Advanced Brushed and Brushless Digital Motor Controllers

User Manual

V3.0, March 8, 2024

visit www.roboteq.com to download the latest revision of this manual

Revision History

Date	Version	Changes
July 10, 2022	3.0	Miscellaneous updates to conform to firmware v3.0
February 10, 2022	2.1a	Miscellaneous updates in order to conform to firmware v2.1a
December 3, 2020	2.1	Miscellaneous updates in order to conform to firmware v2.1
July 8, 2019	2.0	Separated CAN functionality (“CAN Networking Manual”) Separated Microbasic (“Microbasic Scripting Manual”) Separated Roborun+ Utility (“Roborun+ Utility User Manual”) Miscellaneous updates in order to conform to firmware v2.0
August 28, 2017	1.8	Added AC Induction Sections Extended command set
October 15, 2016	1.7	Added Speed Position Mode Major Additions to Brushless Motor Section Added RoboCAN protocol Miscellaneous updates
May 10, 2012	1.2	Added CAN Networking Added Closed Loop Count Position mode, Closed Loop Torque mode Extended command set
January 8, 2011	1.2	Added Brushless Motor Connections and Operation
July 15, 2010	1.2	Extended command set Improved position mode
May 15, 2010	1.1	Added Scripting
January 1, 2010	1.0	Initial release

Introduction

Refer to the Datasheet for Hardware-Specific Issues

This manual is the companion to your controller's datasheet. All information that is specific to a particular controller model is found in the datasheet. These include:

- Number and types of I/O
- Connectors pin-out
- Wiring diagrams
- Maximum voltage and operating voltage
- Thermal and environmental specifications
- Mechanical drawings and characteristics
- Available storage for scripting
- Battery or/and Motor Amps sensing
- Storage size of user variables to Flash or Battery-backed RAM

User Manual Structure and Use

The user manual discusses issues that are common to all controllers inside a given product family. Except for a few exceptions, the information contained in the manual does not repeat the data that is provided in the datasheets.

For CAN please refer to "CAN Networking Manual". For Modbus please refer to "Modbus Manual". For Microbasic scripting please refer to "Microbasic Scripting Manual". For Roborun+ Utility please refer to "Roborun+ Utility User Manual". This manual is divided into 15 sections organized as follows:

SECTION 1 Connecting Power and Motors to the Controller

This section describes the power connections to the battery and motors, the mandatory vs. optional connections. Instructions and recommendations are provided for safe operation under all conditions.

SECTION 2 Safety Recommendations

This section lists the possible motor failure causes and provides examples of prevention methods and possible ways to regain control over motor if such failures occur.

SECTION 3 Connecting Sensors and Actuators to Input/Outputs

This section describes all the types of inputs that are available on all controller models and describes how to attach sensors and actuators to them. This section also describes the connection and operation of optical encoders.

SECTION 4 I/O Configuration and Operation

This section details the possible use of each type of Digital, Analog, Pulse or Encoder inputs, and the Digital Outputs available on the controller. It describes in detail the software configurable options available for each I/O type.

SECTION 5 Roboteq Products Connection and Operation

This section discusses how to interface one or more Roboteq's products (MGS1600, BMS1040, etc.) to the motor controller.

SECTION 6 Command Modes

The controller can be operated using serial, analog or pulse commands. This section describes each of these modes and how the controller can switch from one command input to another. Detailed descriptions are provided for the RC pulse and Analog command modes and all their configurable options.

SECTION 7 Motor Operating Features and Options

This section reviews all the configurable options available to the motor driver section. It covers global parameters such as PWM frequency, overvoltage, or temperature-based protection, as well as motor channel-specific configurations. These include amps limiting, acceleration/deceleration settings, or operating modes.

SECTION 8 Brushless Motor Connections and Operation

This section addresses the installation and operating issues specific to brushless motors. It is applicable only to brushless motor controller models.

SECTION 9 AC Induction MotorOperation

This section discusses the controller's operating features and options when using three-phase AC Induction motors.

SECTION 10 Closed Loop Speed and Speed Position Modes

This section focuses on the closed loop speed mode with feedback using analog speed sensors or encoders. Information is provided on how to setup a closed loop speed control system, tune the PID control loop, and operate the controller.

SECTION 11 Closed Loop Relative and Tracking Position Modes

This section describes how to configure and operate the controller in position mode using analog, pulse, or encoder feedback. In position mode, the motor can be made to smoothly go from one position to the next. Information is provided on how to setup a closed loop position system, tune the PID control loop, and operate the controller.

SECTION 12 Closed Loop Count Position Mode

This section describes how to configure and operate the controller in Closed Loop Count Position mode. Position command chaining is provided to ensure seamless motor motion.

SECTION 13 Closed Loop Torque Mode

This section describes how to select, configure and operate the controller in Closed Loop Torque mode.

SECTION 14 Serial (RS232/RS485/USB/TCP) Operation

This section describes how to communicate to the controller via the RS232, RS485, USB or TCP interface.

SECTION 15 Commands Reference

This section lists and describes in detail all configuration parameters, runtime commands, operating queries, and maintenance commands available in the controller.

SECTION 1

Connecting Power and Motors to the Controller

This section describes the controller’s connections to power sources and motors.

This section does not show connector pin-outs or wiring diagram. Refer to the datasheet for these.

Important Warning

The controller is a high power electronics device. Serious damage, including fire, may occur to the unit, motor, wiring, and batteries as a result of its misuse. Please follow the instructions in this section very carefully. Any problem due to wiring errors may have very serious consequences and will not be covered by the product’s warranty.

Important Note

All products are not serviceable. If damage is suspected, the item must be replaced rather than repaired. Attempting to service or repair the product voids any existing warranty and may pose safety risks.

Consult customer support for more information on replacements.

Power Connections

Power connections are described in the controller model’s datasheet. Depending on the model type, power connection is done via wires, fast-on tabs, screw terminals or copper bars coming out of the controller.

Controllers with wires as power connections have Ground (black), VMot (red) power cables and a Power Control wire (yellow). The power cables are located at the back end of the controller. The various power cables are identified by their position, wire thickness and color: red is positive (+), black is negative or ground (-).

Controllers with tabs, screw terminals or copper bars have their connector identified in print on the controller.

Controller Power

The controller uses a flexible power supply scheme that is best described in Figure 1-1. In this diagram, it can be seen that the power for the Controller's internal microcomputer is separate from this of the motor drivers. The microcomputer circuit is connected to a DC/DC converter which takes power from either the Power Control input or the VMot input. A diode circuit that is included in most controller models, is designed to automatically select one power source over the other and lets through the source that has the highest voltage.

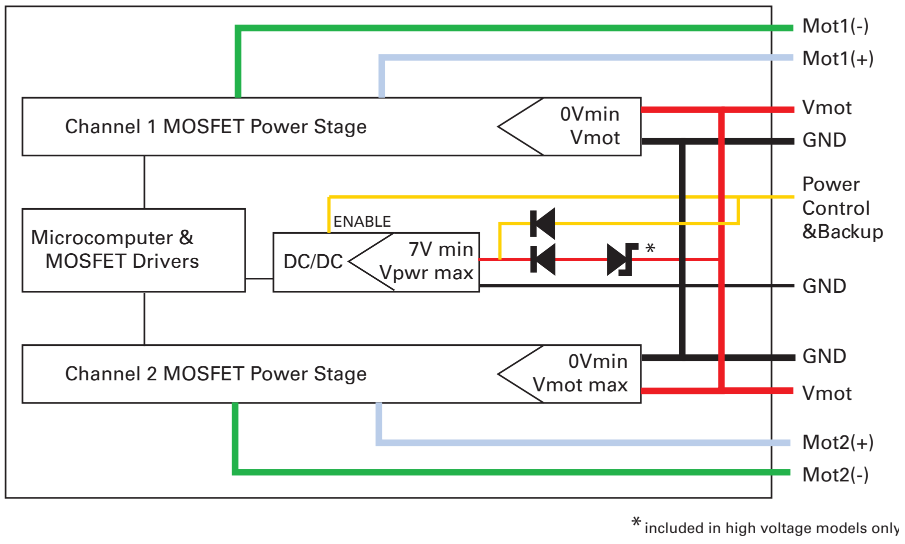


FIGURE 1-1. Representations of the controller's Internal Power Circuits

When powered via the Power Control input only, the controller will turn On, but motors will not be able to turn until power is also present on the VMot wires or Tab.

The Power Control input also serves as the Enable signal for the DC/DC converter. When floating or pulled to above 1V, the DC/DC converter is active and supplies the controller's microcomputer and drivers, thus turning it On. When the Power Control input is pulled to Ground, the DC/DC converter is stopped and the controller is turned Off.

The Power Control input **MUST** be connected to Ground to turn the Controller Off. For turning the controller On, even though the Power Control may be left floating, whenever possible pull it to a 12V or higher voltage to keep the controller logic solidly On. You may use a separate battery to keep the controller alive as the main Motor battery discharges.

On the high voltage controller that is rated above 60V, a zener diode is inserted between the VMot supply and the DC/DC converter. This causes a voltage drop that keeps the voltage at the converter's input within its maximum operating range. However, this diode also increases by around 20V the low voltage threshold at which the controller will start operating when powered from VMot alone.

The table below shows the state of the controller depending on the voltage applied to Power Control and VMot.

TABLE 1-1. Controller Status depending on Power Control and VMot

Power Control input is connected to	**And Main Battery Voltage is**	**Action**
Ground	Any Voltage	Controller is Off. **Required Off Configuration.**
Floating	0V	Controller is Off. **Not Recommended Off Configuration.**
Floating	Above VMotMin (1)	Controller is On. Power Stage is Active (2)
7V to max PwrCtl (3) Volts	Any Voltage	Controller is On. Power Stage is Active (2)
Note 1: VMotMin = 7V on all controller rated up to 60V. VMotMin = 28V on all controllers rated above 60V. See product datasheet Note 2: Power Stage is active but turned off when overvoltage or undervoltage condition. Note 3: 35V max on 30V controllers. 60V max on all products rated above 30V		

Note: All ground terminals (-) are connected to each other inside the controller. On dual channel controllers, the two VMot main battery wires are also connected to each other internally. However, you must never assume that connecting one wire of a given battery potential will eliminate the need to connect the other. When pre-charging the controller's capacitors, the Power Control input must be grounded.

Controller Powering Schemes

Roboteq controllers operate in an environment where high currents may circulate in unexpected manners under certain condition. Please follow these instructions. Roboteq reserves the right to void product warranty if analysis determines that damage is due to improper controller power connection.

The example diagram on Figure 1-2 on page 24 shows how to wire the controller and how to turn power On and Off. All Roboteq models use a similar power circuit. See the controller datasheet for the exact wiring diagram for your controller model.

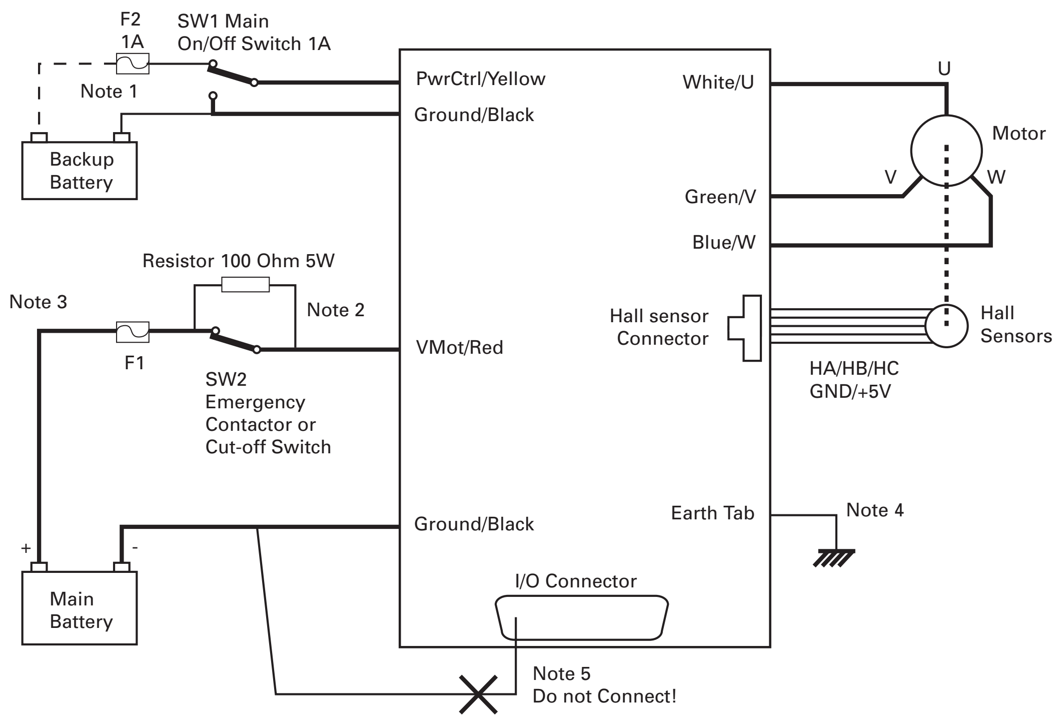


FIGURE 1-2. Brushless DC Controller power wiring diagram

Mandatory Connections

It is imperative that the controller is connected as shown in the wiring diagram provided in the datasheet in order to ensure safe and trouble-free operation. All connections shown as thick black lines are mandatory.

- Connect the thick black wire(s) or the ground terminal to the minus (-) terminal of the battery that will be used to power the motors. Connect the thick red wire(s) or VMot terminal to the plus (+) terminal of the battery. The motor battery may be of 12V up to the maximum voltage specified in the controller model datasheet.
- The controller must be powered On/Off using switch SW1on the Power Control wire/terminal. Grounding this line powers Off the controller. Floating or pulling this line to a voltage will power On the controller. (SW1 is a common SPDT 1 Amp or more switch).
- Use a suitable high-current fuse F1 as a safety measure to prevent damage to the wiring in case of a major controller malfunction. (Littlefuse ATO or MAXI series).
- The battery must be connected in permanence to the controller's Red wire(s) or VMot terminal via a high-power emergency switch SW2 as an additional safety measure. Partially discharged batteries may not blow the fuse, while still having enough power left to cause a fire. Leave the switch SW2 closed at all times and open only in case of an emergency. Use the main On/Off switch SW1 for normal operation. This will prolong the life of SW2, which is subject to arcing when opening under high current with the consequent danger of contact welding.
- If installing in an electric vehicle equipped with a Key Switch where SW2 is a contactor, and the key switch energizes the SW2 coil, then implement SW1 as a relay. Connect the Key Switch to both coils of SW1 and SW2 so cutting off the power to the vehicle by the key switch and SW2 will set the main switch SW1 in the OFF position as well.

Connection for Safe Operation with Discharged Batteries (note 1)

The controller will stop functioning when the main battery voltage drops below 7V. To ensure motor operation with weak or discharged batteries, connect a second battery to the Power Control wire/terminal via the SW1 switch. This battery will only power the controller's internal logic. The motors will continue to be powered by the main battery while the main battery voltage is higher than the secondary battery voltage.

Use precharge Resistor to prevent switch arcing (note 2)

Insert a 100Ohm, 5W resistor across the SW2 Emergency Switch. This will cause the controller's internal capacitors to slowly charge and maintain the full battery voltage by the time the SW2 switch is turned on and thus eliminate damaging arcing to take place inside the switch. Make sure that the controller is turned Off with the Power Control wire grounded while the SW2 switch is off. The controller's capacitors will not charge if the Power Control wire is left floating and arcing will then occur when the Emergency switch is turned on.

Protection against Damage due to Regeneration (notes 3)

The voltage generated by motors rotating while not powered by the controller can cause serious damage even if the controller is Off or disconnected.

- Use the main SW1 switch on the Power Control wire/terminal to turn Off and keep Off the controller. In this way the controller cannot be powered up under any unwanted circumstances.
- Countermeasures should be taken to deal with any regeneration power if the battery or BMS system does not support energy to return back to it.
- Disconnecting the controller from the battery while motors are rotating could lead to a serious damage. In this case a regeneration brake system is needed.

Connect Case to Earth if connecting AC equipment (note 4)

If building a system which uses rechargeable batteries, it must be assumed that periodically a user will connect an AC battery charger to the system. Being connected to the AC main, the charger may accidentally bring AC high voltage to the system's chassis and to the controller's enclosure. A similar danger exists when the controller is powered via a power supply connected to the mains.

Some controller models in metallic enclosures are supplied with an Earth tab, which permits earthing the metal case. Connect this tab to a wire connected to the Earth while the charger is plugged in the AC main, or if the controller is powered by an AC power supply or is being repaired using any other AC equipment (PC, Voltmeter etc.)

Avoid Ground loops when connecting I/O devices (note 5)

When connecting a PC, encoder, switch or actuators on the I/O connector, be very careful that you do not create a path from the ground pins on the I/O connector and the battery minus terminal. Should the controller's main Ground wires (thick black) or terminals be disconnected while the VMot wires (thick red) or terminals are connected, the high current would flow from the ground pins, potentially causing serious damage to the controller and/or your external devices.

- Do not connect a wire between the I/O connector ground pins and the battery minus terminal. Look for hidden connection and eliminate them.
- Have a very firm and secure connection of the controller ground wire and the battery minus terminal.
- Do not use connectors or switches on the power ground cables.

Important Warning

Do not rely on cutting power to the controller for it to turn Off if the Power Control is left floating. If motors are spinning because the robot is pushed or because of inertia, they will act as generators and will turn the controller On, possibly in an unsafe state. ALWAYS ground the Power Control wire terminal to turn the controller Off and keep it Off.

Important Warning

Unless you can ensure a steady voltage that is higher than 7V in all conditions, it is recommended that the battery used to power the controller's electronics be separate from the one used to power the motors. This is because it is very likely that the motor batteries will be subject to very large current loads which may cause the voltage to eventually dip below 7V as the batteries' charge drops. The separate backup power supply should be connected to the Power Control input.

Connecting the Motors

Refer to the datasheet for information on how to wire the motor(s) to a particular motor controller model.

After connecting the motors, apply a minimal amount of power using the Roborun PC utility with the controller configured in **Open Loop speed mode**. Verify that the motor spins in the desired direction. Immediately stop and swap the motor wires if not.

In Closed Loop Speed or Position mode, beware that the motor polarity must match this of the feedback. If it does not, the motors will runaway with no possibility to stop other than switching Off the power. The polarity of the Motor or of the feedback device may need to be changed.

Important Warning

Make sure that your motors have their wires isolated from the motor casing. Some motors, particularly automotive parts, use only one wire, with the other connected to the motor's frame. If you are using this type of motor, make sure that it is mounted on isolators and that its casing will not cause a short circuit with other motors and circuits which may also be inadvertently connected to the same metal chassis.

Single Channel Operation

Dual channel controllers may be ordered with the -S (Single Channel) suffix.

The two channel outputs must be paralleled as shown in the datasheet so that they can drive a single load with twice the power. To perform in this manner, the controller's Power Transistors that are switching in each channel must be perfectly synchronized. Without this synchronization, the current will flow from one channel to the other and cause the destruction of the controller.

This synchronized function is achieved by utilizing a special firmware.

Important Warning

Before pairing the outputs, attach the motor to one channel and then the other. Verify that the motor responds the same way to command changes.

Power Fuses

Power fuses are mandatory, and one should be installed on each power input of the drive. Each fuse must be rated for the maximum expected current of the application. For more details, please consult the motor drive's datasheet.

Caution

Fuses, inherently slow to respond, accommodate normal surge currents during motor operations, such as acceleration and braking. However, due to their delayed reaction, they cannot shield the drive from surges that rise to harmful levels, which might lead to damage. Primarily, fuses are designed to interrupt the electrical circuit in the event of a permanent short circuit, a situation that often suggests the drive has sustained prior damage.

Wire Length Limits

The controller regulates the output power by switching the power to the motors On and Off at high frequencies. At such frequencies, the wires' inductance produces undesirable effects such as parasitic RF emissions, ringing, and overvoltage peaks. The controller has built-in capacitors and voltage limiters that will reduce these effects. However, should the wire inductance be increased, for example by extended wire length, these effects will be amplified beyond the controller's capability to correct them. This is particularly the case for the main battery power wires.

Important Warning

Avoid long connection between the controller and power source, as the added inductance may cause damage to the controller when operating at high currents. Try extending the motor wires instead since the added inductance is not harmful on this side of the controller.

If the controller must be located at a long distance from the power source, the effects of the wire inductance may be reduced by using one or more of the following techniques:

- Twisting the power and ground wires over the full length of the wires
- Use the vehicle's metallic chassis for ground and run the positive wire along the surface
- Add a capacitor (10,000uF or higher) near the controller

Electrical Noise Reduction Techniques

As discussed in the above section, the controller uses fast switching technology to control the amount of power applied to the motors. While the controller incorporates several circuits to keep electrical noise to a minimum, additional techniques can be used to keep the noise low when installing the controller in an application. Below is a list of techniques you can try to keep noise emission low:

- Keep wires as short as possible
- Loop wires through ferrite cores
- Add snubber RC circuit at motor terminals
- Keep controller, wires, and battery enclosed in a metallic body

Battery Current vs. Motor Current

The controller limits the current that flows through the motors and not the battery current. Current that flows through the motor is typically higher than the battery current. This counter-intuitive phenomenon is due to the "flyback" current in the motor's inductance. In some cases, the motor current can be extremely high, causing heat and potentially damage while battery current appears low or reasonable.

The motor's power is controlled by varying the On/Off duty cycle of the battery voltage 16,000 times per second to the motor from 0% (motor off) to 100 (motor on). Because of the inductive flyback effect, during the Off time current continues to flow at nearly the same peak - and not the average - level as during the On time. At low PWM ratios, the peak current - and therefore motor current - can be very high as shown in Figure 1-4, on next page.

The relation between Battery Current and Motor current is given in the formula below:

Motor Current = Battery Current / PWM ratio

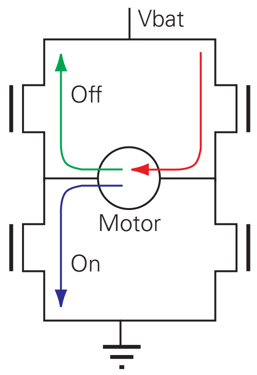


FIGURE 1-3. Current flow during operation

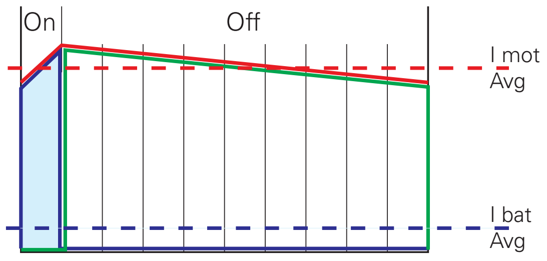


FIGURE 1-4. Instant and average current waveforms

The relation between Battery Current and Motor current is given in the formula below:

Motor Current = Battery Current / PWM Ratio

Example: If the controller reports 10A of battery current while at 10% PWM, the current in the motor is 10 / 0.1 = 100A.

Power Regeneration Considerations

When a motor is spinning faster than it would normally at the applied voltage, such as when moving downhill or decelerating, the motor acts as a generator. In such cases, the current will flow in the opposite direction, back to the power source.

It is therefore essential that the controller is connected to rechargeable batteries. If a power supply is used instead, the current will attempt to flow back in the power supply during regeneration, potentially damaging it and/or the controller.

Regeneration can also cause potential problems if the battery is disconnected while the motors are still spinning. In such a case, the energy generated by the motor will keep the controller On, and depending on the command level applied at that time, the regenerated current will attempt to flow back to the battery. Since none is present, the voltage will rise to potentially unsafe levels. The controller includes an overvoltage protection circuit to prevent damage to the output transistors (see "Using the Controller with a Power Supply" below). However, if there is a possibility that the motor could be made to spin and generate a voltage higher than 30V, a path to the battery must be provided, even after a fuse is blown. This can be accomplished by inserting a diode across the fuse as shown in Figure 1-2 on page 24.

Please download the Application Note "Understanding Regeneration" from the www.roboteq.com for an in-depth discussion of this complex but important topic.

Important Warning

Use the controller only with a rechargeable battery as supply to the Motor Power wires (thick black and red wires). If a transformer or power supply is used, damage to the controller and/or power supply may occur during regeneration. See “Using the Controller with a Power Supply” below for details.

Important Warning

Avoid switching Off or cutting open the main power cables while the motors are spinning. Damage to the controller may occur. Always ground the Power Control wire to turn the controller Off.

Using the Controller with a Power Supply

Using a transformer or a switching power supply is possible but requires special care, as the current will want to flow back from the motors to the power supply during regeneration. As discussed in “Power Regeneration Considerations” above, if the supply is not able to absorb and dissipate regenerated current, the voltage will increase until the over-voltage protection circuit cuts off the motors. While this process should not be harmful to the controller, it may be to the power supply, unless one or more of the protective steps below is taken:

- Use a power supply that will not suffer damage in case a voltage is applied at its output that is higher than its own output voltage. This information is seldom published in commercial power supplies, so it is not always possible to obtain positive reassurance that the supply will survive such a condition.
- Avoid deceleration that is quicker than the natural deceleration due to the friction in the motor assembly (motor, gears, load). Any deceleration that would be quicker than natural friction means that braking energy will need to be taken out of the system, causing a reverse current flow and voltage rise.
- Place a battery in parallel with the power supply output. This will provide a reservoir into which regeneration current can flow. It will also be very helpful for delivering high current surges during motor acceleration, making it possible to use a lower current power supply. Batteries mounted in this way should be connected for the first time only while fully charged and should not be allowed to discharge. The power supply will be required to output unsafe amounts of current if connected directly to a discharged battery. Consider using a decoupling diode on the power supply’s output to prevent battery or regeneration current to flow back into the power supply.
- Place a shunt regulator such as Roboteq’s SR5K. This device will monitor the voltage at the controller and place resistive load in parallel with the power supply in order to absorb the regenerated current. The diagram below shows to wire the SR5K shunt regulator.

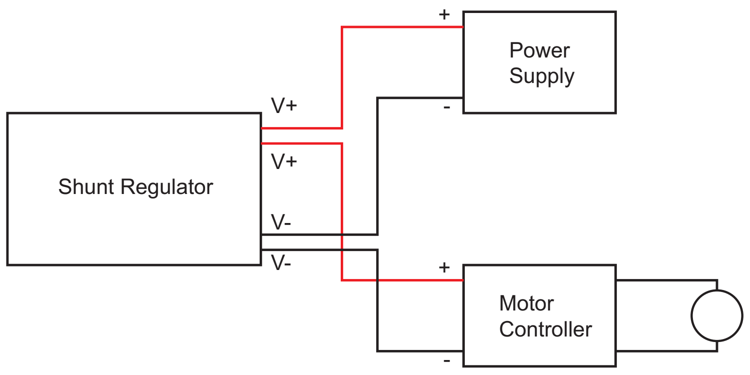


FIGURE 1-5. Shunt Regulator wiring

Note: The schematic above is provided for reference only. It may not work in all conditions.

SECTION 2

Safety Recommendations

In many applications, Roboteq controllers drive high power motors that move parts and equipment at high speed and/or with very high force. In case of malfunction, potentially enormous forces can be applied at the wrong time and/or wrong place causing serious damage to property and/or harm to a person. While Roboteq controllers operate very reliably, and failures are rare, a failure is possible as with any other electronic equipment. If there is any danger that a loss of motor control can cause damage or injury, you must plan on that possibility and implement methods for stopping the motor **independently of the controller operation**.

Below is a list of failure categories, their effect and possible ways to regain control, or minimize the consequences. The list of possible failures is not exhaustive and the suggested prevention methods are provided as examples for information only.

Important Safety Disclaimer

Dangerous uncontrolled motor runaway condition can occur for a number of reasons, including, but not limited to command or feedback wiring failure, configuration error, faulty firmware, errors in user MicroBasic script or in the user program, or controller hardware failure. The user must assume that such failures can occur and must take all measures necessary to make his/her system safe in all conditions. The information contained in this manual, and in this section in particular, is provided for information only. Roboteq will not be liable in case of damage or injury as a result of product misuse or failure.

Possible Failure Causes

The dangerous unintended motor operation could occur for a number of reasons, including, but not limited to:

- Failure in Command device
- Feedback sensors malfunction
- Wiring errors or failure
- Controller configuration error
- Faulty firmware
- Errors or oversights in user MicroBasic scripts
- Controller hardware failure

Motor Deactivation in Normal Operation

In normal operation, the controller is always able to turn off the motor if it detects faults or if instructed to do so from an external command.

There are three kinds of actions in case of faults.

1. **Quick Stop Action:** This action is used in case of faults that have nothing to do with motor control and thus they do not require the instant cut of the motor power. The controller will react to these faults by ramping down the speed of the motor from its current value to 0 RPM, following a deceleration defined by the "Fault Deceleration" (EDEC) parameter. Please note that Quick Stop will switch the system to Speed mode to decelerate the motor. For proper operation, the Speed mode must be configured with the appropriate PID gains. This Action is used for the following faults:
    - Quick Stop, either commanded (QST), or triggered as an action.
    - Dead Man Switch, triggered as an action.
    - Command watchdog expiration
    - Loop Error Detection
2. **Disabling the power stage:** This action is used for critical failures that require the immediate cut of the motor power. To do so, the controller will switch OFF all the H - bridge MOSFETs, preventing any additional power to be applied to the motor. The motor phases will be essentially disconnected, and the motor will be freewheeling until it stops due to its mechanical power loses (vehicle's weight, friction of the moving parts). This action is used for the following faults:
    - Emergency Stop, either commanded (EX), or triggered as an action.
    - Overvoltage
    - Undervoltage
    - Short
    - Sensor Error Detection
    - MOSFail
3. **Electrical Brake:** Contrary to Quick stop, that will break the motor following the defined deceleration, and the disabled power stage, that will leave the motor in free-wheeling, electrical brake will stop the motor in a quick way, by shorting its phases together. To achieve that, the controller will turn ON all bottom MOSFETs. Depending on the speed of the motor - and therefore its back emf - high current surges can be developed that can be harmful for the controller. To avoid that, the motor should be in a relatively low speed when performing electrical braking. If the application requires electrical braking to be performed as a safety action, the regenerative power should be calculated to ensure that is harmless for the controller. This action is used for the following faults:
    - Limit Switches
    - Stall Detection

Important Note

Quick Stop will switch the system to Speed mode to decelerate the motor. For proper operation, the Speed mode must be configured with the appropriate PID gains.

Important Warning:

While cutting the power to the motors is generally the best thing to do in case of major failure, it may not necessarily result in a safe situation.

Motor Deactivation in Case of Output Stage Hardware Failure

The power stage of each controller is composed of 4 or 6 MOSFET paths depending on the motor that they are designed to drive (brushed, brushless or induction motors). In case that a controller's MOSFTET is damaged, it can be permanently ON or OFF without the controller having the ability to change its state. Depending on which MOSFETs are damaged and their failure modes (open or shorted), different faulty conditions can occur. Some of them will not affect the motor when it is idle, but other will permanently short circuit the power stage or even supply constant voltage to the motor windings.

The figures below show all the possible combinations of shorted MOSFETs switches in a brushed DC motor controller.

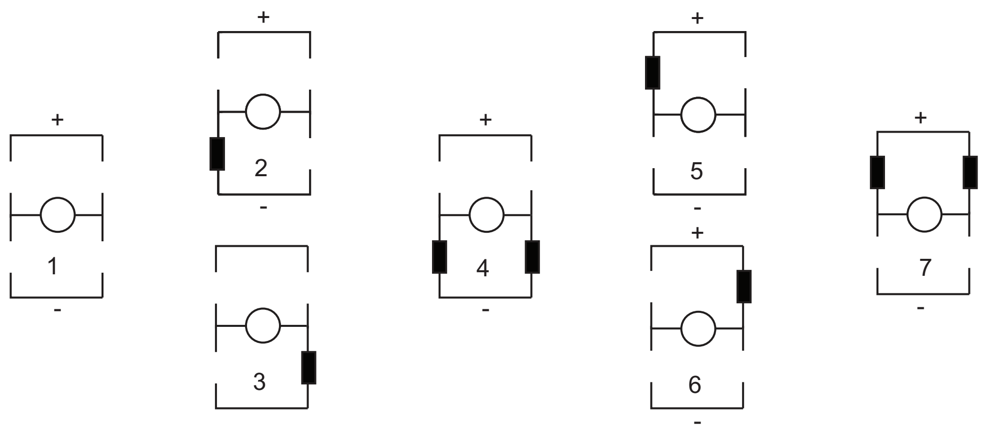


FIGURE 2-1. MOSFET Failures resulting in no motor activation

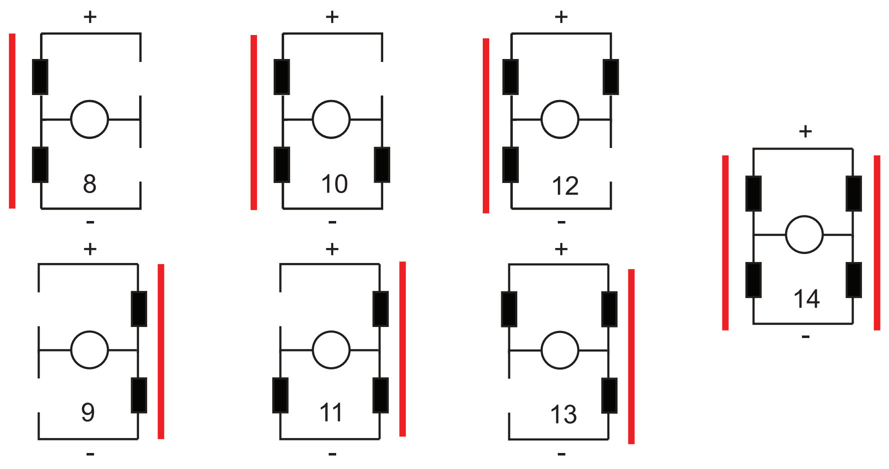


FIGURE 2-2. MOSFET Failures resulting in battery short circuit and no motor activation

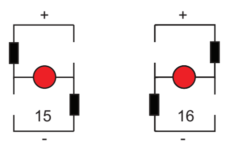


FIGURE 2-3. MOSFET Failure resulting in motor activation

Only in DC controllers, the motor can run out of control regardless of, if the controller is ON or OFF. While these failures conditions (15 and 16) are rare, users must take them into account and provide means to cut all power to the controller's power stage when they occur. The same precautions must be followed on brushless and IM controllers in case that the power stage is short circuited.

The controller will check for shorted MOSFETs every time it powers ON or whenever the STT command is executed (see **STT - STO Self-Test**, page 208). If either of the MOSFETs is found to be shorted (which is usually the initial condition of a damaged MOSFET), the controller will react by disabling the power stage and setting the respective flags (MOSFail bit of the fault flags, EStop and FETs Off bits of the Status flags). Disabling the power stage cannot prevent a battery short circuit if both top and bottom MOSFETs are shorted, so the control logic of the system should detect that condition and act accordingly. The MOSfail alarm will be cleared only if the MOSFET failure goes away and the self-test gets executed (either after the STT command or after power-up).

The MOSFET test can be by-passed by configuration with the SFTS - Safety Switch Connected command, in case a safety switch is connected between the UVW connectors of the controller and the motor.

Manual Emergency Power Disconnect

In systems where the operator is within physical reach of the controller, the simplest safety device is the emergency disconnect switch that is shown in the wiring diagram inside all controller datasheets, and in the example diagram below.

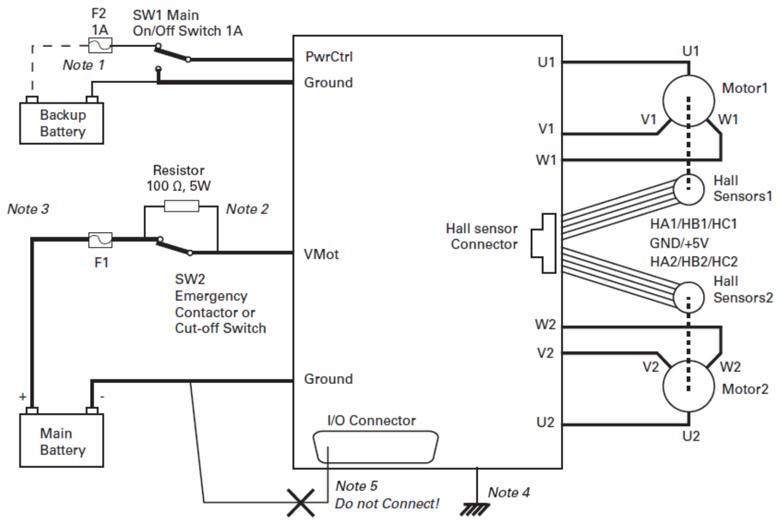


FIGURE 2-4. Example powering diagram

The switch must be placed visibly and be easy to operate. Prefer "mushroom" emergency stop push buttons. Make sure that the switches are rated at the maximum current that can be expected to flow through all motors at the same time.

FIGURE 2-5. "Mushroom" type Emergency Disconnect Switch

Important Note

When the motor is connected to chassis and the chassis is totally isolated from the battery negative terminal (system ground), then it could be transformed to an EMI antenna. For this reason it is highly recommended to use a Y capacitor between the chassis and the battery negative terminal, so as to have the chassis AC coupled to the system ground.

Remote Emergency Power Disconnect

In remote controlled systems, the emergency switch must be replaced by a high power contactor relay as shown in Figure 2-6. The relay must be normally open and be activated using an RC switch on a separate radio channel. The receiver should preferably be powered directly from the system's battery. If powered from the controller's 5V output, keep in mind that in case of a total failure of the controller, the 5V output may or may not be interrupted.

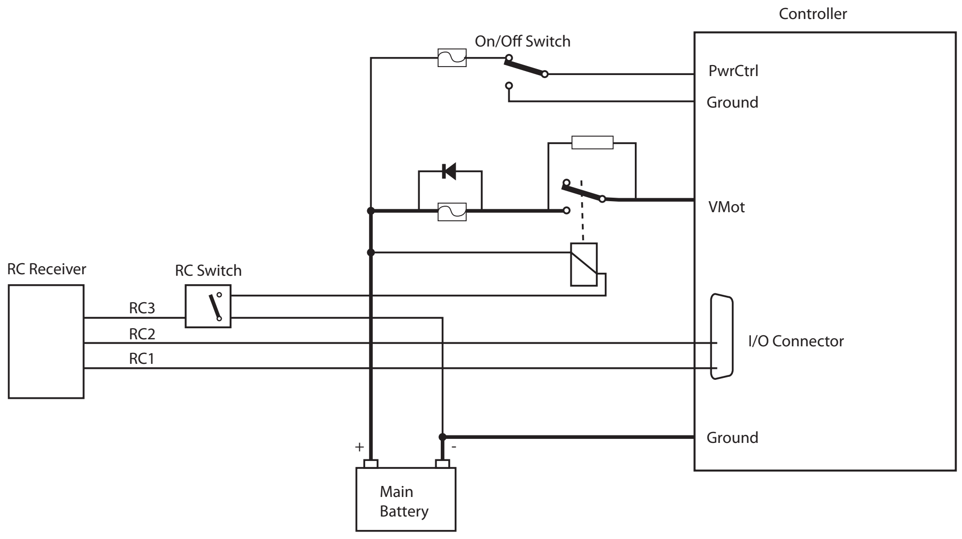


FIGURE 2-6 Example of remotely operated safety disconnect

The receiver must operate in such a way that the contactor relay will be off if the transmitter if off or out of range.

The transmitter should have a visible and easy to reach an emergency switch for the operator. That switch will be used to deactivate the relay remotely. It could also be used to shutdown entirely the transmitter, assuming it is determined for certain that this will deactivate the relay at the controller.

Protection using Supervisory Microcomputer

In applications where the controller is commanded by a PC, a microcomputer or a PLC, that supervisory system could be used to verify that the controller is still responding and cut the power to the controller's power stage in case a malfunction is detected. The supervisory system would only require a digital output or other means to activate/deactivate the contactor relay as shown in the figure below.

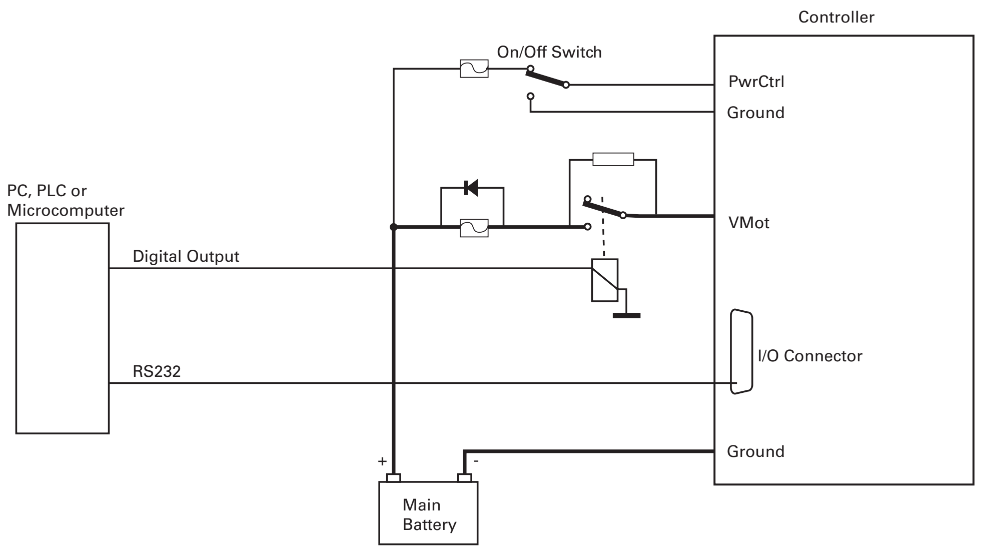


FIGURE 2-7. Example of safety disconnect via supervisory system

Self Protection against Power Stage Failure

If the controller processor is still operational, it can self detect several, although not all, situations where a motor is running while the power stage is off. The figure below shows a protection circuit using an external contactor relay.

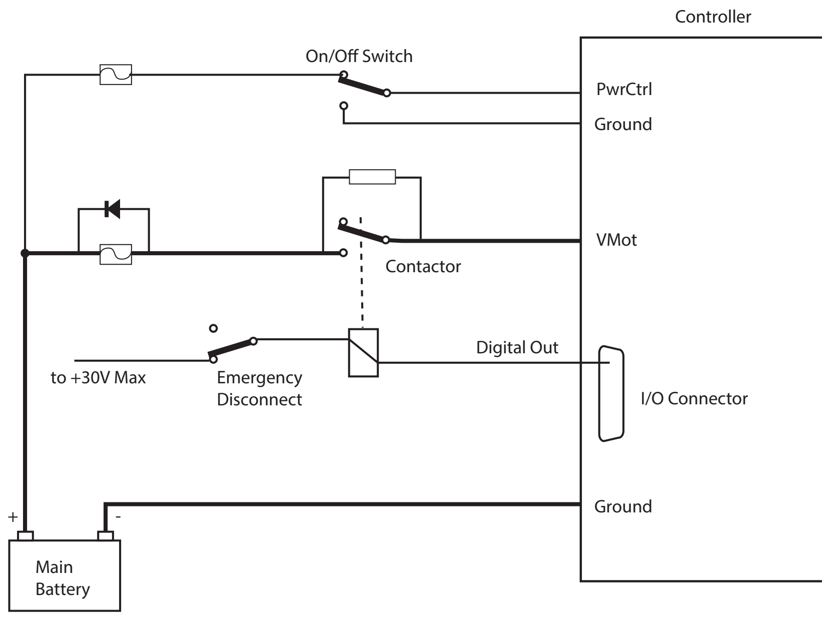


FIGURE 2-8. Self protection circuit using a contactor

Note: Digital outputs are rated 40V max. If the battery voltage is higher than 40V, the relay must be connected to the + of an alternate power source of lower voltage.

The controller must have the Power Control input wired to the battery so that it can operate and communicate independently of the power stage. The controller's processor will then activate the contactor coil through a digital output configured to turn on when the "No MOSFET Failure" condition is true. The controller will automatically deactivate the coil if the output is expected to be off and the battery current is above 500mA to 2.5A (depending on the controller model) for more than 0.5s.

The contactor must be rated high enough so that it can cut the full load current. For even higher safety, additional precaution should be taken to prevent and to detect fused contactor blades.

This contactor circuit will only detect and protect against damaged output stage conditions. It will not protect against all other types of fault. Notice therefore, the presence of an emergency switch in series with the contactor coil. This switch should be operated manually or remotely, as discussed in the Manual Emergency Power Disconnect the Remote Emergency Power Disconnect and the Protection using Supervisory Microcomputer earlier in this section of the manual.

Using this contactor circuit, turning off the controller will normally deactivate the digital output and this will cut the power to the controller's output stage.

Important Warning

Fully autonomous and unsupervised systems cannot depend on electronics alone to ensure absolute safety. While a number of techniques can be used to improve safety, they will minimize but never totally eliminate risks. Such systems must be mechanically designed so that no moving parts can ever cause harm in any circumstances.

Safe Torque-Off (STO)

Safe Torque Off is a safe method for switching controller in a state where no torque is generated, regardless whether the controller is operating normally or is faulty. This function is a mechanism that prevents the drive from restarting unexpectedly. STO has the immediate effect that the drive cannot supply any torque-generating energy. STO can be used wherever the drive will be brought to a standstill in a sufficiently short time by the load torque or friction or where coasting down of the drive is not relevant to safety. STO enables safe working and has a wide range of use in motion control/ systems with moving axes. The advantage of the integrated STO safety function compared with standard safety technology using electromechanical switchgear is the elimination of separate components and the effort that would be required to wire and service them. Because of the rapid electronic switching times, the function has a shorter switching time than the electromechanical components in a conventional solution.

Specific motor controllers implement Safe Torque-Off (STO) circuitry, which is certified from TUV (T-version - Certification No. M6A 104504 0001 Rev. 01). STO is the most common safety function, meant for motor controllers, ensuring that upon trigger no torque will be generated even after the controller power cycle. For controllers without the specific circuit the STO is implemented in firmware alone and digital inputs 1 and 2 are usually used (check controller's datasheet).

Safe Torque Off (STO) on Roboteq Controllers

Two digital inputs on the user IO connector can be used to put the controller in a state where the motor is deprived of energy.

The two inputs, labeled STO1 and STO2 must both be brought and maintained at a logic level 1 for the controller to be active. If any one or both of these lines are at 0, the output is de-energized.

The STO circuit operates independently of the MCU. It will always override the MCU, whether the MCU is processing normally, or is in a hardware of firmware fault condition.

The STO circuit works by controlling the voltage supply to the controller's MOSFET drivers. When both STO1 and STO2 are at logic 1 level, the MOSFET driver are supplied with power. When either or both STO1 and STO2 are at logic level 0, the MOSFET driver power is cut, and the MOFETs gates can no longer be above the ON threshold level, regardless of the MCU activity.

Accordingly, the STO circuit is built with redundancy and will continue to function if any one component is faulty, anywhere in the STO circuit or elsewhere in the controller.

Soft-STO inputs

On controllers that do not have the certified STO circuitry, a soft STO implementation is available, which clones the principles of the real STO without of course being certified.

In that case all the available digital inputs have an extra action "Soft STO". Two of them need to have this action activated and they will act as STO inputs.

Activating STO

By factory default STO is disabled. It must be enabled by removing the jumper located on the controller's PCB. STO is activated by removing power (logic level 0) to both STO inputs. In order for controller to monitor STO state, this function must be activated through Roborun+ Utility or serial command (check command section). The controller will immediately stop generation of torque in the motor. STO functionality is only available in the T version of the controller.

Deactivating STO

STO is deactivated by applying a voltage (logic level 1) to both STO inputs. After this a new start command has to be given to turn the motor. It can be disabled by connecting the jumper located on the controller's PCB. After that user should disable the function through Roborun+ Utility or serial command (check command section).

Constraints when using STO

- The STO feature is only approved for use with Brushless DC or AC Induction motors.
- All voltages attached to the controller need to fulfil SELV or PELV requirements.
- After first installation and at least every 3 months the test as described in chapter 14 has to be conducted.

STO Failure Messages

In case a failure is detected in the STO circuit the following failure message will be visible according on how the user operates the controller.

1. Status LED on Controller

   The status LED pattern will be the below in case of STO failure

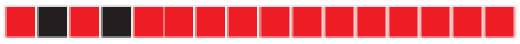
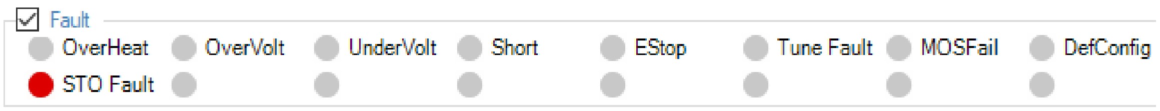


FIGURE 2-9. STO fault status LED pattern

2. STO Fault LED at Roborun+ utility in failure



☑ Fault

OverHeat	OverVolt	UnderVolt	Short	EStop	Tune Fault	MOSFail	DefConfig
● STO Fault							

FIGURE 2-10. STO fault position in Fault Flags

This failure can have several internal and external reasons. If the failure is shown please check the cabling and the signals to STO 1 and STO 2. Both signals much have at all times the same level.

- Check that the STO jumper is set correctly and STO is configured correctly
- Check wiring
- Check cabling for short circuits or open circuits

If the failure persists, contact Roboteq support.

IMPORTANT WARNING

Same status LED pattern is used for undervoltage and overvoltage faults and that should not be confusing. If STO fault appear it is normal for the controller voltage to be off and undervoltage fault to trigger. Either way this Status LED pattern indicates a situation that should be treated with caution.

Firmware implementation

The STO circuit will operate regardless of the MCU activity. However, when operating normally the MCU will perform the following functions:

1. Self-test that the STO circuits, switches and the power MOSFETs are functional. This is done every time the controller is powered on. It can also be done at any time using command STT (see **STT - STO Self-Test**, page 208) from the system's PC or PLC. The self-test can also be initiated by the controller itself using its scripting language, at periodic time intervals, or any other user-define rule(s).

   If the self-test fails, the controller will stop driving the MOSFETs and set a fault flag that can be monitored by the PLC/Computer. It can also activate one of its digital outputs to indicate the fault.

2. The STO inputs are monitored continuously every 1ms. If one or both STO inputs are at level zero and the MOSFET driver supply voltage has not dropped, an STO fault is detected. The STO fault flag is set. A user digital output can be activated to indicate the fault.

Installation – Maintenance

The STO circuit needs to be tested before first installation and at least every 3 months according to the below sequence:

1. Activate STO (both logic level 0)
2. Check that STO is active (through Roborun+ Utility/Serial)
3. Check that there is no STO fault present (through Roborun+ Utility/Serial)
4. Deactivate STO 1 and STO 2 (both logic level 1)
5. Check that STO is not active and that there is no STO fault present (through Roborun+ Utility/Serial).

SAFETY INSTALLATION

It is required that the controller has to be placed in an enclosure that can provide IP54 protection.

SAFETY REGISTRATION

It is required that STO end user should follow up www.roboteq.com **and register to site/forum for any news about safety function.**

STO Voltage source specification attention

In order to have maximum response at STO implementation, user/installer/integrator should use voltage source with low output capacitance. In any other case, latency in activation might occur.

Compliance and Safety Metrics

The STO function is compliant to:

- IEC 61800-5-2:2007, SIL 3
- IEC 61508:2010, SIL 3
- IEC 62061:2005, SIL 3
- ISO 13849-1:2015, Category 3 Performance Level e

TABLE 2-1. STO compliance and safety metrics

Metric acc. To IEC 61508, IEC 61800-5-2, IEC 62061	Value
SIL	Up to 3
PFH	5 FIT
Mission Time and Proof Test Interval	20 years
Performance Level	e
Category	3
MTTFD	>100 years

Technical Data

TABLE 2-2. STO technical data

Specification	Value
STO Input High Level	6V to 30V
STO Input Low Level	0V to 1V
STO Response Time	< 5msec
Operating Temperature	-20°C to 55°C
Storage Temperature	-20°C to 70°C
IP degree	IP40
Humidity	5% to 95% non-condensing
Maximum altitude	2000m
STO cable length	≤ 3m (1)
EMC immunity	According to IEC 61800-3:2017 and IEC 61800-5-2:2007 Annex E
CE Declaration of conformity	Available at www.roboteq.com
All connected cables must have length <3m	

SECTION 3

Connecting Sensors and Actuators to Input/Outputs

This section describes the various inputs and outputs and provides guidance on how to connect sensors, actuators or other accessories to them.

Controller Connections

The controller uses a set of power connections DSub and plastic connectors for all necessary connections.

The power connections are used for connecting to the batteries and motor, and will typically carry large current loads. Details on the controller's power wiring can be found at "Connecting Power and Motors to the Controller" section of this manual.

The DSub and plastic connectors are used for all low-voltage, low-current connections to the Radio, Microcontroller, sensors, and accessories. This section covers only the connections to sensors and actuators.

For information on how to connect the RS232 or the RS485 ports, see "Serial (RS232/ RS485/TCP/USB) Operation" section.

The remainder of this section describes how to connect sensors and actuators to the controller's low-voltage I/O pins that are located on the DSub and plastic connectors.

Controller’s Inputs and Outputs

The controller includes several inputs and outputs for various sensors and actuators. Depending on the selected operating mode, some of these I/Os provide command, feedback and/or safety information to the controller.

When the controller operates in modes that do not use these I/Os, these signals are ignored or can become available via the RS232/RS485/TCP/USB port for user application. Below is a summary of the available signals and the modes in which they are used by the controller. The actual number of the signal of each type, voltage or current specification, and their position on the I/O connector is given in the controller datasheet.

TABLE 3-1. Controller’s IO signals and definitions

Signal	**I/O type**	**Use/Activation**
DOUT1 to DOUTn	Digital Output	- Activated when motor(s) is powered - Activated when motor(s) is reversed - Activated when overtemperature - Activated when overvoltage - Mirror Status LED - Deactivates when output stage fault - User activated (RS232/RS485/TCP/USB or via scripting)
DIN1 to DINn	Digital Input	- Quick Stop - Emergency stop - Motor Stop (deadman switch) - Invert motor direction - Forward or reverse limit switch - Run MicroBasic Script - Load Home counter - Soft STO
AIN1 to AINn	Analog Input	- Command for the motor(s) - Speed or position feedback - Trigger Action similar to Digital Input if under or over user-selectable threshold - Motor Thermistor use
PIN1 to PINn	Pulse Input	- Command for the motor(s) - Speed or position feedback - Trigger Action similar to Digital Input if under or over user selectable threshold
ENC1a/b to ENC2a/b	Encoder Inputs	- Speed or position feedback - Trigger action similar to Digital Input if under or over user-selectable count threshold

Connecting devices to Digital Outputs

Depending on the controller model, 2 to 8 Digital Outputs are available for multiple purposes. The Outputs are Open Drain MOSFET type capable of driving loads up to 30V/1A. For selected models the digital outputs can be configured in order to be high side drivers at 5 Volts or at 24 Volts (For more details see configuration commands **DOT - Digital Output Type and AUXV - Digital Output High Side Drive Voltage Level** and the respective controller's datasheet).

At the Open Drain configuration, the output will be pulled to ground when activated. Thus, the load must be connected between the controller's output and a positive voltage source (e.g. a 24V battery).

Connecting Resistive Loads to Outputs

Resistive or other non-inductive loads can be connected simply as shown in the diagram below.

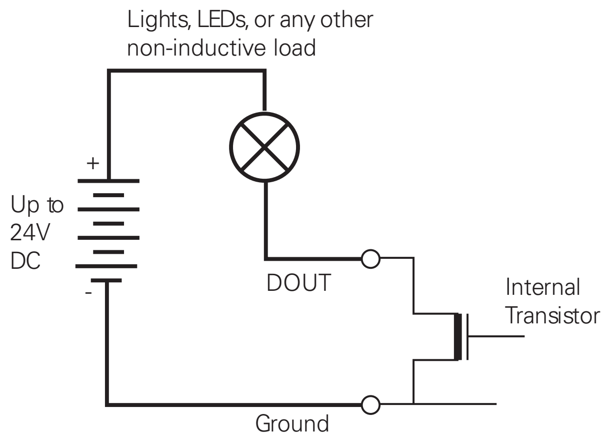


FIGURE 3-1. Connecting resistive loads to Dout pins

Connecting Inductive loads to Outputs

The diagrams on Figure 3-2 show how to connect a relay, solenoid, valve, small motor, or other inductive load to a Digital Output:

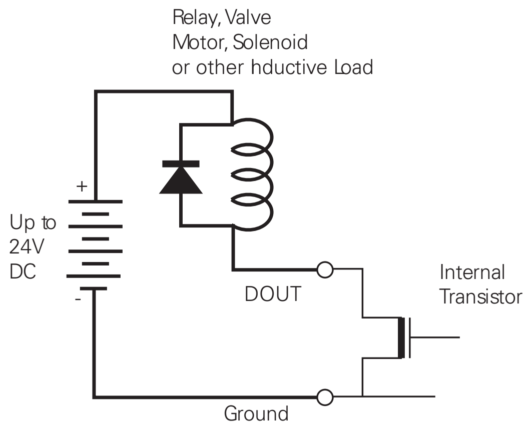


FIGURE 3-2. Connecting inductive loads to Dout pins

Important Warning

Overvoltage spikes induced by switching inductive loads, such as solenoids or relays, will destroy the transistor unless a protection diode is used.

Connecting Switches or Devices to Inputs shared with Outputs

On HBLG2XXX controllers, Digital inputs DIN12 to DIN19 share the connector pins with digital outputs DOUT1 to DOUT8. When the digital outputs are in the Off state, these outputs can be used as inputs to read the presence or absence of a voltage at these pins.

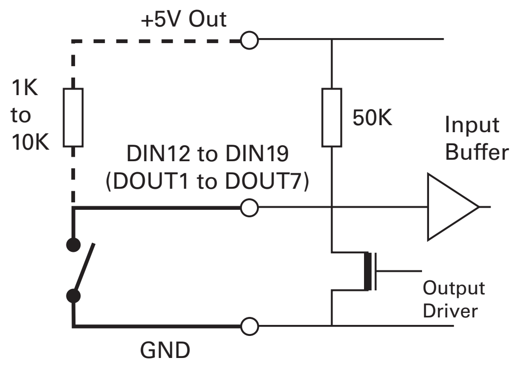


FIGURE 3-3. Switch wiring to inputs shared with outputs

For better noise immunity, an external pull up resistor should be installed even though one is already present inside the controller.

Important Warning

Do not activate an output when it is used as input. If the input is connected directly to a positive voltage when the output is activated, a short circuit will occur. Always pull the input up via a resistor.

Connecting Switches or Devices to direct Digital Inputs

The controller Digital Inputs are high impedance lines with a pull down resistor built into the controller. Therefore it will report an Off state if unconnected, A simple switch as shown in Figure 3-4 can be used to activate it. When a pull up switch is used, for better noise immunity, an external pull down resistor should be installed even though one is already present inside the controller.

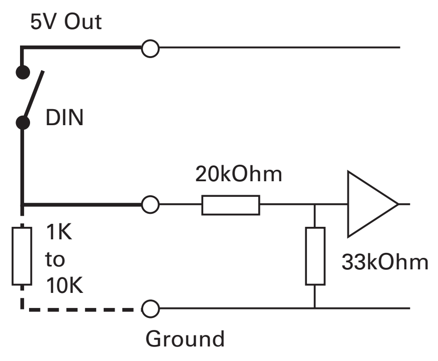


FIGURE 3-4. Pull up (Active High) switch wirings to DIN pins

A pull up resistor must be installed when using a pull down switch.

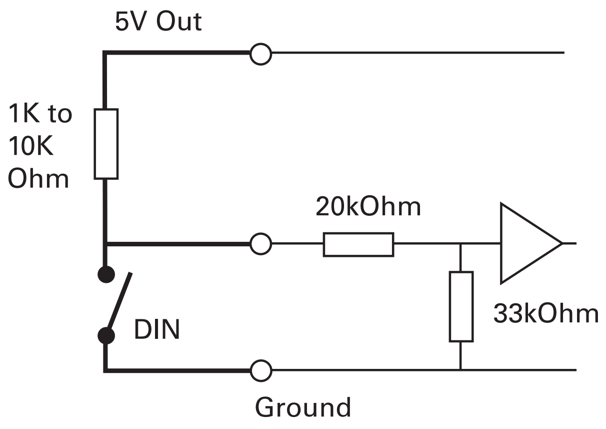


FIGURE 3-5. Pull down (Active Low) switch wirings to DIN pins

Connecting a Voltage Source to Analog Inputs

Connecting sensors with variable voltage output to the controller is simply done by making a direct connection to the controller's analog inputs. When measuring absolute voltages, configure the input in "Absolute Mode" using the PC Utility.

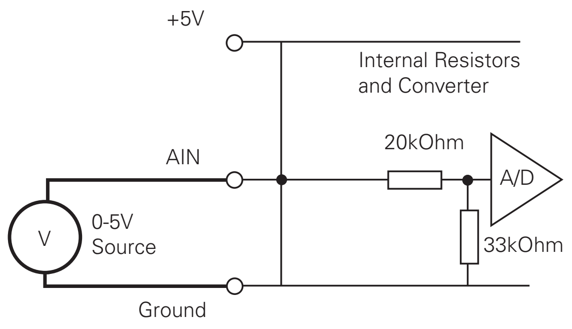


FIGURE 3-6. 0-5V Voltage source connected to Analog inputs

Using external resistors, it is possible to alter the input voltage range to 0V/10V or -10V/+10V.

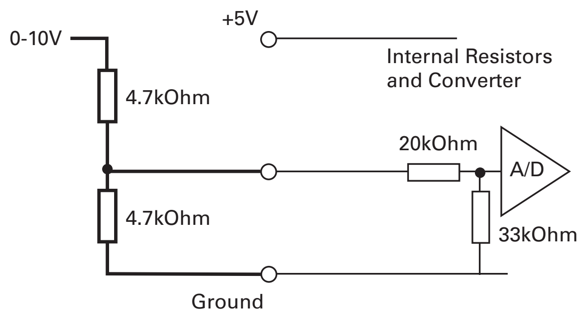


FIGURE 3-7. External resistor circuit for 0 to 10V capture range

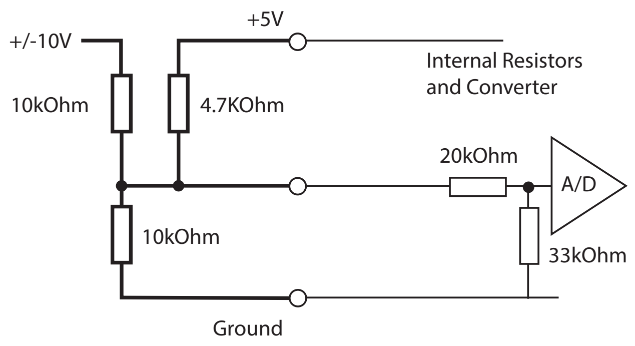


FIGURE 3-8. External resistors circuit for -10V to 10V capture range

Important Notice

Activating the pulse mode on input will also enable a pull up resistor on that input. If the input is also used for analog capture, the analog reading will be wrong. Make sure the pulse mode is disabled on that input.

Reducing noise on Analog Inputs

The Analog inputs are very fast and have a high input resistance. They will therefore easily be disturbed by ambient electrical noise and this will cause the analog reading to be fluctuating. Use shielded cables between the input and the analog sensor. Add a 1uF capacitor between the input pin and the GND pin. With good shielding and filtering, a signal stable to withing +/-5V or better can generally be achieved.

Connecting Potentiometers to Analog Inputs

Potentiometers mounted on a foot pedal or inside a joystick are an effective method for giving the command to the controller. In closed loop mode, a potentiometer is typically used to provide position feedback information to the controller.

Connecting the potentiometer to the controller is as simple as shown in the diagram in Figure 3-9.

The potentiometer value is limited at the low end by the current that will flow through it and which should ideally not exceed 5 or 10mA. If the potentiometer value is too high, the analog voltage at the pot's middle point will be distorted by the input's resistance to ground of 53K. A high value potentiometer also makes the input sensitive to noise, particularly if wiring is long. Potentiometers of 1K or 5K are recommended values.

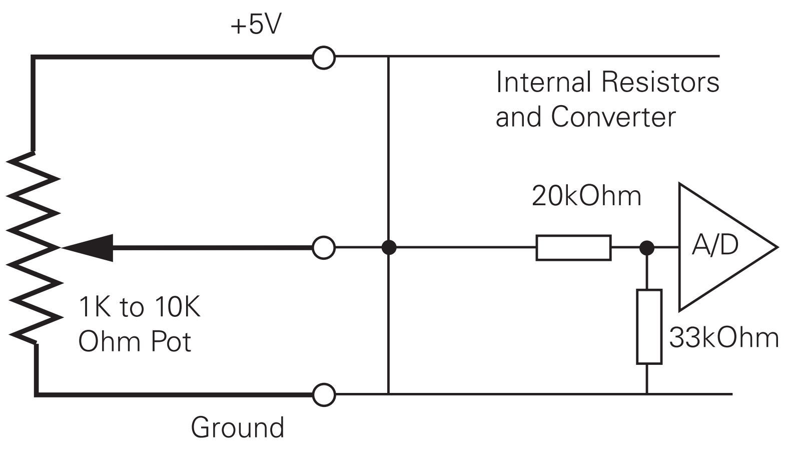


FIGURE 3-9. Potentiometer wiring

Because the voltage at the potentiometer output is related to the actual voltage at the controller's 5V output, configure the analog input in "Relative Mode". This mode measures the actual voltage at the 5V output in order to eliminate any imprecision due to source voltage variations. Configure using the PC Utility.

Connecting Potentiometers for Commands with Safety band guards

When a potentiometer is used for sensing a critical command (Speed or Brake, for example) it is critically important that the controller reverts to a safe condition in case wiring is sectioned. This can be done by adding resistors at each end of the potentiometer so that the full 0V or the full 5V will never be present, during normal operation, when the potentiometer is moved end to end.

Using this circuit shown below, the Analog input will be pulled to 0V if the two top wires of the pot are cut, and pulled to 5V if the bottom wire is cut. In normal operation, using the shown resistor values, the analog voltage at the input will vary from 0.2V to 4.8V.

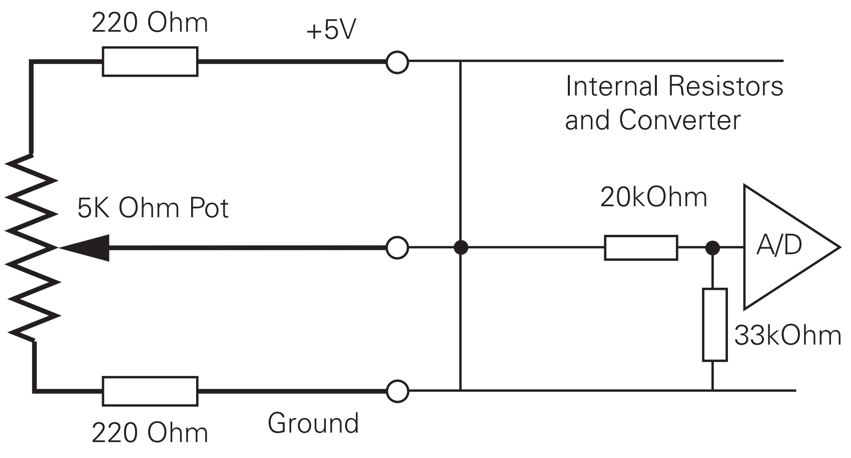


FIGURE 3-10. Potentiometer wiring in Position mode

The controller's analog channels are configured by default so that the min and max command range is from 0.25V to 4.75V. These values can be changed using the PC configuration utility. This ensures that the full travel of the pot is used to generate a command that spans from full min to full max.

If the Min/Max safety is enabled for the selected analog input, the command will be considered invalid if the voltage is lower than 0.1V or higher than 4.9. These values cannot be changed.

Connecting External Thermistor to Analog Inputs

The analog inputs can be used to read thermistor sensors. In order to set it up, the following steps must be followed:

- Connect the sensor with a pull-up resistor (value 10KOhm) to the 5Volt supply, as shown in FIGURE 3-11.
- Use the R25 and B25 parameters of the sensor in the Motor Thermistor and configure accordingly the respective configuration commands (see **B25 - Thermistor Temperature Coefficient β25** and **R25 - Thermistor Resistance at 25oC**) along with the overtemperature limit (see **OTL** - **Over Temperature Limit**).
- Set the conversion type of the Analog input to Absolute (see AMOD).
- Set Input Use to Motor Temperature.
- Select which motor channel should be used (see AINA).
- After the successful configuration, the motor temperatures can be read using the Temperature query (see **T - Read Temperature**).

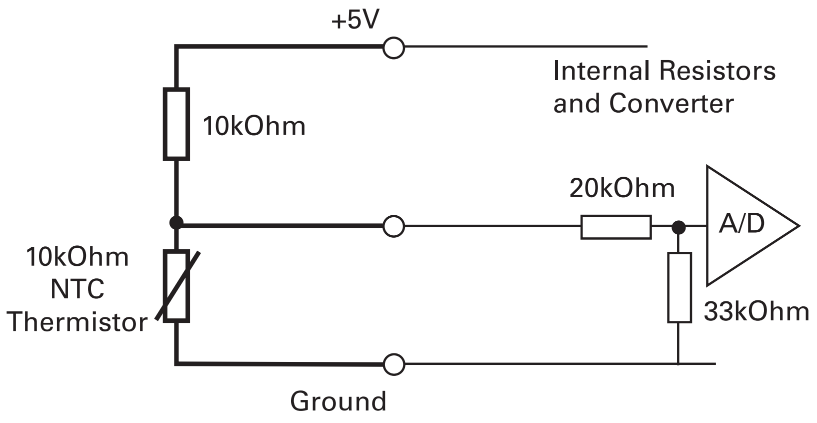


FIGURE 3-11. NTC Thermistor wiring diagram

The firmware uses Steinhart-Hart method to implement the temperature estimation from a thermistor. According to the simplified model the temperature can be estimated by the following formula:

$$R = R_{25}e^{\beta_{25}(\frac{1}{T}-\frac{1}{T_{25}})} \Rightarrow$$
$$\frac{1}{T} = \frac{1}{T_{25}} + \frac{1}{\beta_{25}}ln(\frac{R}{R_{25}})$$

Where:

- R(Ω) is the measured resistance of the thermistor,
- T(Kelvin) is the estimated temperature,
- T_{25} = 298K (25^{o}C),
- R_{25}(Ω) is the resistance in T_{25} temperature and
- β_{25}(Kelvin) is the temperature coefficient.

The last two are characteristics of the thermistor. They can be derived out of their datasheet and set to the respective R25 and B25 configuration commands.

Temperature coefficient (β_{25}) estimation

Temperature coefficient is temperature dependent and is specified for the range between two temperature points, where the model will provide better accuracy.

- Some manufactures specify β as $B_{T1/T2}$, where T_1 and T_2 are the two temperatures mentioned above.
- Other manufacturers give a Resistance - Temperature table from which T_1 and T_2 can be chosen to derive β from the following formula:

$$\beta = \frac{T_1T_2}{T_2 - T_1}ln(\frac{R_{T_1}}{R_{T_2}})$$

where R_{T1} and R_{T2} are the resistances of the thermistor in T_1 and T_2 temperatures accordingly.

In both cases, T_1 should be chosen equal to T_{25} = 298K, thus R_{T1} = R_{25} and β = β_{25}. The formula ends up in the following form:

$$\beta_{25} = \frac{T_{25}T_2}{T_2 - T_{25}}ln(\frac{R_{25}}{R_{T_2}})$$

A typical β_{25} could be:

$$\beta_{25} = B_{25^{\circ}/100^{\circ}} = \frac{T_{25}T_{100}}{T_{100} - T_{25}}ln(\frac{R_{25}}{R_{100}})$$

Note that β_{25} is negative in case of PTC thermistors and positive in case of NTC thermistors.

Using the Analog Inputs to Monitor External Voltages

The analog inputs may also be used to monitor the battery level or any other DC voltage. If the voltage to measure is up to 5V, the voltage can be brought directly to the input pin. To measure higher voltage, insert two resistors wired as a voltage divider. The figure shows a 10x divider capable of measuring voltages up to 50V.

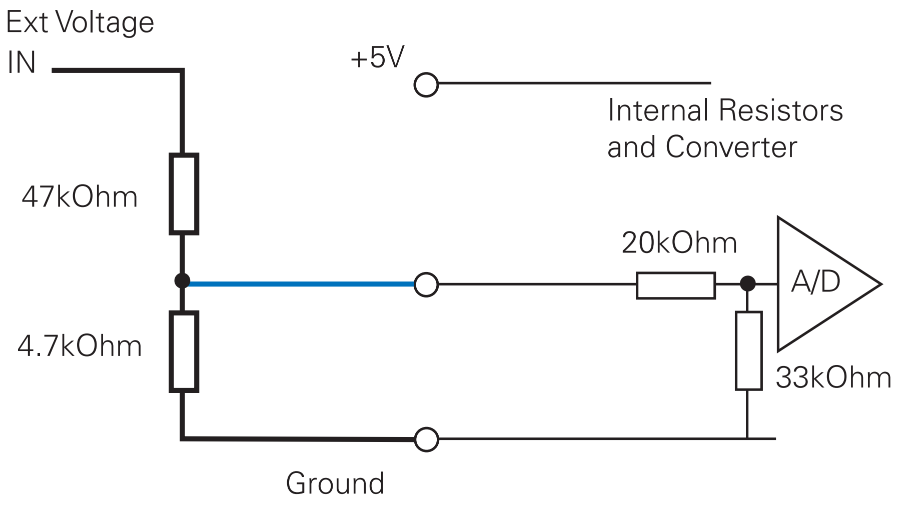


FIGURE 3-12. Battery Voltage monitoring circuit

Connecting Sensors to Pulse Inputs

The controller has several pulse inputs capable of capturing Pulse Length, Duty Cycle or Frequency with excellent precision. Being a digital signal, pulses are also immune to noise compared to analog inputs.

Important Notice

On newer motor controllers models, activating the pulse mode on input will also enable a pull up resistor on that input. If the input is also used for analog capture, the analog reading will be wrong.

Connecting to RC Radios

The pulse inputs are designed to allow direct connection to an RC radio without additional components.

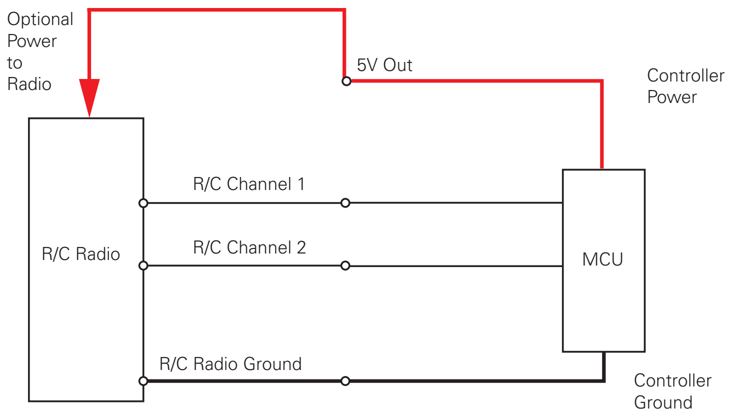


FIGURE 3-13. RC Radio powered by controller electrical diagram

Connecting to PWM Joysticks and Position Sensors

The controller’s pulse inputs can also be used to connect to sensors with PWM outputs. These sensors provide excellent noise immunity and precision. When using PWM sensors, configure the pulse input in Duty Cycle mode. Beware that the sensor should always be pulsing and never output a steady DC voltage at its ends. The absence of pulses is considered by the controller as a loss of signal.

Connecting SSI Sensors

SSI Sensors Overview

SSI sensors are absolute encoders that send their data using Synchronous Serial Interface (SSI). Using SSI protocol offers reduced wiring and EMI immunity. Being absolute encoders, the SSI sensors will report a signal respective to the shaft position. They can be found in both multi-turn and single-turn variations. Roboteq controllers support both (multi-turn and single-turn versions) with a frame size up to 47 bits.

Connecting the SSI Sensor

SSI Sensors connect directly to pins present on the controller’s connector. The connector provides 5V power to the sensors and has inputs for the two data and the two clock signals for each sensor. The figure below shows the connection to the SSI Sensor.

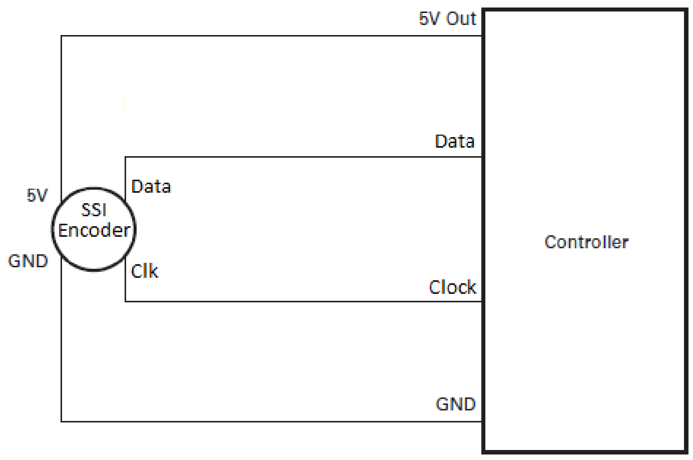


FIGURE 3-14. Controller Connection to typical SSI Encoder

SSI Sensor Clock Polarity

The supported SSI sensors that can be used should have positive clock polarity.

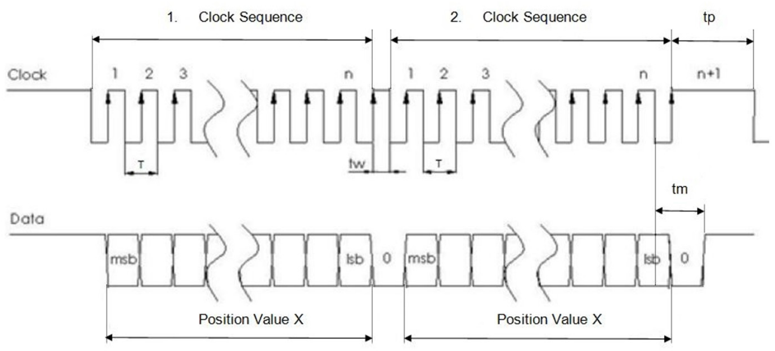


FIGURE 3-15. SSI frame with positive clock

If the polarity is negative there is a workaround described below in order to make them work.

If the sensor has 12 bits and the clock polarity is negative, the following steps need to be performed:

1. Swap Clk+ and Clk- (This will invert the clock polarity but it will ignore one bit)
2. Configure SSI/SPI Number of bits: 13 (so increased by 1 in order to fix the ignored bit)
3. Counter start bit position: 1
4. Counter number of bits: 12

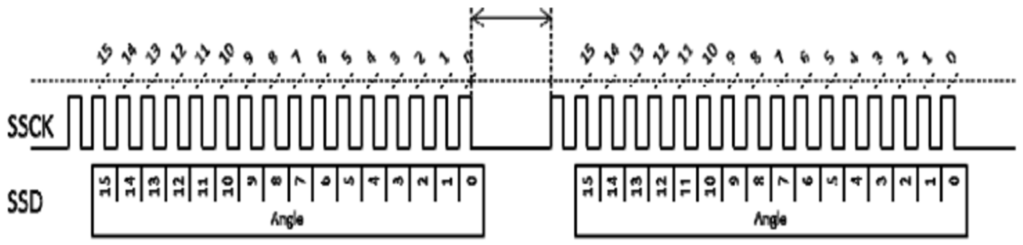


FIGURE 3-16. SSI frame with negative clock and clk signals inverted

Connecting Optical Encoders

Optical Incremental Encoders Overview

Optical incremental encoders are a means for capturing speed and traveled distance on a motor. Unlike absolute encoders which give out a multi-bit number (depending on the resolution), incremental encoders output pulses as they rotate. Counting the pulses tells the application how many revolutions, or fractions of, the motor has turned. Rotation velocity can be determined from the time interval between pulses or by the number of pulses within a given time period. Because they are digital devices, incremental encoders will measure distance and speed with perfect accuracy.

Since motors can move in forward and reverse directions, it is necessary to differentiate the manner that pulses are counted so that they can increment or decrement a position counter in the application. Quadrature encoders have dual channels, A and B, which are electrically phased 90° apart. Thus, the direction of rotation can be determined by monitoring the phase relationship between the two channels. In addition, with a dual-channel encoder, a four-time multiplication of resolution is achieved by counting the rising and falling edges of each channel (A and B). For example, an encoder that produces 250 Pulses per Revolution (PPR) can generate 1,000 Counts per Revolution (CPR) after quadrature.

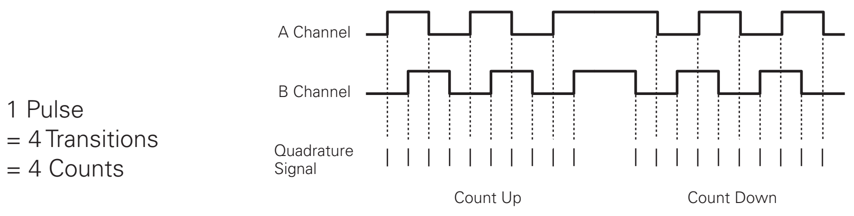


FIGURE 3-17. Quadrature encoder output waveform

The figure below shows the typical construction of a quadrature encoder. As the disk rotates in front of the stationary mask, it shutters light from the LED. The light that passes through the mask is received by the photo detectors. Two photo detectors are placed side by side at so that the light making it through the mask hits one detector after the other to produces the 90° phased pulses.

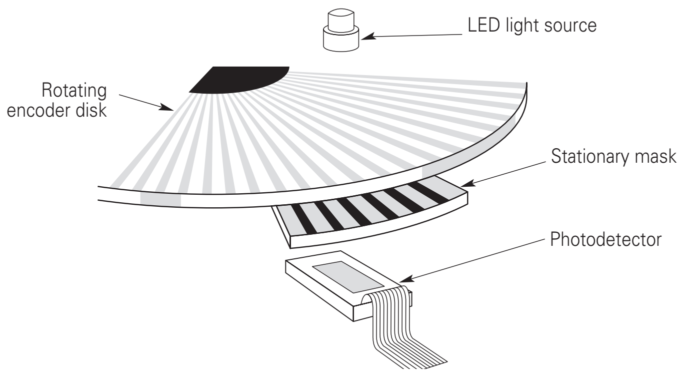


FIGURE 3-18. Typical quadrature encoder construction

Unlike absolute encoders, incremental encoders have no retention of absolute position upon power loss. When used in positioning applications, the controller must move the motor until a limit switch is reached. This position is then used as the zero references for all subsequent moves.

Recommended Encoder Types

The module may be used with most incremental encoder modules as long as they include the following features:

- Two quadrature outputs (Ch A, Ch B), single ended
- 3.8V minimum swing between 0 Level and 1 Level on quadrature output
- 5VDC operation. 50mA or less current consumption per encoder

More sophisticated incremental encoders with index and other features may be used, however these additional capabilities will be ignored.

The choice of encoder resolution is very wide and is constrained by the module's maximum pulse count at the high end and measurement resolution for speed at the low end.

Specifically, the controller's encoder interface can process 1 million counts per second, unless otherwise specified in the product datasheet.

Commercial encoders are rated by their numbers of "Pulses per Revolution" (also sometimes referred to as "Number of Lines" or "Cycles per Revolution"). Carefully read the manufacturer's datasheet to understand whether this number represents the number of pulses that are output by each channel during the course of a 360 degrees revolution rather than the total number of transitions on both channels during a 360 degrees revolution. The second number is 4 times larger than the first one.

The formula below gives the pulse frequency at a given RPM and encoder resolution in Pulses per Revolution.

Pulse Frequency in counts per second = (RPM/60)*PPR*4

Example: a motor spinning at 10,000 RPM max, with an encoder with 200 Pulses per Revolution would generate:

10,000 / 60 * 200 * 4 = 133.3 kHz which is well within the 1MHz maximum supported by the encoder input.

An encoder with 200 Pulses per Revolutions is a good choice for most applications.

A higher resolution will cause the counter to count faster than necessary and possibly reach the controller's maximum frequency limit.

An encoder with a much lower resolution will cause speed to be measured with less precision.

Connecting the Encoder

Encoders connect directly to pins present on the controller's connector. The connector provides 5V power to the encoders and has inputs for the two quadrature signals from each encoder. The figure below shows the connection to the encoder.

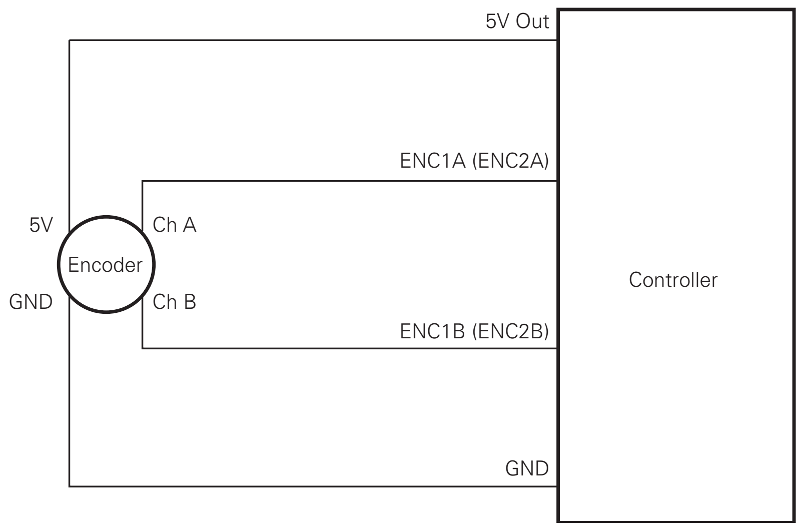


FIGURE 3-19. Controller connection to typical Encoder

For several controller models, the Encoder inputs are by default mapped in Molex connector. In order to be able to use encoders and SSI sensors at the same time (see "MLX" in the command reference section), the encoders can be mapped to DB25 connector pins where pulse inputs are.

Cable Length and Noise Considerations

The cable should not exceed one 3' (one meter) to avoid electrical noise to be captured by the wiring. A ferrite core filter should be inserted near the controller for length beyond 2' (60 cm). For longer cable length use an oscilloscope to verify signal integrity on each of the pulse channels and on the power supply.

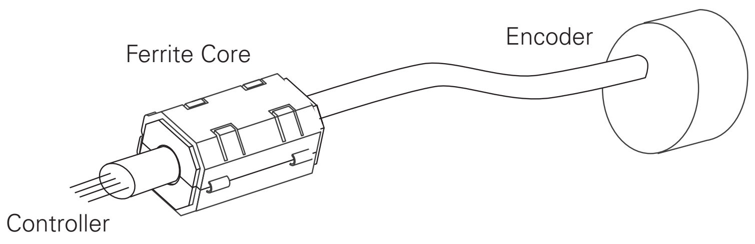


FIGURE 3-20. Use ferrite core on cable length beyond 2' or 60cm

Important Warning

Excessive cable length will cause electrical noise to be captured by the controller and cause erratic functioning that may lead to failure. In such a situation, stop operation immediately.

Motor - Encoder Polarity Matching

When using encoders for closed loop speed or position control, it is imperative that when the motor is turning in the forward direction, the counter increments its value and a positive speed value is measured. The counter value can be viewed using the PC utility.

If the Encoder counts backward when the motor moves forward, correct this by either:

1- Swapping Channel A and Channel B on the encoder connector. This will cause the encoder module to reverse the count direction,

2- Enter a negative number in the PPR configuration will also cause the counter to count in the reverse direction

3- Swapping the leads on the motor. This will cause the motor to rotate in the opposite direction.

SECTION 4

I/O Configuration and Operation

This section discusses the controller's digital and analog inputs and output and how they can be used.

Basic Operation

The controller's operation can be summarized as follows:

- Receive commands from a radio receiver, joystick or a microcomputer
- Activate the motor according to the received command
- Perform continuous check of fault conditions and adjust actions accordingly
- Report real-time operating data

The diagram below shows a simplified representation of the controller's internal operation. The most noticeable feature is that the controller's serial, digital, analog, pulse, and encoder inputs may be used for practically any purpose.

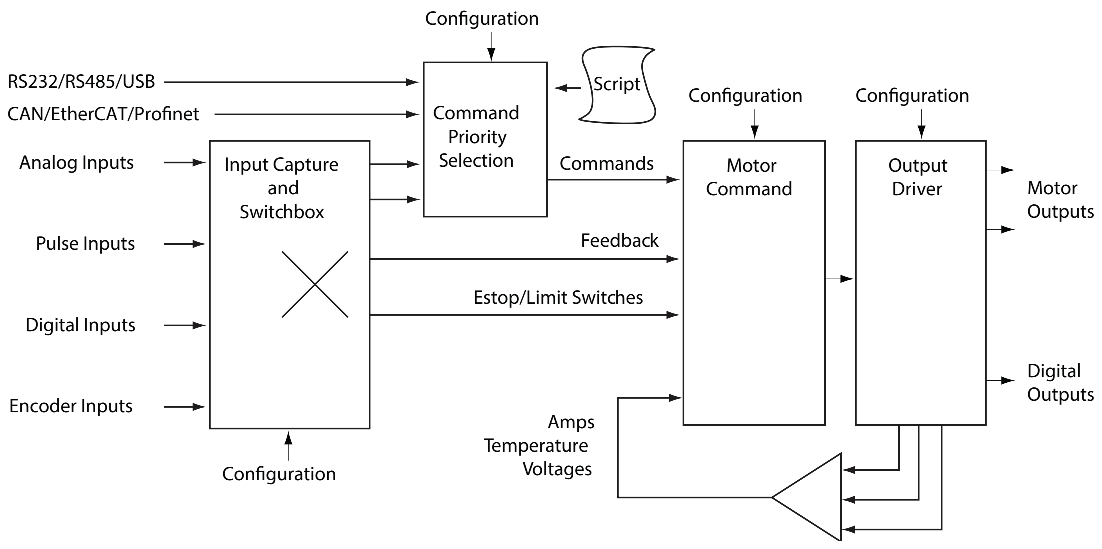


FIGURE 4-1. Simplified representation of the controller's internal operation

Practically all operating configurations and parameters can be changed by the user to meet any specific requirement. This unique architecture leads to a very high number of possibilities. This section of the manual describes all the possible operating options.

Input Selection

As seen earlier in the controller's simplified internal operating diagram on Figure 4-1, any input can be used for practically any purpose. All inputs, even when they are sharing the same pins on the connector, are captured and evaluated by the controller. Whether input is used, and what it is used for, is set individually using the descriptions that follow.

Important Notice

On shared I/O pins, there is nothing stopping one input to be used as analog or pulse at the same time or for two separate inputs to act identically or in conflict with one another. While such an occurrence is normally harmless, it may cause the controller to behave in an unexpected manner and/or cause the motors not to run. Care must be exercised in the configuration process to avoid possible redundant or conflictual use.

Digital Inputs Configurations and Uses

Each of the controller's digital Inputs can be configured so that they are active high or active low. Each output can also be configured to activate one of the actions from the list in the table below. In multi-channel controller models, the action can be set to apply to any or all motor channels.

TABLE 4-1. Digital Input Action List

Action	Applicable Channel	Description
No Action	-	Input causes no action
Quick Stop	Selectable	Stops the respective motor by controlling speed using Fault Deceleration. The motor remains in Quick Stop until an idle motor command is given (0 in case of speed modes, equal to feedback in case of position modes).
Emergency stop	All	Sets all the MOSFETs that drive the respective motor to float. So no power is applied to the motor and the motor is about to stop due to friction.
Motor Stop (deadman switch)	Selectable	Stops the respective motor by controlling speed using Fault Deceleration. Motor resumes when input becomes inactive
Invert motor direction	Selectable	Inverts the motor direction, regardless of the command mode in use
Forward limit switch	Selectable	Stops the motor until the command is changed to reverse
Reverse limit switch	Selectable	Stops the motor until the command is changed forward
Run script	NA	Start execution of MicroBasic script
Load Home counter	Selectable	Load counter with a Home value
Soft STO	Selectable	Configure input to act as an STO input

Configuring the Digital Inputs and the Action to use can be done very simply using the PC Utility.

Wiring instructions for the Digital Inputs can be found in "Connecting Switches or Devices to Inputs shared with Outputs" on page 48

Analog Inputs Configurations and Use

The controller can do extensive conditioning on the analog inputs and assign them to a different use.

Each input can be disabled or enabled. When enabled, it is possible to select the whether capture must be an absolute voltage or relative to the controller's 5V Output. Details on how to wire analog inputs and the differences between the Absolute and Relative captures can be found in "Using the Analog Inputs to Monitor External Voltages" page 54.

TABLE 4-2. Analog Capture Modes

Analog Capture Mode	Description
Disabled	Analog capture is ignored (forced to 0)
Absolute	Analog capture measures real volts at the input
Relative	Analog captured is measured relative to the 5V Output which is typically around 4.8V to 5.1V depending on the controller model and the load. Correction is applied so that an input voltage measured to be the same as the 5V Output voltage is reported at 5.0V

The raw Analog capture then goes through a series of processing shown in the diagram below.

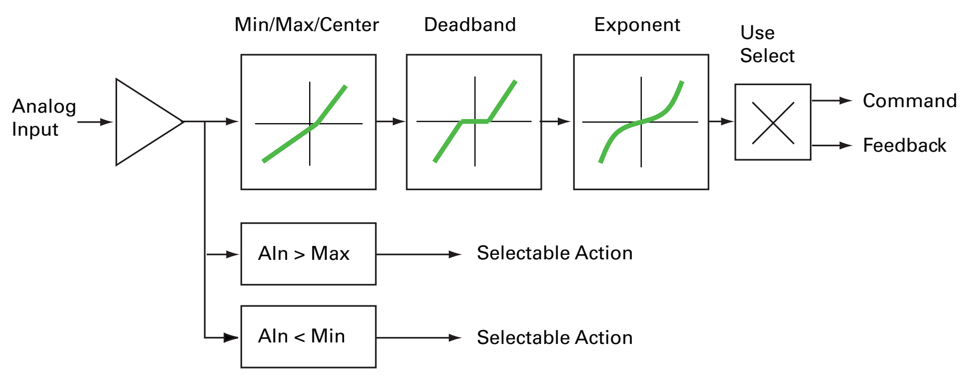


FIGURE 4-2. Analog Input processing chain

Analog Min/Max Detection

An analog input can be configured so that an action is triggered if the captured value is above a user-defined Maximum value and/or under a user-defined Minimum value. The actions that can be selected are the same as these that can be triggered by the Digital Input. See the list and description in Table 4.1, "Digital Input Action List" on page 62.

Min, Max and Center adjustment

The raw analog capture is then scaled into a number ranging from -1000 to +1000 based on user-defined Minimum, Maximum and Center values for the input. For example, setting the minimum to 500mV, the center to 2000mV, and the maximum to 4500mV, will produce the output to change in relation to the input as shown in the graph below

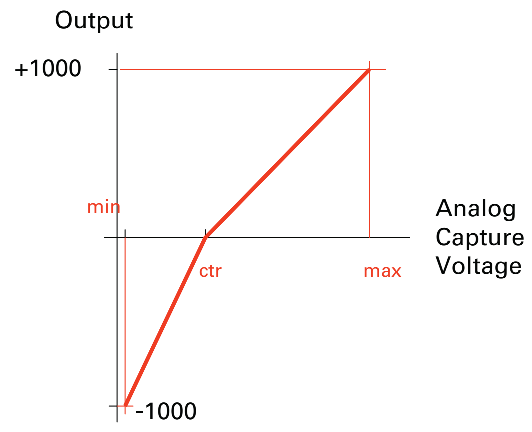


FIGURE 4-3. Analog Input processing chain

This feature allows capturing command or feedback values that match the available range of the input sensor (typically a potentiometer).

For example, this capability is useful for modifying the active joystick travel area. The figure below shows a transmitter whose joystick's center position has been moved back so that the operator has a finer control of the speed in the forward direction than in the reverse position.

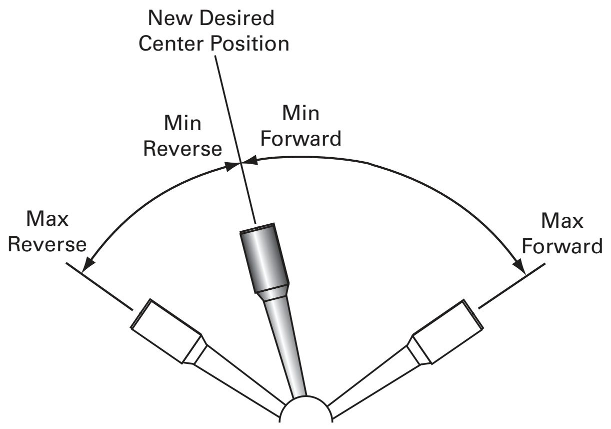


FIGURE 4-4. Calibration example where more travel is dedicated to forward motion

Setting the center value to be the same as the min value makes the input capture only commands in a positive direction. For example if Min = Center = 200 and Max = 4500, the input will convert into 0 when 200 and below, and 1000 above 4500.

The Min, Max, and Center values are defined individually for each input. They can be easily entered manually using the Roborun PC Utility. The Utility also features an Auto-calibration function for automatically capturing these values.

Deadband Selection

The adjusted analog value is then adjusted with the addition of a deadband. This parameter selects the range of movement change near the center that should be considered as a 0 command. This value is a percentage from 0 to 50% and is useful, for example, to allow some movement of a joystick around its center position before any power is applied to a motor. The graph below shows output vs input changes with a deadband of approximately 40%.

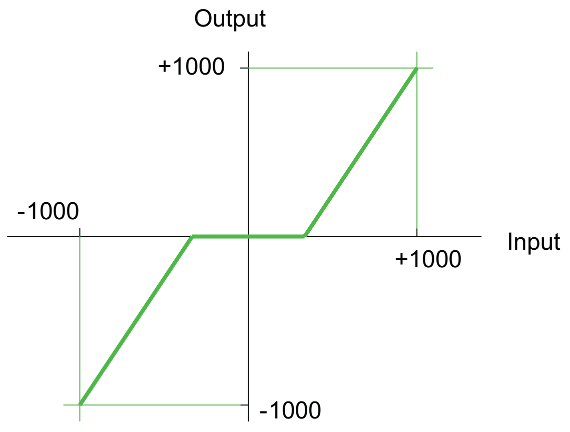


FIGURE 4-5. Effect of deadband on the output

Note that the deadband only affects the start position at which the joystick begins to take effect. The motor will still reach 100% when the joystick is in its full position. An illustration of the effect of the deadband on the joystick action is shown in Figure 4-6 below.

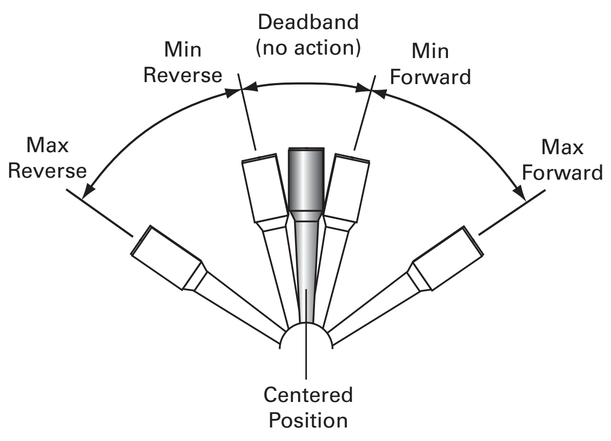


FIGURE 4-6. Effect of deadband on joystick position vs. motor command

The deadband value is set independently for each input using the PC configuration utility.

Command Correction

An optional exponential or a logarithmic adjustment can then be applied to the signal. The exponential correction will make the commands change less at the beginning and become stronger at the end of the joystick movement. The logarithmic correction will have a stronger effect near the start and lesser effect near the end. The linear selection causes no change to the input. There are 3 exponential and 3 logarithmic choices: weak, medium and strong. The graph below shows the output vs input change with exponential, logarithmic and linear corrections.

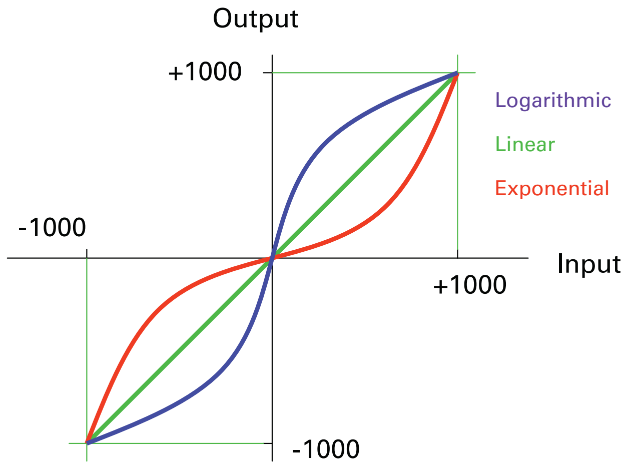


FIGURE 4-7. Effect of exponential / logarithmic correction on the output

The exponential or log correction is selected separately for each input using the PC Configuration Utility.

Use of Analog Input

After the analog input has been fully processed, it can be used as a motor command or, if the controller is configured to operate in a closed loop, as a feedback value (typically speed or position).

Each input can therefore be configured to be used as command or feedback for any motor channel(s). The mode and channel(s) to which the analog input applies are selected using the PC Configuration Utility.

Pulse Inputs Configurations and Uses

The controller's Pulse Inputs can be used to capture pulsing signals of different types.

TABLE 4-3. Pulse Inputs Capture Modes

Capture Mode	Description	Typical use
Disabled	Pulse capture is ignored (forced to 0)	
Pulse	Measures the On time of the pulse	RC Radio

TABLE 4-3. (*continued*)

Capture Mode	Description	Typical use
Duty Cycle	Measures the On time relative to the full On/Off period	Hall position sensors and joysticks with pulse output
Frequency	Measures the repeating frequency of pulse	Encoder wheel
MagSensor	Gets Data from MagSensor (MultiPWM, see section 5)	Magnetic Sensor (MGS1600)
BMS	Gets Data from BMS (MultiPWM, see section 5)	Battery Management System (BMS1040)
Pulse Count	Counts the number of Pulses[1]	Quadrature Encoder
Flow Sensor	Gets Data from FlowSensor (MultiPWM, see section 5)	Flow Sensor (FLW100)
[1] The counter will increment every time a pulse is received. The count can be read using the PI query. In order to reset the counter de-configure capture mode to disabled and then back to pulse count.		

The capture mode can be selected using the PC Configuration Utility.

The captured signals are then adjusted and can be used as command or feedback according to the processing chain described in the diagram below.

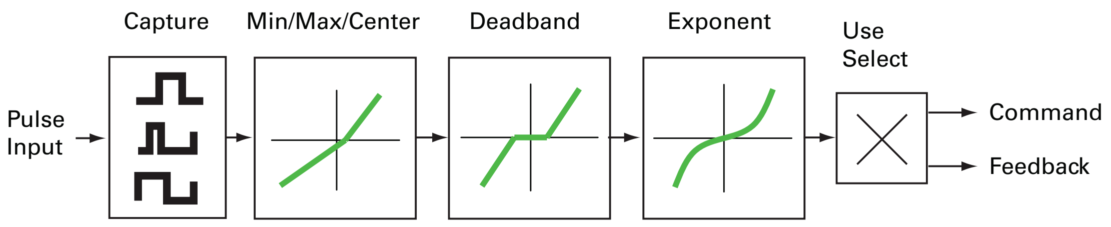


FIGURE 4-8. Pulse Input processing chain

Except for the capture, all other steps are identical to those described for the Analog capture mode.

Use of Pulse Input

After the pulse input has been fully processed, it can be used as a motor command or, if the controller is configured to operate in a closed loop, as a feedback value (typically speed or position).

Each input can therefore be configured to be used as command or feedback for any motor channel(s). The mode and channel(s) to which the analog input applies are selected using the PC Configuration Utility.

Digital Outputs Configurations and Triggers

The controller's digital outputs can individually be mapped to turn On or Off based on the status of user-selectable internal status or events. The table below lists the possible assignment for each available Digital Output.

TABLE 4-4. Digital Outputs action modes

Action	Output activation	Typical Use
No action	Not changed by any internal controller events.	Output may be activated using Serial commands or user scripts
Motor(s) is on	When selected motor channel(s) has power applied to it.	Brake release
Motor(s) is reversed	When selected motor channel(s) has power applied to it in reverse direction.	Back-up warning indicator
Overvoltage	When battery voltage above over-limit	Shunt load activation
Overtemperature	When over-temperature limit exceeded	Fan activation. Warning buzzer
Status LED	When status LED is ON	Place Status indicator in visible location.

Encoder Configurations and Use

On controller models equipped with encoder inputs, external encoders enable a range of precision motion control features. See "Connecting Optical Encoders" page 56 for a detailed discussion on how optical encoders work and how to physically connect them to the controller. The diagram below shows the processing chain for each encoder input.

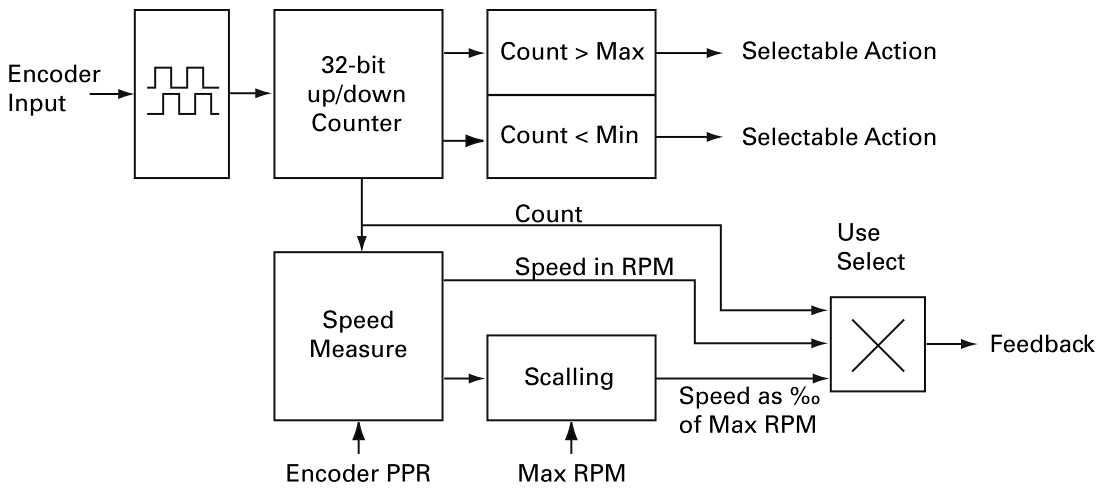


FIGURE 4-9. Encoder input processing

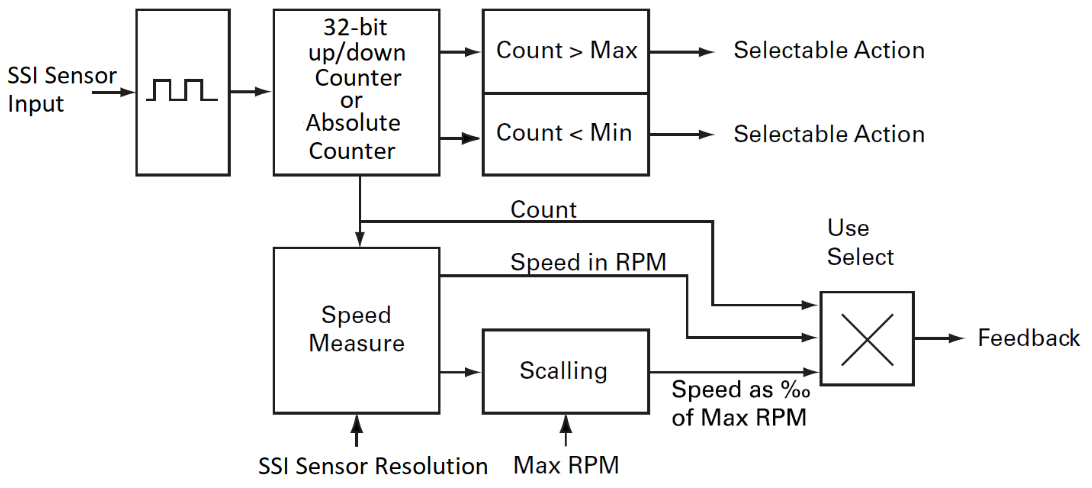
The encoder's two quadrature signals are processed to generate up and down counts depending on the rotation direction. The counts are then summed inside a 32-bit counter. The counter can be read directly using serial commands and/or can be used as a position feedback source for the closed loop position mode.

The count information is also used to measure rotation speed. Using the Encoder Pulse Per Rotation (PPR) configuration parameter, the output is a speed measurement in actual RPM that is useful in closed loop speed modes where the desired speed is set as a numerical value, in RPM, using a serial command.

Configuring the encoder parameters is done easily using the PC Configuration Utility.

SSI Configuration and Use

On controller models equipped with SSI sensor inputs, SSI sensors enable a range of precision motion control features. See "Connecting SSI Sensors" page 56 for a detailed discussion on how SSI sensors work and how to physically connect them to the controller. The diagram below shows the processing chain for each encoder input.



FIGUER 4-10. SSI Sensor input processing

The SSI sensor's signal is processed to generate up and down counts depending on the rotation direction. The counts are then summed inside a 32-bit counter. The counter can be read directly using serial commands and/or can be used as a position feedback source for the closed loop position mode.

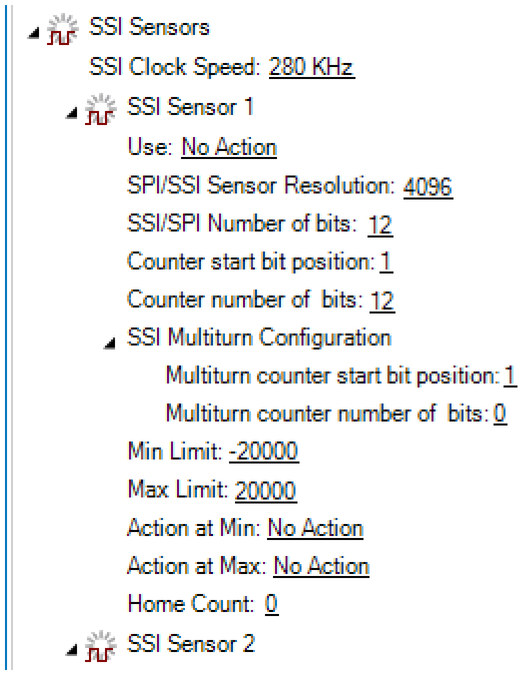
Also taking into consideration that SSI sensors are absolute sensors, the sensor can be configured in order to indicate the absolute position of the rotor shaft (Absolute Feedback). In that case, the counter holds the absolute position of the rotor shaft.

The count information is also used to measure rotation speed. Using the SSI Sensor Counter number of bits (SLEN) configuration parameter, the output is a speed measurement in actual RPM that is useful in closed loop speed modes where the desired speed is set as a numerical value, in RPM, using a serial command.

Configuring the SSI Sensor parameters is done easily using the PC Configuration Utility.

The SSI configuration commands provide support for multiturn encoders. The user can configure the length and position of the angle counter and the multiturn counter even when the sensor has extra status bits. The raw frame of the sensor can be retrieved using the SRF query. See below the configuration fields used for that purpose:

- SSI Clock Speed (SCLK), it is common for all SSI sensor inputs
- SSI/SPI Number of bits (SLEN), the total number of bits of the SSI sensor frame.
- Counter start bit position (SSTA), the position of the first bit of the angle counter.
- Counter number of bits (SCLE), the number of bits of the angle counter.
- Multi-turn counter start bit position (MSTA), the position of the first bit of the multi-turn counter.
- Multi-turn counter number of bits (MCLE), the number of bits of the multi-turn counter.



SSI Sensors
SSI Clock Speed: 280 KHz
SSI Sensor 1
Use: No Action
SPI/SSI Sensor Resolution: 4096
SSI/SPI Number of bits: 12
Counter start bit position: 1
Counter number of bits: 12
SSI Multiturn Configuration
Multiturn counter start bit position: 1
Multiturn counter number of bits: 0
Min Limit: -20000
Max Limit: 20000
Action at Min: No Action
Action at Max: No Action
Home Count: 0
SSI Sensor 2

FIGURE 4-11. SSI Sensor Configuration

Configuration Examples:

a. 12-bit encoder without multi-turn or status bits

^SLEN 1 12: Total number of bits is 12

^SSTA 1 1: The angle counter starts at bit 1

^SCLE 1 12: The resolution (length) of the counter is 12 bits.

b. 12-bit encoder without multi-turn and 2 status bits at LSB (12bit Angle, 2bit Status)

^SLEN 1 14: Total number of bits is 14

^SSTA 1 3: The angle counter starts at bit 3

^SCLE 1 12: The resolution (length) of the counter is 12 bits.

c. 12-bit encoder with 12-bit multi-turn counter and no status bits (12bit Multi-turn,12bit Angle Counter)

^SLEN 1 12: Total number of bits is 24

^SSTA 1 1: The angle counter starts at bit 1

^SCLE 1 12: The resolution (length) of the counter is 12 bits.

^MSTA 1 13: The angle counter starts at bit 13

^MCLE 1 12: The resolution (length) of the counter is 12 bits.

d. 16-bit encoder with 12-bit multi-turn counter and 2-bit status bits (12bit Multi-turn, 16bit Angle Counter, 2bit Status)

^SLEN 1 30: Total number of bits is 30

^SSTA 1 3: The angle counter starts at bit 3

^SCLE 1 16: The resolution (length) of the counter is 16 bits.

^MSTA 1 18: The angle counter starts at bit 18

^MCLE 1 12: The resolution (length) of the counter is 12 bits.

Hall and other Rotor Sensor Inputs

The Hall or other Rotor position sensor that is used to switch power around the motor windings, are also used to measure speed and distance traveled.

Speed is evaluated by measuring the time between transition of the Hall Sensors. A 32 bit up/down counter is also updated at each Hall Sensor transition.

Speed information picked up from the Hall Sensors can be used for closed loop speed operation without any additional hardware.

Sensor Min Max values

Each Encoder counter, SSI Sensor Counter or Internal Sensor Counter counter can be compared to the respective user-defined Min and/or Max values and trigger an action if these limits are reached. The type actions are the same as these selectable for Digital Inputs and described in "Digital Inputs Configurations and Use" page 62.

Relative Speed

The speed information is also scaled to produce a number ranging from -1000 to +1000 relative to a user-configured arbitrary Max RPM value. For example, with the Max RPM configured as 3000 and the motor rotating at 1500 RPM, the measured relative speed will be 500. Relative speed is useful for closed loop speed mode that uses Analog or Pulse inputs as speed commands.

Brake Release

Each digital output can be connected to a brake solenoid and control it by setting the output's action to "Motor is ON" using the Roborun+ utility. In that case when there is a non-idle motor command the DOUT is activated and the brake is released. When the motor command becomes idle (0 in case of speed modes, equal to feedback in case of position modes) and the motor stops moving, then the DOUT is deactivated and the brake is engaged.

A delay has been introduced from the time that the motor stops until engaging the brake. The delay time is configurable, and its purpose is to ensure that the motor is on equilibrium before engaging the brake (for more details see **BKD - Brake activation delay in ms**, in page 341).

The user has the ability to override the above-mentioned brake functionality by using the runtime command BRK (or more details see BRK - Brake Override, in page 191). That way, the motor brake can be immediately engaged by setting the BRK parameter to 2 and it can be immediately disengaged skipping the brake delay, by setting the BRK parameter to 1. Setting the BRK parameter to 0 will return the brake control to the previous automatic mode.

A general overview of the handling of the brake functionality (Motor is on) and the brake override can be seen in the figure below:

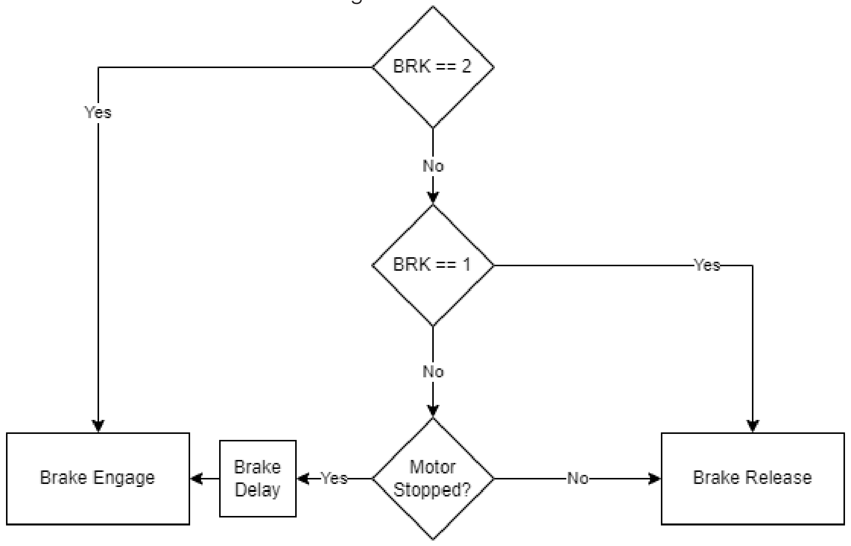


FIGURE 4-12. Brake (Motor Is On) functionality

Some boards have special outputs (Brk+, Brk-) that can activate the brake by using PWM instead of providing a constant voltage. That way a higher voltage can be initially applied to energize the coil (for more details see **BRV - Brake Release Voltage**, in page 343), and then reduced to a lower voltage level (for more details see **BHV - Brake Hold Voltage**, in page 343), in order to maintain the brake released while consuming less energy. Between the two steps there is a time delay which defines for how much time the Brake Release voltage will be applied (for more details see **BDT - Brake Delay Time**, in page 344).

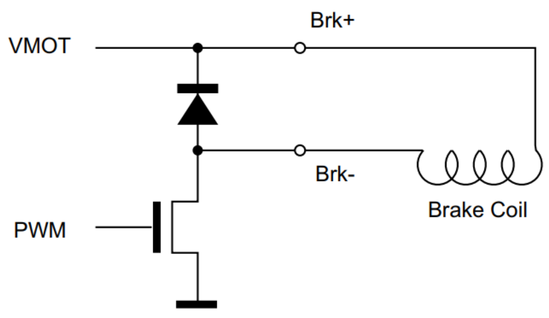


FIGURE 4-12. Brake drive circuit and connection

The configuration is similar since the brake outputs are shared with digital outputs. For more details see product’s datasheet.

SECTION 5

Roboteq Products Connection and Operation

This section discusses how to interface one or more Roboteq's products to the motor controller. For the moment the supported products are:

- Magnetic or Optical Sensor (MGS(W)1600, MSW3200, MGSW4800, OTS1600),
- FlowSensor (FLW100),
- Battery Management System (BMS10X0).

Details of each of the above products' operation can be found in the products' datasheets.

Introduction to MGS1600 Magnetic Guide Sensor

Roboteq's Magnetic Guide Sensor is a sensor capable of detecting and reporting the position of a magnetic field along its horizontal axis. The sensor is intended for applications in Automatic Guided Vehicles using inexpensive adhesive magnetic tape to form a guide on the floor. The tape creates an invisible field that is immune to dirt and unaffected by lighting conditions. The sensor can be interfaced directly to any of Roboteq's motor controllers in order to create an effective AGV solution with just two components.

The sensor generates the following information about the track:

- Tape Detect
- Position of Left Track
- Position of Right Track
- Presence of Left Marker
- Presence of Right Marker

Introduction to FLW100 Flowsensor

Roboteq's FLW100 is a high-resolution sensor especially designed for accurate contact-less X-Y motion sensing over a surface. The FLW100 is intended as a navigation sensor for a wheeled mobile robot. The sensor works similarly to an optical mouse, but with higher resolution, accuracy and at a greater distance from the reference surface. The sensor uses an embedded infrared camera that is pointed to the floor and measures the displacement distance and speed along the X and Y axis by comparing images at each frame. Distance is measured with 0.1mm resolution with excellent accuracy.

The sensor generates among others the following information:

- X axis distance in mm,
- Y axis distance in mm.

Introduction to BMS10X0 Battery Management System

RoboteQ's BMS10x0 is a battery management and protection system for building cost-effective, ultra-efficient and high current power sources using Lithium battery cells. The BMS connects to an array of battery cells at one end, and to a user load at the other. Available in a 40V and 60V versions, it is optimized for 6-cell to 15-cell battery packs.

The product generates among others the following information:

- BMS State of Charge in AmpHours (Ah),
- BMS State Of Charge in percentage,
- BMS Status Flags,
- BMS Switch States.

Available Interfaces

All the above data can be transmitted to the Roboteq controller and other devices using one of the following methods:

TABLE 5-1. Available Interfaces between Roboteq Products

Method	To Roboteq Controllers	To PLCs	To PCs
MultiPWM	Preferred	Unsuitable	Unsuitable
Serial	Not Recommended	Preferred	Suitable
CANbus	Suitable	Preferred	Suitable(1)
USB	N/A	Unsuitable	Suitable
Notes: 1: PC must be fitted with CAN adapter			

MultiPWM interface

The recommended interfacing method to Roboteq motor controller is the MultiPWM mode. As the name implies, this proprietary method uses a succession of variable duty-cycle pulses to carry the data sent by the Magnetic sensor, the FlowSensor or the Battery Management System.

Any of the controller's pulse input can be configured as a MultiPWM input. The diagram below shows how simple this one-wire interfacing is.

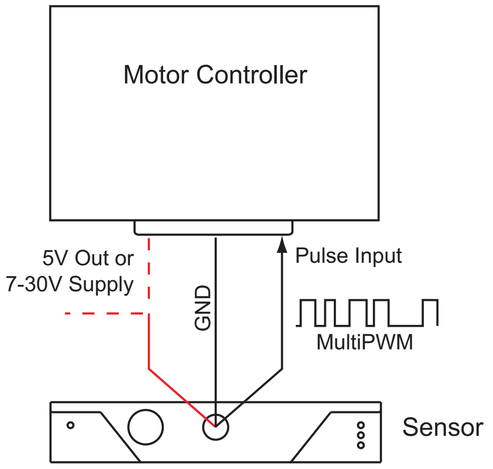


FIGURE 5-1. One-wire interfacing using MultiPWM

Enabling MultiPWM Communication

Magnetic Sensor and Flow Sensor are set to MultiPWM mode in its factory default configuration, while the Battery Management System needs to be explicitly configured. To enable the capture, the selected pulse input on the controller must be configured to the respective option when using the PC utility.

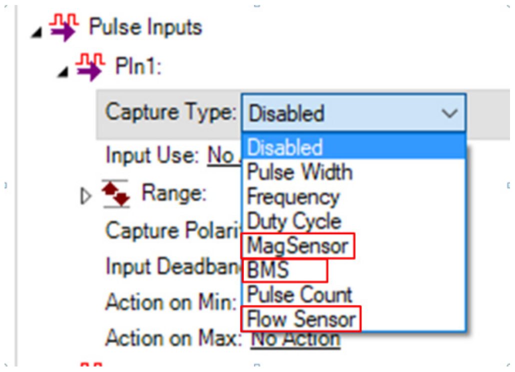

Pulse Inputs
PIn1:
Capture Type: Disabled
Input Use: No
Range:
Capture Polari
Input Deadban
Action on Min:
Action on Max: No Action

FIGURE 5-2. Multi-PWM configuration

When changing via the console use

^PMOD cc nn to enable pulse input cc in MultiPWM mode.

Where nn:

4: For Magnetic Sensor

5:For Battery Management System

7: For Flow Sensor

Accessing Sensor Information

Once enabled, the pulses are sent continuously by the sensor 100 times per second. The pulses are captured and parsed by the motor controller as they arrive. A real time mirror image of sensor data is then present inside the controller. From there the sensor information can be read using serial, USB, CAN or MicroBasic scripts like any other of the controller's operational parameters.

The following Motor Controller queries are available for reading the captured sensor data.

TABLE 5-2. Accessing Roboteq Sensor Information

Magnetic Sensor	
?MGD	Read tape detect
?MGT nn	Read left track when nn= 1 or right track when nn= 2
?MGM nn	Read left marker when nn=1 or right track when nn= 2
Flow Sensor	
?FLW nn	Read X Counter in mm when nn= 1 or Y Counter in mm when nn= 2
Battery Management System	
?BMC	Read BMS State Of Charge in AmpHours (Ah)
?BSC	Read BMS State Of Charge in per cent
?BMF	Read BMS status flags
?BMS	Read BMS switch states

Details on these commands can be found in the Commands Reference section of this manual

Connecting Multiple Similar Sensors

More than one similar sensors can be connected to a single motor controller. For Magnetic sensors, this can be useful in AGV designs that must be able to move in the forward and reverse direction along with the guide. Connecting multiple sensors can be done by connecting each sensor to one of the available pulse input, as shown in the figure below.

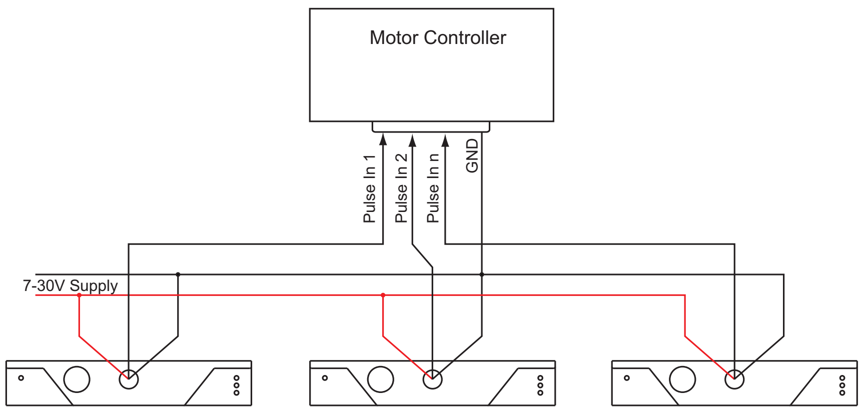


FIGURE 5-3. Connecting multiple sensors to a motor controller

Accessing Multiple Sensor Information Sequentially

Two methods are available for accessing each sensor's data when multiple sensors are connected.

The first method is to only have one sensor enabled at any one time. This is done by enabling and disabling pulse inputs via serial commands or MicroBasic scripting. Examples:

^PMOD 1 0 : Serial command to Disable Sensor on pulse input 1

Setconfig(_PMOD, 1, 0) : Microbasic instruction to disable sensor on Pulse input 1

^PMOD 2 4 : Enable Sensor on Pulse input 2

Setconfig(_PMOD, 2, 4) : Microbasic instruction to enable sensor on Pulse input 2

The sensor information can then be accessed with the respective queries as discussed above (?MGD, ?MGT, ?MGM, ?FLW).

Accessing Multiple Sensor Information Simultaneously

It is possible to have all sensors enabled at the same time by having their respective pulse input configured accordingly.

When more than one pulse input is configured that way, the sensor data is accessible using the ?MGD, ?MGT, ?MGM, ?MGY or ?FLW queries as follows, where x is the pulse input number (1, 2, 3 etc.).

Reading Tape Detect

?MGD x or GetValue(_MGD, x)

Returns the Tape Detect state of Sensor at Pulse input x

Example:

?MGD 2 : Returns the Tape Detect state of Sensor 2

Reading Marker Detect

?MGM 2*(x-1)+1 or GetValue(_MGM, 2*(x-1)+1)

Returns the state of the Left Marker Detect state of Sensor at Pulse input x

?MGM 2*(x-1)+2 or GetValue(_MGM, 2*(x-1)+2)

Returns the state of the Right Marker Detect state of Sensor at Pulse input x

Examples:

?MGM 1 : Returns the Left Marker Detect state of Sensor at input 1

?MGM 2 : Returns the Right Marker Detect state of Sensor at input 1

?MGM 3 : Returns the Left Marker Detect state of Sensor at input 2

?MGM 4 : Returns the Right Marker Detect state of Sensor at input 2

Reading Track Positions

?MGT 3*(x-1)+1 or GetValue(_MGT, 3*(x-1)+1)

Returns the Left Track Position of Sensor at input x

?MGT 3*(x-1)+2 or GetValue(_MGT, 3*(x-1)+2)

Returns the Right Track Position of Sensor at input x

?MGT 3*(x-1)+3 or GetValue(_MGT, 3*(x-1)+3)

Returns the Active (Left or Right) Track Position of Sensor at input x

Examples:

?MGT 1 : Returns the Left Track Position of Sensor at input 1

?MGT 2 : Returns the Right Track Position of Sensor at input 1

?MGT 4 : Returns the Left Track Position of Sensor at input 2

?MGT 5 : Returns the Right Track Position of Sensor at input 2

?MGT 7 : Returns the Left Track Position of Sensor at input 3

Reading Flow Sensor Counters

?FLW 2*(z-1)+1 or GetValue(_FLW, 2*(z-1)+1)

Returns the Counter of the X axis of Sensor at Pulse input z,

?FLW 2*(z-1)+2 or GetValue(_FLW, 2*(z-1)+2)

Returns the Counter of the Y axis of Sensor at Pulse input z

Examples:

?FLW 1 : Returns the Counter of the X axis of Sensor at Pulse input 1

?FLW 2 : Returns the Counter of the X axis of Sensor at Pulse input 1

?FLW 3 : Returns the Counter of the Y axis of Sensor at Pulse input 2

?FLW 4 : Returns the Counter of the Y axis of Sensor at Pulse input 2

SECTION 6

Command Modes

This section discusses the controller's normal operation in all its supported operating modes.

Input Command Modes and Priorities

The controller will accept commands from one of the following sources

- Serial data (RS232, RS485, TCP, USB)
- Pulse (R/C radio, PWM, Frequency)
- Analog signal (0 to 5V)
- Network Interface (CAN/EtherCAT/Profinet)
- Microbasic Script

One, many or all command modes can be enabled at the same time. When multiple modes are enabled, the controller will select which mode to use based on a user selectable priority scheme and the hardcoded priorities concerning the Network interface and the Microbasic script. Setting the priorities is done using the PC configuration utility.

This scheme uses a priority table containing three parameters and let you select which mode must be used in each priority order. During operation, the controller reads the first priority parameter and switches to that command mode. If that command mode is found to be active, that command is then used. If no valid command is detected, the controller switches to the mode defined in the next priority parameter. If no valid command is recognized in that mode, the controller then repeats the operation with the third priority parameter. If no valid command is recognized in that last mode, the controller applies a default command value that can be set by the user (typically 0).

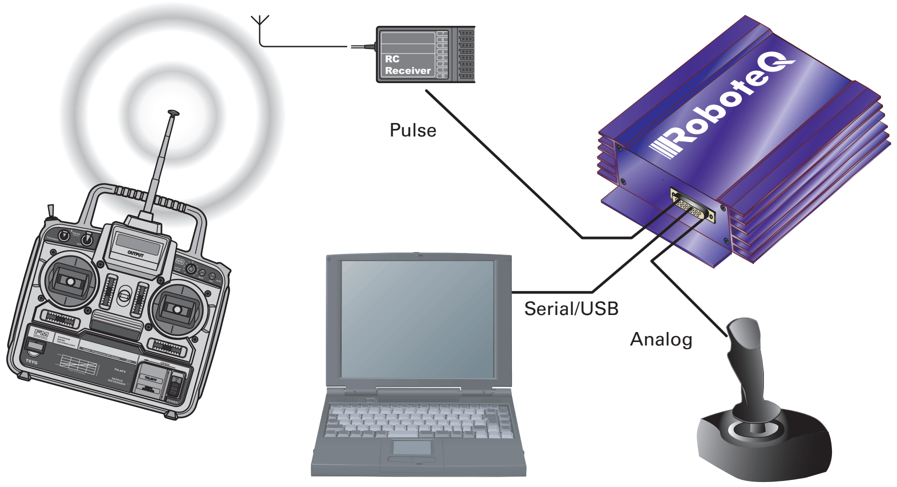


FIGURE 6-1. Controller's possible command modes

In the Serial mode, the mode is considered as active if commands

- !G - Go to Speed or to Relative Position
- !MS - Stop in all modes
- !S - Set Motor Speed
- !TC - Target Torque
- !GIQ - Set Torque Amps
- !GID - Set Flux Amps

arrive within the watchdog timeout period via the RS232, RS485, TCP or USB ports. The mode will be considered inactive, and the next lower priority level will be selected as soon as the watchdog timer expires. Note that disabling the watchdog will cause the serial mode to be always active after the first command is received, and the controller will never switch to a lower priority mode.

If the above mentioned commands are called from a script then the Script mode is enabled and if they are called from either network the Network mode is enabled.

In the pulse mode, the mode is considered active if a valid pulse train is found and remains present.

In analog mode, the mode is considered active at all time, unless the Center at Start safety is enabled. In this case, the Analog mode will activate only after the joystick has been centered. The Keep within Min/Max safety mode will also cause the analog mode to become inactive, and thus enable the next lower priority mode, if the input is outside of a safe range.

The example in Figure 6-1 shows the controller connected to a microcomputer, a RC radio, and an analog joystick. If the priority registers are set as in the configuration below:

1- Serial
2- Pulse
3- Analog

then the active command at any given time is given in the table below.

TABLE 6-1. Priority resolution example

Microcomputer Sending commands	Valid Pulses Received	Analog joystick within safe Min/Max	Command mode selected
Yes	Don't care	Don't care	Serial
No	Yes	Don't care	RC mode
No	No	Yes	Analog mode
No	No	No	User selectable default value

Note that it is possible to set a priority level to "None". For example, the priority table

1 - Serial
2 - RC Pulse
3 - None

will only arbitrate and use Serial or RC Pulse commands.

USB vs Serial Communication Arbitration

Commands may arrive through the RS232, RS485, TCP or the USB port at the same time. They are executed as they arrive in a first come first served manner. Commands that are arriving via USB are replied on USB. Commands arriving via the RS232 are replied on the RS232 and so on. Redirection symbol for redirecting outputs to the other port exists (e.g. a command can be made to respond on USB even though it arrived on RS232).

Network Commands Arbitration

On controllers fitted with a Network interface, commands are processed as they arrive regardless if any other mode, apart from Script mode, is active at the same time. Network mode has the highest priority from all other modes apart from script mode.

Commands issued from MicroBasic scripts

When sending a Motor command from a MicroBasic script, it will be interpreted by the controller with higher priority than any other interface. If a serial command is received from the serial/USB port at the same time a command is sent from the script, the script command will prevail.

Important Warning

Script and CAN commands are also subject to the serial Watchdog timer. Script commands have the highest priority and CAN commands similar priority to serial commands as shown below:

TABLE 6-2. Command Priorities

Priority	Mode	Configurable
1	Script Mode	No
2	Network Mode	No
3, 4, 5	Serial Mode(RS232,RS485,TCP,USB)	Yes (see CPRI)
	Pulse Mode	
	Analog Mode	

Operating the Controller in RC mode

The controller can be directly connected to an R/C receiver. In this mode, the speed or position information is contained in pulses whose width varies proportionally with the joysticks' positions. The controller mode is compatible with all popular brands of RC transmitters.

The RC mode provides the simplest method for remotely controlling a robotic vehicle: little else is required other than connecting the controller to the RC receiver and powering it On.

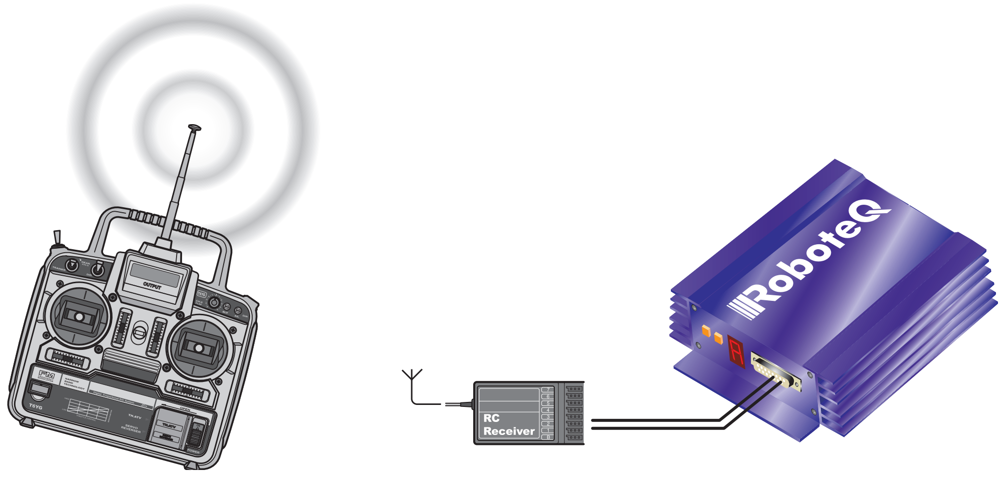


FIGURE 6-2. R/C radio control mode

The speed or position information is communicated to the controller by the width of a pulse from the RC receiver: a pulse width of 1.0 millisecond indicates the minimum joystick position and 2.0 milliseconds indicates the maximum joystick position. When the joystick is in the center position, the pulse should be 1.5ms.

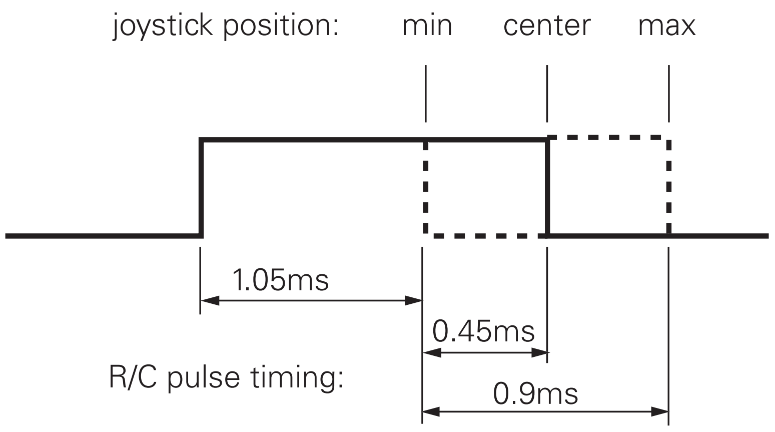


FIGURE 6-3. Joystick position vs. pulse duration default values

The controller has a very accurate pulse capture input and is capable of detecting changes in joystick position (and therefore pulse width) as small as 0.1%. This resolution is superior to the one usually found in most low cost RC transmitters. The controller will therefore be able to take advantage of the better precision and better control available from a higher quality RC radio, although it will work fine with lesser expensive radios as well.

Input RC Channel Selection

The controllers feature several inputs that can be used for pulse capture. See product datasheet for an actual number of pulse input. Any RC input can be used as a command for any motor channels. The controller's factory default defines two channels for RC capture (one input on single channel products). Which channel and which pin on the input connector depends on the controller model and can be found in the controller's datasheet.

Changing the input assignment is done using the PC Configuration utility.

Input RC Channel Configuration

Internally, the measured pulse width is compared to the reference minimum, center, and maximum pulse width values. From this is generated a command number ranging from -1000 (when the joystick is in the min. position), to 0 (when the joystick is in the center position) to +1000 (when the joystick is in the max position). This number is then used to set the motor' desired speed or position that the controller will then attempt to reach.

For best results, reliability, and safety, the controller will also perform a series of corrections, adjustments and checks to the R/C commands, as described below.

Joystick Range Calibration

The Joystick min, max, and center position are adjustable. For best control accuracy, the controller can be calibrated to capture and use your radio's specific timing characteristics and store them into its internal Flash memory. This is done using a simple calibration procedure described page 64.

Deadband Insertion

The controller allows for a selectable amount of joystick movement to take place around the center position before activating the motors. See the full description of this feature at "Deadband Selection" page 65

Command Correction

The controller can also be set to translate the joystick motor commands so that the motor responds differently depending on whether the joystick is near the center or near the extremes. Five different exponential or logarithmic translation curves may be applied. Since this feature applies to the R/C, Analog and RS232 modes, it is described in detail in "Command Correction" page 66, in the General Operation section of the manual.

Reception Watchdog

Immediately after it is powered on, if in the R/C mode, the controller is ready to receive pulses from the RC radio.

If valid pulses are received on any of the enabled Pulse input channels, the controller will consider the RC Pulse mode as active. If no higher priority command is currently active (See "Input Command Modes and Priorities" page 79), the captured RC pulses will serve to activate the motors.

If no valid RC pulses reach the controller for more than 500ms, the controller no longer considers it is in the RC mode and a lower priority command type will be accepted if present.

Important Warning

Some receivers include their own supervision of the radio signals and will move their servo outputs to a safe position in case of signal loss. Using these types of receiver, the controller will always be receiving pulses even with the transmitter off.

Using Sensors with PWM Outputs for Commands

The controller’s Pulse inputs can be used with various types of angular sensors that use contactless Hall technology and that output a PWM signal. These type of sensors are increasingly used inside joysticks and will perform much more reliably, and typically with higher precision than traditional potentiometers.

The pulse shape output from these devices varies widely from one sensor model to another and is typically different from this of RC radios:

- They have a higher repeat rate, up to a couple of kHz.
- The min and max pulse width can reach the full period of the pulse

Care must therefore be exercised when selecting a sensor. The controller will accommodate any pulsing sensor as long as the pulsing frequency does not exceed 250Hz. The sensor should not have pulses that become too narrow - or disappear altogether - at the extremes of their travel. Select sensors with a minimum pulse width of 10us or higher. Alternatively, limit mechanically the travel of the sensor to keep the minimum pulse width within the acceptable range.

A minimum of pulsing must always be present. Without it, the signal will be considered as invalid and lost.

Pulses from PWM sensors can be applied to any Pulse input on the controller’s connector. Configure the input capture as Pulse or Duty Cycle.

A Pulse mode capture measures the On time of the pulse, regardless of the pulse period.

A Duty Cycle mode capture measures the On time of the pulse relative to the entire pulse period. This mode is typically more precise as it compensates for the frequency drifts o the PWM oscillator.

PWM signals are then processed exactly the same way as RC pulses. Refer to the RC pulse paragraphs above for reference.

Operating the Controller In Analog Mode

Analog Command is the simplest and most common method when the controller is used in a non-remote, human-operated system, such as Electric Vehicles.

Input Analog Channel Selection

The controller features 4 to 11 inputs, depending on the model type, that can be used for analog capture. Using different configuration parameters, any Analog input can be used as command for any motor channel.

Changing the input assignment is done using the PC Configuration utility. See "Analog Inputs Configurations and Use" on page 63.

Input Analog Channel Configuration

An Analog input can be Enabled or Disabled. When enabled, it can be configured to capture absolute voltage or voltage relative to the 5V output that is present on the connector. See "Analog Inputs Configurations and Use" on page 63.

Analog Range Calibration

If the joystick movement does not reach full 0V and 5V, and/or if the joystick center point does not exactly output 2.5V, the analog inputs can be calibrated to compensate for this. See "Min, Max and Center adjustment" on page 64 and "Deadband Selection"on page 65.

Using Digital Input for Inverting direction

Any digital input can be configured to change the motor direction when activated. See "Digital Inputs Configurations and Uses" on page 62. Inverting the direction has the same effect as instantly moving the command potentiometer to the same level the opposite direction. The motor will first return to 0 at the configured deceleration rate and go to the inverted speed using the configured acceleration rate.

Safe Start in Analog Mode

By default, the controller is configured so that in Analog command mode, no motor will start until all command joysticks are centered. The center position is the one where the input equals the configured Center voltage plus the deadband.

After that, the controller will respond to changes to the analog input. The safe start check is not performed again until power is turned off.

Protecting against Loss of Command Device

By default, the controller is protected against the accidental loss of connection to the command potentiometer. This is achieved by adding resistors in series with the potentiometer that reduces the range to a bit less than the full 0V to 5V swing. If one or more wires to the potentiometer are cut, the voltage will actually reach 0V and 5V and be considered a fault condition, if that protection is enabled. See "Connecting Potentiometers for Commands with Safety band guards" on page 51.

Safety Switches

Any Digital input can be used to add switch-activated protection features. For example, the motor(s) can be made to activate only if a key switch is turned On, and a passenger is present on the driver's seat. This is done using by configuring the controller's Digital inputs. See "Digital Inputs Configurations and Uses" page 62.

Monitoring and Telemetry in RC or Analog Modes

The controller can be fully monitored while it is operating in RC or Analog modes. If directly connected to a PC via RS232, RS485, TCP or USB, the controller will respond to operating queries (Amps, Volts, Temperature, Power Out, ...) without this having any effect on its response to Analog or RC commands. The PC Utility can therefore be used to visualize in real time all operating parameters as the controller runs. See "Run Tab" in Roborun+ Utility User Manual.

In case the controller is not connected via a bi-directional link, and can only send information one-way, typically to a remote host, the controller can be configured to output a user-selectable set of operating parameters, at a user selectable repeat rate. See "Query History Commands" on page 295.

MicroBasic scripting can also be used to generate a periodic text string containing parameters to monitor.

Using the Controller in Serial (USB/RS232/RS485/TCP) Mode

The serial mode allows full control over the controller's entire functionality. The controller will respond to a large set of commands. These are described in detail in "Serial (RS232/RS485/USB/TCP) Operation" in Section 14.

SECTION 7

Motor Operating Features and Options

This section discusses the controller's operating features and options relating to its motor outputs.

Power Output Circuit Operation

The controller's power stage is composed of high-current MOSFET transistors that are rapidly pulsed on and off using Pulse Width Modulation (PWM) technique in order to deliver more or less power to the motors. The PWM ratio that is applied is the result of a computation that combines the user command and safety related corrections. In closed-loop operation, the command and feedback are processed together to produce the adjusted motor command. The diagram below gives a simplified representation of the controller's operation.

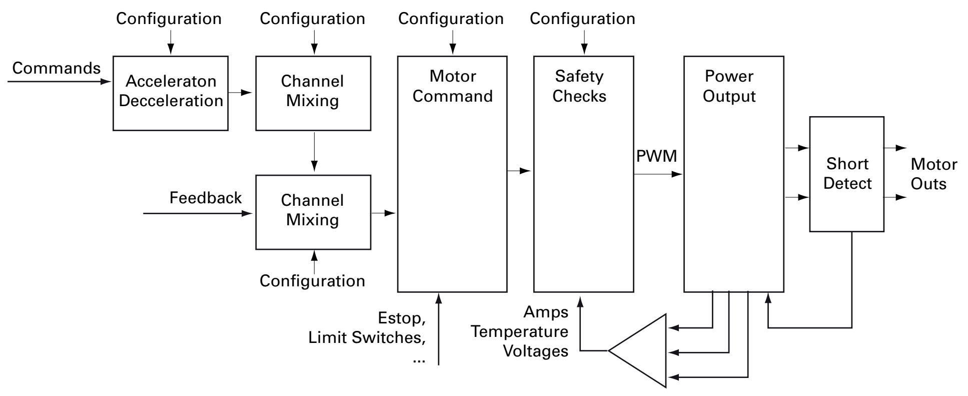


FIGURE 7-1. Simplified diagram of power output operation

Global Power Configuration Parameters

PWM Frequency

The power MOSFETs are switched at 16kHz.

Overvoltage Protection

The controller includes a battery voltage monitoring circuit that will cause the output transistors to be turned Off if the main battery voltage rises above a preset Over Voltage threshold. The value of that threshold is set by default and may be adjusted by the user. The default value and settable range is given in the controller model datasheet.

This protection is designed to prevent the voltage created by the motors during regeneration to be “amplified” to unsafe levels by the switching circuit.

The controller will resume normal operation when the measured voltage drops below the Over Voltage threshold minus a user definable hysteresis voltage, when Automatic Fault Clearance is enabled (see FLCL - Automatic Fault Clearance).

The controller can also be configured to trigger one of its Digital Outputs when an Over Voltage condition is detected. This Output can then be used to activate a Shunt load across the VMot and Ground wires to absorb the excess energy if it is caused by regeneration. This protection is particularly recommended for a situation where the controller is powered from a power supply instead of batteries.

Undervoltage Protection

In order to ensure that the power MOSFET transistors are switched properly, the controller monitors the internal preset power supply that is used by the MOSFET drivers. If the internal voltage drops below a safety level, the controller’s output stage is turned Off. The rest of the controller’s electronics, including the microcomputer, will remain operational as long as the power supply on VMot is above the minimum voltage specified in the product datasheet or if Power Control is above 11V.

Additionally, the output stage will be turned off when the main battery voltage on VMot drops below a user configurable level that is factory preset at 5V. The fault clears as long as the voltage goes above the undervoltage Limit, when Automatic Fault Clearance is enabled (see FLCL - Automatic Fault Clearance).

Temperature-Based Protection

The controller features active protection which automatically reduces power, based on measured operating temperature. This capability ensures that the controller will be able to work safely with practically all motor types and will adjust itself automatically for the various load conditions.

There are 3 types of temperatures:

- MCU temperature
- Heatsink Temperature which is sensed inside the product, close to the MOSFETs.
- Motor temperature which is measured via analog inputs.

When the measured temperature reaches 5°C below the Over temperature limit, the controller’s maximum allowed power output begins to drop by 20% for every degree until the temperature reaches the Over temperature limit. Above this limit, the controller’s power stage turns itself off completely. For the MCU temperature the overtemperature limit is hardcoded at 95°C.

Note that the measured temperature is measured on the heat sink near the Power Transistors and will rise and fall faster than the outside surface.

The time it takes for the heat sink's temperature to rise depends on the current output, ambient temperature, and available air flow (natural or forced). The fault clears as long as the temperature goes below the limit, when Automatic Fault Clearance is enabled (see FLCL - Automatic Fault Clearance).

Current Limiting

The controller does not allow the motor current to go beyond the configuration value ALIM (see **ALIM - Amps Limit**). So even if the command requirements are higher the controller goes into current control and limits the power applied to the motor in order to keep the current inside the configured limits.

I2T Protection

I2T protection is a more indirect but more responsive method for protecting motors mainly from overheating. The concept is based on one number which is the I2T accumulator and is calculated by the following formula:

$$\text{I2T\_accumulator} \mathrel{+}= (I_{now}^{2} - I_{Nom}^{2}) * \mathit{time}.$$

where

- I_{now} is the current that is measured for specific time.
- I_{Nom} is the nominal current of the motor. This is the current under which the motor can run continuously. This value is set using the configuration command NOMA (NOMA - Nominal Current).

So when motor draws more current than the nominal then I2T accumulator increases. When motor draws less current than the nominal, the I2T accumulator decreases. I2T accumulator is compared with a maximum value called I2T limit and is calculated by the following formula.

$$\text{I2T\_limit} = (\text{IPeak}^{2} - \text{INom}^{2}) * \text{peak\_time}.$$

where

- I_{Peak} is the maximum current the motor can handle for specific time. This value is set using the configuration command ALIM (see **ALIM - Amps Limit**).
- peak_time is the maximum time that the motor can handle current similar to IPeak. This value is set using the configuration command TPAL (see **TPAL - Time for Amps Limit**).

If I2T accumulator becomes bigger than I2T limit then the controller will stop limiting the current at ALIM and start limiting the current at 80% of the nominal current (NOMA). In that way the motor will cool down and I2T accumulator will respectively decrease. This limitation will remain until I2T accumulator goes below the 10% of I2T limit.

Respectively the I2T limit of the controller is calculated based on their specifications as stated in the datasheet. In order to protect the controller we compare the two I2T limits (the controller's hardcoded one and the motor's configured one) and will implement the feature using the smaller value of the two limits.

Important Note

If TPAL value is set to 0 then I2T protection is deactivated.

Short Circuit Protection

The controller has two levels of short circuit protection, software and hardware protection.

The firmware utilizes the current sensors in order to detect short circuit. How the firmware will behave depends on the **THLD - Short Circuit Detection Sensitivity**.

- If set as High Sensitivity (0, default), then it works as an over-current threshold detector. The current threshold is defined at each product's datasheet. So if current goes above that threshold the short fault is triggered.
- If set as Medium Sensitivity (1) then when the current goes beyond the current threshold, the controller deactivates the MOSFETs for 5ms and then recovers. If that happens more than 3 times in a 128ms period then short fault is triggered.
- If set as Low Sensitivity (2) then when the current goes beyond the current threshold, the controller deactivates the MOSFETs for 5ms and then recovers. If that happens more than 7 times in a 128ms period then short fault is triggered.

The short fault will remain until the next idle motor command is given (0 in case of speed modes, equal to feedback in case of position modes).

The following shorts can be detected (not in all boards, please refer to the specific datasheet for the "Short Circuit Detection threshold"):

- Between 2 motor output phases of the same channel
- Between motor output phase and VMOT
- Between motor output phase and PWR_GND

It needs also to be noted that the software short circuit protection can also be triggered due to high current ripple conditions, which are mostly due to low motor inductance values. Please refer to the datasheet for the minimum inductance value.

Closed Loop Error Protection

The controller will detect large tracking errors due to mechanical or sensor failures and stop the motor. The detection mechanism looks for the size of the tracking error and the duration the error is present. Three predefined levels of sensitivity are provided in the controller configuration along with a custom option:

0: Disabled

1: 250ms and Error > 100 units

2: 500ms and Error > 250 units.

3: 1000ms and Error > 500 units.

4: Custom

Where Error represent:

- Ramped Command – Feedback in RPM in Closed Loop Speed mode,
- Ramped Command – Feedback in Amps*10 in Closed Loop Torque mode,

- Track – Feedback in Counts in Closed Loop Speed Position mode and Closed Loop Count Position mode,
- Track – Feedback in Fraction of position counts (-1000 – 1000) in Closed Loop Position Relative mode and Closed Loop Tracking Position mode.

When an error is triggered, the motor channel is stopped by controlling speed using Fault Deceleration. The motor remains stopped until being cleared by issuing an idle motor command (0 for speed modes or equal to feedback for position modes).

Clearing the loop error make the motor available for moving again. However, this does not mean that the loop error will not happen again. Configuration tuning is necessary in order to prevent from having Loop Error again. The loop error value can be monitored in real time using the Roborun PC utility.

If Custom option is configured then the loop error will work based on the values in the fields Loop Error Limit and Loop Error Time (see configuration commands FEW - Loop Error Limit and FET - Loop Error Time for more details).

These values can also be modified during runtime using the respective runtime commands FEW and FET, or via the respective SDOs in DS402 mode (applicable only for position modes).

Important Note

The predefined levels might not be applicable in Closed Loop Count Position, especially in case the sensor, used as feedback, has high resolution. In cases like that the loop error detection can be disabled and a custom method can be implemented via scripting or use the FET and FEW runtime commands via DS402 mode.

Similarly for Closed Speed Position mode, since the command is speed and the feedback is position, it is recommended to have loop error detection level to disabled.

Mixed Mode Select

Mixed mode is available as a configuration option in dual channel controllers to create tank-like steering when one motor is used on each side of the robot: Channel 1 is used for moving the robot in the forward or reverse direction. Channel 2 is used for steering and will change the balance of power on each side to cause the robot to turn. Figure 7-2 below illustrates how the mixed mode motor arrangement.

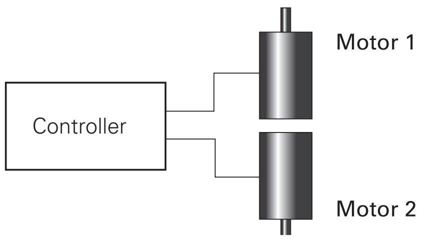


FIGURE 7-2. Effect of commands to motor examples in mixed mode

The controller supports 3 mixing algorithms with different driving characteristics. The table below shows how each motor output responds to the two commands in each of these modes.

TABLE 7-1. Mixing Mode characteristics

Input		Mode 1		Mode 2		Mode 3	
Throttle	**Steering**	**M1**	**M2**	**M1**	**M2**	**M1**	**M2**
0	0	0	0	0	0	0	0
0	300	300	-300	300	-300	300	-300
0	600	600	-600	600	-600	600	-600
0	1000	1000	-1000	1000	-1000	1000	-1000
0	-300	-300	300	-300	300	-300	300
0	-600	-600	600	-300	300	-600	600
0	-1000	-1000	1000	-1000	1000	-1000	1000
300	300	600	0	600	0	522	90
300	600	900	-300	900	-300	762	-120
300	1000	1000	-700	1000	-1000	1000	-400
300	-300	0	600	0	600	90	522
300	-600	-300	900	-300	900	-120	762
300	-1000	-700	1000	-1000	1000	-400	1000
600	300	900	300	900	300	708	480
600	600	1000	0	1000	-200	888	360
600	1000	1000	-400	1000	-1000	1000	200
600	-300	300	900	300	900	480	708
600	-600	0	1000	-200	1000	360	888
600	-1000	-400	1000	-1000	1000	200	1000
1000	300	1000	700	1000	400	900	1000
1000	600	1000	400	1000	-200	1000	1000
1000	1000	1000	0	1000	-1000	1000	1000
1000	-300	700	1000	400	1000	1000	900
1000	-600	400	1000	-200	1000	1000	1000
1000	-1000	0	1000	-1000	1000	1000	1000

Motor Channel Parameters

User Selected Current Limit Settings

The controller has current sensors at each of its output stages. This current is measured and a correction to the current control is applied if higher than the user preset value.

The current limit may be set using the supplied PC utility. The maximum limit is dependent on the controller model and can be found on the product datasheet.

The limitation is performed on the Motor current and not on the Battery current. See "Battery Current vs. Motor Current" on page 28 for a discussion of the differences.

Selectable Amps Threshold Triggering

The controller can be configured to detect when the Amp on a motor channel exceeds a user-defined threshold value and trigger an action if this condition persists for more than a preset amount of time.

The list of actions that may be triggered is shown in the table below.

TABLE 7-2. Possible Action List when Amps threshold is exceeded

Action	Applicable Channel	Description
No Action	-	Input causes no action
Quick Stop	Selectable	Stops the respective motor by controlling speed using Fault Deceleration. The motor remains in Quick Stop until an idle motor command is given (0 in case of speed modes, equal to feedback in case of position modes).
Emergency stop	All	Sets all the MOSFETs that drive the respective motor to float. So no power is applied to the motor and the motor is about to stop due to friction.

This feature is very different from amps limiting. Typical uses for it are for stall detection or "soft limit switches". When, for example, a motor reaches an end and enters stall condition, the current will rise, and that current increase can be detected and the motor be made to stop until the direction is reversed.

Programmable Acceleration & Deceleration

When changing speed command, the controller will go from the present speed to the desired one at a user selectable acceleration. This feature is necessary in order to minimize the surge current and mechanical stress during abrupt speed changes.

This parameter can be changed by using the PC utility. Acceleration can be different for each motor. A different value can also be set for the acceleration and for the deceleration. The acceleration value is entered in RPMs per second. In open loop installation, where speed is not actually measured, the acceleration value is relative to the Max RPM parameter. For example, if the Max RPM is set to 1000 (default value) and acceleration to 2000, this means that the controller will go from 0 to 100% power in 0.5 seconds. In closed loop Torque Mode the Acceleration and Deceleration values are entered in miliAmps per second. In case of Quick Stop the motor will ramp down based on the Fault Deceleration configuration value.

In order to by-pass the ramping process, either the acceleration or the deceleration values need to be set to 0.

Important Warning

Depending on the load's weight and inertia, a quick or no acceleration can cause considerable current surges from the batteries into the motor. A quick or no deceleration will cause an equally large, or possibly larger, regeneration current surge. Always experiment with the lowest acceleration value first and settle for the slowest acceptable value.

Furthermore, by by-passing command ramp the controller cannot protect itself from high currents.

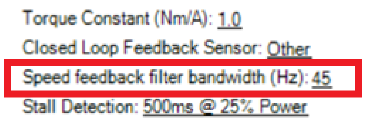
Forward and Reverse Power Adjustment Gain

This parameter lets you select the scaling factor for the power output as a percentage value. This feature is used to connect motors with a voltage rating that is less than the battery voltage. For example, using a factor of 50% it is possible to connect a 12V motor onto a 24V system, in which case the motor will never see more than 12V at its input even when the maximum power is applied.

Note: With this feature the voltage applied is not stepped down. The limit is applied in the duty cycle. So as in the above mentioned example, the duty cycle will be limited to 50% with 24V, thus giving in the end 12V.

Speed feedback filter

For stable and smooth controller closed loop speed operation, a first order Infinite Impulse Response (IIR) low pass filter has been implemented in order to effectively filter the speed oscillations provoked from sensor noise, non-linearity, or mechanical system resonances, adding very small phase delay to the output filtered signal. The user can configure the low pass filter bandwidth by setting the following parameters in the configuration tab:


Torque Constant (Nm/A): 1.0
Closed Loop Feedback Sensor: Other
Speed feedback filter bandwidth (Hz): 45
Stall Detection: 500ms @ 25% Power

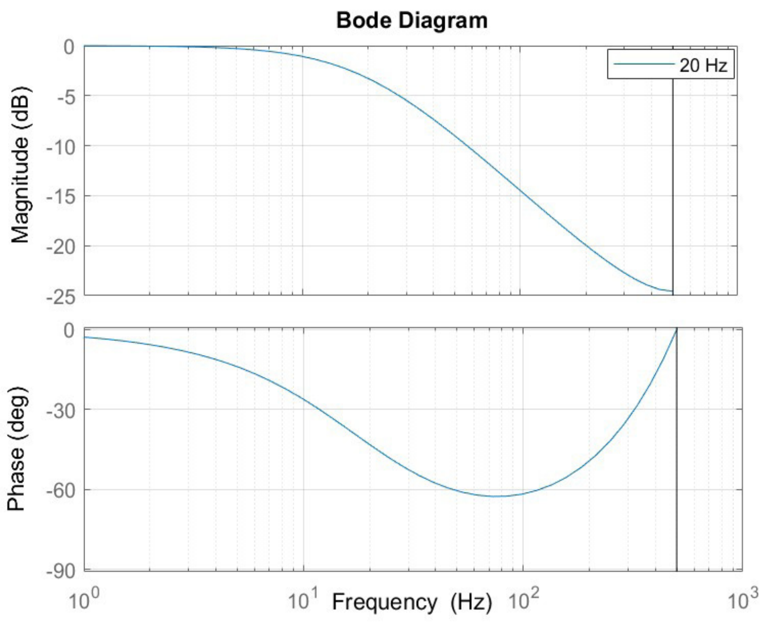
or by sending the following configuration commands from Console:

^LPFB ch nn,

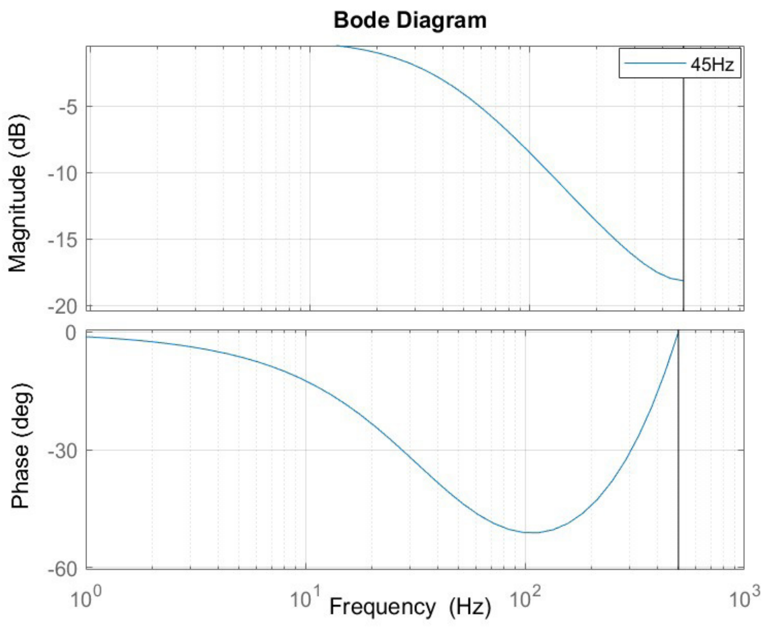
where ch: motor channel, nn: low pass filter bandwidth (Hz)

The available bandwidth range is from 1 Hz to 150 Hz. Therefore, the user could select the most appropriate bandwidth value for the application, compensating between speed filtering and phase delay. Default value has been selected equal to 45 Hz. Below are the respective bode diagrams of the implemented IIR filter for bandwidth values equal to 20, 45 and 80 Hz, illustrating the filtering capability and filtering capability.

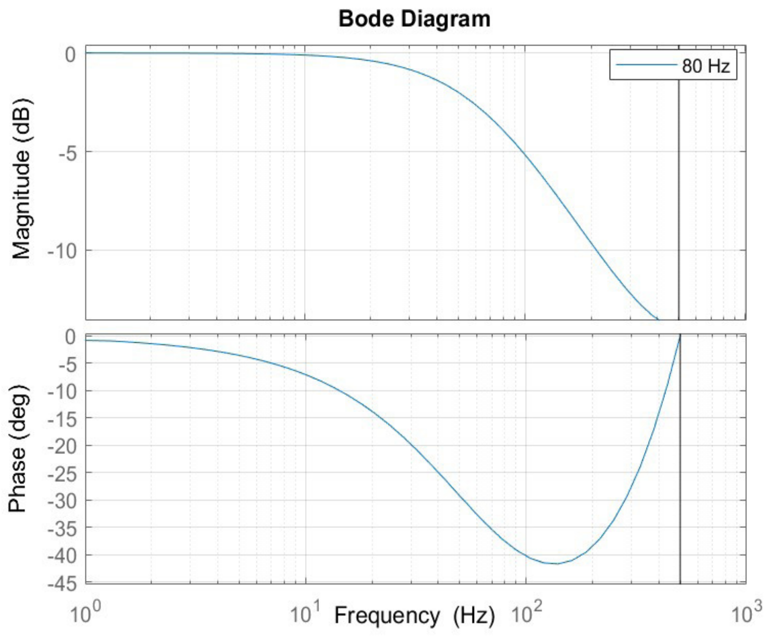
It is also recommended for stability issues the speed filter bandwidth to be quite higher than the speed PI control loop bandwidth. This low pass filter is applicable for encoder, SSI, sin/cos and resolver angle sensors, when used as speed feedback.


Bode Diagram
20 Hz
Magnitude (dB)
0
-5
-10
-15
-20
-25
Phase (deg)
0
-30
-60
-90
10^0
10^1
Frequency (Hz)
10^2
10^3

(a)


Bode Diagram
45Hz
Magnitude (dB)
-5
-10
-15
-20
Phase (deg)
0
-30
-60
10^0
10^1
Frequency (Hz)
10^2
10^3

(b)



(c)

FIGURE 7-3. Frequency response of speed feedback low pass filter for (a) 20 Hz (b) 45 Hz and (c) 80 Hz bandwidth rate.

Selecting the Motor Control Modes

For each motor, the controller supports multiple motion control modes. The controller's factory default mode is Open Loop Speed control for each motor. The mode can be changed using the Roborun PC utility.

Open Loop Speed Control

In this mode, the voltage supplied to the motor is directly proportional to the command given to the motor. The actual motor speed is not controlled, which means that the motor speed will vary with load changes. This mode should only be used for the initial configuration and testing of the motor, and it should not be employed in the final implementation. The reason is that the motor's current is not controlled, and it can increase rapidly in response to significant variations in motor voltage or back electromotive force.

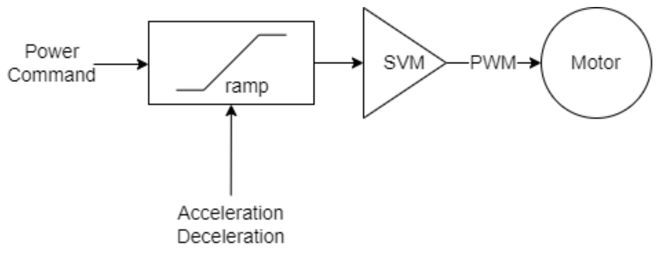


FIGURE 7-4. Open loop mode

Important Note

Open Loop should only be used for the initial configuration and testing of the motor, and it should not be employed in the final implementation.

Closed Loop Speed Control

In this mode, illustrated in Figure 7-5, an optical encoder (typical) or an analog tachometer is used to measure the actual motor speed. If the speed changes because of changes in load, the controller automatically compensates the power output. This mode is preferred in precision motor control and autonomous robotic applications. Details on how to wire the tachometer can be found in "Connecting Tachometer to Analog Inputs" on page 52. Closed Loop Speed control operation is described in "Closed Loop Speed Mode" in Section 10. On brushless motors, speed may be sensed directly from the motor's Hall or other internal Sensors and closed loop operation is possible without additional hardware.

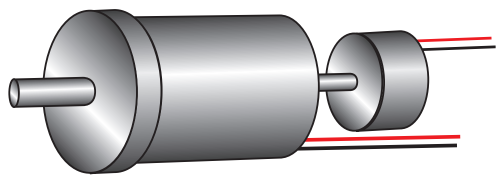


FIGURE 7-5. Motor with tachometer or Encoder for Closed Loop Speed operation

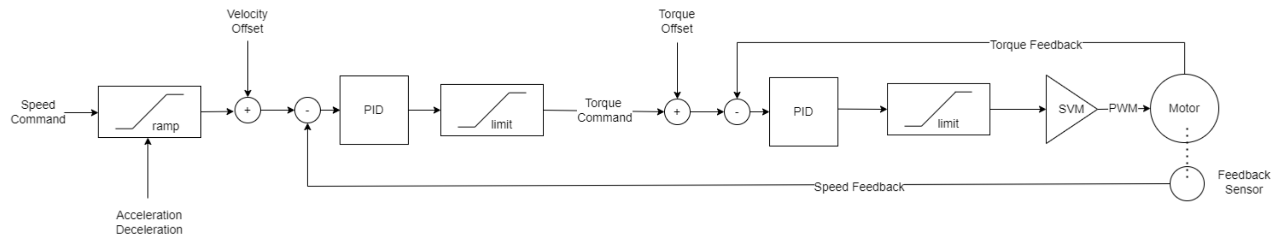


FIGURE 7-6. Closed loop Speed mode

Closed Loop Speed Position Control

In this mode, the controller computes the position at which the motor must be at every 1ms. Then a position loop compares that expected position with the current position and applies the necessary power level in order for the motor to reach that position. This mode is especially effective for accurate control at very slow speeds. Details on this mode can be found in "Closed Loop Speed and Speed-Position Modes"w in Section 10.

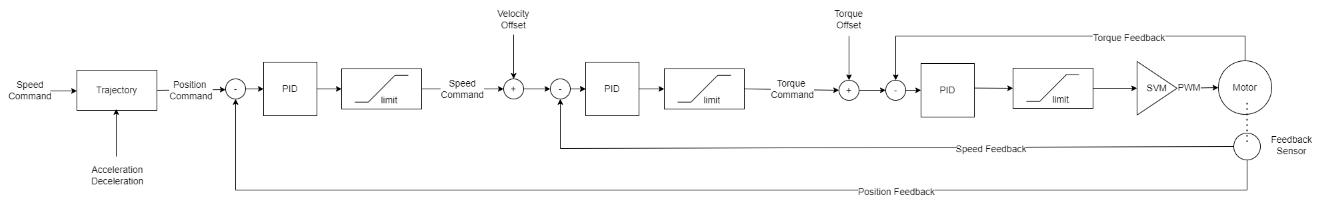


FIGURE 7-7. Closed Loop Speed Position mode

Closed Loop Position Relative Control

In this mode, illustrated in Figure 7-8, the axle of a geared down motor is typically coupled to a position sensor that is used to compare the angular position of the axle versus a desired position. The motor will move following a controlled acceleration up to a user defined velocity and decelerate to smoothly reach the desired destination. This feature of the controller makes it possible to build ultra-high torque "jumbo servos" that can be used to drive steering columns, robotic arms, life-size models and other heavy loads. Details on how to wire the position sensing potentiometers and operating in this mode can be found in "Closed Loop Relative and Tracking Position Modes" in Section 11.

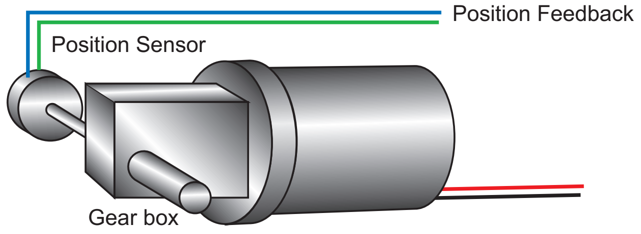


FIGURE 7-8. Motor with potentiometer assembly for position operation

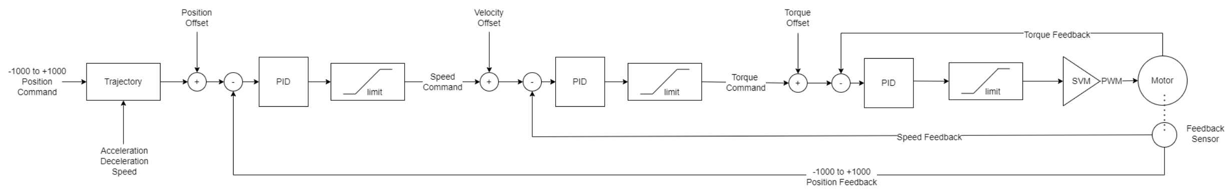


FIGURE 7-9. Closed Loop Position Relative mode

Closed Loop Count Position

In this mode, an encoder is attached to the motor as for the Speed Mode of Figure 7-9. Then, the controller can be instructed to move the motor to a specific number of counts, using a user-defined acceleration, velocity, and deceleration profile. Details on how to configure and use this mode can be found in "Closed Loop Count Position Mode" in Section 12. On brushless motors, the hall sensors can be also be used for position measurement.

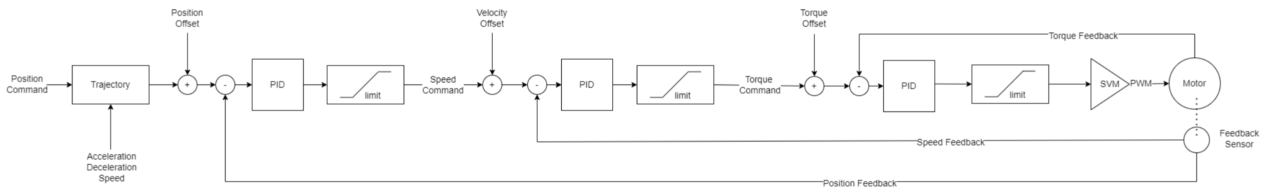


FIGURE 7-10. Closed Loop Count Position mode

Closed Loop Position Tracking

This mode uses the same feedback sensor mount as this of Figure 7-10. In this mode, the motor will be moved until the final position measured by the feedback sensor matches the command. The motor will move as fast as it possibly can, using maximum physical acceleration. This mode is best for systems where the motor can be expected to move as fast as the command changes. Details on this operating mode can be found in "Closed Loop Relative and Tracking Position Modes" in Section 11.

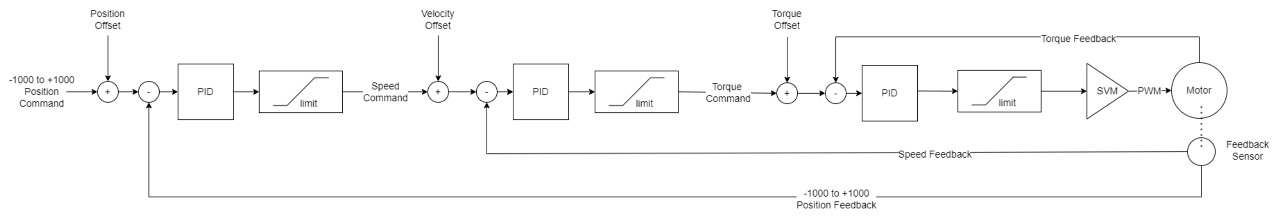


FIGURE 7-11. Closed Loop Position Tracking mode

Torque Mode

In this closed loop mode, the motor is driven in a manner that it produces a desired amount of torque regardless of speed. This is achieved by using the motor current as of the feedback. Torque mode does not require any specific wiring. The detail on this operating mode can be found in "Closed Loop Torque Mode" in Section 13.

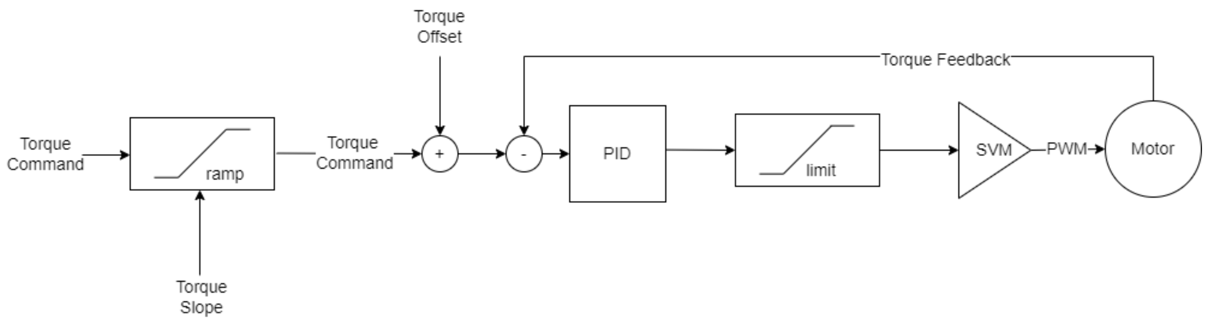


FIGURE 7-12. Closed Loop Torque mode

Motion Control Modes Overview

In the figure 7-13, the block diagram of the whole motion control modes architecture supported in RoboteQ controllers. The motion control mechanism utilizes a cascaded PID control technique for position, speed and current control modes. More details for each control mode operation and tuning are included in separate chapters at sections 10-13, respectively.

Note: The dashed lines at the output of position and speed controllers illustrate the alternative operating scheme of each mode in case the speed or FOC torque gains respectively are equal to zero.

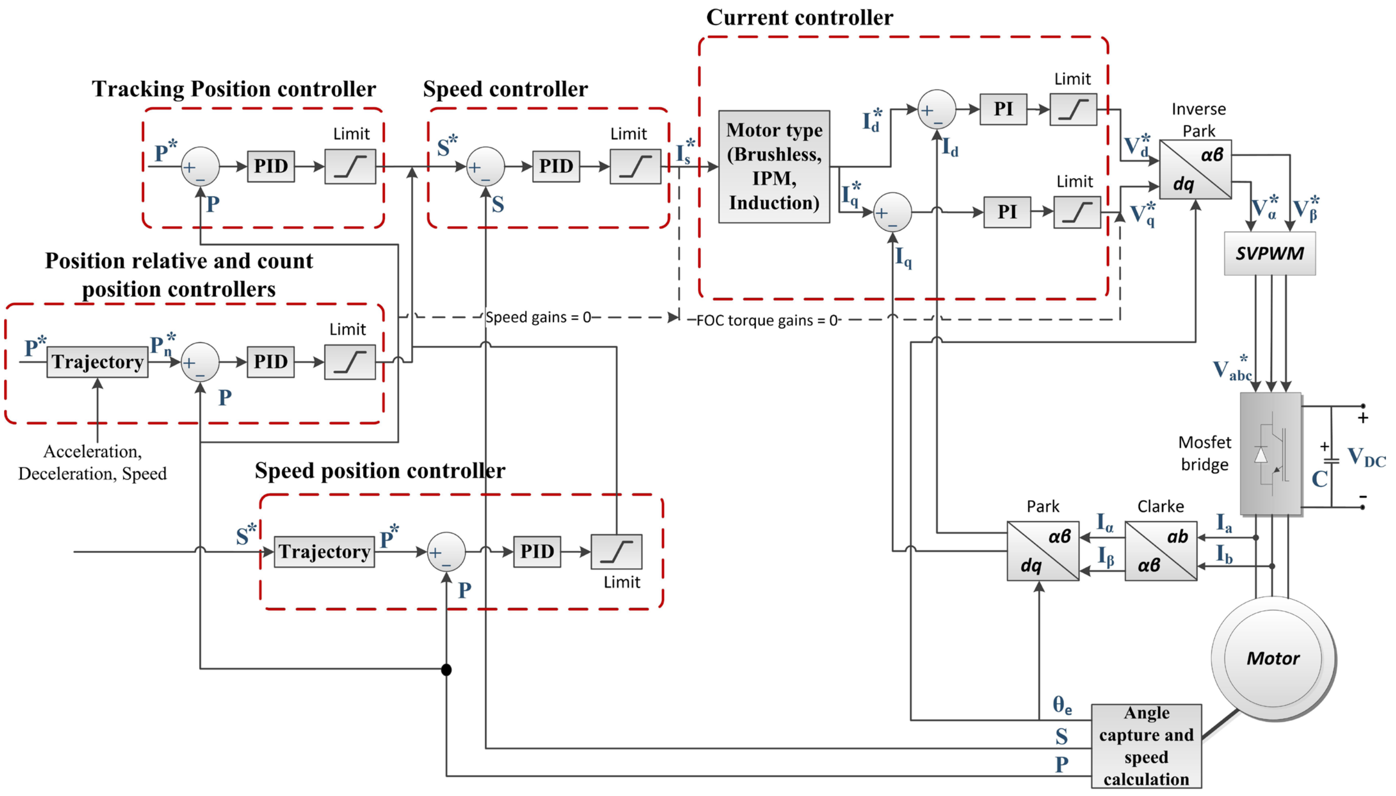

Current controller
Tracking Position controller
Speed controller
Position relative and count position controllers
Speed position controller
Limit
PID
PI
Trajectory
Acceleration, Deceleration, Speed
Motor type (Brushless, IPM, Induction)
Speed gains = 0
FOC torque gains = 0
Inverse Park
SVPWM
Mosfet bridge
Park
Clarke
Motor
Angle capture and speed calculation
V_{DC}
C

FIGURE 7-13. Close loop operating modes diagram

Below is the description of the parameters illustrated in Figure 7-13.

P* = position reference command value, in counts at count position and speed position modes and in -1000/1000 command at tracking position and position relative modes

P = measured position from motor sensor, in counts at count position and speed position modes and in -1000/1000 command at tracking position and position relative modes

P_n = position reference command value every msec, calculated from trajectory function according to configured acceleration, deceleration and speed

S* = speed reference command value in rpm

S = measured motor mechanical speed in rpm

I_s* = reference current command value in Ampere

I_q* = q-axis reference current command value in Ampere (Torque Amps command)

I_d* = d-axis reference current command value in Ampere (Flux Amps command)

I_q* = q-axis current measurement in Ampere (Torque Amps)

I_d* = d-axis current measurement in Ampere (Flux Amps)

V_q* = q-axis reference voltage command Volts

V_d* = d-axis reference voltage command in Volts

V_{abc}* = reference instantaneous applied 3-phase voltage command at mosfet bridge in Volts

I_a = measured motor phase A instantaneous current

I_b = measured motor phase B instantaneous current

θ_e = measured motor electric angle

Feedforward terms

Feedforward control is a powerful method to increase the performance of the speed and position loops. In particular, there are two types of feedforward control working in parallel with the PID regulators, the acceleration feedforward control applied at speed loop and velocity feedforward control applied at position loop.

Acceleration feedforward control

The acceleration feedforward control is applied by extension in the Closed loop Speed mode and in cascaded position control modes, that use it. The purpose of the acceleration feedforward terms is to increase the responsiveness of the speed loop during transition states (acceleration, deceleration), working in parallel with the existing PI controller. The function is applicable to BL products. An overview of the function is illustrated in below figure.

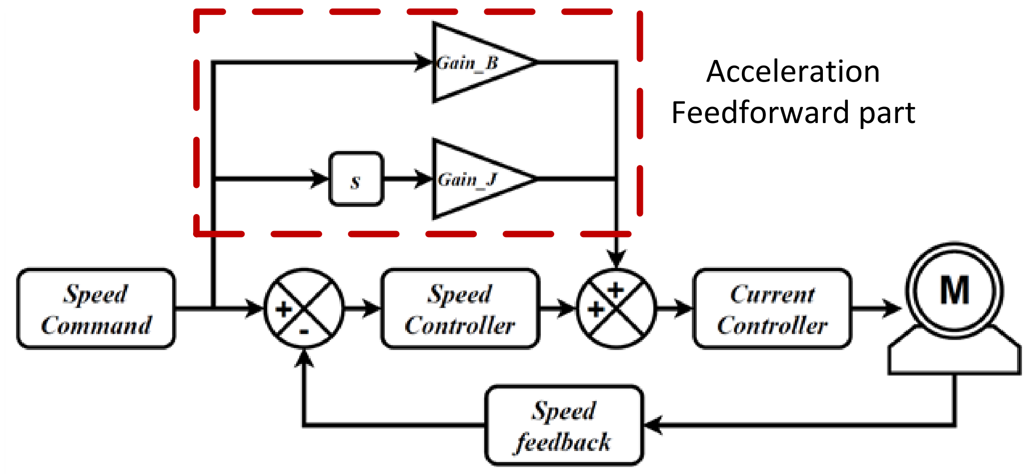


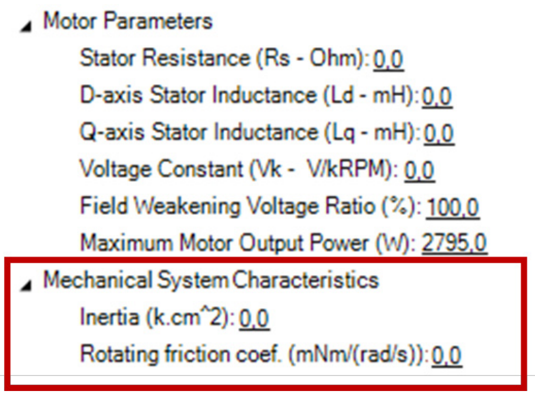
FIGURE 7-14. Block diagram of speed control with acceleration feedforward.

Below the description of the parameters illustrated in above figure.

Gain B: mechanical system rotating friction coefficient term

Gain J: mechanical system inertia term

It is evident that the respective feedforward parameters for speed control loop are mainly affected by mechanical system characteristics. In order to enable the acceleration feedforward control, the following parameters in the configuration needed to be set:



Motor Parameters
- Stator Resistance (Rs - Ohm): 0.0
- D-axis Stator Inductance (Ld - mH): 0.0
- Q-axis Stator Inductance (Lq - mH): 0.0
- Voltage Constant (Vk - V/kRPM): 0.0
- Field Weakening Voltage Ratio (%): 100.0
- Maximum Motor Output Power (W): 2795.0

Mechanical System Characteristics
- Inertia (k.cm^2): 0.0
- Rotating friction coef. (mNm/(rad/s)): 0.0

or by sending the following configuration commands from Console:

^JR ch nn,

where ch: motor channel, nn: mechanical system inertia (kg*m^2) * 10000000

^BR ch nn

where ch: motor channel, nn: mechanical system rotating friction coefficient (Nm/(rad/s)) * 10000000

These parameters could be automatically calculated from the Motor Sensor and Tuning setup wizard, supported in Roborun+ v3.0 utility (see Roborun+ Utility User Manual for more details). By setting both the above parameters with values higher than 0, the acceleration feedforward control is ***automatically*** enabled for speed control and position control modes when speed gains are used. Disabling can be done by zeroing the mechanical system inertia or friction coefficient. Default values for mechanical system inertia/friction are zero, meaning the acceleration feedforward algorithm is by default OFF.

Example

Below is a representative speed response test implemented. The PI parameters were got from the motor sensor and tuning setup wizard.

J : $9.44 * 10^{-5}$ kg · m^2

B : $2.55 * 10^{-4}$ Nm/($\frac{rad}{sec}$)

Control mode: Closed loop speed

Speed loop bandwidth selected: 10Hz

Speed command applied: 0-1500RPM

Acceleration Ramp: 14000 RPM/sec

Current control bandwidth: 800 Hz

Current control Decoupling terms: Enabled

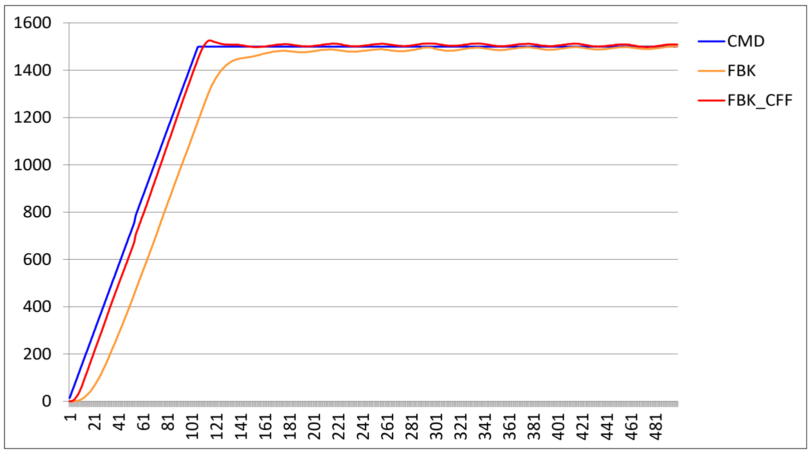


FIGURE 7-15. Speed response test result.

From the above results the much quicker speed response can be observed with the acceleration feedforward terms enabled during the acceleration transient stage.

Velocity feedforward control

The velocity feedforward control is applied in all Closed loop Position modes. The purpose of velocity feedforward term is to enhance the responsiveness of the position loop during transition states (acceleration, deceleration), working in parallel with the existing PI controller. The function is applicable to BL products. An overview of the function for count position mode is illustrated in below figure, where the Velocity Gain is the feedforward gain set by the user. A similar feedforward implementation exists in other position modes.

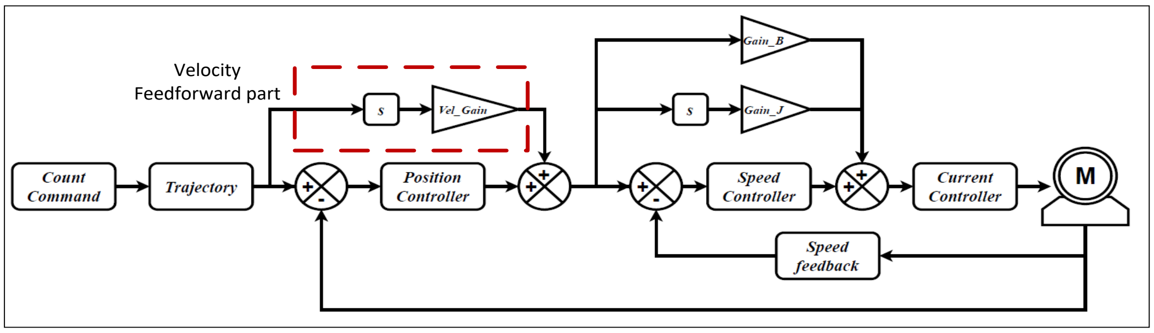
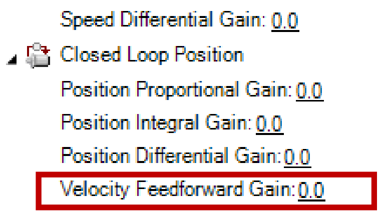


FIGURE 7-16. Block diagram of count position control with velocity feedforward control.

In order to enable the velocity feedforward control, velocity feedforward parameter in the configuration needs to be set:


Speed Differential Gain: 0.0
Closed Loop Position
Position Proportional Gain: 0.0
Position Integral Gain: 0.0
Position Differential Gain: 0.0
Velocity Feedforward Gain: 0.0

The parameter can be also set by sending the following configuration commands from Console:

^VELFF ch nn,

where ch: motor channel, nn: feedforward gain value *1000

By setting the above parameter with value higher than 0, the velocity feedforward control is ***automatically*** enabled for position control modes. Disabling can be done by zeroing the velocity feedforward gain value. The default value for the velocity feedforward gain is zero, meaning that the velocity feedforward algorithm is by default OFF. The typical velocity feedforward gain value is 1.0. For values higher than 1.0, the position loop is more aggressive, while for values lower than 1.0 the position loop is more tolerant in system disturbances but less responsive.

Example

Below is a representative position response test. The parameters of the test are the following:

Control mode: Closed loop Count position

Feedforward gain: 1.0

Acceleration feedforward control: Enabled

Decoupling current control: Enabled

Current loop: 800 Hz

Speed loop bandwidth: 10 Hz

Position loop bandwidth: 1 Hz

Position command applied: 1 revolution (16384 counts)

Acceleration Ramp: 14000 RPM/sec

Deceleration Ramp: 14000 RPM/sec

Encoder counts per revolution: 16384

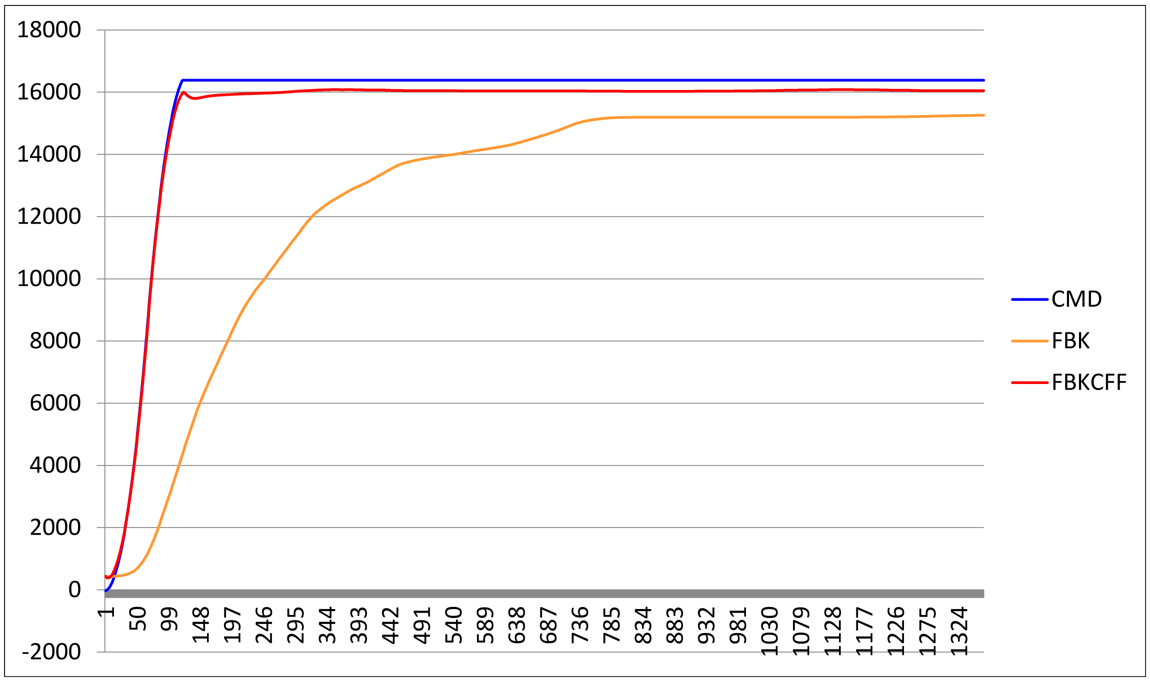


FIGURE 7-17. Position response test result.

From the above results the much quicker position response can be observed with the velocity feedforward terms enabled. It is noted that the effectiveness of the respective feedforward terms is evident during acceleration/deceleration stage.

DS402 Homing Function

Roboteq drives support homing using a homing sensor in accordance with the DS402 standard. The homing methods supported are 17-30 and 35, as defined in the standard. The homing switch can be configured on any supported digital input by setting the input action to 'Load home counter.' Consequently, the 'Home count' parameter will be loaded with the sensor counter value. The default value for this parameter is zero. Configuring the homing switch can be done through serial configuration commands, Roborun+ utility and CANOpen. It should be noted that the digital inputs scan the sensor at 1 ms intervals and do not operate on an interrupt basis.

Table 7.3 lists the relevant serial commands and corresponding CANopen objects for configuring the homing sensor. For more detailed information, please refer to the serial commands reference section and Roboteq's CAN Networking manual.

TABLE 7-3. Homing Command References

Command	CANopen ID	Description
HMD	0X6098	Homing Method
HSP	0X6099	Homing Speed
DINA	0x301A	Digital Input Action
DINL	0x3010	Digital Input Active Level
BHOME	0x303C	Brushless Internal Sensor Home Count
EHOME	0x304F	Encoder Home Count
SHOME	0x30DB	Sensor Home Count

SECTION 8

Brushless Motor Connections and Operation

This section addresses installation and operating issues specific to brushless motors. It is applicable only to brushless motor controller models.

Important Warning

This Manual refers to Firmware 3.x of Roboteq Motor Controllers. Many of the described features are either not available or do not work the same way (PID gains in particular) than in Firmware 2.x or prior. Refer to Manual v2.1a for earlier Firmware.

Introduction to Brushless Motors

Brushless motors, or more accurately Brushless DC Permanent Magnet Synchronous motors (since there are other types of motors without brushes) contain permanent magnets and electromagnets. The electromagnets are arranged in groups of three and are powered in sequence in order to create a rotating field that drives the permanent magnets. The electromagnets are located on the non-rotating part of the motor, which is normally in the motor casing for traditional motors, in which case the permanent magnets are on the rotor that is around the motor shaft. On hub motors, such as those found on electric bikes, scooters and some other electric vehicles, the electromagnets are on the fixed center part of the motor and the permanent magnets on the rotating outer part.

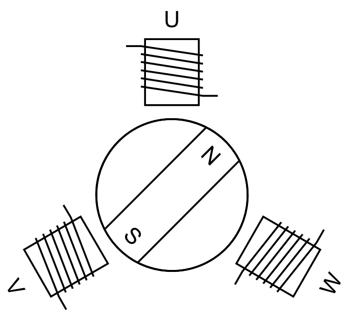


FIGURE 8-1. Permanent Magnet Synchronous Motor construction

As the name implies, Brushless motors differ from traditional DC motors in that they do not use brushes for commutating the electromagnets. Instead, it is up to the motor controller to apply, in sequence, current to each of the 3 motor windings in order to cause the rotor to spin. There are fundamentally two methods of generating the rotating magnetic field in the motor's winding:

- Trapezoidal Commutation
- Sinusoidal Commutation

Within each commutation method is then a method for detecting the actual position of the rotor in order to synchronize the generated rotating field. These are:

- Hall sensors
- Encoders (Absolute or relative)

All Roboteq brushless controllers support Trapezoidal with Hall sensor feedback and sinusoidal with various sensors for angle estimation and feedback.

Number of Poles

One of the key characteristics of a brushless motor is the number of poles of permanent magnets pairs it contains. A full 3-phase cycling of motor's electromagnets will cause the rotor to move to the next permanent magnet pole. A full 3-phase cycle is known as electrical turn which will be different from the physical (mechanical) turn of the shaft if the motor number of pole pairs is greater than one: increasing the number of pole pairs will cause the motor to rotate more slowly for a given rate of change on the winding's phases.

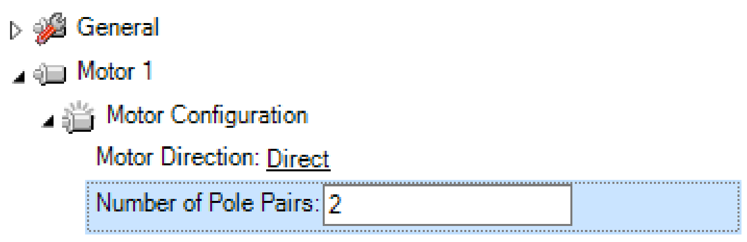
Roboteq controllers use the number of motor pole pairs to measure the number of turns a motor has made as well as motor speed.

Determining the Number of Poles

The number of pole pairs on a particular motor is usually found in the motor's specification sheet. The number of pole pairs can also be measured by applying a low DC current (around 1A) between any two of the three motor phase wires and then counting the number of cogs you feel when rotating the motor by hand for a full turn. It can also be determined by rotating the motor shaft by hand a full turn. Then take the number of counts reported by the hall counter in the Roborun PC utility, and divide it by 6.

#Pole Pairs = Hall Counts per turn / 6

The number must be entered using the Number of Pole Pairs menus in the in the Roborun PC utility.



General
Motor 1
Motor Configuration
Motor Direction: Direct
Number of Pole Pairs: 2

FIGURE 8-2. Number of pole pairs configuration

Or by sending the configuration command:

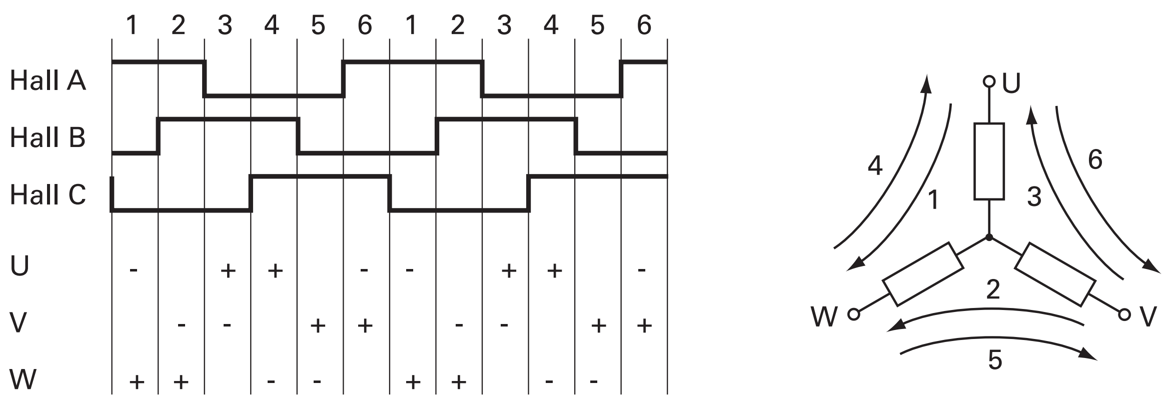
^BPOL channel nn

See “BPOL” in the command reference section for details. This parameter is not needed for basic trapezoidal motor operation with Hall Sensor feedback and can be left at its default value. It is needed if accurate speed reporting is required or to operate in Closed Loop Speed or Position modes . The number of pole pairs is a critical configuration in sinusoidal mode.

Entering a negative number of pole pairs will reverse the measured speed and the count direction. It is useful when operating the motor in closed loop speed mode and if otherwise a negative speed is measured when the motor is moved in the positive direction.

Trapezoidal Switching

In trapezoidal switching, the controller applies current to two of the 3 motor wires, in turn and in alternating direction. A total of 6 combinations of current flow are possible, resulting in the rotor getting a changing magnetic field every 60 degrees of electrical rotation. The controller must therefore know where the rotor is in relation to the electromagnets so that current can be applied to the correct winding at any given point in time. The simplest and most reliable method is to use three Hall sensors inside the motor. The diagram below shows the direction of the current in each of the motor’s windings depending on the state of the 3 hall sensors.



	1	2	3	4	5	6	1	2	3	4	5	6
U	-		+	+		-	-		+	+		-
V		-	-		+	+		-	-		+	+
W	+	+		-	-		+	+		-	-	

FIGURE 8-3. Hall sensors sequence

Hall Sensor Wiring

Hall sensors connection requires 5 wires on the motor:

- Ground
- Sensor A Output
- Sensor B Output
- Sensor C Output
- +5V power supply

Sensor outputs are generally Open Collector, meaning that they require a pull up resistor in order to create the logic level 1. Pull up resistor of 4.7K ohm to +5V are incorporated inside all controllers. Additionally, 1nF capacitors to ground are present at the controller's input in order to remove high frequency spikes which may be induced by the switching at the motor wires.

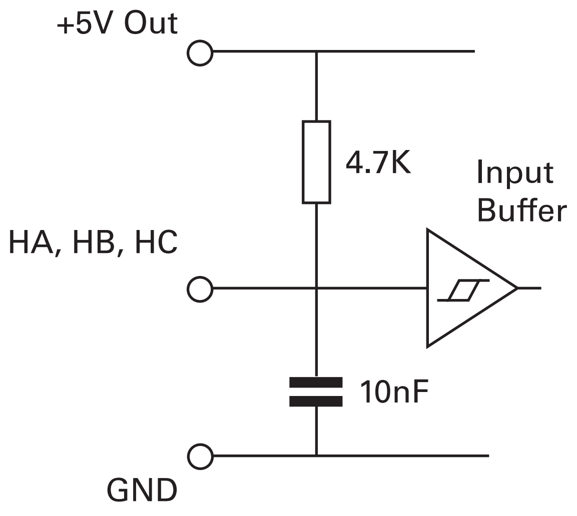


FIGURE 8-4. Hall sensor inputs equivalent circuit

Both 120 degrees and 60 degrees Hall sensors spacing, are supported (see "HPO" in the command reference section). Hall sensors can typically be powered over a wide voltage range. The controller supplies 5V for powering the Hall sensors.

Important Notice

120 degrees Hall sensor spacing is by far the most common. Use 60 degrees only if that is specified in the motor's documentation or label.

Unless specified otherwise in the datasheet, Hall sensor connection to the controller is done using Molex Microfit 3.0 connectors. These high quality connectors provide a reliable connection and include a lock tab for secure operation. The connector pin-out is shown in the controller model's datasheet.

In several controller models, the Hall inputs can be alternatively mapped to digital inputs on the I/O connector. This makes it possible to use the hall sensors for rotor commutation, and SSI sensors for other purposes at the same time (see "MLX" in the command reference section). Note that in the case where digital inputs are configured as Hall inputs, pull-up resistor from the input pin to the +5V must be added externally. Use 4.7K resistors wired as shown in Figure 8.4.

Important Warning

Keep the Hall sensor wires away from the motor wires. High power PWM switching on the motor leads will induce spikes on the Hall sensor wires if located too close. On hub motors where the Hall sensor wires are inside the same cable as the motor power wires, separate the two sets of wires the nearest from the motor as possible.

Important Notice

Make sure that the motor sensors have a digital output with the signal either at 0 or at 1. Sensors that output are gradually changing are typically analog signals will cause the motor to run imperfectly.

Hall Sensor Verification

Hall Sensor miswiring is a very common cause when the motor is not running. You can send the following query to verify that the hall sensors are seen by the controller:

?HS [channel]

The reply is one or two numbers, depending on the number of channels, of values between 0 and 7 with each bit representing the state of each of the HA, HB and HC sensors.

Turn the motor slowly by hand while sending frequent ?HS queries. Verify that all valid combinations appear at one time or the other and that none of the invalid combination ever show.

For 120 degrees spaced sensors, 1-2-3-4-5-6 are valid combinations, while 0 and 7 are invalid combinations. For 60 degrees spaced Hall sensors, 0-1-3-4-6-7 are valid combinations, while 2 and 5 are invalid combinations.

Note that HS query does not work on the first generation HBL and VBL family of products.

Hall Sensor Wiring Order

The order of the Hall sensors and these of the motor connections must match in order for the motor to spin. Unfortunately, there is no standard naming and ordering convention for brushless motors.

The Hall Sensor and Motor Phases naming convention used in Roboteq controllers is A, B and C for the sensors and U, V and W for the motor phases. When rotating the motor shaft clockwise by hand, the controller expects the sensor A to be a mirror of the voltage generated between wires U and W, sensor B between V and U, sensor C between W and

V. See figure 8-5. The sine wave voltage will be inverted when turning the motor in the opposite direction.

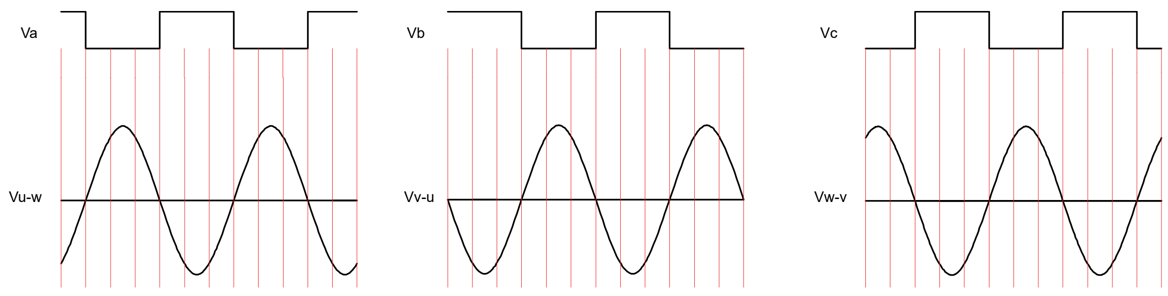


FIGURE 8-5. Relation between hall sensor and U V W windings

Determining the Wiring Order Empirically

While probing with an oscilloscope gives the definite order, a simpler and quicker way is to find the correct combination by trial and error. To do this, you can either connect the motor wires permanently and then try different combination of Hall sensor wiring, or you can connect the Hall sensors permanently and try different combinations of motor wiring. There is a total of 6 possible combinations of wiring three sensors on three controller inputs. There are also 6 possible combinations of wiring three motor wires on three controller outputs. Only one of the 6 combinations will work correctly and smoothly while allowing the controller to drive the motor in both directions.

Alternatively, instead of swapping Hall sensors or motor phases, you can use the "Hall Sensor Map" configuration from the PC Utility menu (see "HSM" in the command reference section)., or from the following console command:

```
^HSM ch nn
```

Try each of the 6 available values of HSM (0-5) and retain the one that will make the motor spin in both directions while drawing the same low current.

When testing a combination, apply a low amount of power (5 to 10%). Applying too high power may trigger the stall protection. Once a combination that make the motor spin is found, increase the power level and verify that rotation is smooth, at very slow speed and at high speed and in both directions.

Important Notice

Beware that while only one combination is valid, there may be other combinations that will cause the motor to spin. When the motor spins with the wrong wiring combination, it will do so very inefficiently. Make sure that the motor spins equally smoothly in both directions. Try all 6 combinations and select the best.

Important Notice

It is not possible to change the motor direction by changing the Hall/Phase order. If the motor is not turning in the desired direction, chose "Inverted" in the "Motor Direction" configuration menu in the PC Utility.

Hall Sensor Alignment

It is very important that the hall sensors be precisely aligned vs the electromagnets inside the motor so that commutation be done exactly at the right time. Bad alignment will cause the motor to run inefficiently.

A first, and generally reliable clue that Hall Sensors are not properly aligned is to run the motor in the forward and then reverse direction while in Open Loop. Verify that for a given command level in open loop, the motor reaches the identical speed and consumes the same amount of current.

Another simple verification method is to use an oscilloscope to view the shape of the phase voltage. While the motor is running, place a probe between ground and any of the motor phases. Verify that the voltage looks like the shape on the figure 8-6. Look for symmetrical ramps on the left and right. An imbalance in the ramps indicates that the commutation happens at the wrong time because of bad Hall Sensor position.

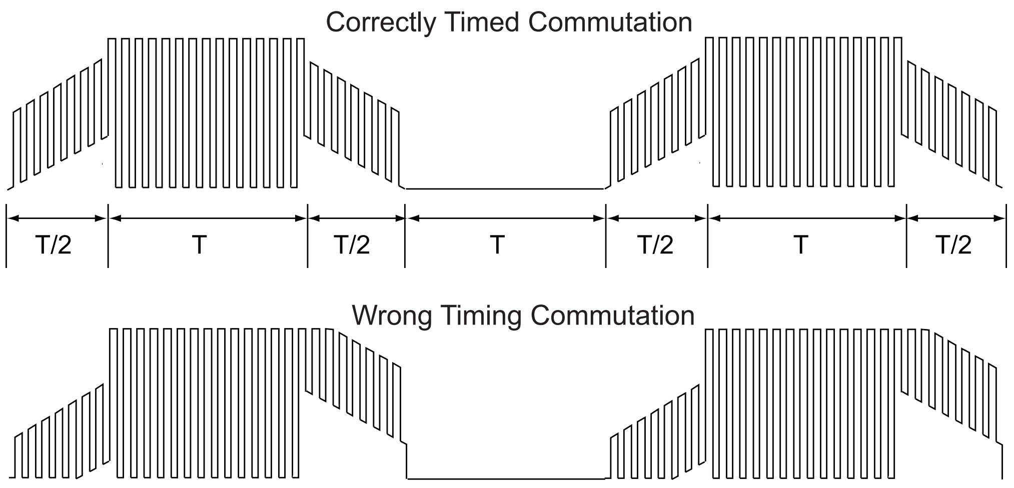


FIGURE 8-6. Ground to Phase voltage waveform on motor with correct and wrong commutations

The most precise evaluation of the Hall Sensors alignment is done using an oscilloscope and the circuit described in figure 8.7. Compare the shape of the Hall Sensor signal to this of the voltage that is generated on the motor phases as the shaft is rotated by an external force. Verify that the zero-crossing of the phase voltages is occurring at exactly the same time as the Hall Sensor transitions, as shown in figure 8-5.

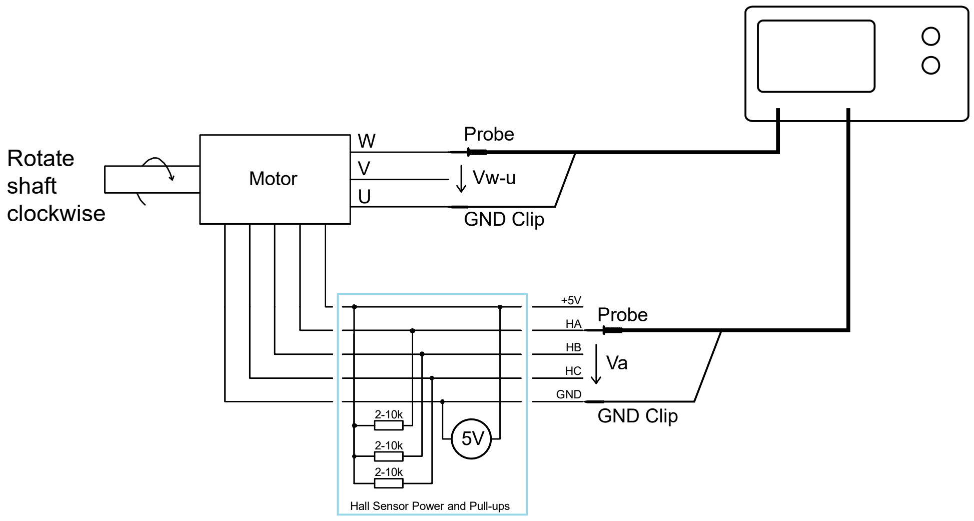


FIGURE 8-7. Use an oscilloscope and the circuit in figure to place the probes and generate these signals

Important Warning

Hall Sensor misalignment cannot be corrected by the controller. Contact the motor manufacturer for remedy.

Sinusoidal Commutation

In sinusoidal commutation, all three wires are permanently energized with a sinusoidal current that is 120 degrees apart on each phase as shown in figure 8-8.

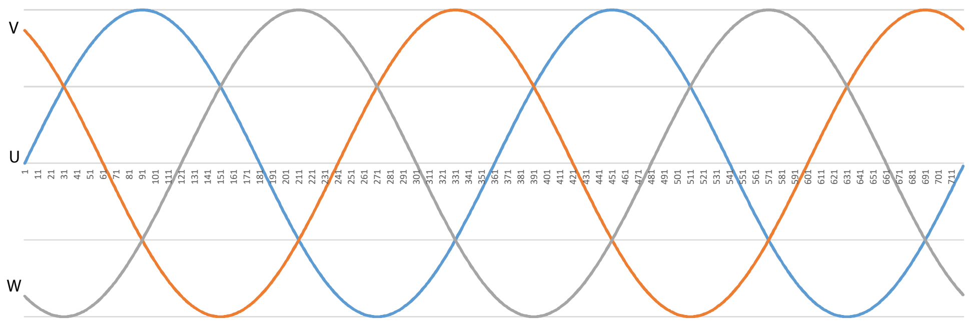


FIGURE 8-8. Three phase current creating a rotating magnetic field

At its most basic operation, the controller measures the rotor’s angular position, adds or subtracts 90o depending on the desired rotation direction, and applies the result to the 3-phase PWM generator.

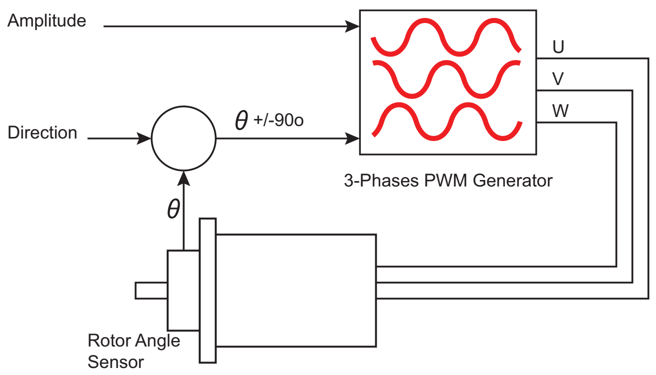


FIGURE 8-9. Simplified 3-phase sinusoidal model

As the motor turns, the phase on each wire is changed in order for the magnetic field to always be perpendicular, and therefore create the maximum radial force to the rotor.

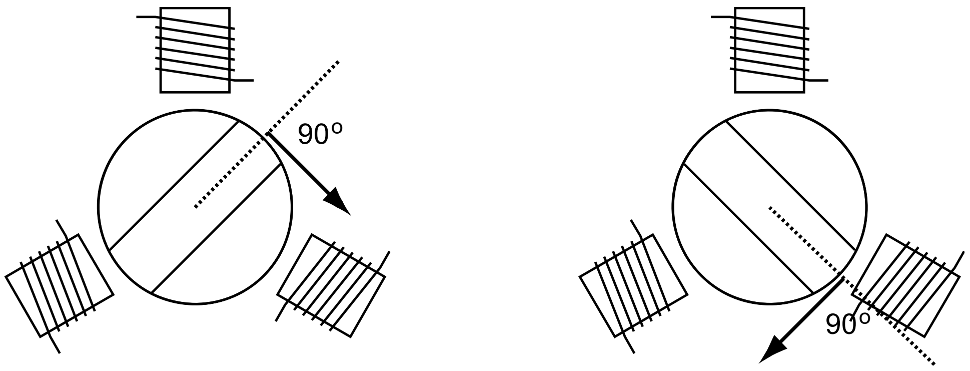


FIGURE 8-10. Magnetic field perpendicular to rotor magnets

The principle benefit of sinusoidal commutation is the quiet, rumble-free, motor operation resulting from the smoothly rotating and always aligned magnetic field.

However, sinusoidal commutation is more complex to configure, calibrate, tune and operate. For best and fastest results it is recommended that you follow these step rigorously:

1- Configure the Controller for Sinusoidal Commutation

2- Select and Configure a Supported Angle Sensors

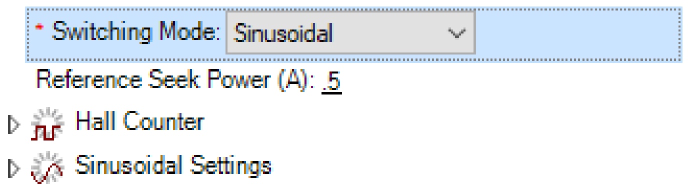
4- Prepare for Automatic Sensor Setup

5- Run the Automatic Sensor Setup

6- Set, Test and Troubleshoot FOC-Field Oriented Control

Configuring the Controller for Sinusoidal Commutation

Sinusoidal mode is selected via the Switching Mode configuration menu in the Roborun PC utility.



* Switching Mode: Sinusoidal

Reference Seek Power (A): .5

FIGURE 8-11. Sinusoidal configuration

Or by sending the configuration command:

^BMOD channel 1

Configuring the Number of Motor Pole Pairs

The number of Motor Pole Pairs is a critical parameter for sinusoidal combination. A full 3-phase cycling of motor's electromagnets will cause the rotor to move to the next permanent magnet pole. A full 3-phase cycle is known as electrical turn which will be different from the physical (mechanical) turn of the shaft if the motor number of pole pairs is greater than one. The figure below show the relationship between mechanical degree and electrical degree in the case of a two pole pairs motor.

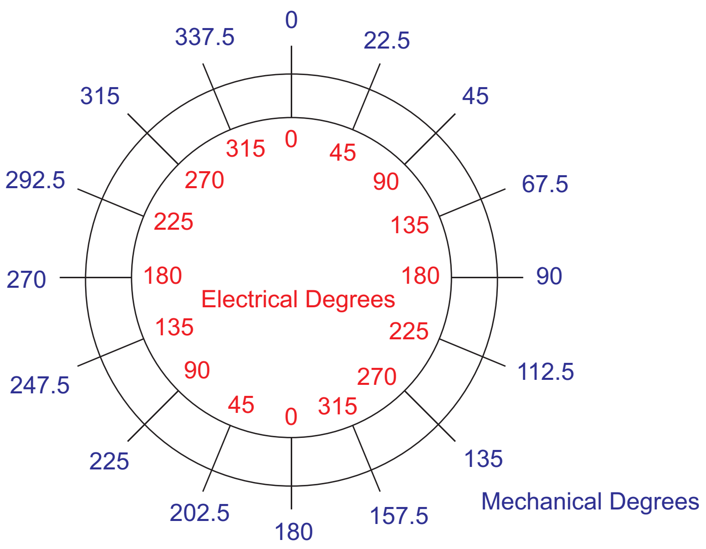
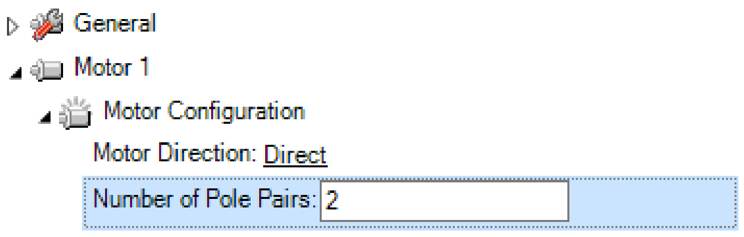


FIGURE 8-12. Mechanical vs electrical degrees

See "Determining the Number of Poles" on page 100 for details on how to determine the number of pole pairs and configuring the controllers.

The number must be entered using the Number of Pole Pairs menus in the in the Roborun PC utility.



Motor Direction: Direct
Number of Pole Pairs: 2

FIGURE 8-13. Number of pole pairs configuration

Or by sending the configuration command:

```
^BPOL channel nn
```

Entering a negative number of pole pairs will reverse the measured speed and the count direction. It is useful when operating the motor in closed loop speed mode and if otherwise a negative speed is measured when the motor is moved in the positive direction.

Configuring Number of Sensor Poles

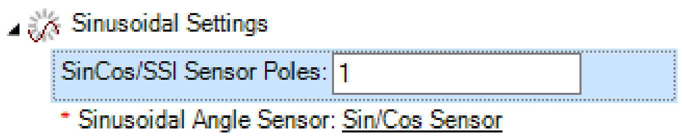
Single pole absolute sensors like Resolvers, sin/cos and SSI encoders typically return an angular value that is equal to the mechanical angle. The controller then converts the measured angle into the electrical angle for its internal operation, using the formula:

Electrical Angle = (Sensor Angle * Number of Poles) modulo 360

When used on motors with a high number of pole pairs, single pole sensors will measure the electrical angle with degrading resolution as the number of motor poles is higher. For example, a single pole sensor with 1o resolution on a 10 pole motor will produce an electric angle evaluations with 10o resolution, which will result in less than optimal operation.

To resolve this problem, some absolute sensors are available in multiple pole versions. In that case, the sensor will outputs a 0-360 angle value multiple times during a full turn. Note that the controller will not work if the number of sensor poles is higher than the number of motor poles.

Enter the number of Sensor Poles in the SinCos/SSI Sensor Poles configuration menu in the Roborun PC utility.



SinCos/SSI Sensor Poles: 1
Sinusoidal Angle Sensor: Sin/Cos Sensor

FIGURE 8-14. Sin/Cos/SSI Sensor Poles configuration

Or by sending the configuration command:

```
^SPOL ch poles
```

You can determine or verify the number of sensor pole pairs by following these steps:

1 Set Motor Poles to 1

2 Launch the Calibration/Setup

3 Make one full rotation of the motor shaft by hand and monitor the Angle value reported in Roborun Utility.

4 Check how many times the Angle range (0-511) rolls over. This gives the number of sensor poles.

5 Restore the correct values of motor and sensor poles

6 Launch the Calibration/Setup again

Important Notice

The number of motor poles and sensor poles are very important configuration parameters in sinusoidal mode. Using the wrong values will produce erratic behavior and possibly damage.

Selecting and Configuring Supported Angle Sensors

In order for the proper voltage and phase to be applied to each of the 3 motor wires, the rotor angular position must be known with precision at all times. Roboteq controllers support several sensors for achieving this.

Then select the method for capturing the rotor angle of the rotor as the motor spins. This is done using the Sinusoidal Angle Sensor configuration menu in the utility.

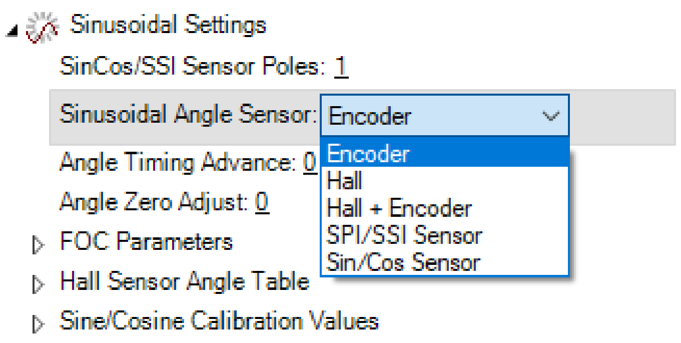

Sinusoidal Settings
SinCos/SSI Sensor Poles: 1
Sinusoidal Angle Sensor: Encoder
Angle Timing Advance: 0
Angle Zero Adjust: 0

FIGURE 8-15. Encoder Sinusoidal configuration

Or by sending the configuration command:

```
^BFBK ch mode
```

Where mode:

0: Encoder

1: Hall

2: Hall+Encoder

3: SSI sensor

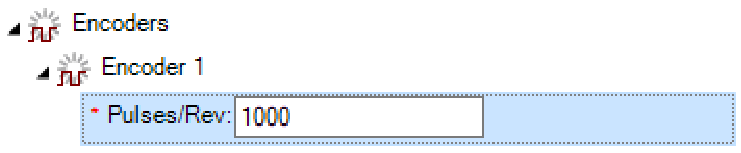
4: Sin/Cos Sensor

5: Resolver

Each mode requires a various amount of additional setup and/or calibration as described in the following sections.

Incremental Encoder-Only

A quadrature encoder can be used to determine the rotor position. Enter the Encoder PPR using the PC Utility.



Encoders
Encoder 1
* Pulses/Rev: 1000

FIGURE 8-16. Encoder PPR configuration

Or by sending the configuration command:

^EPPR channel nn

Optimally, the encoder should have a PPR that is at least 128 x the number of pole pairs. For example a motor with 4 pole pairs should have a 128 x 4 = 512 Pulse per revolution. This will result in 2048 counts for a full turn of the rotor, and therefore the electrical angle to be measured with 360 / 2048 * 4 = 0.7 degrees, resulting in a very smooth changing sinusoidal drive to the motor. A significantly lower resolution encoder will results in a stepping sinusoid. A higher resolution encoder will not improve the waveform.

Since encoders do not give an absolute position, a reference search sequence is performed automatically by the controller at every power up or when the controller switches from a different mode to sinusoidal mode with encoder feedback

The search can also be forced manually by pressing the Sensor Setup button on the Diagnostic tab of the PC utility, or by sending the following maintenance or runtime command from the console or the serial/USB port. The runtime command can also be executed via CANOpen.

%CLMOD 2 or !MSS 1 (for channel 1) and
%CLMOD 3 or !MSS 2 (for channel 2).

Before trusting that reference search will be successful at every power up, try repeatedly to send the sensor tuning command under real-life load conditions. After reference search is completed, verify that the motor turns with the same efficiency in the forward and reverse direction.

Proper operation of the encoder can be verified by viewing the counter with the query:

```
?C [channel]
```

And verifying that it increments by the Encoder's CPR (counts per revolution, or PPR *4) when making a full turn, and returns to its original value after a full turn in the reverse direction.

Hall-Only

In this mode, the Hall sensors are used to determine the angular position of the rotor. Since transitions of the Hall pattern occur at every 60 degrees only, the controller will estimate the current angle by interpolating in between two transition based on the current motor speed. This technique works well as long as speed is stable and changes are relatively slow. It also requires that the magnets and sensors are positioned with precision inside the motor, which is not always the case in low cost motors. Compared to Trapezoidal mode, this mode will result is quieter motor operation because of the sinusoidal commutation.

Proper operation of the Hall sensors can be verified by checking that the hall counter changes by 6 * the number of poles over a full mechanical turn in the PC Utility or by using the query:

```
?CB [channel]
```

Alternatively, the following query can be run to view the state of the logic level of each Hall inputs.

```
?HS [channel]
```

The Hall-only mode requires only a one time tuning during which their actual angular position of each Hall sensor will be detected and stored. See Automatic Sensor Setup procedure below.

Hall + Encoder

If the motor is fitted with Hall sensors and an Incremental Encoder, the controller can be configured to use both sensors together. The Encoder's PPR must be configured as described above. In this mode, the controller's operation is identical to when an Encoder alone is used for feedback, except that there is no need for the reference search sequence described above. When first energized, the motor will operate using the Hall sensor until the first change to the Hall pattern is detected. This will set the angle reference for the encoder. For this mode, it is critical that both the number of Encoder PPRs and the motor number of pole pairs be entered correctly. Both counters must count in the same direction.

The Hall Encoder mode requires only a one time tuning during which their actual angular position of each Hall sensor will be detected and stored. See Automatic Sensor Setup procedure below.

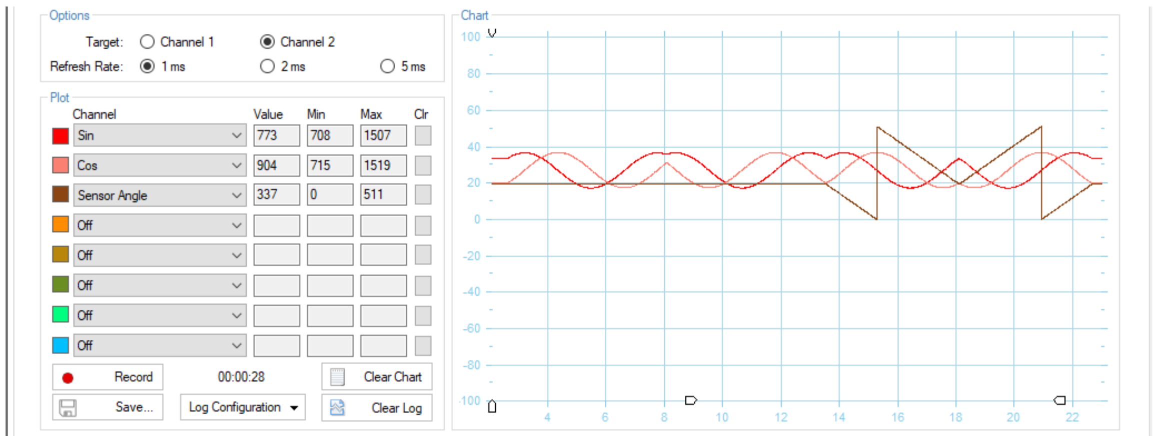
See above and below how to verify that the Encoder and Hall sensors are operating correctly.

Sin/Cos Analog

Some controller models can be interfaced to absolute position sensors with Sine/Cosine output. These sensors are usually made using Hall technology and are built into the motor. They provide two analog voltage output that are usually 90 degrees apart. The rotor angle is determined by measuring the voltage ratio between the two signals. The controller can compensate for differences in amplitudes between the two signals.

There are sensors whose signals are not 90 degrees apart. The controller can be configured for use with sensors that have practically any phase shift. This is done using the PSA - Phase Shift Angle configuration parameter. Note that angles are in 0-511 degrees notation.

The proper operation of the Sin/Cos sensor can be verified by plotting in real time the voltages of the sin and cos signals inside the Diagnostic tab of the PC utility


Options

Target: Channel 1 [✓ Channel 2]

Refresh Rate: [✓ 1 ms] 2 ms 5 ms

Plot

	Channel	Value	Min	Max	Clr
🟥	Sin	773	708	1507	
🟥	Cos	904	715	1519	
🟫	Sensor Angle	337	0	511	
🟧	Off				
🟨	Off				
🟩	Off				
🟩	Off				
🟦	Off				

Record 00:00:28 Clear Chart

Save... Log Configuration Clear Log

FIGURE 8-17. Diagnostic Tab with Auto Setup

Or using the following queries.

?ASI [channel]

Sin/cos sensors require a one-time setup and calibration. See Automatic Sensor Setup procedure below.

Important Warning

The Tuning Fault LED will be on in the Roborun screen and the motor will NOT start if the Sin/Cos has not been setup/calibrated.

Important Notice

Electrical noise on the sensor output will cause wrong angle readings. Shield the wires and keep them as far as possible from the motor wires. If noise persists, add a 10nF ceramic capacitor between the input pin and ground pin on the controller's connector

Synchro Resolver

Synchro Resolvers are a form of Sine/Cosine sensor based on transformer technology. It is composed of a fixed primary coil, and two secondary coils positioned at 90o from each other and that rotate with the rotor. A fixed frequency excitation voltage is fed in the primary. As the secondary coils turn, and take turn being parallel with the fixed primary, the voltage amplitude induced in each varies as shown in the figure below.

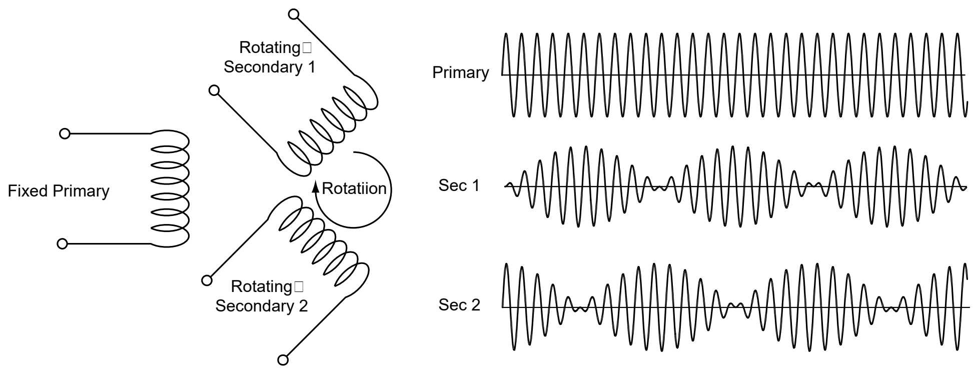


FIGURE 8-18. Resolver equivalent diagram and signals

Roboteq controllers models supporting resolvers use one output to generate the excitation. The secondaries are then fed to two analog inputs. Exact wiring depends on the controller model. Please consult the controller data sheet for pinout location. Resolvers require one time calibrations similar to these for the sin/cos sensors and can be tested the same way.

Important Warning

The Tuning Fault LED will be On in the Roborun Screen and the motor will NOT start if the Resolver has not been setup/calibrated.

Digital Absolute Encoder (SSI) Feedback

Some advanced motors, incorporate an absolute position sensor with a high speed serial interface based on the SSI protocol.

Roboteq controllers support SSI encoders with various resolutions. The SSI encoders give the rotor absolute position. Nevertheless, a calibration must be done once in order to capture offset between the motor's and the sensor's 0 degrees position. In case of multi-turn encoders the angle counter is taken into consideration during the calibration.

The SSI sensor's angle counter is configured based on the following configuration fields:

- SSI Clock Speed (SCLK), it is common for all SSI sensor inputs

- SSI/SPI Number of bits (SLEN), the total number of bits of the SSI sensor frame.
- Counter start bit position (SSTA), the position of the first bit of the angle counter.
- Counter number of bits (SCLE), the number of bits of the angle counter.

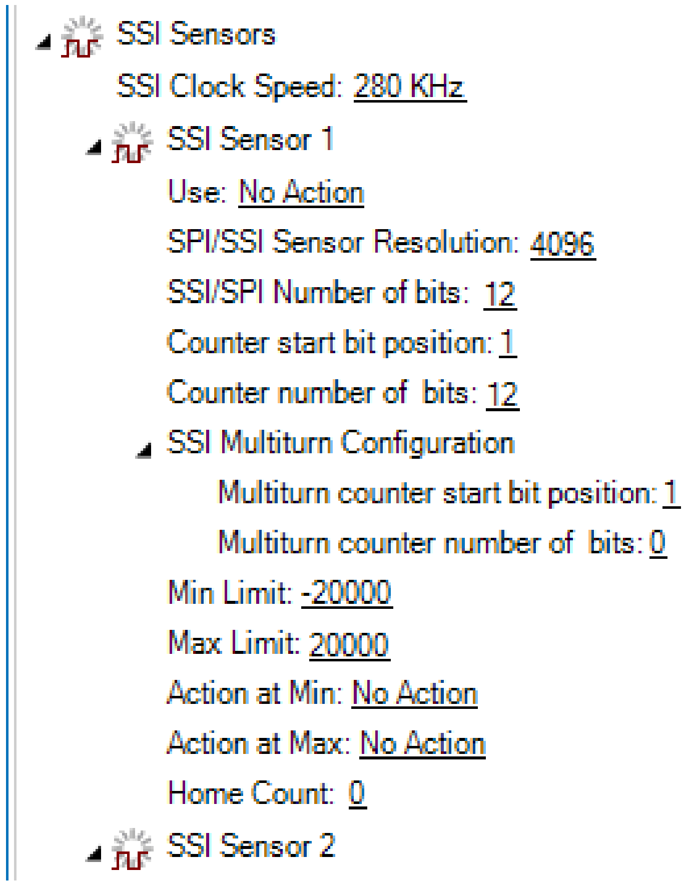

SSI Sensors
SSI Clock Speed: 280 KHz
SSI Sensor 1
Use: No Action
SPI/SSI Sensor Resolution: 4096
SSI/SPI Number of bits: 12
Counter start bit position: 1
Counter number of bits: 12
SSI Multiturn Configuration
Multiturn counter start bit position: 1
Multiturn counter number of bits: 0
Min Limit: -20000
Max Limit: 20000
Action at Min: No Action
Action at Max: No Action
Home Count: 0
SSI Sensor 2

FIGURE 8-19. SSI Sensor Configuration

Or by sending the configuration commands:

```
^SCLK nn
^SLEN channel nn
^SSTA channel nn
^SCLE channel nn
```

After enabling the Sinusoidal Mode and SSI Angle Feedback, verify first that the SSI Counter displays a stable number that is different from zero. This will indicate that data is output from the sensor and captured by the controller. Rotate the motor by hand to verify that the counter changes.

The raw value of the SSI sensor can be read using the query:

```
?SFR
```

The continuous 32-bit counter and speed that is driven by the SSI sensor can be read using the following queries respectively:

?CSS [channel]

?SS [channel]

Typically, SPI encoders are single pole sensors, meaning that they output 0 to their maximum value over a full mechanical turn.

Preparation for Automatic Sensor Setup

The rotor's angle sensor is a critical element for good sinusoidal commutation. Wrong or unstable angle reading can cause excessive current consumption, vibration, or even damage. The sensor must be correctly and firmly attached to the motor so that it never slips during operation. The table below shows the setup and calibration steps required for each sensor type.

TABLE 8-1. Setup and calibration steps

Setup	Encoder	Hall	Hall+ Encoder	SSI	Sin/ Cos	Resolver
Min/Max Range Calibration	No	No	No	No	Yes	Yes
Zero reference search	Yes(1)	No	No	Yes	Yes	Yes
Hall Position Mapping	No	Yes(2)	Yes(2)	No	No	No
Linearity Correction Map	No(3)	No	No(3)	Yes(3)	Yes	Yes
Sensor/Winding Order	Yes	Yes	Yes	Yes	Yes	Yes

Note 1: Zero reference search must be performed at every power-up for Encoder mode
Note 2: Hall position mapping is optional but recommended for best results

Note 3: Linearity Correction is optional for digital encoders

Sensor setup and calibration is generally a one-time procedure. Sensor information is stored in the controller's configuration and calibration flash.

Reference Search Power

During the automatic sensor setup phase, the controller will drive the motor coils with a slow-changing three phase current, creating a rotating magnetic field inside the motor. The rotor's magnets are attracted to the field, causing the rotor to follow turn. For best accuracy the rotor is driven for a full turn in the forward direction, and another full turn in the reverse direction.

For the reference search to work, the current that is injected into the motor must be strong enough to pull the rotor in perfect alignment. The motor must not be loaded during this sequence. For best results, the current should be set equal to the motor's nominal amps rating.

Enter the value in Amps s in the "Reference Seek Power" configuration menu in the Roborun PC utility.

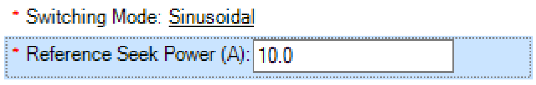

* Switching Mode: Sinusoidal
* Reference Seek Power (A): 10.0

FIGURE 8-20. Reference Seek Power configuration

Or by sending the configuration command:

^SREF channel amps (in amps * 10)

Important Warning

Do not select an amperage value that is above the maximum nominal value published in the motor's datasheet.

Important Warning

Calibration data is specific to a motor+sensor set. Changing the motor and/or sensor requires recalibration.

Sensor Min/Max Range

Analog sensors like Sin/Cos and Resolvers have voltage output swings that can vary from one sensor to another. During Automatic Setup, the motor is forced into rotation and the min and max voltages for each sensor output signal is captured over a full turn of the sensor. These values then determine offsets and multiplier that are saved in calibration memory and subsequently used to scale the signals as necessary to produce a correct angle measurement.

The min/max range calibration is a one time operation for a given sensor.

Zero Reference Search for Absolute Sensors

All absolute sensors (Sin/Cos, Resolver, SSI) cannot be trusted to be mounted so that their 0 degree reference position exactly matches this of the motor's windings. During Zero Reference search, the motor is driven over known angular positions. The sensor measurement is compared with the expected rotor position and an Angle Adjustment vale is computed and stored in configuration memory.

Important Notice

All angles values entered or displayed by the controller are in 0-511 value where 0 = 0 degrees or radians, and 511 is 360º or 2ρ radians.

Zero Reference Search for Incremental Encoders

Incremental Encoders do not output an absolute position value. The Zero Reference search is therefore performed every time the controller is powered on. See also "Incremental Encoder-Only" on page 111.

Hall Sensors Position Mapping

Hall sensors are often not precisely located at the 0, 60, 120, 180, ... positions. While some imprecision has little effect in trapezoidal mode, misalignment of even a few degrees will have disruptive effect when operating in sinusoidal mode. When used with Hall sensors, the automatic setup procedure captures and maps the angle at which each of the six hall transitions occurs within an electrical turn, regardless of their wiring order.

Linearity Correction Map

Sensor are not alway totally linear. This causes the sensed angle to be measured with periodic errors along the sensor travel. During the automatic setup process, the controller maps the values read from the sensor for every mechanical degree over a full turn. It then build a correction table which is applied in real-time as the motor spins.

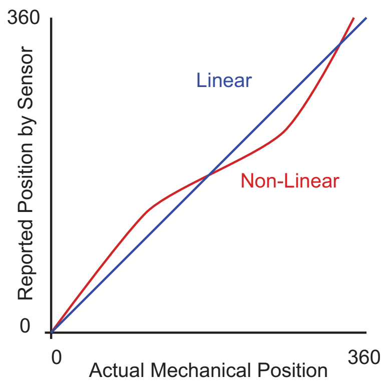


FIGURE 8-21. Sensor non-linearity correction

Windings and Sensor Order.

The wiring order of the U, V, W motor cable, the A, B, C Hall Sensors inputs, the A, B Encoder inputs and/or the Sin, Cos signals are all related to each other. Swapping any of the motor wires will make the motor turn in the opposite direction. Swapping any sensor cables affects the angle and counting direction.

The Automatic Setup will detect the wiring order of the motor and sensors, and set the corresponding bits in the SWD configuration register in order to virtually swap the connections so that all are moving/sensing in the same direction.

Running the Automatic Sensor Setup

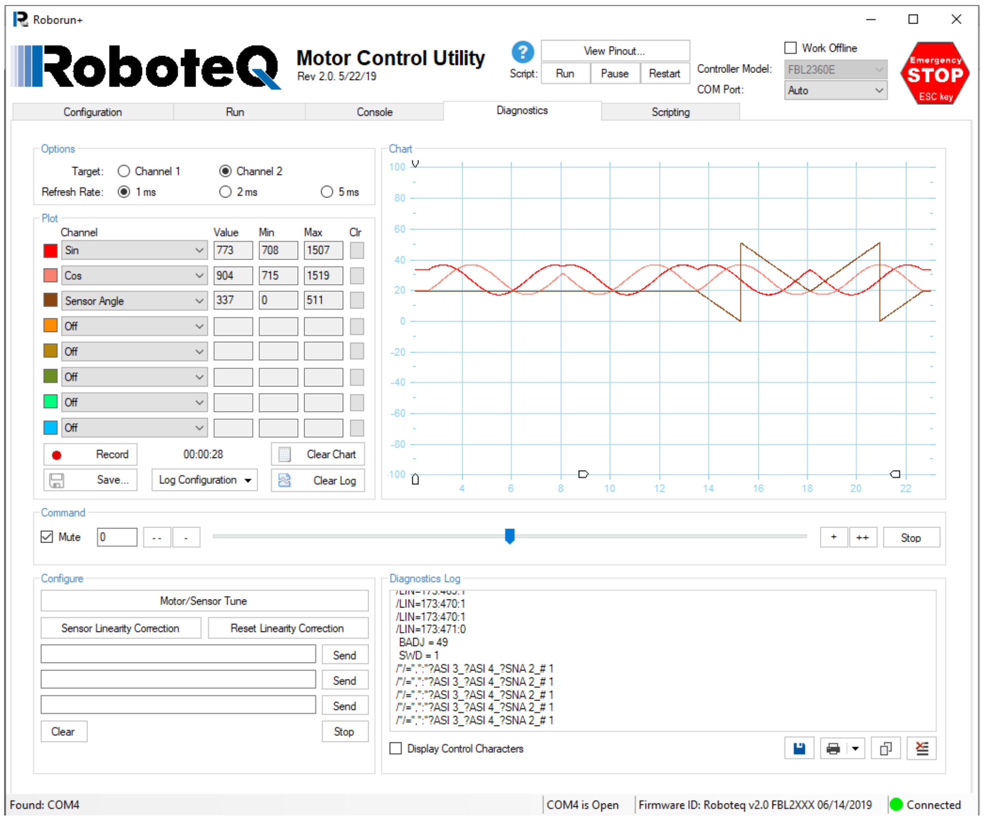
The V2.x firmware has a new feature for Automatic Setup and Calibration of all the supported rotor sensor types. With a simple click, it performs the following:

a. Find the sensor polarity with respect to stator winding connection and set the SWD Swap Windings configuration parameter.
b. Find angle offset from the stator zero degrees reference axis and store the value in the “BADJ” Angle Zero Adjust configuration register.
c. Map the angular position of the Hall sensors and store the values in the HSAT Hall Sensor Angle configuration table
d. Find the min, max and zero levels of the analog signals in case of sin-cos and resolver for normalization of sine and cosine. The values are stored in the ZSMC calibration register

After configuring the controller for sinusoidal mode and after selecting and configuring the Angle Sensor, click on the Diagnostic tab of the Roborun+ PC Utility.

Important Notice

All angles values entered or displayed by the controller are in 0-511 value where 0 = 0 degrees or radians, and 511 is 360o or 2ρ radians.



Target: Channel 1 [✓ Channel 2]
Refresh Rate: [✓ 1 ms] 2 ms 5 ms

	Channel	Value	Min	Max	Clr
🟥	Sin	773	708	1507	
🟥	Cos	904	715	1519	
🟫	Sensor Angle	337	0	511	
🟧	Off				
🟫	Off				
🟩	Off				
🟩	Off				
🟦	Off				

00:00:28

```
/LIN=173:470:1
/LIN=173:470:1
/LIN=173:471:0
 BADJ = 49
 SWD = 1
/"/=",":"?ASI 3_?ASI 4_?SNA 2_# 1
/"/=",":"?ASI 3_?ASI 4_?SNA 2_# 1
/"/=",":"?ASI 3_?ASI 4_?SNA 2_# 1
/"/=",":"?ASI 3_?ASI 4_?SNA 2_# 1
/"/=",":"?ASI 3_?ASI 4_?SNA 2_# 1
```

Found: COM4 | COM4 is Open | Firmware ID: Roboteq v2.0 FBL2XXX 06/14/2019 | Connected

FIGURE 8-22. Diagnostic Tab with Auto Setup

Select which motor channel to setup. Then click on the Motor/Sensor Setup button to launch the Automatic process. The process can also be initiated using the serial commands %CLMOD 2 or !MSS 1 for Channel 1 and %CLMOD 3 or !MSS 2 for Channel 2.

The motor will rotate at a RPM speed of 60/Number of Pole pairs. It will perform a full turn in each direction and then stop. If the motor doesn't turn, or turns with hesitations, Click on the Cancel button (or send %CLMOD 0 or !MSS 0). Make sure that the motor is not loaded. Adjust the Reference Search Current and try again. See "Reference Search Power" on page 108.

After setup is finished, depending on the sensor type, the Utility will print the following captured parameters on the console tab indicating that the process is complete:

TABLE 8-2. Motor/Sensor Setup parameters

Sensor	Captured Values	Description
All types	BADJ SWD	Zero Reference Winding/Sensor Order
Hall, Hall+Encoder	HSAT	Hall Sensor Map
Sin/Cos, Resolver	ZSMC	Min/Max/Zero

Perform the Setup a few times to verify that it produces the same values. If the Zero Reference in particular varies significantly, increase the Adjust the Reference Search Current and try again. See "Reference Search Power" on page 116.

The reference search will settle on a given electrical angle location. On a two pole motor, this electrical angle value exists in 2 mechanical locations for that motor. After performing a first search, rotate the motor shaft and repeat the search. On a perfectly constructed motor, the search will settle at the same electrical angle on any of the other poles. In practice, it is expected that the values will be off by one or two degrees from one pole to another.

If the value is consistent from one measurement to another, and difference is larger than a few degrees, this means the poles are not placed with precision in the motor and the motor will not run efficiently. If the difference is very large (20 degrees or more), it is likely that the angle sensor is not working correctly or that the number of poles of the motor and/or sensor are not configured correctly.

Real-Time Charting

The Diagnostic tab includes a Fast-Updating that can be used to monitor useful information during the Setup phase. For instance for observing that the Sin and Cos inputs see a Sinusoidal shaped signal as the motor turns, or that the captured angle is changing steadily without noise or glitches. The chart is also useful for testing the motor immediately after Setup. The table below shows the parameters that can be monitored in the chart.

TABLE 8-3. Diagnostics monitor values

Parameter	Description
Sin	Voltage at Sin input of Sin/Cos or resolver
Cos	Voltage at Sin input of Sin/Cos or resolver
Electrical Angle	Electrical Angle
Calibration Angle	Angle of the forced field applied during auto-setup
Sensor Angle	Angle measured by the sensor
Motor Power	PWM Level applied to the motor
Motor Amps	Motor Amps
FOC Flux Amps	Flux Amp, Direct Current, Id (cause no torque)
FOC Torque Amps	Torque Amps, Quadrature Current, Iq (cause torque)
Hall Status	State of all 3 Hall into a single 0-7 value
Hall A	State of Hall A input
Hall B	State of Hall B input

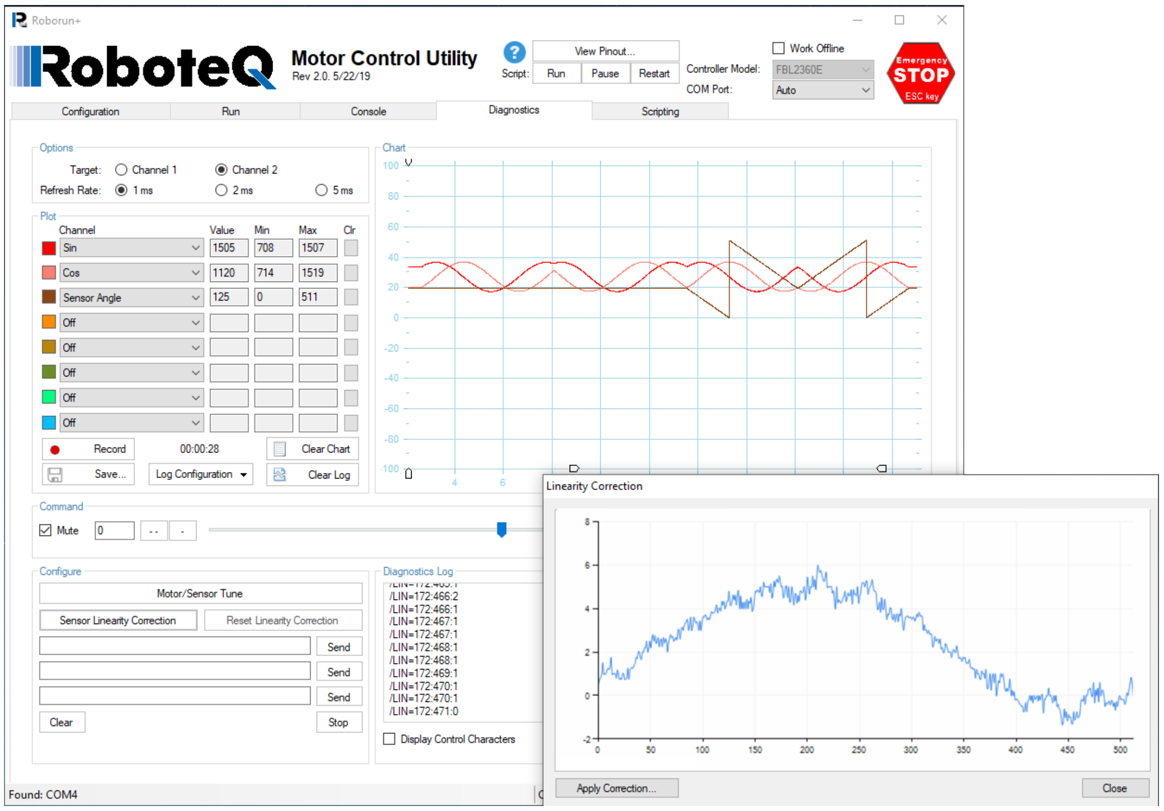
Parameter	Description
Hall C	State of Hall C input
FOC Angle Correction	Correction determined and applied by FOC
Battery Volts	Battery Volts
Power Per Phase U	PWM on Phase U
Power Per Phase V	PWM on Phase V
Power Per Phase W	PWM on Phase W
Sense U	Sensed voltage level at output U
Sense V	Sensed voltage level at output V
Sense W	Sensed voltage level at output W
BEMF U	BEMF voltage on floating phase U (sensorless-only)
BEMF V	BEMF voltage on floating phase V (sensorless-only)
BEMF W	BEMF voltage on floating phase W (sensorless-only)
BEMF Integrator	BEMF Integrator (sensorless-only)

Important Notice

All angles values entered or displayed by the controller are in 0-511 value where 0 = 0 degrees or radians, and 511 is 360o or 2ρ radians.

Sensor Linearity Correction

For Sin/Cos, Resolver and SSI encoders, it is possible to automatically check and eventually apply automatic correction to the sensor linearity. From the diagnostics tab, select channel 1 or 2 and press the “Sensor Linearity Correction” button. The motor will spin for a few electrical revolutions and then stop.



```
/LIN=172:466:2
/LIN=172:466:1
/LIN=172:467:1
/LIN=172:467:1
/LIN=172:468:1
/LIN=172:468:1
/LIN=172:469:1
/LIN=172:470:1
/LIN=172:470:1
/LIN=172:471:0
```

FIGURE 8-23. Sensor Linearity Capture and Correction

The linearity of the sensor will be displayed on the screen. If the graph is flat at around zero value, the sensor is linear and no correction is needed. If the shape is not a flat line and is clean (i.e. without random noise and glitches), press "Apply Correction" button. It can be beneficial to capture the linearity curve a few times and verify that it is consistent, before applying it. If the graphs is noisy, has glitches or looks otherwise unusual, then press close an do the Sensor Linearity procedure again. If the performance is not better then the correction can be cleared by pressing the "Reset Linearity Correction" button.

Basic Motor Check

After the Automatic Setup completed successfully, the motor is ready to run. Use the sliders on the Diagnostics Tab or on the Run Tab to apply power. Always run first in open loop, and preferably with no or light load. Apply first a small command (100-200) and verify that the motor spins without noise or rumble while using low current. Run the motor in the opposite direction and verify that for the same command value, it reaches the same speed and draws the same current. If the motor runs smoothly and with symmetrical performance, repeat the test with higher command value.

If the performance is different in the forward vs reverse direction, the zero reference may be wrong. Run the setup again. Noise or rumble typically points to problems with the sensor.

Field Oriented Control (FOC)

In sinusoidal modes, using the rotor angle to determine the voltage to apply to each of the 3 motor phase works well at low frequencies, and therefore at low rotation speed. At higher speed, the effect of the winding inductance, back EMF and other effect from the motor rotation, create a shifting current. The resulting magnetic field is then no longer optimally perpendicular to the rotor's permanent magnets.

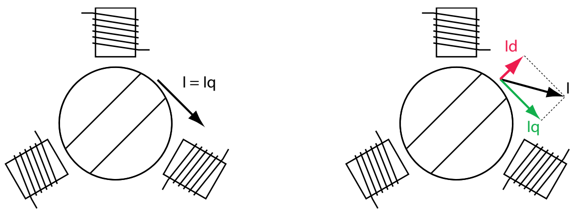


FIGURE 8-24. Perpendicular and non-perpendicular fields

As can be seen in figure 8-24, when the magnetic field is at an angle other than exactly perpendicular to the rotor's magnets, the rotor is pulled by a force that can be decomposed in two forces:

Lateral force causing torque, and therefore rotation. This force results from the Quadrature current Iq, which is also called Torque current.

Parallel force that pulls the rotor outwards, creating no motion. This force results from the Direct Current Id, which is also called Flux current.

Field Oriented Control is a technique that measures the useful Torque current and wasted Flux current component of the motor current. It then automatically adjust the power (output voltage amplitude) and phase angle applied to each motor phase in order to eliminate the wasted Flux current.

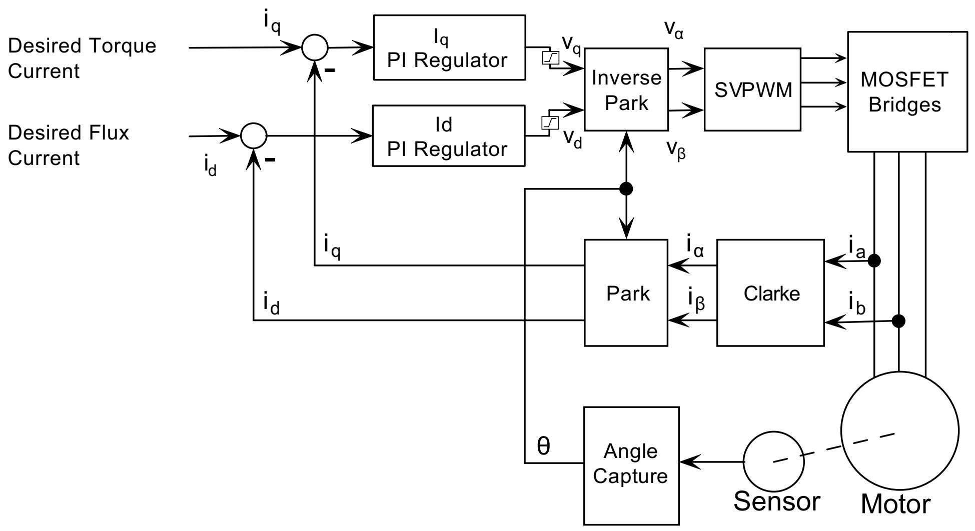
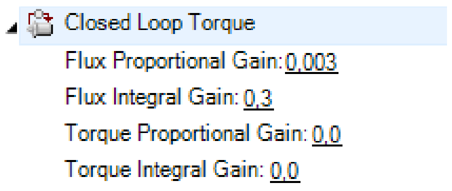


FIGURE 8-25. FOC operation

Field Oriented Control is available on all models of Roboteq motor controllers. It uses a classical implementation as described in the figure 8-25. The current in the motor phase is captured, along with the rotor's angle. From this are computed the useful Iq and wasteful Id. Two Proportional-Integral (PI) regulators then work in parallel to control the output voltage amplitude and phase so that the desired Torque (Iq) and Flux (Id) currents are met. The desired Flux current is typically set to 0 for SPM brushless dc motors, and so the regulator will work to totally eliminate the Flux current. The output of the PI d-q axis current controllers yield the d-q axis reference voltage commands (Vd, Vq) applied to the motor through the Space Vector Pulse Width Modulation (SVPWM) technique.

Both PI regulator have user-settable gains. They can be changed from the menus in the RoborunPlus utility.



Closed Loop Torque
Flux Proportional Gain: 0,003
Flux Integral Gain: 0,3
Torque Proportional Gain: 0,0
Torque Integral Gain: 0,0

FIGURE 8-26. Current PI Gains

Or by sending the configuration command for single Channel Controllers:

```
^KPF 1 nn   = Proportional Gain for Channel 1 Flux
^KPF 2 nn   = Proportional Gain for Channel 1 Torque
```

```
^KIF 1 nn  = Integral Gain for Channel 1 Flux
^KIF 2 nn  = Integral Gain for Channel 1 Torque
```

Or by sending the configuration command for dual Channel Controllers:

```
^KPF 1 nn   = Proportional Gain for Channel 1 Flux
^KPF 2 nn   = Proportional Gain for Channel 2 Flux
^KPF 3 nn   = Proportional Gain for Channel 1 Torque
^KPF 4 nn   = Proportional Gain for Channel 2 Torque
```

```
^KIF 1 nn  = Integral Gain for Channel 1 Flux
^KIF 2 nn  = Integral Gain for Channel 2 Flux
^KIF 3 nn  = Integral Gain for Channel 1 Torque
^KIF 4 nn  = Integral Gain for Channel 2 Torque
```

Where nn = Gain * 10000, e.g. 125000 = 12.5 only (on version 3.x of firmware)

Note that these gains can be modified also during runtime by using the respective runtime command CPG and CIG.

FOC Gains Determination & Tuning

The FOC Proportional and Integral gains for brushless dc motors could be automatically calculated from "Motor characterization" tool at Roborun+ (see Roborun+ Utility User Manual for more details).

Good PI gains are important for the controller to quickly reach and stabilize the desired Id and Iq current. A very good approximation of the gain values can be calculated from the motor's Resistance and Inductance using the formulas:

Flux Proportional gain = Motor Phase Inductance(Henry) * Bandwidth

Flux Integral gain = Motor Phase Resistance(Ohm) * Bandwidth

Bandwidth is in rad/sec and according to Nyquist criteria the current loop bandwidth cannot be more than the half of the current loop sampling time. Most commonly the current loop bandwidth is set to the 1/10-1/20 of the current loop sampling time. The current loop sampling time is at 16 kHz. So if we choose as current loop bandwidth the 300Hz then:

1Hz = 2π rad/sec

So for 300Hz Bandwidth = 300*2π rad/sec = 1885

Usually even smaller bandwidth can be as effective as the 300Hz. It is better to start with the smaller possible gains and then tune according to the behavior of the motor. Test in open loop with caution.

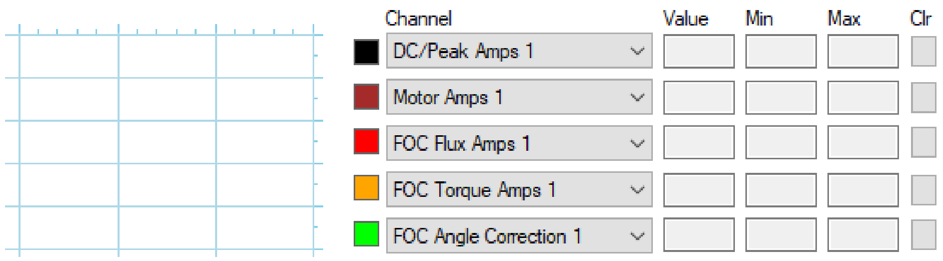
Example calculation for 300Hz bandwidth, 11mOhm Phase resistance and 90uH Phase Inductance

Ki = 0.011 * 1885 = 20.735

Kp = 0,00009 * 1885 = 0.16965

FOC Testing and Troubleshooting

Verify that FOC is operating correctly by monitoring the following values with the PC Utility:



	Channel	Value	Min	Max	Clr
⬛	DC/Peak Amps 1				
🟫	Motor Amps 1				
🟥	FOC Flux Amps 1				
🟧	FOC Torque Amps 1				
🟩	FOC Angle Correction 1				

FIGURE 8-27. FOC performance monitoring

Optimal performance is achieved when:

- FOC Flux Amps are close to 0.
- Motor Amps and FOC Torque Amps values are close.
- FOC Angle Correction is stable for stable motor Power (output voltage amplitude).

Check also when changing motor power how fast FOC Flux Amps is corrected to zero. Tune FOC PI as necessary.

Unstable FOC correction is generally a sign of imperfections of the angle sensing (noise, non-linearity, bad calibration, ...). Bad angle sensing causes the wrong values of Id and Iq to be measured, which the FOC algorithm falsely attempts to correct, worsening the current stability.

Decoupling Current Control

Decoupling Current Control is a powerful feedforward control technique, running in parallel with the traditional FOC Id and Iq PI regulators, as shown in the following figure. Decoupling control exploits the synchronous motor d-q equivalent circuit steady state equations in order to estimate the required d-q axis voltage, based on the commanded currents, measured speed and motor parameters L_d, L_q and voltage constant. Therefore, this particular technique realizes the high response of speed control system by giving the ability to control the d-axis and q-axis independently.

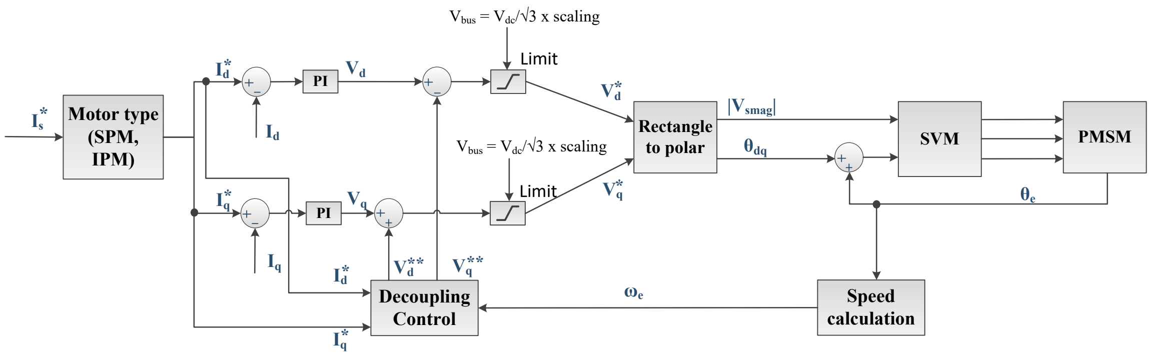


FIGURE 8-28. FOC control including decoupling terms.

Below is a description of the parameters illustrated in Figure 8-…

ω_e: electrical speed (rad/sec)

L_d: d-axis motor inductance (H)

L_q: q-axis motor inductance (H)

I_q^* : q-axis motor current command (A)

I_d^* : d-axis motor current command (A)

I_q : q-axis motor current measurement (A)

I_d : d-axis motor current measurement (A)

$|V_{smag}|$: Commanded voltage vector magnitude (V)

θ_{dq}: FOC angle correction (rad)

θ_e: electrical angle (rad)

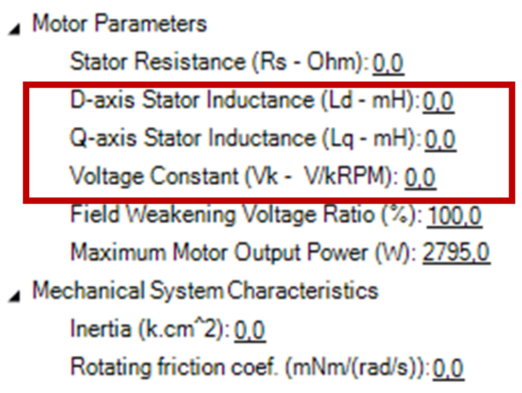
V_q^* : q-axis voltage command (V)

V_d^* : d-axis voltage command (V)

V_q^{**} : Decoupling control q-axis voltage estimation (V)

V_d^{**} : Decoupling control d-axis voltage estimation (V)

In order to enable the decoupling control, the following parameters need to be set through Roborun+ utility:



Motor Parameters
- Stator Resistance (Rs - Ohm): 0.0
- D-axis Stator Inductance (Ld - mH): 0.0
- Q-axis Stator Inductance (Lq - mH): 0.0
- Voltage Constant (Vk - V/kRPM): 0.0
- Field Weakening Voltage Ratio (%): 100.0
- Maximum Motor Output Power (W): 2795.0

Mechanical System Characteristics
- Inertia (k.cm^2): 0.0
- Rotating friction coef. (mNm/(rad/s)): 0.0

or by sending the following configuration commands from Console:

^LD ch nn,

where ch: motor channel, nn: D-axis motor inductance in Henry * 1000000

^LQ ch nn,

where ch: motor channel, nn: Q-axis motor inductance in Henry * 1000000

^VK ch nn,

where ch = motor channel and nn = peak value of motor induced phase to phase voltage amplitude (produced back-emf) per krpm speed * 1000

The d and q axis inductances can be automatically calculated also through Motor Characterization tool in Motor Sensor and Tuning setup wizard, supported in Roborun+ v3.0 utility (see Roborun+ Utility User Manual for more details). By setting all the previous parameters with values higher than 0, the decoupling control is ***automatically*** enabled for all closed loop modes including FOC torque and flux gains. Disabling can be done by zeroing the d-q axis inductances and voltage constant. Default values for d-q axis inductances and voltage constant are zero, meaning the decoupling current control algorithm is by default OFF.

Furthermore, the switching mode should be set to "Sinusoidal" and the FOC gains should be inserted according to the method described in "FOC Gains Determination & Tuning" chapter at Section 8. It is noted that the decoupling control algorithm applies only at close loop operating modes, including FOC torque and flux gains.

Example

Below is a representative current response implemented test result in a G series product. The parameters were got from speed PI auto-tune process:

Phase resistance Rs : 0.048 Ω

D-axis inductance L_d : 0.375 mH

Q-axis inductance L_q : 0.383 mH

Voltage constant K_e: 37.75 V/krpm

Control mode : Torque mode

Current loop PI regulators bandwidth selected : 800 Hz

Commanded current: 0-30 A (Step)

Motor speed: 500 rpm

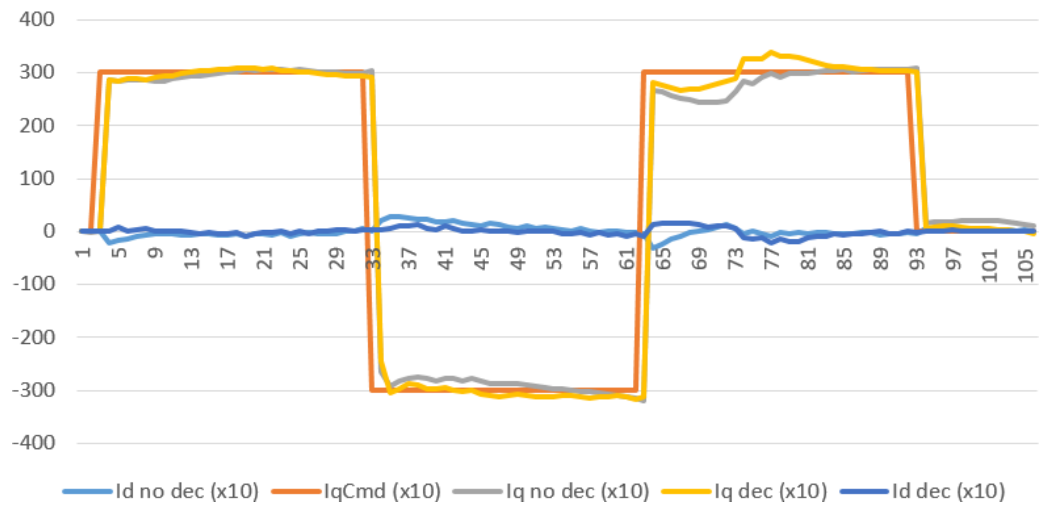


FIGURE 8-29. Current response test result.

From the above results the quicker response, as well as the lower fluctuation in Flux current can be observed with the decoupling control terms enabled.

Next, the same motor is tested in close loop speed mode, in order to evaluate the speed response. Below the parameters of the test is illustrated:

Current loop bandwidth selected : 800 Hz

Speed loop bandwidth selected: 10 Hz

Command through ramp: 0-1100 rpm

Acceleration: 2000 rpm/sec

Deceleration: 2000 rpm/sec

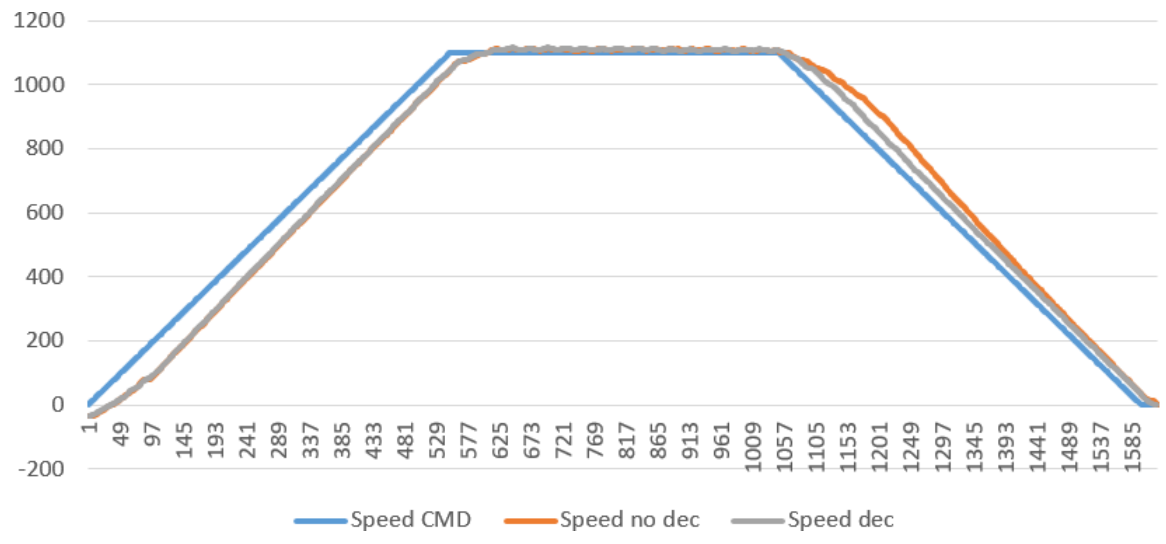


FIGURE 8-30. Speed response test result.

From the above test results important improvement is speed response is obvious, especially during deceleration.

Field Weakening

Field weakening is a technique that is used in order to increase the operating speed range of motor drive, as shown in the torque-speed curve below. At Constant Torque region the synchronous motor operates under Maximum Torque per Ampere mode, while in Constant Power region the synchronous motor operates under Flux Weakening mode in order to keep stator voltage constant. This can be done by applying some Flux (I_d) current, even though this also increases somehow the total consumed current from the motor and the source.

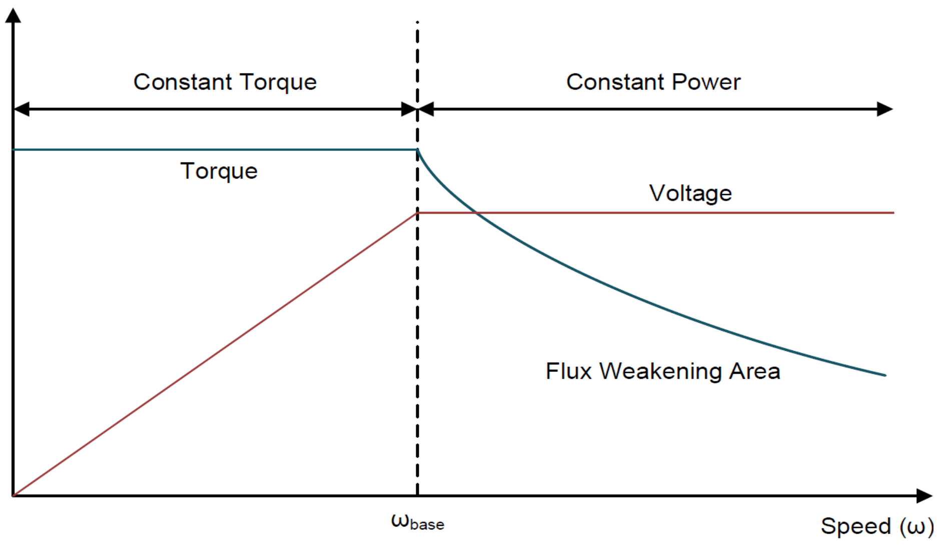


FIGURE 8-31. Example of a torque speed curve of a motor operating over the rated speed with field weakening.

RoboteQ controllers are capable of operating at field weakening region through two methods, manual and automatic, described below. It is noted that only one method, manual or automatic can be applied each time from user.

Manual Field Weakening

Manual field weakening is implemented by applying a non-zero set point for the Flux current. This can be done from the console, the serial port, or from a MicroBasic script with the command:

!GID ch Amps*10

The amount of Flux current should be different at low and high speed, typically starting with zero, and increasing after a given RPM threshold is reached. Below is an example of a MicroBasic script that changes the Flux set point according to such a rule.

```
top:
Speed = abs(getvalue(_S, 1)) ' Read motor speed from Encoders
if (Speed > 5000) ' check if above 5000 RPM
FluxSetpoint = (Speed - 5000) / 100 ' 1A per 100 RPM above 5000
else
FluxSetpoint = 0 ' No Flux current below 5000 RPM
end if
if (FluxSetpoint > 100) then FluxSetpoint = 100 ' Cap to 10.0 Amps
setcommand(_GID, 1, FluxSetpoint) ' Apply Flux setpoint
wait(10)
goto top ' repeat every 10ms
```

Automatic Field Weakening

Below block diagram describes the automatic field weakening function. The field weakening I_d command is regulated from a voltage PI additionally to a feedforward field weakening function for optimized response.

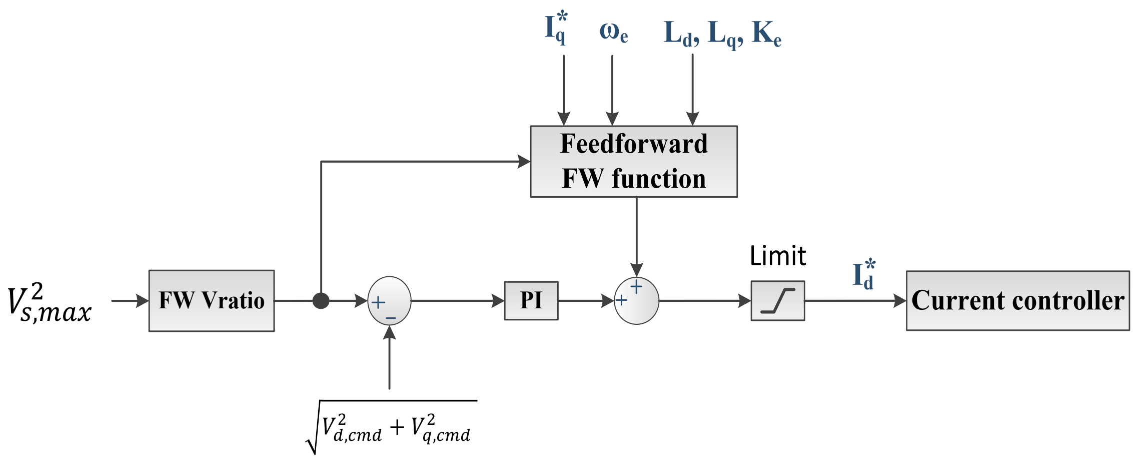


FIGURE 8-32. Automatic field weakening control block diagram.

Below is a description of the parameters illustrated in Figure 8-32.

ω_e: electrical speed (rad/sec)

L_d: d-axis motor inductance (H)

L_q: q-axis motor inductance (H)

I_q^* : q-axis current command (A)

I_d^* : d-axis current command (A)

K_e: Voltage constant (V/rad/sec)

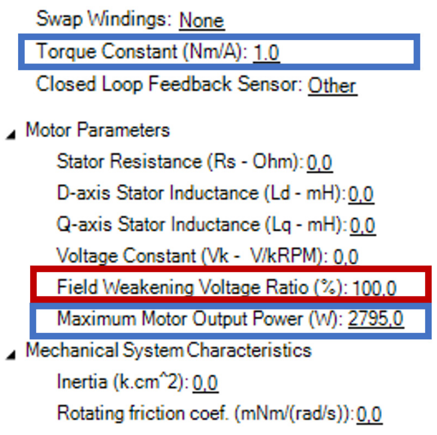
$V_{d,cmd}$: d axis voltage command(V)

$V_{q,cmd}$: q axis voltage command (V)

$V_{s,max}$: maximum applied stator voltage (V)

FW Vratio: the (%) of the maximum permitted stator PWM voltage to be regulated from field weakening algorithm

The field weakening control is applicable for both surface and interior permanent magnet motors, for all closed loop modes with sinusoidal commutation and configured FOC torque, flux, and speed gains. In order to enable the automatic field weakening control, the following parameters needed to be set:



Swap Windings: None
Torque Constant (Nm/A): 1.0
Closed Loop Feedback Sensor: Other

Motor Parameters
- Stator Resistance (Rs - Ohm): 0.0
- D-axis Stator Inductance (Ld - mH): 0.0
- Q-axis Stator Inductance (Lq - mH): 0.0
- Voltage Constant (Vk - V/kRPM): 0.0
- Field Weakening Voltage Ratio (%): 100.0
- Maximum Motor Output Power (W): 2795.0

Mechanical System Characteristics
- Inertia (k.cm^2): 0.0
- Rotating friction coef. (mNm/(rad/s)): 0.0

Or by sending the following configuration commands from Console:

^FWVR ch nn,

where ch: motor channel, nn: field weakening voltage ratio (%) * 10

^TNM ch nn,

where ch: motor channel, nn: torque constant (Nm/A_{peak}) * 1000

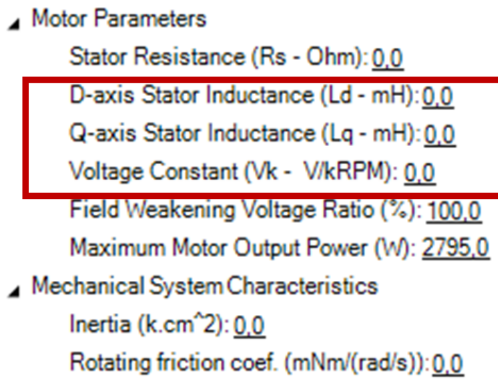
^MXPW ch nn,

where ch = motor channel and nn = maximum motor output power (W) * 10

By setting the field weakening voltage ratio with values < 100%, the field weakening control is ***automatically*** enabled for all closed loop modes including FOC torque, flux, and speed gains (no additional Enable/Disable input command required). Disabling can be done by setting field weakening voltage ratio at 100 %. Default values for field weakening voltage ratio is 100%, meaning that field weakening control is by default OFF. Value range for field weakening voltage ratio is 75% - 100%. Typical field weakening voltage ratio values should be around 90-97 %, depending on the application.

Two other important parameters for field weakening operation, that need to be configured, are the maximum output motor power (W) and torque constant (Nm/A_{peak}). As described above, these parameters will define the I_q current limit at constant power region. It is noted that the q-axis current limitation at constant power region is necessary in order to release d-axis current for the field weakening process.

By configuring the above mentioned parameters field weakening voltage ratio, maximum output motor power and torque constant the field weakening control operates only with the voltage PI regulator shown in figure below. In order to enable the feedforward field weakening function, the following parameters needed to be set:


Motor Parameters
Stator Resistance (Rs - Ohm): 0,0
D-axis Stator Inductance (Ld - mH): 0,0
Q-axis Stator Inductance (Lq - mH): 0,0
Voltage Constant (Vk - V/kRPM): 0,0
Field Weakening Voltage Ratio (%): 100,0
Maximum Motor Output Power (W): 2795,0
Mechanical System Characteristics
Inertia (k.cm^2): 0,0
Rotating friction coef. (mNm/(rad/s)): 0,0

or by sending the following configuration commands from Console:

^LD ch nn,

where ch: motor channel, nn: D-axis motor inductance in Henry * 1000000

^LQ ch nn,

where ch: motor channel, nn: Q-axis motor inductance in Henry * 1000000

^VK ch nn,

where ch = motor channel and nn = peak value of motor induced phase to phase voltage amplitude (produced back-emf) per krpm speed * 1000

The d and q axis inductances can be automatically calculated also through Motor Characterization tool in Motor Sensor and Tuning setup wizard, supported in Roborun+ v3.0 utility (see Roborun+ Utility User Manual for more details). By setting both the following parameters with values higher than 0, the feedforward field weakening is ***automatically*** enabled, provided that the field weakening voltage ratio has been set less than 100%. Disabling can be done by zeroing the d-q axis inductances and voltage constant. Default values for d-q axis inductances and voltage constant are zero, meaning the feedforward field weakening algorithm is by default OFF.

Example

Below is a representative torque vs speed test result, showing the significant increase in operating speed range.

Current limit: 33 A

Input dc voltage : 24 V

Field weakening voltage ratio: 95%

Control mode : Close loop torque

Test speed range: 200 – 3700 rpm

Current loop bandwidth selected : 800 Hz

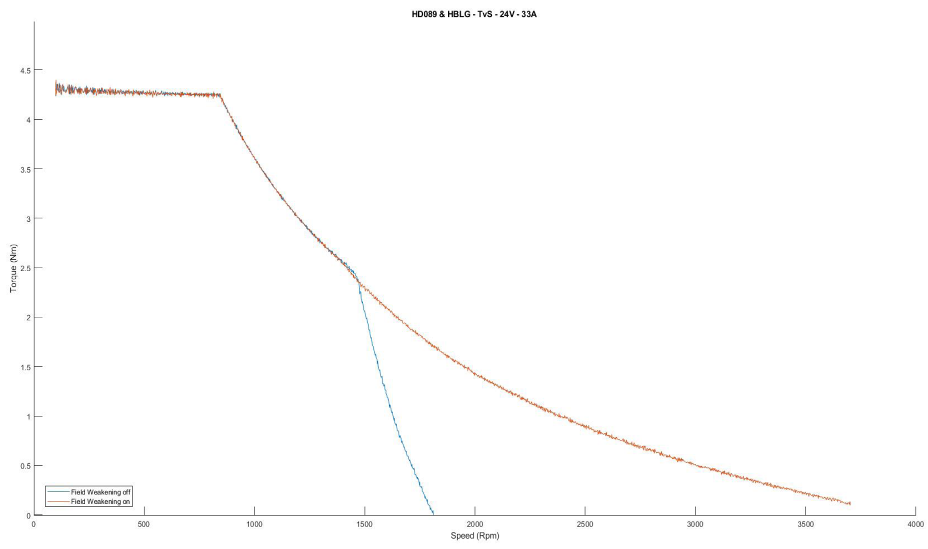
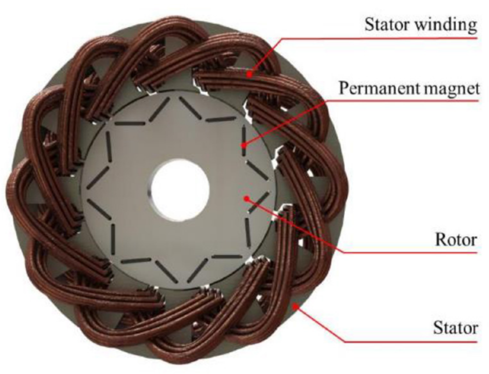


FIGURE 8-33. Torque vs speed curve with and without field weakening control.

Interior Permanent Magnet Motor Operation

IPM (interior permanent magnet) motor is an Alternating Current (AC) synchronous motor, where the permanent magnets are inserted inside the rotor, while in brushless dc motors the permanent magnets are mounted on the surface of the rotor, as illustrated in figure 8-34.



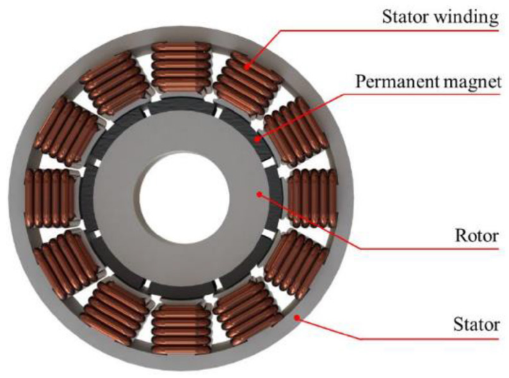


FIGURE 8-34. (a) IPM motor (b) brushless dc motor configurations

Below the basic structural and operating differences between the IPM and BLDC (Surface Permanent Magnet - SPM) motor:

IPM

- Reduce the risk of a magnet being peeled off by centrifugal force
- Higher total torque (IPM motor produces both magnetic and reluctance torque)
- Higher speed range (IPM motor has higher field weakening capability, due to the insertion of the permanent magnets inside the rotor)
- Reduce the risk of Permanent Magnets demagnetization
- Higher efficiency (lower permanent magnet Eddy current losses)

BLDC

- The permanent magnets are mounted on the surface of the rotor
- PMs are glued (provokes aging due to centrifugal forces and heat) or banded
- Lower total torque (BLDC produces only magnetic alignment torque)
- Lower speed range (BLDC motor has lower field weakening capability, due to the PMs location at the airgap)
- Lower efficiency (higher PM Eddy current losses)

Typical IPM motor torque-speed and power-speed curves are described in below figures, consisting of Constant Torque Region (CTR) and Constant Power Region (CPR). It is evident that in the CPR the torque reduction for the same motor current is low, especially for lower loading conditions, due to the increased reluctance torque added to the magnet torque.

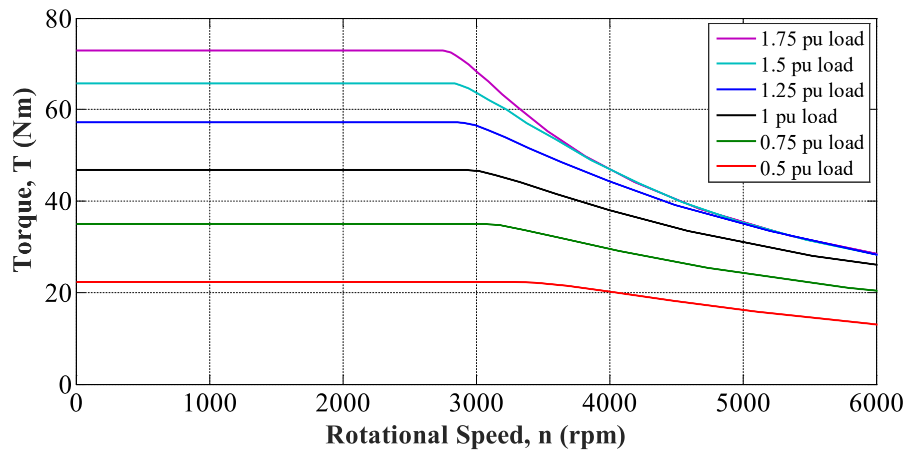


FIGURE 8-35. Typical IPM motor torque-speed curve.

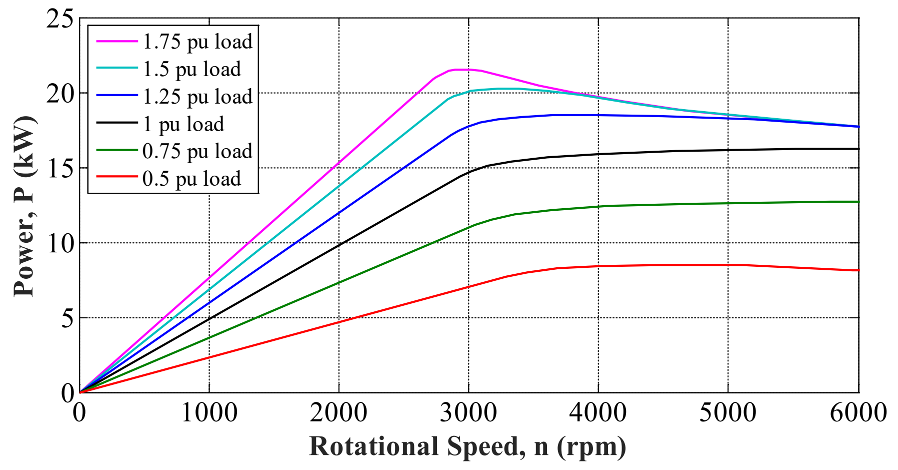


FIGURE 8-36. Typical IPM motor power-speed curve.

Constant Torque Region IPM motor control algorithm

For optimal efficiency during the whole IPM motor operating range the control technique employed for the CTR – that is the region up to rated speed - is the Maximum Torque Per Ampere (MTPA). According to that method both torque (Iq) and field weakening (Id) currents need to be adjusted for each torque demand as shown in figure 8-37. The calculations for the required currents are implemented automatically by the controller, by utilizing the IPM motor equivalent d-q axis (Park transformation) circuit equations. In CTR, the only absolute limit applied in the motor is the motor thermal current maximum limit. As shown in figure 8-38 the current vector is below current limit and voltage limit loops.

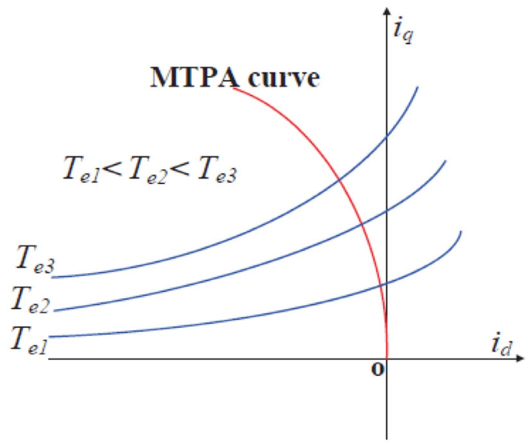


FIGURE 8-37. IPM motor required Iq, Id with produced electromagnetic torque.

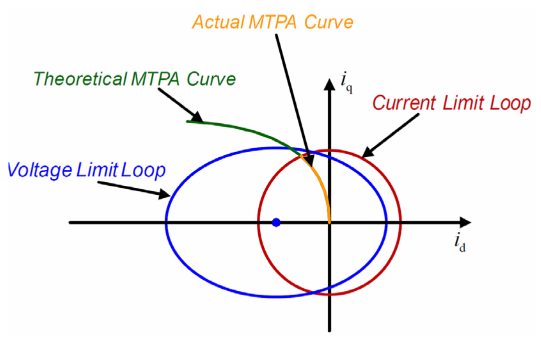
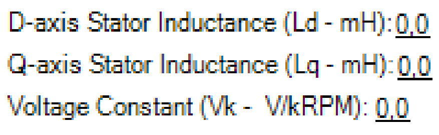


FIGURE 8-38. IPM motor MTPA d-q current control strategy on CTR - low speed region.

In order to effectively operate the IPM motors at MTPA curve, the following data required to be set during controller configuration from Roborun+ utility:



D-axis Stator Inductance (Ld - mH): 0.0

Q-axis Stator Inductance (Lq - mH): 0.0

Voltage Constant (Vk - V/kRPM): 0.0

or by sending the configuration command from Console tab:

^LD cc nn

where cc = motor channel and nn = motor d-axis inductance (H) * 1000000

^LQ cc nn

where cc = motor channel and nn = motor q-axis inductance (H) * 1000000

^VK cc nn

where cc = motor channel and nn = peak value of motor induced phase to phase voltage amplitude (produced back-emf) per krpm speed * 1000

Above values are necessary in order to calculate the optimal Id, Iq current commands needed to be applied to FOC motor operation. Typically, these values are included in each IPM motor technical datasheet. Alternatively, the d-q axis motor inductances can be calculated from "Motor characterization" tool at Roborun+ utility (see Roborun+ Utility User Manual for more details).

Furthermore, the switching mode should be set to "Sinusoidal" and the FOC gains should be inserted according to the method described in "FOC Gains Determination & Tuning" chapter at Section 8 of Motor Controllers Manual, considering as phase inductance the d-axis inductance Ld for flux proportional gain and q-axis inductance Lq for torque proportional gain, respectively. It is noted that the IPM motor operation algorithm applies only at close loop operating modes, including FOC torque and flux gains. In all other cases, IPM motor can operate as a brushless dc motor.

Example:

Consider the IPM motor with the following characteristics provided from manufacturer:

R (phase resistance) = 85 mOhm

Ld (d-axis inductance) = 580 uH

Lq (q-axis inductance) = 850 uH

The FOC gains for 300 Hz current loop bandwidth are calculated as follows:

Torque proportional gain = Q-axis inductance (H) * Bandwidth = 0.000850 * 1884 = 1.602

Torque integral gain = Phase resistance (Ohm) * Bandwidth = 0.085 * 1884 = 160.14

Flux proportional gain = D-axis inductance (H) * Bandwidth = 0.000580 * 1884 = 1.092

Flux integral gain = Phase resistance (Ohm) * Bandwidth = 0.085 * 1884 = 160.14

Constant Power Region IPM motor control algorithm

When the motor operates in the high speed region - CPR, the MTPA *d-q* axis motor currents should be appropriately regulated, in order to satisfy the voltage limitation defined from battery dc voltage and PWM modulation method utilized, in conjunction with the current limitation, as shown in below graph. Therefore, the field weakening Id current should be increased, in absolute value, at this mode of operation to weaken the permanent magnet field and thus, the produced back-EMF of the motor.

The field weakening current required for the constant power region is calculated from the automatic field weakening function described analytically in chapter "Field Weakening" at Section 8, which is the same for SPM and IPM motor type.

Therefore, in order to operate the IPM motor at full speed range (CTR and CPR) all the requested parameters for MTPA operation and field weakening control should be properly configured. It is also noted that in IPM motor configuration, external manual field weakening current is not supported.

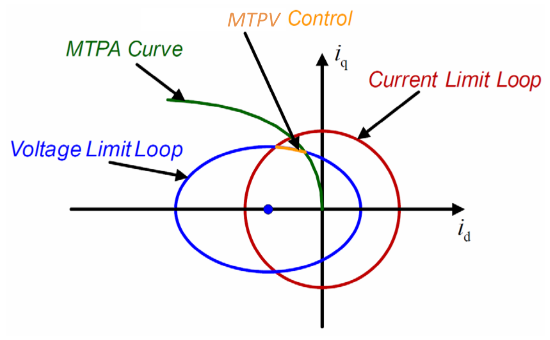


FIGURE 8-39 IPM motor field weakening/MTPV control strategy on CPR - high speed region.

Operating Brushless Motors

This section covers operating features that are common to Brushless Motor control, regardless of the commutation mode.

Stall Detection

The rotor sensors and the encoders can be used to detect whether the motor is spinning or not. The controller includes a safety feature that will stop the motor power if no rotation is detected while a given amount of power is applied for a certain time. Three combinations of power and time are available:

- 250ms at 10% power
- 500ms at 25% power
- 1s at 50% power

If the power applied is higher than the selected value and no motion is detected for the corresponding amount of time, the power to the motor is cut until the next idle motor command is given (0 in case of speed modes, equal to feedback in case of position modes). This function is controlled by the BLSTD - Brushless Stall Detection parameter (see "BLSTD - Brushless Stall Detection" in Command Reference section). Do not disable the stall protection.

A stall condition is indicated with the "Stall" LED on the Roborun PC utility screen.

The detection uses the speed measurement from the rotor sensor.

Important Notice

In close loop modes, it is quite possible to have the motor stopped while power is applied to them. That could happen while stopped uphill, for example. Select the appropriate triggering level for your application

Sensor Error Detection

Sensor Error Detection methods have been implemented based on the kind of sensors used for commutation.

The user can select one of three options, Disabled, Tolerant and Strict (see "SED - Sensor Error Detection", page 366).

- Disabled: All sensor errors are ignored
- Tolerant: The fault will be active after 5 errors. The counter is reset if there is no sensor error for 128 ms
- Strict: The fault will be activate on the first sensor error occurrence.

If Hall sensor is used (Hall Trapezoidal and Hall Sinusoidal) we consider as a sensor error:

- An invalid hall state (0 or 7 for 120 degrees sensors, 2 or 6 for 60 degrees sensors).
- An out of sequence hall state. When the motor moves and the hall state has a specific value, only two out of the rest 5 hall states can be considered as expected values. Which of the two, depends on direction.

If Hall+Encoder sensors are used (Hall+Encoder Sinusoidal), we consider as sensor error:

- Whatever is considered as error in case of hall error.
- When the electrical angle of the encoder and the electrical angle of the hall sensor are more than 45 degrees apart in three sequential hall transitions.

Note: In case of Hall+Encoder Sinusoidal, whenever an error occurs concerning the encoder, the controller resynchronizes hall sensor and encoder, recovering the error.

If SinCos or Resolver sensors are used (SinCos Sinusoidal or Resolver Sinusoidal), we consider as sensor error:

- Upon configuration and the sensor is uncalibrated (see ZSMA - Cos Amplitude and ZSMC - SinCos Calibration). Calibration is done automatically during Motor/Sensor setup.

In all other switching modes Sensor Error Detection is not applicable.

Important Note

In case of dual channel controllers, and since the default value of SED is Strict, the Sensor Error flag will be triggered even if only one channel is used. This will be mainly due to 2nd channel being in default configuration (Hall Trapezoidal and SED to strict) and no hall sensor connected to the respective connector. So if you wish to run with only one channel make sure to disable the SED in the second one.

Speed Measurement using the angle feedback Sensors

Information from Hall, SPI/SSI and sin/cos sensors is used by the controller to compute the motor's rotation speed.

When Hall sensors are used, speed is determined by measuring the time between Hall sensor transitions. This measurement method is very accurate, but requires that the motor be well constructed and that the placement between sensors be accurate. On precision motors, this results in a stable speed being reported. On less elaborate motors, such as low-cost hub motors, the reported speed may oscillate by a few percent.

Speed measurement is very precise with digital absolute sensors (SSI). Sin/Cos sensors operating without noise also give a very precise value.

The motor's number of poles must be entered as a controller parameter in order to produce an accurate RPM value. See discussion above. The speed information can then be used as feedback in a closed loop system. Motor with a more precise Hall sensor positioning will work better in such a configuration than less precise motors.

If the reported speed is negative when the slider is moved in the positive direction, you can correct this by putting a negative number of poles in the motor configuration. This will be necessary in order to operate the motor in closed loop speed mode using hall sensor speed capture.

Distance Measurement using Hall, SSI or other Sensors

When Hall sensors are used, the controller automatically detects the direction of rotation, keeps track of the number of Hall sensor transition and updates a 32-bit up/down counter. The number of counts per revolution is computed as follows:

Counts per Revolution = Number of Poles * 6

With Resolver or Sin/Cos sensors, the controller accumulates the sensor angle data to recreate an accurate and high resolution 32-bit counter. For these sensors, the number of counts per revolution is:

Counts per Revolution = Number of Sensor Poles * 512

With SSI sensor, the controller accumulates the SSI data difference to recreate an accurate and high resolution 32-bit counter. For these sensors, the number of counts per revolution is:

Counts per Revolution = Number of Sensor Poles * (1 << SSI Counter number of bits)

The counter information can then be read via the Serial/USB port, CAN bus, or can be used from a MicroBasic script. The counter can also be used to operate the brushless motor in a Closed Loop Position mode, within some limits

SECTION 9

AC Induction MotorOperation

This section discusses the controller's operating features and options when using three phase AC Induction motors.

Introduction to AC Induction Motors

Three phase induction motors are the most common types of electrical motors. They have a very simple construction composed of a stator covered with electromagnets, and a rotor composed of conductors shorted at each end, arranged as a "squirrel cage". They work on the principle of induction where a rotating electro-magnetic field it created by applying a three-phase current at the stators electromagnets. This in turn induces a current inside the rotor's conductors, which in turns produces rotor's magnetic field that tries to follow stator's magnetic field, pulling the rotor into rotation.

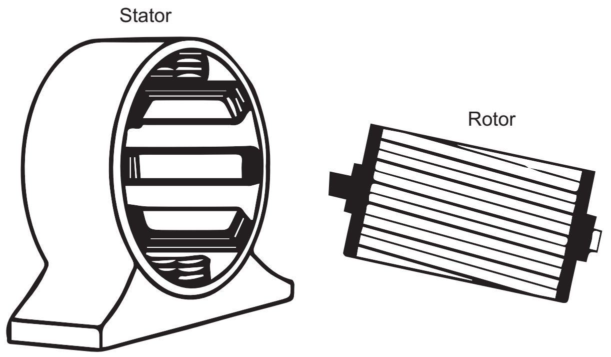


FIGURE 9-1. AC-Induction Motor

Benefits of AC Induction Motors are:

- Induction motors are simple and rugged in construction. They are more robust and can operate in any environmental condition.
- Induction motors are cheaper in cost due to simple rotor construction, absence of brushes, commutators, and slip rings

- They are maintenance free motors unlike dc motors due to the absence of brushes, commutators and slip rings.
- Induction motors can be operated in polluted and explosive environments as they do not have brushes which can cause sparks

Asynchronous Rotation and Slip

AC Induction motors are Asynchronous Machines meaning that the rotor does not turn at the exact same speed as the stator's rotating magnetic field. Some difference in the rotor and stator speed is necessary in order to create the induction into the rotor. The difference between the two is called the slip.

Slip is measured in Hertz. It is the difference of the frequency generated by the controller, and the rotor's frequency, as determined by the formula

f = ((RPM / 60) * NumberOfPolePairs)

Optimal slip varies from the motor to motor and is in the range of typically 2 to 10Hz.

As seen from the figure below, when the slip is 0, i.e. the rotor turns at exactly the same speed as the stator field, torque totally disappears. Within the stable operating region, the Torque is proportional to the Slip. The torque and motor efficiency then quickly drops when the slip grows past its optimal value.

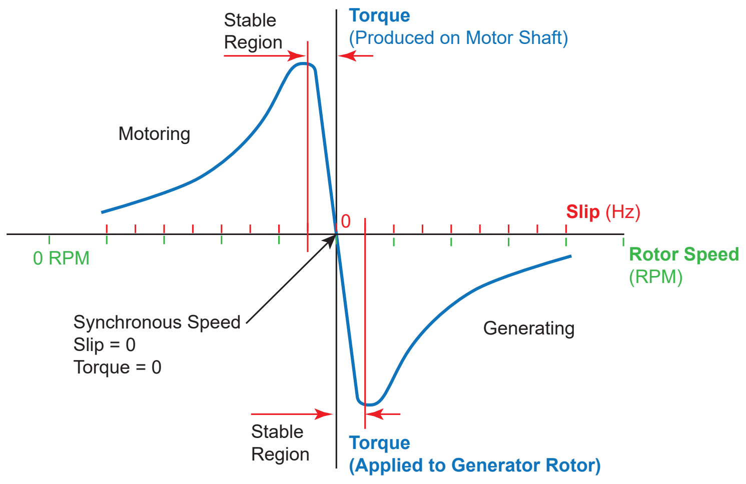


FIGURE 9-2. Torque vs. Rotor Speed graph for ACIM

The main task of the motor controller is to generate a rotating magnetic field whose frequency and strength is such that the rotor will operate within the motor's optimal slip range. Three techniques are supported by Roboteq for achieving this:

Scalar, Volts per Hertz (VPH)

Constant Slip

Field Oriented Control

Each of these techniques, benefits, and limitations are described in the following sections

Connecting the Motor

An AC Induction motors have just 3 power wires which must be connected to the controller's U V and W terminals. The connection order is not important. However, swapping any two motor connections will make the motor turn in the opposite direction.

Selecting and Connecting the Encoder

A speed sensor must be used to measure and control the motor's slip when running in Constant Slip mode and Torque/Speed FOC mode. This is done using an incremental encoder. Most AC induction motors come with a built-in quadrature encoder. These encoders typically have a relatively low number of counts. 32 or 64 Pulses Per Revolution (PPR) rotation are typical values. A low count encoder results in low frequency pulses.

When using encoders up to 128 Pulses Per Revolution, the controller evaluates the rotation speed by measuring the time between encoder pulses. This results in measurement with a resolution of 0.1Hz even at full speed.

When using encoders with higher PPR, speed is measured by counting the number of encoder signal transitions over a 10ms period. Prefer therefore a high-count encoder of around 1000 PPR for better speed measurement resolution.

Unless otherwise noted in the product's datasheet, all Roboteq's AC Induction Motor Controllers have to pull up resistors that can connect to open collector encoder outputs. Controllers also have capacitors to help filter out any electrical noises that contribute to fake encoder readings.

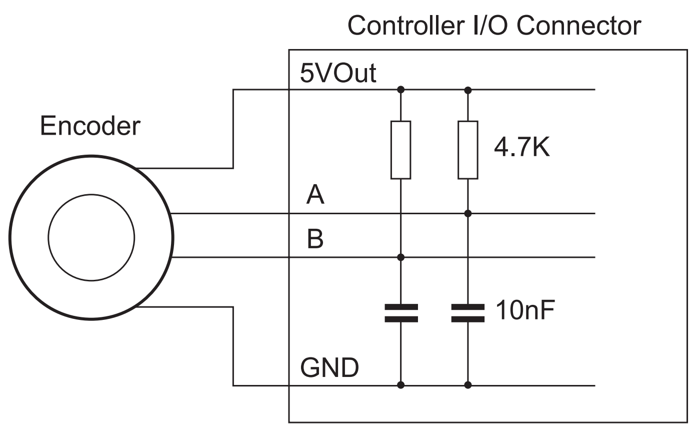


FIGURE 9-3. Encoder connection

Testing the Encoder

To test the encoder, use the PC utility to enable the encoder and set its number of Pulses Per Revolution (PPR). Go to the Utility's Run tab, enable the Encoder Count in the chart. Make a full turn of the motor shaft by hand. Verify that the counter has changed by the number of PPR * 4.

Prior to enabling Constant Slip or FOC Modes, operate the motor in open loop Volts per Hertz mode with slip control disabled. To disable slip control, set the encoder as No Action in the configuration menu.

Apply a positive motor command. Verify that the motor shaft is moving in the desired direction. If the motor moves in the opposite direction, swap any two of the three motor cables, or change the motor direction to Inverted in the motor configuration.

If the motor moved in the desired direction, then verify that the encoder counter increments when a positive motor command is applied. If the counter decrements, then either swap the A and B encoder wires or enter a negative number of PPRs in the encoder configuration.

Open Loop Variable Frequency Drive Operation

In its simplest operating mode, the controller will output to the motor a three-phase sinusoid whose voltage and frequency change together at a fixed ratio. This mode is called Scalar because of the fixed ratio between the Voltage and Frequency that is applied to the motor.

The ratio is set by the VPH - Volts Per Hertz configuration parameter.

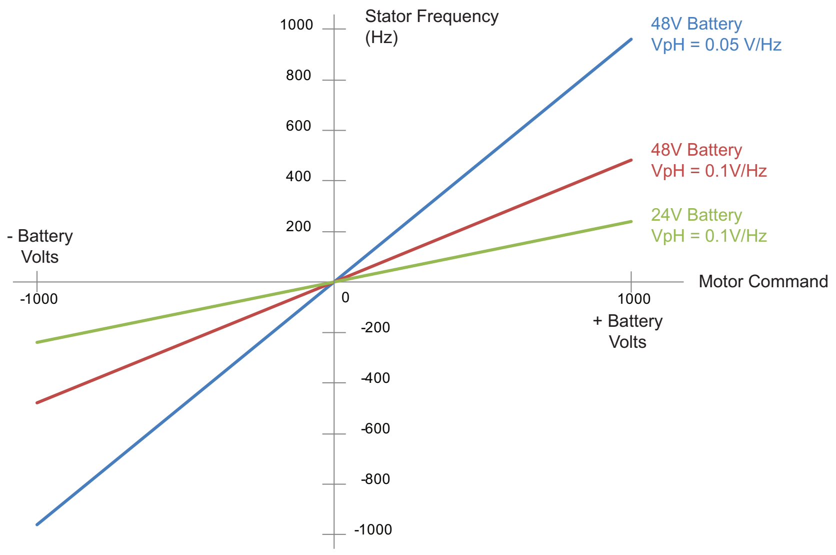


FIGURE 9-4. VPH graph for ACIM

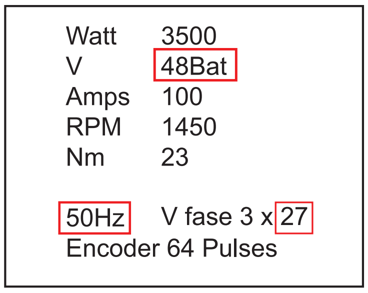
The figure above shows an example of the resulting stator frequency for a given motor command. In open loop mode, Motor commands range from -1000 to +1000 and result in the output voltage to range between -VBat to +VBat respectively

As long as the motor is not overloaded, the rotor RPM will be

((Stator Frequency - Slip) / Number of Pole Pairs) * 60

Figuring the Motor's Volts per Hertz

Each motor has a value for the optimal Volts per Hertz ratio. It can be determined by the operating frequency and rated voltage written of the motor's label. The figure below shows values from a real motor.



Watt	3500
V	48Bat
Amps	100
RPM	1450
Nm	23

50Hz V fase 3 x 27
Encoder 64 Pulses

FIGURE 9-5. ACIM label

For this motor, the VPH can be determined by dividing the 48 V phase to phase motor voltage amplitude by the 50Hz frequency. In this case 0.96 Volts per Hertz. It is worth mentioning that the rated voltage in the calculated VPH is related to the amplitude of the phase to phase voltage (not RMS value). Therefore, if the motor manufacturer provides the nominal stator voltage in RMS, it should be transformed to peak voltage value.

You can validate that the rated VPH is applied if the motor at no load reaches the rated speed when applying maximum motor command (Power =1000).

Note that this value is for the optimal torque as rated on the label. If the load is a lot lighter, the VpH will be too high and result in excessive current consumption. If the load is a lot heavier, the VpH will be too low and the motor will not be able to drive it. The VpH will therefore need to be different from this computed based on actual load conditions. Always first monitor the motor consumption at no load. Adjust the VpH to a lower value if the no load current appears too high.

Maintaining Slip within Safe Range

Open Loop, or Scalar, the mode does not require encoders for its operation. If the load is known to always be within the motor's max torque, the motor can be trusted to always be able to drive it. In this case, an encoder does not need to be connected. If an encoder is connected - to measure and report speed, for example - then it must be configured as "No Action" in the PC Utility.

For added safety and better performance, however, an encoder can be installed and enabled to measure the rotor's speed, and therefore the slip, in real-time. When the encoder is enabled and configured as "Feedback", the controller will lower the voltage and frequency if the slip exceeds twice the value stored in the Optimal Slip configuration.

Prior to enabling the encoder as Feedback, verify that the encoder count direction has the same polarity as the motor command.

Closed Loop Speed Mode with Constant Slip Control

In this mode, the controller will automatically adjust the voltage and frequency in order to reach and maintain the desired speed, even as the load is changing, while operating within the optimal slip range (which is the half of the configured maximum slip).

The "Optimal slip" value in the configuration is referred to the optimal slip at rated frequency where the motor produces the maximum breakdown torque Sk. The induction motor slip is controlled to the half of the optimal slip for stability reasons, in order to be on the safe side in case of a quick transient situation.

To configure this mode, first set the controller in open loop mode as described in the previous section. Verify that the encoder is working and is counting with the correct polarity.

Once the encoder is verified to work and the motor spins in the open loop, follow these steps. Using the Roborun PC utility:

- Select Close Loop Constant Slip Mode
- Set the PID gains found in the Motor Output, Closed Loop Speed Parameters menus (do not use the FOC PID gains). Try first with gains of P=0, I=1, D=0. These values will produce adequate results in most cases. Additional turning may be needed.
- Set the Max RPM configuration to the speed that must be reached at full throttle (ie when command = 1000). Make sure to enter a value that is within the physical reach of the motor under the expected maximum load condition.
- Enter the lowest acceleration rate that is acceptable for the application. Rapid changes will create current surges and should therefore not be allowed to be higher than necessary.
- Save the settings to the controller.

The motor speed can now be set to be any value between 0 and plus/minus the maximum RPM configured above when sending a command ranging from -1000 to +1000 using the serial, analog or pulse inputs.

The motor speed can also be set to an absolute RPM value by sending the S (Speed) command via serial, USB, CAN or Scripting.

When exercising the motor with the PC utility, monitor the Slip, the Rotor RPM, Stator RPM and the Motor Amps.

The slip will stabilize at the Optimal Slip setting.

Field Oriented Control (FOC) mode Operation

Field Oriented Control (or Vector Drive) is a technique by which the magnetic field generated in the stator is adjusted in relation to the field induced in the rotor in a manner to generate optimal torque at all times and all load conditions.

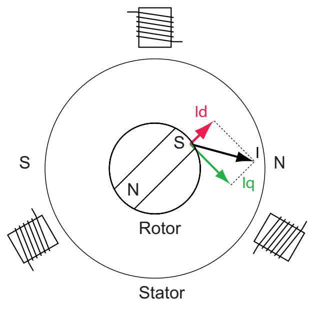


FIGURE 9-6. FOC on ACIM

The optimal rotation occurs when the magnetic field induced in the rotor is perpendicular with this of the stator. Practically the fields the two fields are never exactly perpendicular. As shown in the diagram above, the angled field I am made of an outward pulling flux field (Id) and perpendicular pulling torque field. The torque field is the one that causes the rotation and that the controller will maximize. Flux field is not causing any rotation and therefore must be minimized. Some flux is necessary at all times, however, in order to create the induction in the rotor.

The challenge in induction motors is that the rotor flux's absolute position cannot be measured physically. It is determined mathematically using known speed, voltage and current, and a model representation of the motor's main parameters shown in the figure below.

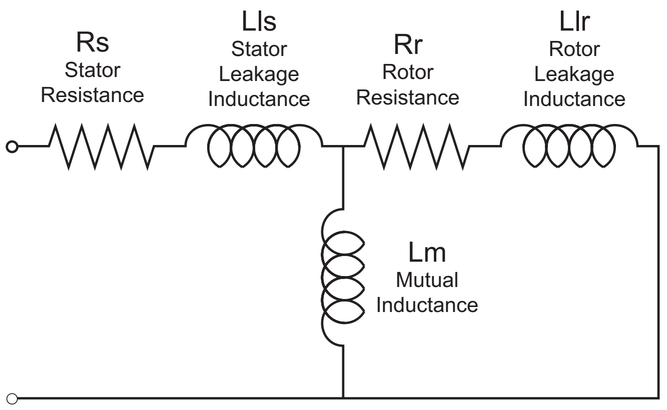


FIGURE 9-7. Electrical representation of ACIM

These parameters include per phase rotor resistance 'Rr', rotor leakage inductance 'Llr', mutual inductance 'Lm' and rotor leakage inductance 'Llr'. Usually, motor manufacturer will provide you an equivalent circuit of the induction motor that contains Rr, Rs, Lm, Llr, Lls

In FOC, therefore, rotor flux and motor torque can be individually controlled regardless of load and speed. FOC offers better dynamic performance, accurate current control and ensures maximum efficiency, unlike traditional scalar control methods such as Open Loop VpH and Constant Slip Control.

Under FOC operation, AC induction motors can be run in either FOC torque mode or FOC speed mode. FOC torque mode allows users to command a torque to a motor in terms of Amps. While FOC speed will regulate the speed at the command/desired value.

Note that, with a bad tuning of FOC (flux and torque) PID, the motor Amps can be higher than the Amps Limit. If that happened, try use lower bandwidth for FOC tuning (see in Chapter *FOC Gains Determination & Tuning*).

Configuring FOC Torque Mode

To configure FOC mode, first set the controller in Open Loop (Volts per Hertz) mode as described in one of the previous sections. Verify that the motor is spinning in the desired direction and that encoder is working and is counting with the correct polarity. See Testing the Encoder section above.

Once the encoder is verified to work and the motor spins in the open loop, follow these steps. Using the Roborun PC utility:

- Enter the motor parameters under the section "Motor Parameters" in RoboRun configuration. Usually, motor manufacturer will provide you an equivalent circuit of the induction motor that contains Rr, Rs, Lm, Llr, Lls.
- Set the Flux Amps. In order to find the optimal flux amps, run the motor in Volts per Hertz (VpH) mode with small/no load using the motor's rated VpH ratio. Then watch the flux amps in the PC utility. Enter this value in the configuration. This flux amps can be increased (Flux Strength) for low speed and high torque operation requirements and can be decreased (Field Weakening) for high speed and low torque operation requirements.

- Set the operating mode to FOC torque. Set Amps limit according to the application's need but do not exceed the motor specifications.
- Save the settings to the controller.
- Next step is to tune FOC (Flux and Torque) PID. Start with low proportional gain e.g. 0.1, and then set some Integral gain. Integral gain is more important in this case. Alternatively use the method as described in chapter "FOC Gains Tuning" in Section 8, using the Stator Resistance (Rs) and the Stator Inductance (Ls).
- Monitor and Record the Flux Amps and Torque Amps for the desired motor channel when tuning the FOC PID.
- Put some high load on the rotor and command a step Torque Amps from the slider bar (say 10A). Record the "FOC Torque Amps" reading on the chart. If the step response reaches the desired (10A) steady state fast enough then the PID is can be considered tuned. If it is slow then increase integral gain. If the Torque and Flux Amps show noise at high speed or motor produces noisy sound, then lower your proportional gain Kp.
- Once FOC/current PID is tuned, FOC torque mode is ready to operate and FOC speed mode can then be tuned. Note: It is important to know the value of Flux Amps the motor is designed to operate under. Flux Amps stay the same during entire FOC operation unless field weakening is used.

Now motor Torque can be set to any desired value from 0 to plus or minus the value stored in the Amps Limit configuration parameter, by sending a command of -1000 to +1000 using the slider, analog input, pulse input, scripting, CAN or any other command mode.

Note that in Torque Mode, the Max Speed RPM configuration parameter is used to limit the motor speed if the motor is not loaded and the desired torque is below the torque that can actually be reached by the motor under the current load conditions. For example, a torque command of 50A on an unloaded motor (that will never draw 50A) will cause the voltage to increase to the maximum value, and therefore the motor to maximum speed, unless the speed is limited by the Max RPM parameter. For more details see chapter "Speed Limiting in FOC Torque Mode" below.

Note that, with bad tuning of FOC (flux and torque) PID, the motor might draw much higher current than expected. If so, try using lower bandwidth for FOC gains. For more details see chapter "**FOC Gains Determination & Tuning**", below.

FOC Gains Determination & Tuning

Good PI gains are important for the controller to quickly reach and stabilize the desired I_d and I_q current. A very good approximation of the gain values can be calculated from the motor's Resistance and Inductance as follows:

Torque FOC gains (iq current): consider the lock rotor test mode (check below figure) when mainly iq current flows.

Torque FOC K_p : $(L_{lr} + L_{ls}) \times 2\pi \times BW$

Torque FOC K_i : $(R_r + R_s) \times 2\pi \times BW$

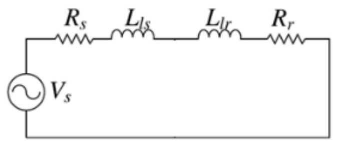


FIGURE 9-8. Locked rotor equivalent circuit

Flux FOC gains (id current): consider no-load test mode (check below figure) when mainly id current flows.

Due to the high magnetizing inductance the Flux current PI loop shall run faster than Torque current PI loop.

Flux FOC K_p gain: 2 x $K_{p,\ torque}$ (Torque FOC gain)

Flux FOC K_i gain: 2 x $K_{i,\ torque}$ (Torque FOC gain)

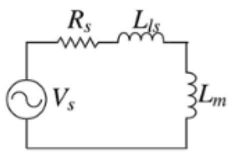


FIGURE 9-9. Locked rotor equivalent circuit

Bandwidth is in rad/sec and according to Nyquist criteria the current loop bandwidth cannot be more than the half of the current loop sampling time. Most commonly the current loop bandwidth is set to the 1/10-1/20 of the current loop sampling time. The current loop sampling time is at 16 kHz. So if we choose as current loop bandwidth the 300Hz then:

1Hz = 2π rad/sec

So for 300Hz Bandwidth = 300*2π rad/sec = 1885

Usually even smaller bandwidth can be as effective as the 50Hz. It is better to start with the smaller possible gains and then tune according to the behavior of the motor. Test in FOC torque mode with caution.

Example calculation for 50Hz bandwidth, 48mOhm Phase resistance (R_r + R_s) and 152uH Phase Inductance (L_{lr} + L_{ls}).

Torque FOC gains:

K_i = 0.048 * 1885 = 90.48

K_p = 0.000152 * 1885 = 0.2865

Flux FOC gains:

K_i = 2 * $K_{i,torque}$ = 180.96

K_p = 2 * $K_{p,torque}$ = 0.573

Configuring FOC Speed Mode

To configure FOC Speed mode, configure first the FOC Torque mode as described in the section above.

- Set the controller to FOC Speed Mode
- Tune speed loop PID in a similar manner as was done for FOC PID. Use the PID gains found in the Motor Output, Closed Loop Speed Parameters menus (do not use the FOC PID gains). It can be started with K_p term and introduce small K_d term. Once transient response on the graph seems reasonable then Ki can be used to get rid of steady state error.

Now motor Speed can be set to any desired value from 0 to plus or minus the value stored in the Max RPM configuration parameter, by sending a command of -1000 to +1000 using the slider, analog input, pulse input, scripting, CAN or any other command mode.

E.g. use the Roborun slider, or the console command

```
!G 1 800
```

to set motor RPM to 800 on channel 1 when MaxSpeed is set to 1000 RPM. Correspond-ing if MaxSpeed is set to 2000 RPM, any value on the slider will give 2 times RPM.

Speed can be also set as an absolute RPM value using the S command from Serial, USB, CAN or Microbasic.

Speed Limiting in FOC Torque Mode

AC Induction Controllers provide a way of smoothly limiting the speed in FOC torque mode to prevent motor runaways. The method for limiting the speed is based on PID speed over-ride control which provides very smooth motor output but requires PID tuning.

The speed loop PID tuning can either be done in "FOC torque mode" at the speed limit or in "FOC Speed Mode" by looking at the response time.

- When in "FOC Torque Mode", command some torque to the motor and cause the motor to spin to the speed limit. Some low frequency ripple will be noticed in the speed which should be minimized by increasing PID gains. Increase the PID gains to a point where no ripple is seen.
- If in "FOC Speed Mode", tune the speed PID gains by looking at the response time of the motor. And then move to case 1) to check the ripple in the speed.

Induction Motor Parameters Calculation

The motor parameters of Induction Motors are crucial when the Field Oriented Control Close loop torque/speed control is needed. More specifically, the rotor induced flux abso-lute position is determined mathematically using known speed, voltage and current, and a model representation of the motor's main parameters shown in the figure below.

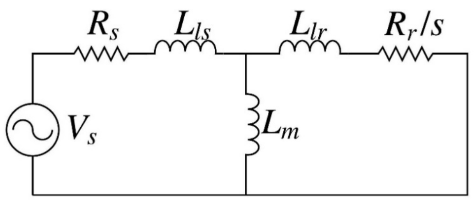


FIGURE 9-10. Electrical Representation of ACIM for motor characteristics calculation

These parameters include per phase rotor resistance 'R_r', rotor leakage inductance 'L_{lr}', mutual inductance 'L_m' and rotor leakage inductance 'L_{lr}' (core losses R_c are neglected). Furthermore, in order to set the FOC (Flux and Torque) PID gains, the Stator Resistance (R_s) and Stator Inductance (L_{ls}) needed, according to the method as described in chapter "FOC Gains Tuning" at Section 8. The most common ways, to manually estimate induc-tion motor parameters through the controller, are to test induction motor under no-load and locked rotor conditions.

No load testing

The no-load test, like the open circuit test on a transformer, gives information about exciting flux I_d current, the magnetizing inductance L_m and rotational losses. The test is performed by applying balanced rated voltage on the stator windings at the rated frequency. The small power provided to the motor is due to core losses, friction and winding loses. The motor will consume the necessary flux I_d current in order to establish the appropriate magnetic field. Motor will rotate at almost a synchronous speed, which makes slip nearly zero (s≈0). Therefore, the motor equivalent circuit is expressed as follows:

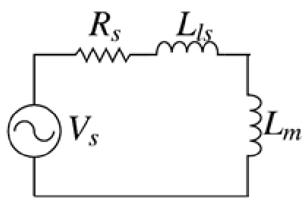


FIGURE 9-11. Electrical Representation of ACIM during no load testing

Assuming that the R_s (Ω), L_s(H) are much lower than the magnetizing inductance L_m (H), the following equation can be extracted:

$$L_m = \frac{V_s}{2\pi f_s I} \qquad (9.1)$$

where V_s (V) is the phase stator voltage applied (RMS value), f_s (Hz) is the stator frequency and I (A) is the RMS motor amps.

In order to implement the above mentioned no load test with RoboteQ induction motor controllers, please follow these steps:

- Set the operating mode to "Volts per Hertz" mode. Encoder feedback action needed.
- Configure the Volt per Hertz setting according to motor nominal voltage (peak value of stator voltage according to manufacturer) and frequency.
- Run the motor without load up to the maximum available voltage (Command=1000), utilizing small acceleration/deceleration value. Be sure that the configured volt per hertz is correct, by checking the "Stator speed RPM" to be approximately equal to manufacturer's synchronous rated speed.
- Calculate the applied V_s by the following equation:

$$V_s = \frac{m_a \times V_{dc}}{1000 \times \sqrt{2} \times \sqrt{3}} \qquad (9.2)$$

where m_a is the output PWM level (%1000) applied which is equal to Motor Power output measurement in Roborun+ utility (-1000 to 1000 range) and V_{dc} is the battery volts (V). In the considered no-load test the m_a is 1000, therefore the applied voltage can be calculated as $V_s = \frac{V_{dc}}{\sqrt{2} \times \sqrt{3}}$

- Measure the motor RMS current from the utility (Motor Amps). Furthermore, the peak amps value should be configured as "Rotor Flux current" in the configuration tab.
- Calculate the magnetizing inductance for the no load test according to equation (9.1).

Example:

The following data have been given from the induction motor manufacturer:

P = 400 W (nominal power)

I_N = 23 A (nominal stator current)

V_{dc} = 24 V (Input dc voltage at controller)

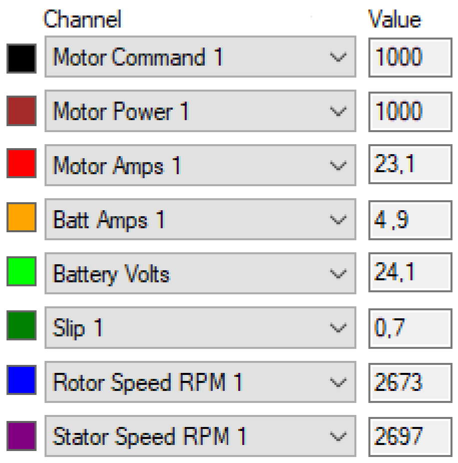
f_s = 90 Hz (Nominal stator frequency)

n_s = 2700 rpm (Synchronous speed)

p= 2 (pole pairs)

Therefore, the Volts per Hertz ratio introduced in configuration tab is $\frac{V_{dc}}{f_s} = \frac{24}{90} = 0.267$

Then the motor run volts per Hertz and the respective results from Roborun+ utility are illustrated below.



	Channel	Value
⬛	Motor Command 1	1000
🟫	Motor Power 1	1000
🟥	Motor Amps 1	23,1
🟧	Batt Amps 1	4,9
🟩	Battery Volts	24,1
🟩	Slip 1	0,7
🟦	Rotor Speed RPM 1	2673
🟪	Stator Speed RPM 1	2697

FIGURE 9-12. No load testing monitoring

From the above results, it can been observed that the stator speed is close to the synchronous speed (2700 rpm), while the slip is very small due to the no-load operation.

Therefore, by applying equation (9.1) the magnetizing inductance is:

$$L_m = \frac{V_s}{2\pi f_s I} = \frac{\frac{24.1}{\sqrt{3} \times \sqrt{2}}}{2 \times \pi \times 90 \times 23.1} = 754uH$$

Furthermore, the rotor "Rotor Flux current" should be set in the configuration as (Motor Amps) x $\sqrt{2}$ = 32.7 A

Locked rotor testing

The locked rotor test, like short circuit test on a transformer, provides the information about leakage impedances and rotor resistance. Rotor is at the stand still, while low voltage is applied to stator windings up to rated current. Due to the fact that the magnetizing inductance L_m is much higher that leakage phase inductances L_{ls}, L_{lr}, it can be assumed that there is no current is floating to L_m parallel branch. Typically, leakage inductances L_{ls}, L_{lr} should be around 2-10% of the magnetizing inductance L_m. Since there is no rotation slip (rotor at standstill) s=1, which gives us the following equivalent circuit.

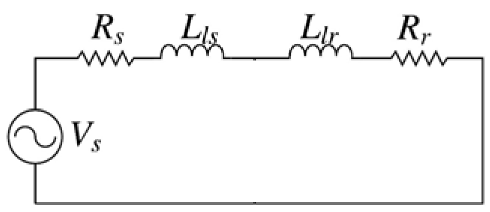


FIGURE 9-13. Electrical Representation of ACIM during locked rotor testing

Therefore, the phase L_{ls} (H) , L_{lr} (H), R_s (Ω), R_r (Ω) motor parameters are calculated as follows:

$$\cos(\varphi) = \frac{P_s}{V_s \times I} \quad (9.3)$$

$$Z = \frac{V_s}{I} \quad (9.4)$$

$$R_s + R_r = Z \times \cos\varphi \quad (9.5)$$

$$2\pi f_s \left(L_{ls} + L_{lr}\right) = Z \times \sin\varphi \quad (9.6)$$

$$R_s = R_r \quad (9.7)$$

$$L_{ls} = L_{lr} \quad (9.8)$$

where P_s (W) is the input phase motor power, V_s (V) is the phase stator voltage applied (peak value), f_s (Hz) is the stator frequency, cosφ is the power factor, I (A) is the motor current (RMS value), Z (Ω) is the equivalent phase impedance at locked rotor test.

According to equations (9.7) and (9.8), it is assumed that rotor resistance is equal to the stator resistance, as well as rotor leakage inductance is equal to stator leakage inductance.

In order to implement the above mentioned locked rotor test with induction motor controllers, please follow these steps:

- Set the operating mode to "Volts per Hertz" mode. Encoder feedback action needed.

- Configure the Volt per Hertz setting five times lower than the motor's nominal voltage (peak value of stator voltage according to manufacturer) and frequency (1/5 of nominal V/f). The reason for configuring lower V/f ratio than in no-load test is to appropriately weaken the induced field and reduce the produced torque at startup, in order to be easier to lock the rotor for the test.
- Lock the rotor by appropriate tool/device and increase the command up reaching the 80% of the rated motor current. If the produced torque is high and the rotor cannot be locked, then reduce appropriately the Volt per Hertz at configuration and repeat the test.
- Calculate the applied V_s by utilizing equation (9.2).
- Calculate the input phase motor power P_s by utilizing the following equation:

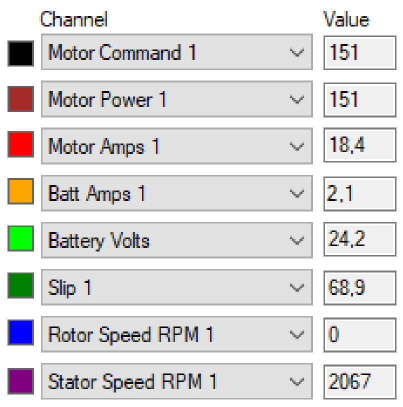
$$P_s = \frac{V_{dc} \times I_{dc} \times \eta}{3} \tag{9.9}$$

where V_{dc} (V) is the battery dc volts, I_{dc} (A) is the battery dc current, η is the controller efficiency (assume 0.95 efficiency for RoboteQ controllers). Battery volts and amps can be measured from Roborun+ utility.

- Measure the I_q (A) current from the Roborun+ utility (FOC Torque Amps).
- Calculate the L_{ls}, L_{lr} , R_s, R_r motor parameters by applying the equations (9.3) - (9.8).

Example:

For the same induction motor at no load test example, the Volts per Hertz ratio is set 5 times lower than the nominal, that is 0.053. The respective results taken from Roborun+ utility are shown below:



	Channel	Value
⬛	Motor Command 1	151
🟫	Motor Power 1	151
🟥	Motor Amps 1	18,4
🟧	Batt Amps 1	2,1
🟩	Battery Volts	24,2
🟩	Slip 1	68,9
🟦	Rotor Speed RPM 1	0
🟪	Stator Speed RPM 1	2067

FIGURE 9-14. Locked rotor testing monitoring

Therefore, the input phase motor power P_s is equal to

$$P_s = \frac{V_{dc} \times I_{dc} \times \eta}{3} = \frac{24.2 \times 2.1 \times 0.95}{3} = 16.1W$$

The power factor is $\cos(\varphi) = \frac{P_s}{V_s \times I} = \frac{16.1}{\frac{151 \times 24.2}{1000\sqrt{2} \times \sqrt{3}} \times 18.4} = 0.59$, while the equivalent phase impedance is Z = 0.081 Ω according to equation (9.4).

Therefore, the motor stator and rotor resistances are $R_s = R_r = 24\ m\Omega$ according to equations (9.5) and (9.7) and the leakage inductances according to equation (9.6) are:

$$\left(L_{ls}+L_{lr}\right)=\frac{Z\times\sin\varphi}{2\pi f_s}=\frac{0.081\mathrm{x}0.81}{2\times\pi\times\frac{2067\times2}{60}}=152\mathrm{uH}$$

It is noted that the synchronous frequency in locked rotor test is different than the synchronous frequency in no-load test. Finally, the $L_{ls} = L_{lr} = 76$ uH according to equation (9.8).

Optimal slip calculation

After estimating L_m, L_{ls}, L_{lr} , R_s, R_r motor parameters from no-load and locked-rotor tests, the optimal slip where the motor produces the maximum torque s_{maxT} can be estimated as follows:

$$s_{maxT}\approx\frac{R_r}{2\pi f_s L_m} \tag{9.10}$$

Next, in order to apply at configuration tab the optimal slip in Hz, the following transformation needed:

$$s=s_{maxT}\times f_s \tag{9.11}$$

Example:

For the calculated $L_m = 754$ uH, motor leakage inductances $L_{ls} = L_{lr} = 76$ uH and resistance $R_s = R_r = 24\ m\Omega$, the optimal slip is calculated equal to s = 5.1 Hz at rated motor speed.

After making the following simplification out of (9.10) and (9.11):

$$s=\frac{R_r}{2\cdot\pi\cdot f_s\cdot L_m}\cdot f_s \Rightarrow s=\frac{R_r}{2\cdot\pi\cdot L_m}$$

SECTION 10

Closed Loop Speed and Speed-Position Modes

This section discusses the controller's Closed Loop Speed modes.

Modes Description

Close loop speed modes ensure that the motor(s) will run at a precisely desired speed. If the speed changes because of changes in load, the controller automatically compensates the output voltage so that the motor maintains a constant speed. Two closed loop speed modes are available:

Closed Loop Speed Mode

This mode is the traditional closed loop technique where speed is measured with a speed sensor. The speed is compared to the desired speed and the speed PID control loop output provides the reference current (if FOC torque gains are set) or voltage commands (if the FOC torque gains are zero), in order to reach and maintain that speed. The internal current control is described in the Field Oriented Control (FOC) operation chapter.

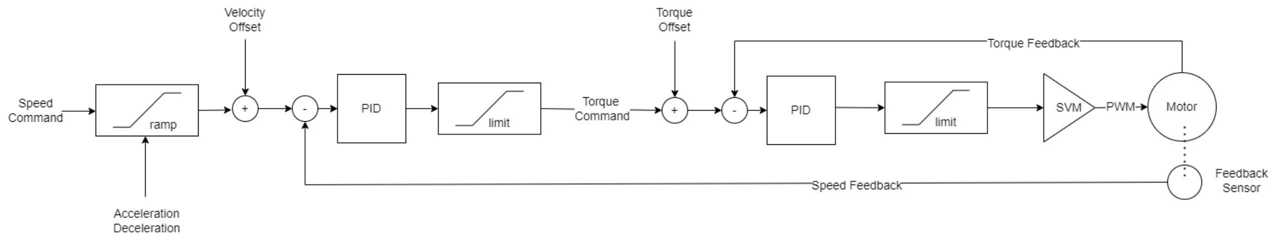


FIGURE 10-1. Closed Loop Speed Mode

Closed Loop Speed Position Control

In this mode, the controller computes the position at which the motor must be at every 1ms. Then a PID compares that expected position with the current position and applies the necessary reference speed (if speed and FOC torque gains are set) or voltage (if the speed and FOC torque gains are zero) command in order for the motor to reach that position. This mode is especially effective for accurate control at very slow speeds.

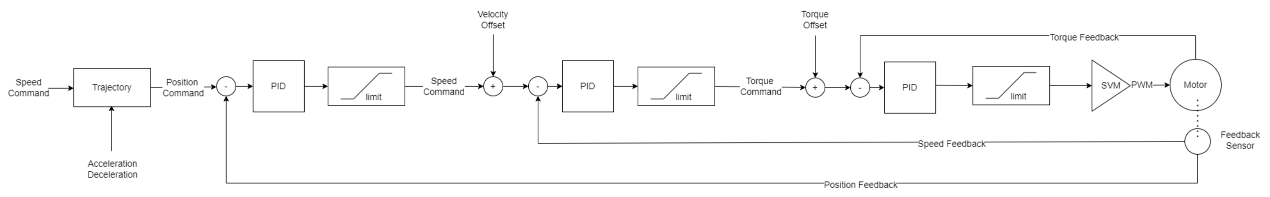


FIGURE 10-2. Closed Loop Speed Position Mode

The controller incorporates a full-featured Proportional, Integral, Derivative (PID) control algorithm for quick and stable speed control.

The closed loop speed mode and all its tuning parameters may be selected individually for each motor channel.

Motor Sensors

The controller may be used with the following kinds of sensors:

- Analog Tachometers
- Either of the supported sensors'

Digital Optical Encoders may be used to capture accurate motor speed. Analog tachometers are another technique for sensing speed. See "Connecting Tachometo Analog Inputs" on page 52.

Tachometer or Encoder Mounting

Proper mounting of the speed sensor is critical for an effective and accurate speed mode operation. Figure 10-1 shows a typical motor and tachometer or encoder assembly. It is always preferable to have the encoder connected to the motor shaft rather than at the output of a gearbox. If the encoder must be mounted after a gear box considers the effect of the gear backlash. A higher count encoder will typically be required to compensate for the lower rotation speed.

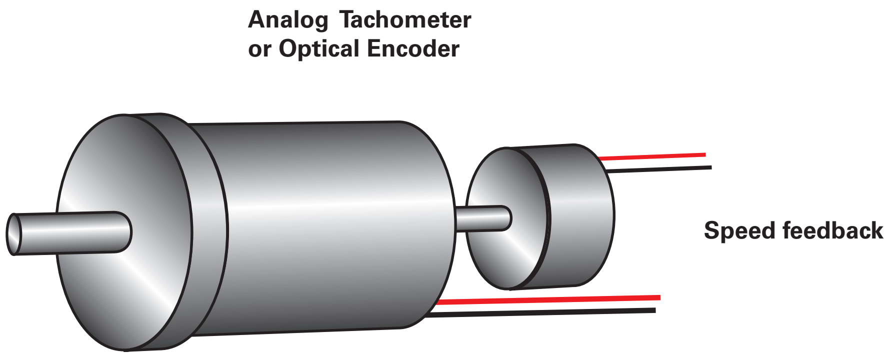


FIGURE 10-3. Motor and speed sensor assembly needed for Close Loop Speed mode

Tachometer wiring

The tachometer must be wired so that it creates a voltage at the controller's analog input that is proportional to rotation speed: 0V at full reverse, +5V at full forward, and 0 when stopped.

Connecting the tachometer to the controller is as simple as shown in the diagram below.

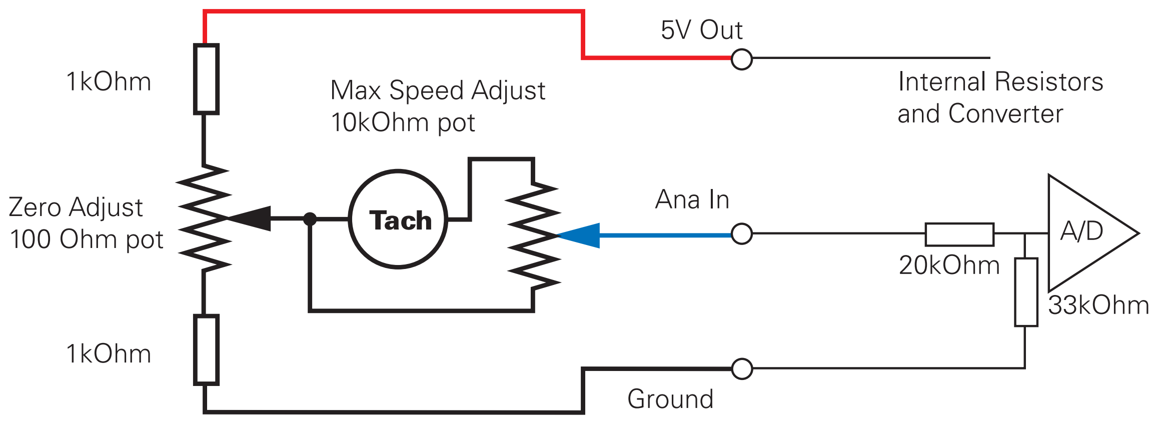


FIGURE 10-4. Tachometer wiring diagram

Hall Sensors as Speed Sensors

The Hall Sensors and most other types of rotor position sensors that are used to switch power around the motor windings, can also used to measure speed and distance traveled.

Speed is evaluated by measuring the time between the transition of the Hall Sensors. A 32 bit up/down counter is also updated at each Hall Sensor transition.

Speed information picked up from the Hall Sensors can be used for closed loop speed operation without any additional hardware on both sinusoidal and trapezoidal commutation. Likewise, the position counter that is updated at every Hall transition can also be used to operate the motor in Speed Position mode.

Speed Sensor and Motor Polarity

The sensor's polarity (i.e. which rotation direction produces positive or negative speed information) is related to the motor's rotation speed and the direction the motor turns when power is applied to it.

In the Closed Loop Speed mode, the controller compares the actual speed, as measured by the sensor, to the desired speed. If the motor is not at the desired speed and direction, the controller will apply power to the motor so that it turns faster or slower, until reached.

Important Warning

The speed sensor polarity must be such that a positive voltage is generated to the controller's input when the motor is rotating in the forward direction. If the polarity is inverted, this will cause the motor to run away to the maximum speed as soon as the controller is powered.

Determining the right polarity is best done during the motor sensor setup. The result can be validated experimentally using the Roborun utility (see "Roborun+ Utility User Manual") and following these steps:

1. Configure the controller in Open Loop Mode using the PC utility. This will cause the motor to run in Open Loop for now.
2. Configure the sensor you plan to use as speed feedback. If an analog tachometer is used, map the analog channel on which it is connected as "Feedback" for the selected motor channel. If an encoder is used, configure the encoder channel with the encoder's Pulses Per Revolution value. On the brushless motor, if the rotor sensor (Hall, Sin/Cos, ..) sensors are used, configure the correct number of motor pole pairs.
3. Click on the Run tab of the PC utility. Configure the Chart recorder to display the speed information if an encoder is used. Display Feedback if an analog sensor is used.
4. Verify that the motor sliders are in the "0" (Stop) position.
5. If a tachometer is used, verify that the reported feedback value read is 0 when the motors are stopped. If not, adjust the Analog Center parameter.
6. Move the cursor of the desired motor to the right so that the motor starts rotating, and verify that a positive speed is reported. Move the cursor to the left and verify that a negative speed is reported.
7. If the reported speed polarity is the same as the applied command, the wiring is correct.
8. If the tachometer polarity is opposite of the command. If an encoder is used, swap its ChA and ChB outputs. Alternatively, swap the motor leads if using a brushed DC motor only. The speed polarity can also be inverted by entering a negative number of encoder PPR. On brushless motors, entering a negative number of poles will invert the speed measured by the Hall, SinCos, or Resolver sensor. If using SSI sensor, the speed polarity can be inverted by entering a negative number of SSI sensor resolution.
9. Set the controller operating mode to Closed Loop Speed mode using the Roborun utility.
10. Move the cursor and verify that speed stabilizes at the desired value. If speed is unstable, tune the PID values.

Important Warning

It is critically important that the tachometer or encoder wiring be extremely robust. If the speed sensor reports an erroneous speed or no speed at all, the controller will consider that the motor has not reached the desired speed value and will gradually increase the applied power to the motor until the closed loop error is triggered and the motor is then stopped.

Controlling Speed in Closed Loop

When using either of the supported sensors' feedback, the controller will measure and report speed as the motor's actual RPM value.

When using analog or pulse as input command, the command value will range from 0 to +1000 and 0 to -1000. In order for the max command to cause the motor to reach the desired actual max RPM, an additional parameter must be entered in the configuration. The Max RPM parameter is the speed that will be reported as 1000 when reading the speed in relative mode. Max RPM is also the speed the controller will attempt to reach when a max command of 1000 is applied.

When sending a speed command via serial, network or scripting, the command may be sent as a relative speed (0 to +/-1000) or actual RPM value.

PID Description

The controller performs both Closed Loop Speed modes using a full featured Proportional, Integral and Derivative (PID) algorithm. This technique has a long history of usage in control systems and works on performing adjustments to the Power Output based on the difference measured between the desired speed or position (set by the user) and the actual speed or position (captured by the sensor on the motor).

Figure 9-3 shows a representation of the PID algorithm. Every 1 millisecond, the controller measures the actual motor speed or position and subtracts it from the desired speed or position to compute the error.

The resulting error value is then multiplied by a user selectable Proportional Gain. The resulting value becomes one of the components used to command the motor. The effect of this part of the algorithm is to regulate current (if torque FOC gains are set) or voltage (if torque FOC gains are zero) of the motor proportionally with the difference between the current and desired speed or position: when far apart, high power is applied, with the power being gradually reduced as the motor moves to the desired speed or destination.

A higher Proportional Gain will cause the algorithm to apply a higher level of power for a given measured error thus making the motor react more quickly to changes in commands and/or motor load.

The Derivative component of the algorithm computes the changes to the error from one 1 ms time period to the next. This change will be a relatively large number every time an abrupt change occurs on the desired speed value or the measured speed value. The value of that change is then multiplied by a user selectable Derivative Gain and added to the output. The effect of this part of the algorithm is to give a boost of extra power when starting the motor due to changes to the desired speed or position value. The Derivative component will also help dampen any overshoot and oscillation.

The Integral component of the algorithm performs a sum of the error over time. In Speed mode, this component helps the controller reach and maintain the exact desired speed when the error is reaching zero (i.e. measured speed is near to, or at the desired value). In Speed Position mode, the Integral parameter can help maintain a slightly tighter difference between the desired and actual position, but makes no significant difference and can be omitted altogether.

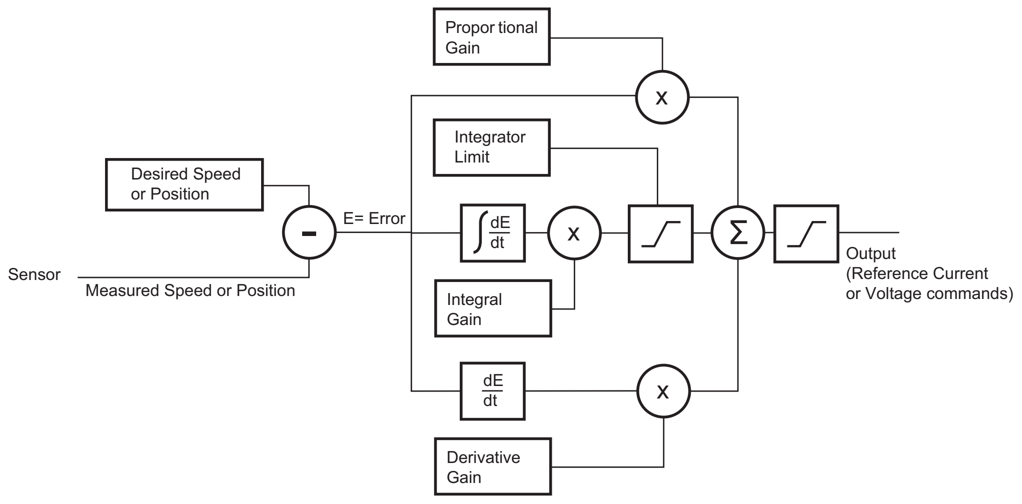

Propor tional Gain
Integrator Limit
Desired Speed or Position
E= Error
Sensor
Measured Speed or Position
dE/dt
Integral Gain
Derivative Gain
Output (Reference Current or Voltage commands)

FIGURE 10-5. PID algorithm used in close loop speed modes

PID tuning in Closed Loop Speed Mode

As discussed above, three parameters - Speed Proportional Gain, Speed Integral Gain, and Speed Derivative Gain - can be adjusted to tune the Closed Loop Speed control algorithm. The ultimate goal in a well tuned PID is a motor that reaches the desired speed quickly without overshoot or oscillation. The speed mode PI gains are essentially affected from electric motor and mechanical system characteristics, such as torque constant, inertia, rotating friction coefficient, load torque.

The speed loop gains can be automatically, as well as manually, regulated from the Motor Sensor and Tuning setup wizard supported in Roborun+ v3.0 utility (see Roborun+ Utility User Manual for more details).

The Roborun PC utility makes this experimentation easy by providing one screen for changing the Proportional, Integral and Derivative gains and another screen for running and monitoring the motor. First, configure the torque loop gains properly and run the motor with the preset values. Then experiment with different values until a satisfactory behavior is found.

In Speed Mode, the Integral component of the PID is of high importance, eliminating the loop error. The Proportional and Derivative components will help improve the response time and loop stability.

Try initially to only use a small value of P and I with no D:

P = 0 .1
I = 0.2
D = 0

These values practically always work, but they may cause the motor to be slowly reaching the desired speed. Increase the P gain to improve responsiveness and I gain to eliminate loop error, but keeping them below the level at which the motor begins to oscillate or perform overshoots/undershoots during speed transitions.

In the case where the load moved by the motor is not fixed, tune the PID with the minimum expected load and tune it again with the maximum expected load. Then try to find values that will work in both conditions. If the disparity between minimal and maximal possible loads is large, it may not be possible to find satisfactory tuning values. In this case, consider changing the PID gains on the fly during motor operation with serial/CAN commands of with a MicroBasic script.

In slow systems, use the integrator limit parameter to prevent the integrator to reach saturation prematurely and create overshoots. Beware to set speeds that can physically be reached by the motor under load. If the motor is not physically able, there will be a loop error, which if it becomes too large, will cause a fault to be detected and the motor to be stopped.

PID Tuning in Speed Position Mode

As discussed, in Closed Loop Speed Position mode, every millisecond, the controller computes a successive desired position. The PID then works to make the motor follow the computed trajectory. This mode works much better than the regular Closed Loop Speed mode when the motor must operate at very low speed. When the motor is stopped, it will maintain its position even if pulled, as for example on a robot stopped downhill.

The PID therefore must be tuned for position mode. In position mode (the Position Gains are used), most of the work is done by the proportional gain. It acts essentially as an imaginary rubber belt between the controller's internal destination counter and the motor: the higher the difference, the more the belt is stretched, and the stronger the motor will turn. Once the imaginary belt has stiffened the motor will run at the desired speed.

In speed position mode, the position loop P gain can be calculated through the following equation, utilizing the zero-pole cancelation method, considering sensor resolution also as the PI control is applied in absolute position counts:

$$K_p = 2\pi f_{BW} \times \frac{60}{\text{Sensor resolution}}$$

where:

K_p: Position loop P gain.

f_{BW}: Bandwidth of the position control loop (Hz)

Sensor resolution: The utilized sensor resolution for one full mechanical revolution. Below is the sensor resolution for each supported feedback type:

Encoder: Pulses per revolution x 4

SPI/SSI: Sensor counts resolution

Hall: Number of pole pairs x 6

Sin/Cos: 16384

Resolver: 16384

Example

For an encoder with 4096 pulses/rev and 1 Hz bandwidth selected the proportional position gain should be:

K_p = (2 x 3.141 x 1 x 60)/(4096 x 4) = 0.023

It is noted that the speed gains should be configured first, in order to enable the internal speed loop and operate in cascaded speed position mode. The speed loop gains can be automatically, as well as manually, regulated from the Motor Sensor and Tuning setup wizard supported in Roborun+ v3.0 utility (see Roborun+ Utility User Manual for more details).

Recommended initial bandwidth for position control loop is 0.5Hz, considering also that the typical speed loop range is 2-5 Hz.

With the controller configured in Speed Position mode and the motor stopped, do a first check of the PID's stiffness by attempting to rotate the motor by hand. It should feel increasingly hard to rotate away from the rest position. With a higher P gain, it will become harder to move than lower gains. As a rule of thumb, on a mobile robot, use a gain that makes it very hard to move the wheel more than a quarter turn away from the rest position. Test then by applying a speed command and verifying the motor runs smoothly under all load conditions.

The I and D gain can generally be omitted in Speed Position mode.

Beware to set speeds that can physically be reached by the motor under load. The Closed Loop Speed Position mode relies on the fact that the motor will actually be able to follow the computed trajectory. If the motor is not able, the controller will pause updating the destination counter until the motor caught up. This will result in inaccurate speed and can be a problem in mobile robot applications depending on precise control of their left and right side motor.

SECTION 11

Closed Loop Relative and Tracking Position Modes

This section describes the controller's Position Relative and Position Tracking modes, how to wire the motor and position sensor assembly and how to tune and operate the controller in these modes.

Modes Description

In these two position modes, the axle of a geared-down motor is coupled to a position sensor that is used to compare the angular position of the axle versus a desired position. The controller will move the motor so that it reaches this position.

This feature makes it possible to build ultra-high torque "jumbo servos" that can be used to drive steering columns, robotic arms, life-size models and other heavy loads.

The two position modes are similar and differ as follows:

Position Relative Mode

The controller accepts a command ranging from -1000 to +1000, from serial/USB, analog joystick, pulse input or Network command. The controller reads a position feedback sensor and converts the signal into a -1000 to +1000 feedback value at the sensor's min and max range respectively. The controller then moves the motor so that the feedback matches the command, using a controlled acceleration, set velocity, and controlled deceleration. This mode requires several settings to be configured properly but results in very smoothly controlled motion.

Position Tracking Mode

This mode is identical to the Position Relative mode in the way that commands and feedback are evaluated. However, the controller will move the motor simply using a PID comparing the command and feedback, without controlled acceleration and as fast as possible. This mode requires fewer settings but often results in a motion that is not as smooth and harder to control overshoots.

Selecting the Position Modes

The two position modes are selected by changing the Motor Control parameter to Closed Loop Position. This can be done using the corresponding menu in the Power Output tree in the Roborun utility. It can also be done using the associated serial (RS232/RS485/TCP/USB) command. See "MMOD - Operating Mode" on page 358. The position mode can be set independently for each channel.

Position Feedback Sensor Selection

The controller may be used with the following kinds of sensors:

- Potentiometers
- Hall effect angular sensors
- Supported Sensors (Hall sensors, Optical Encoders, SSI, SinCos, Resolver).

The first two are used to generate an analog voltage ranging from 0V to 5V depending on their position. They will report an absolute position information at all times.

Modern position Hall sensors output a digital pulse of the variable duty cycle. These sensors provide an absolute position value with high precision (up to 12-bit) and excellent noise immunity. PWM output sensors are directly readable by the controller and therefore are a recommended choice.

Optical encoders report incremental changes from a reference which is their initial position when the controller is powered up or reset. Before they can be used for reporting position, the motors must be moved in open loop mode until a home switch is detected and resets the counter. Encoders offer the greatest positional accuracy possible.

Sensor Mounting

Proper mounting of the sensor is critical for an effective and accurate position mode operation. Figure 11-1 shows a typical motor, gear box, and sensor assembly.

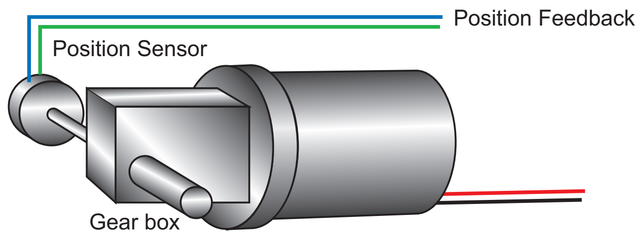


FIGURE 11-1. Typical motor/Potentiometer/assembly in Position Mode

The sensor is composed of two parts:

- a body which must be physically attached to a non-moving part of the motor assembly or the robot chassis, and
- an axle which must be physically connected to the rotating part of the motor you wish to position.

A gear box is necessary to greatly increase the torque of the assembly. It is also necessary to slow down the motion so that the controller has the time to perform the position control algorithm. If the gearing ratio is too high, however, the positioning mode will be very sluggish.

A good ratio should be such that the output shaft rotates at 1 to 10 rotations per second (60 to 600 RPM) when the motor is at full speed.

The mechanical coupling between the motor and the sensor must be as tight as possible. If the gear box is loose, the positioning will not be accurate and will be unstable, potentially causing the motor to oscillate.

Some sensors, such as potentiometers, have a limited rotation range of typically 270 degrees (3/4 of a turn), which will in turn limit the mechanical motion of the motor/potentiometer assembly. Consider using a multi-turn potentiometer as long as it is mounted in a manner that will allow it to turn throughout much of its range, when the mechanical assembly travels from the minimum to maximum position. When using encoders, best results are achieved when the encoder is mounted directly on the motor shaft.

Feedback Sensor Range Setting

Regardless of the type of sensor used, feedback sensor range is scaled to a -1000 to +1000 value so that it can be compared with the -1000 to +1000 command range.

On analog and pulse sensors, the scaling is done using the min/max/center configuration parameters.

When encoders are used for feedback, the encoder count is also converted into a -1000 to +1000 range. In the encoder case, the scaling uses the Encoder min and max limit parameters. See "Serial (RS232/ RS485/USB/TCP) Operation" in Section 14 for details on these configuration parameters. Beware that encoder counters produce incremental values. The encoder counters must be set using a homing procedure before they can be used as position feedback sensors. A typical homing process involves switching the operating mode to Open Loop or Closed Loop Speed and moving the motor in one direction until the homing condition is met (for example, detecting the encoder's homing pulse). Afterward, the encoder counter should be set to the desired value. For more information on accessing the encoder counter, refer to the 'C' runtime command.

Important Notice

Potentiometers are mechanical devices subject to wear. Use better quality potentiometers and make sure that they are protected from the elements. Consider using a solid state hall position sensor in the most critical applications. Optical encoders may also be used, but require a homing procedure to be used in order to determine the zero position.

Important Warning

If there is a polarity mismatch, the motor will turn in the wrong direction and the position will never be reached. The motor will turn until the Closed Loop Error detection is triggered. The motor will then stop until the error disappears, the controller is set to Open Loop, or the controller is reset.

Determining the right polarity is best done during the motor sensor setup. The result can be validated experimentally using the Roborun utility (see "Roborun+ Utility User Manual") and following these steps:

1. Configure the controller in Open Loop Speed mode.
2. Configure the position sensor input channel as position feedback for the desired motor channel.
3. Click on the Run tab.
4. Enable the Feedback channel in the chart recorder.
5. Move the slider slowly in the positive direction and verify that the Feedback in the chart increases in value. If the Feedback value decreases, then the sensor is backward and you should either invert using configuration commands, invert the sensor physically, it or swap the motor wires so that the motor turns in the opposite direction.
6. Move the sensor off the center position and observe the motor's direction of rotation.
7. Go to the max position and verify that the feedback value reaches 1000 a little before the end of the physical travel. Modify the min and max limits for the sensor input if needed.
8. Repeat the steps in the opposite direction and verify that the -1000 is reached a little before the end of the physical travel limit.

Important Safety Warning

Never apply a command that is lower than the sensor's minimum output value or higher than the sensor's maximum output value as the motor would turn forever trying to reach a position it cannot. Configure the Min/Max parameter for the sensor input so that a value of -1000 to +1000 is produced at both ends of the sensor travel.

Adding Safety Limit Switches

The Position mode depends on the position sensor providing accurate position information. If the sensor is damaged or one of its wires is cut, the motor may spin continuously in an attempt to reach a fictitious position. In many applications, this may lead to serious mechanical damage.

To limit the risk of such breakage, it is recommended to add limit switches that will cause the motor to stop if unsafe positions have been reached independently of the sensor reading. Any of the controller's digital inputs can be used as a limit switch for any motor channel.

Using Current Trigger as Protection

The controller can be configured to trigger an action when the current reaches a user configurable threshold for more than a set amount of time. This feature can be used to detect that a motor has reached a mechanical stop and is no longer turning. The triggered action can be an emergency stop or a simulated limit switch.

Operating in Closed Loop Relative Position Mode

This position algorithm allows you to move the motor from an initial position to the desired position. The motor starts with a controlled acceleration, reaches the desired velocity, and decelerates at a controlled rate to stop precisely at the end position. The graph below shows the speed and position vs. time during a position move.

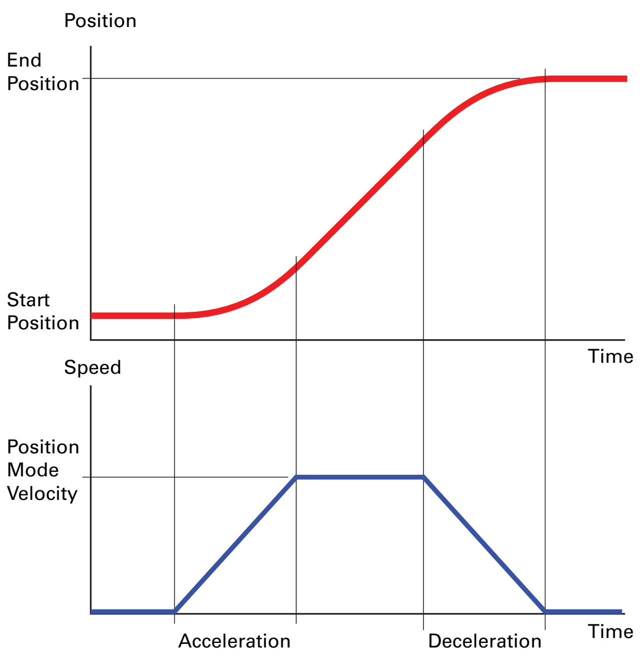


FIGURE 11-2. Position Mode Functionality

When turning the controller on, the default acceleration, deceleration and velocity are parameters retrieved from the configuration EEPROM. In most applications, these parameters can be left unchanged and only change in commands used to control the change from one position to the other. In more sophisticated systems, the acceleration, deceleration and velocity can be changed on the fly using Serial/USB commands or from within a MicroBasic script.

When using Encoders as feedback sensors, the controller can accurately measure the speed and the number of motor turns that have been performed at any point in time. The complete positioning algorithm can be performed with the parameters described above.

When using analog or pulse sensors as feedback, the system does not have a direct way to measure speed or number of turns. It is therefore necessary to configure an additional parameter in the controller which determines the number of motor turns between the point the feedback sensor gives the minimum feedback value (-1000) to the maximum feedback value (+1000).

In the Closed Loop Relative Position mode, the controller will compute the position at which the motor is expected to be at every millisecond in order to follow the desired acceleration and velocity profile. This computed position becomes the setpoint that is compared with the feedback sensor and a correction is applied at every millisecond.

For troubleshooting, the computed position can be monitored in real time by enabling the Tracking channel in the PC utility's chart recorder.

Beware not to use accelerations and max velocity that are beyond the motor's physical reach at full load. This would result in a loop error which will stop the system if growing too large.

Operating in Closed Loop Tracking Mode

In this mode, the controller makes no effort to compute a smooth, millisecond by millisecond position trajectory. Instead, the current feedback position is periodically compared with the requested destination and power is applied to the motor using these two values in a PID control loop.

This mode will work best if changes in the commands are smooth and not much faster than what the motor can physically follow.

Position Mode Relative Control Loop Description

The controller performs the Relative Position mode using a full featured Proportional, Integral and Derivative (PID) algorithm (using the Position Gains). This technique has a long history of usage in control systems and works on performing adjustments to the speed command based on the difference measured between the desired position (set by the user) and the actual position (captured by the position sensor).

Figure 10-4 shows a representation of the PID algorithm. Every 1 millisecond, the controller measures the actual motor position and subtracts it from the desired position to compute the position error.

The resulting error value is then multiplied by a user selectable Proportional Gain. The resulting value becomes one of the components used to command the motor. The effect of this part of the algorithm is to calculate the speed command to the motor control that is proportional with the distance between the current and desired positions: when far apart, high speed command is generated, with the power being gradually reduced and stopped as the motor moves to the final position. The Proportional feedback is the most important component of the PID in Position mode.

If the PID of the speed PI controller is not configured, then the output of the position PID corresponds to the applied output voltage (power). In this case, it is highly recommended to configure only the flux FOC gains and set the torque FOC gains to zero.

A higher Proportional Gain will cause the algorithm to apply a higher level of speed/voltage commands for a given measured error, thus making the motor move quicker. Because of inertia, however, a faster moving motor will have more difficulty stopping when it reaches its desired position. It will therefore overshoot and possibly oscillate around that end position.

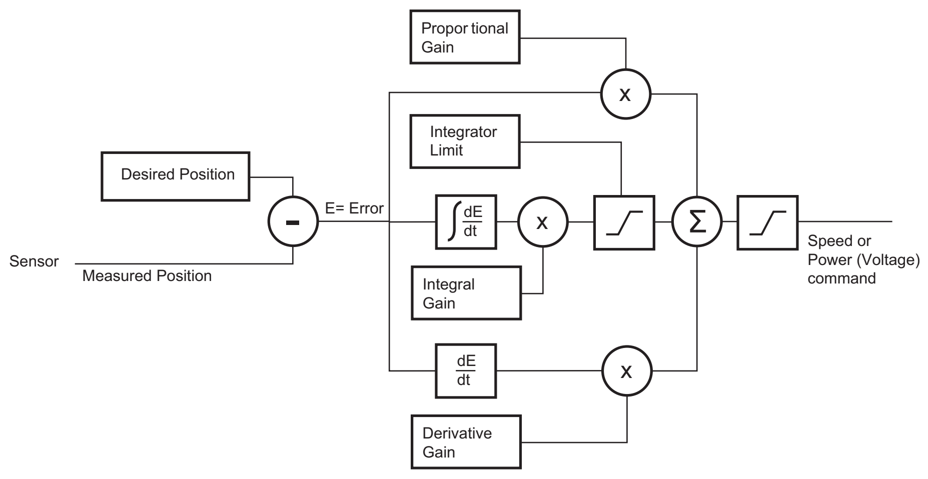


FIGURE 11-3. PID algorithm used in Position Mode

The Derivative component of the algorithm computes the changes to the error from one ms time period to the next. This change will be a relatively large number every time an abrupt change occurs on the desired position value or the measured position value. The value of that change is then multiplied by a user-selectable Derivative Gain and added to the output. The effect of this part of the algorithm is to give a boost of extra power when starting the motor due to changes to the desired position value. The Derivative component will also help dampen any overshoot and oscillation.

The Integral component of the algorithm performs a sum of the error over time. In the position mode, this component helps the controller reach and maintain the exact desired position when the error would otherwise be too small to energize the motor using the Proportional component alone. Only a very small amount of Integral Gain is typically required in this mode.

In systems where the motor may take a long time to physically move to the desired position, the integrator value may increase significantly causing then difficulties to stop without overshoot. The Integrator Limit parameter will prevent that value from becoming unnecessarily large.

PID tuning in Position Relative and Tracking Position Modes

As discussed above, three parameters - Position Proportional Gain, Position Integral Gain and Position Derivative Gain - can be adjusted to tune the position control algorithm. The ultimate goal in a well tuned PID is a motor that reaches the desired position quickly without overshoot or oscillation.

In closed loop position relative and tracking position modes, the position loop P gain can be calculated through the following equation, utilizing the zero-pole cancelation method, considering sensor resolution, as the PI control is applied in absolute position counts and position feedback normalization process from -1000 to +1000:

$$K_p = 2\pi f_{BW} \times \frac{60}{\text{Sensor resolution}} \times \frac{\text{MaxLimit} - \text{MinLimit}}{2000}$$

where:

K_p : Position loop P gain.

f_{BW} : Bandwidth of the position control loop (Hz)

Sensor resolution: The utilized sensor resolution for one full mechanical revolution. Below is the sensor resolution for each supported feedback type:

Encoder: Pulses per revolution x 4

SPI/SSI: Sensor counts resolution

Hall: Number of pole pairs x 6

Sin/Cos: 16384

Resolver: 16384

Example

For an encoder with 4096 pulses/rev, 20000 counts max limit, -20000 counts min limit and 1 Hz bandwidth selected the proportional position gain should be:

K_p = (2 x 3.141 x 1 x 60)/(4096 x 4) x (20000 + 20000)/2000 = 0.46

It is noted that the speed gains should be configured first, in order to enable the internal speed loop and operate in cascaded speed position mode. The speed loop gains can be automatically, as well as manually, regulated from the Motor Sensor and Tuning setup wizard supported in Roborun+ v3.0 utility (see Roborun+ Utility User Manual for more details).

Recommended initial bandwidth for position control loop is 0.5Hz, considering also that the typical speed loop range is 2-5 Hz.

Because many mechanical parameters such as motor power, gear ratio, load and inertia are difficult to model, tuning the PID needs also a manual validation process that takes experimentation.

The Roborun PC utility makes this experimentation easy by providing one screen for changing the Proportional, Integral and Derivative gains and another screen for running and monitoring the motor.

When tuning the motor, first start with the Integral and Derivative Gains at zero, increasing the Proportional Gain until the motor overshoots and oscillates. Typically the Integral and Derivative gains should be equal to zero, especially when utilizing cascaded position mode including the speed and torque control loops.

To set the Proportional Gain, which is the most important parameter, use the Roborun utility to observe the three following values:

- Command Value
- Actual Position
- Actual Speed

With the Integral Gain set to 0, the PID output should be:

PID Output = (Command Value - Actual Position) * Proportional Gain

The PID output can be the speed command or the applied power (voltage), depending on whether the speed gains are set or zeroed respectively. Experiment first with the motor electrically or mechanically disconnected and verify that the controller is measuring the correct position and is applying the expected amount of power (voltage) to the motor depending on the command given.

Verify that when the Command Value equals the Actual Position, the Applied Power (Voltage) equals to zero. Note that the Applied Power (Voltage) value is shown without the sign in the PC utility.

In the case where the load moved by the motor is not fixed, the PID must be tuned with the minimum expected load and tuned again with the maximum expected load. Then try to find values that will work in both conditions. If the disparity between minimal and maximal possible loads is large, it may not be possible to find satisfactory tuning values.

PID Tuning Differences between Position Relative and Position Tracking

The PID works the same way in both modes in that the desired position is compared to the actually measured position.

In the Closed Loop Relative mode, the desired position is updated every ms and so the PID deal with small differences between the two values.

In the Closed Loop Tracking mode, the desired position is changed whenever the command is changed by the user according to the acceleration and deceleration values configured.

Tuning for both modes requires the same steps. However, the position loop bandwidth expected may be different from one mode to the other.

SECTION 12

Closed Loop Count Position Mode

In the Closed Loop Position mode, the controller can move a motor a precise number of counts, using a predefined acceleration, constant velocity, and deceleration. This mode requires that an encoder be mounted on the motor.

Mode description

The desired position is given in the number of counts. Using acceleration, deceleration and top velocity, the controller computes the position at which the motor is expected to be at every one millisecond interval. A PID then computes the speed to give to the motor in order to maintain that position. A comparator looks at the desired position and the computed current position and issues a Destination Reached flag. The figure below shows a representation of this mode.

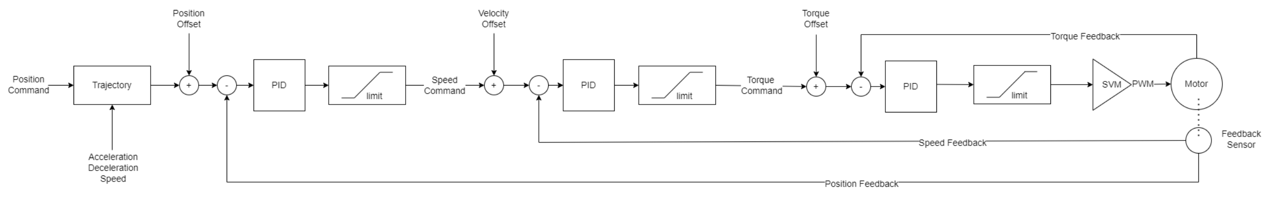


FIGURE 12-1 Closed Loop Position mode

Sensor Types and Mounting

In position mode, best results are achieved with encoders directly mounted on the motor shaft.

It is not advised to mount encoders at the output of a geared motor as the gear box often introduces backlash. If the encoder must be mounted at the output, then it must typically have a higher count to compensate the lower speed rotation at that location.

Quadrature encoders typically provide the highest resolution since they can be ordered with a line resolution of several hundred or thousands of counts per revolution. Hall encoders built in brushless motors give a relatively low, but often adequate count of 6* NumberOfPolePairs per mechanical resolution. SSI digital sensors give a pole resolution equal to their SSI sensor resolution. Other brushless rotor sensors, such as sin/cos or resolver sensors will give up to 512 counts per pole and can therefore be used instead of encoders.

Encoder Home reference

Beware that encoders do not give an absolute position information. It is therefore necessary to perform a search of the zero reference position at least once after every power up. This is typically achieved by moving the motor up to a limit switch and loading the counter with a fixed value at that location. A home search sequence can easily be implemented using a MicroBasic script. The search and counter loading must be done while the motor is operated in an open loop.

SSI Sensor Home reference

When SSI sensors are used as relative encoders, it is as well necessary to perform a search of the zero reference position, as stated above. When SSI sensors are used as absolute encoders (Absolute Feedback), Home reference is used as an offset to the SSI Counter.

Important Warning

Changing the counter with value while the motor is operated in a closed loop can cause violent and dangerous jumps. Always revert to open loop to change the counter value.

Preparing and Switching to Closed Loop

To enter this mode you will first need to configure the encoder so that it is used as feedback for motor1, and feedback for motor2 on the other encoder in a dual motor system. On brushless motors, selecting "Other" in Closed Loop Feedback Sensor, the encoder or the SSI sensor, if either present and properly configured, will be used as feedback. Selecting "Internal Sensor", Hall, sin/cos or resolver sensor, depending on which is configured, will be used as feedback.

Use the PC Utility to set the default acceleration, deceleration, position mode velocity as long as maximum and minimum speed in the motor menu. These values can then be changed on the fly if needed.

While in Open Loop, enable the Speed channel in the Roborun Chart Recorder. Move the slider in the positive direction and verify that the measured speed polarity is also positive. If a negative speed is reported, swap the two encoder wires to change the measured polarity, or swap the motor leads to make the motor spin in the opposite direction.

Then use the PC Utility to select the Closed Loop Position Mode. After saving to the controller, the motor will operate in Closed Loop and will attempt to go to the 0 counter position. Beware therefore that the motor has not already turned before switching to Closed Loop. Reset the counter if needed prior to closing the loop.

Count Position Commands

Moving the motor is done using a set of simple commands.

To go to an absolute encoder position value, use the !P command

To go to a relative encoder position count that is relative to the current position, use the !PR command.

The Acceleration, Deceleration and Velocity are fixed parameters that can be changed using the ^MAC, ^MDEC and ^MVEL configuration settings. These can also be changed on the fly, any time using the !AC and !DC commands.

The velocity can also be changed at any time using the !S command:

New position destination command can be issued at any time. If the previous destination is not reached while the new is sent, the motor will move to the new destination. If this causes a change of direction, the motor will do the change using the current acceleration and deceleration settings. See the Commands Reference section for details on all these commands

Position Command Chaining

It is possible to chain position commands in order to create seamless motion to a new position after an initial position is reached. To do this, the controller can store the next goto position with, optionally, a new set of acceleration, deceleration and velocity values.

The commands that set the "next" move are identical to these discussed in the previous section, with the addition of an "X" at the end. The full command list is:

!PX nn mm	Next position absolute
!PRX nn mm	Next position relative
!ACX nn mm	Next acceleration
!DCX nn mm	Next deceleration
!SX	Next velocity
Example:	!PX 1 -50000 will cause the motor to move to that new destination once the previous destination is reached. !PRX -10000 will cause the motor to move 10000 count back from the previous end destination. If the next acceleration, next deceleration or next velocity are not entered, the value(s) used for the previous motion will be used.

Beware that the next commands must be entered while the motor is moving, since the next commands will only be taken into account at the end of the current motion.

To chain more than two commands, use a MicroBasic script or an external program to load new "next" command when the previous "next" commands become active. The ?DR query can be used to detect that this transition has occurred and that a new next command can be sent to the controller.

The chart below shows a typical chaining flow.

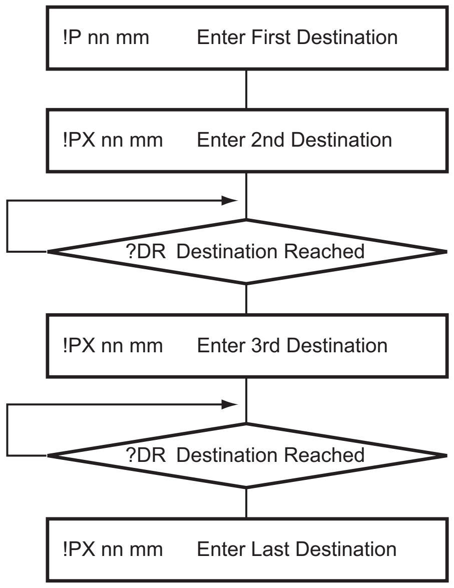

!P nn mm Enter First Destination
!PX nn mm Enter 2nd Destination
?DR Destination Reached
!PX nn mm Enter 3rd Destination
?DR Destination Reached
!PX nn mm Enter Last Destination

FIGURE 12-2 Command Chaining flow

Position Accuracy Considerations

In the position mode, the controller computes a trajectory that the motor then attempts to follow using a PID. For this technique to work well, the motor must first be physically able to run as fast as dictated by the trajectory calculation. If not, a loop error (ie desired position - actual position) will accumulate and eventually grow to trigger an error that will stop the motor. Make sure that the velocity setting is always under the max speed that can be reached by the motor while running at full load, in open loop.

Some difference between the desired and actual position, i.e. a loop error, is always to be expected when using a PID. The PID gains must be tuned to minimize the loop error while keeping smooth motion. The expected position and loop error can be monitored in real time using the PC utility's Tracking and Loop Error channels, respectively, in the chart recorder.

Beware that the Destination Reached flag will become true when the result of the trajectory computation equals the desired destination. In most practical situations, the motor will still be on its way to actually reach that destination. This can be an important consideration when chaining commands, as the new command will become active before the motor has actually reached the previous destination

PID Tuning in Count Position Mode

As for any position control loop, the dominant PID parameter is the Proportional gain, with typically no Integral and Derivative gain. The tuning process for count position control is similar to the position relative and tracking position control PID tuning described in page 183 , except from the Proportional gain equation due to the absolute position feedback utilized in count position mode.

Therefore, in closed loop count position mode, the position loop P gain can be calculated through the following equation, utilizing the zero-pole cancelation method, considering sensor resolution, as the PI control is applied in absolute position counts:

$$K_p = 2\pi f_{BW} \times \frac{60}{\text{Sensor resolution}}$$

where:

K_p : Position loop P gain.

f_{BW} : Bandwidth of the position control loop (Hz)

Sensor resolution: The utilized sensor resolution for one full mechanical revolution. Below is the sensor resolution for each supported feedback type:

Encoder: Pulses per revolution x 4

SPI/SSI: Sensor counts resolution

Hall: Number of pole pairs x 6

Sin/Cos: 16384

Resolver: 16384

<u>Example</u>

For an encoder with 4096 pulses/rev and 1 Hz bandwidth selected the proportional position gain should be:

K_p = (2 x 3.141 x 1 x 60)/(4096 x 4) = 0.023

It is noted that the speed gains should be configured first, in order to enable the internal speed loop and operate in cascaded speed position mode. The speed loop gains can be automatically, as well as manually, regulated from the Motor Sensor and Tuning setup wizard supported in Roborun+ v3.0 utility (see Roborun+ Utility User Manual for more details).

Recommended initial bandwidth for position control loop is 0.5Hz, considering also that the typical speed loop range is 2-5 Hz.

SECTION 13

Closed Loop Torque Mode

This section describes the controller's operation in Torque Mode.

Torque Mode Description

The torque mode is a special case of closed loop operation where the motor command controls the current that flows through the motor regardless of the motor's actual speed.

In an electric motor, the torque is directly related to the current. Therefore, controlling the current controls the torque.

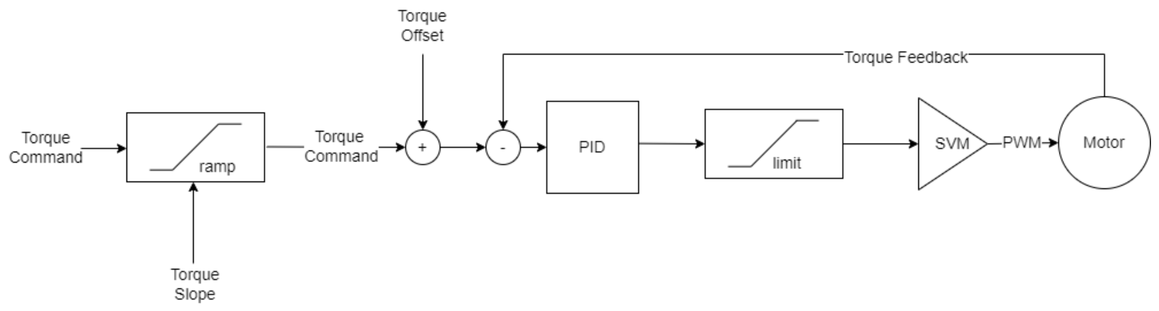


FIGURE 13-1. Torque mode

Torque mode is mostly used in electric vehicles since applying a higher command gives more "push", similarly to how a gas engine would respond to stepping on a pedal. Likewise, releasing the throttle will cause the controller to adjust the power output so that the zero amps flow through the motor. In this case, the motor will coast and it will take a negative command (i.e. negative amps) to brake the motor to a full stop. When the sinusoidal mode is selected, the field oriented current control is utilized, in order to regulate the Iq (Torque Amps) and Id (Flux Amps) at the reference values for each kind of motor selected (brushless dc, IPM, Induction).

Torque Mode Selection, Configuration and Operation

Use the PC utility and the menu "Operating Mode" to select Torque Mode. The controller will now use user commands from RS232, RS485, TCP, USB, Network, Analog or Pulse to command the motor current.

Torque commands can be given:

- **G** command, with range -1000 to +1000. The command for brushed controllers is then scaled using the amps limit configuration value. For example, if the amps limit is set to 100A, a user command of 500 will cause the controller to energize the motor until 50A are measured. For brushless controllers and in sinusoidal mode, the G command is scaled based on the torque (quadrature) amps limit. If flux amps are always at 0 (no field weakening) then the scaling is the same.
- **TC** command, which is similar to G command but the command is given per thousand of rated torque. The rated torque is derived out of the nominal current (NOMA) and torque constant (TNM).
- **GIQ** command (applicable only for brushless motor controller and sinusoidal mode), with which one can give the Torque (quadrature) amps command in Amps*10.
- **GID** command (applicable only for brushless motor controller and sinusoidal mode), with which one can give the Flux amps command in Amps*10.

Torque Mode Tuning

In Torque Mode, the measured Motor Amps (or Flux and Torque Amps in sinusoidal mode) become the feedback in the closed loop system. The PID then operates the same way as in the other Closed Loop modes described in this manual (See "PID tuning in Closed Loop Speed Mode" on page 172).

In sinusoidal mode, torque mode uses the PID that is regulating the Field Oriented Control, which are the same with Closed Loop Torque PID parameters. See also the KIF and KPF configuration commands or CIG and CPG runtime commands, respectively, in the Commands Reference section. The Field Oriented Control PI parameters tuning process is described in "FOC Gains Determination & Tuning" chapter in page 130 of the manual.

Speed Limiting

All Controllers provide a way of smoothly limiting the speed in torque mode to prevent motor runaways. The method for limiting the speed is based on PID speed over-ride control which provides very smooth motor output but requires PID tuning.

Therefore for the torque loop we use the Torque Proportional Gain (KPF) and the Torque Integral Gain (KIF), in Closed Loop Torque parameters, and the speed limit is tuned using the Speed Proportional Gain (KPG), the Speed Integral Gain (KIG) and the Speed Derivative Gain (KDG), in Closed Loop Speed parameters. The speed loop PID tuning can either be done in "Closed Loop Torque Mode" at the speed limit or in " Closed Loop Speed Mode" by looking at the response time.

SECTION 14

Serial (RS232/ RS485/USB/TCP) Operation

This section describes how to communicate to the controller via the RS232, RS485, USB or TCP Interface. This information is useful if you plan to write your own controlling software on a PC or microcomputer.

The full set of commands accepted by the controller is provided in "Commands Reference" in Section 15.

If you wish to use your PC simply to set configuration parameters and/or to exercise the controller, you should use the RoborunPlus PC utility.

Use and benefits of Serial Communication

The serial communication allows the controller to be connected to PCs, PLC, microcomputers or wireless modems. This connection can be used to both send commands and read various status information in real-time from the controller. The serial mode enables the design of complex motion control system, autonomous robots or more sophisticated remote controlled robots than is possible using the RC mode. RS232 and RS485 commands are very precise and securely acknowledged by the controller. They are also the method by which the controller's features can be accessed and operated to their fullest extent.

When operating in RC or analog input, serial communication can still be used for monitoring or telemetry.

When connecting the controller to a PC, the serial mode makes it easy to perform simple diagnostics and tests, including:

- Sending precise commands to the motor
- Reading the current consumption values and other parameters
- Obtaining the controller's software revision and date
- Reading inputs and activating outputs
- Setting the programmable parameters with a user-friendly graphical interface
- Updating the controller's software

Serial Port Configuration

The controller's default serial communication port is set as follows:

- 115200 bits/s
- 8-bit data
- 1 Start bit
- 1 Stop bit
- No Parity

Communication is done without flow control, meaning that the controller is always ready to receive data and can send data at any time.

Connector RS232 Pin Assignment

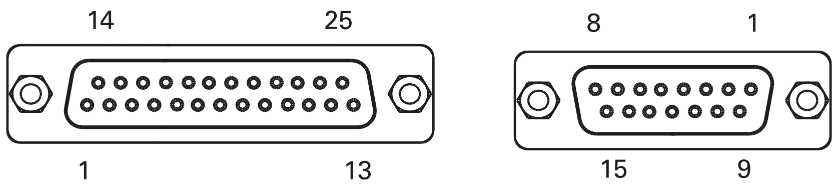


FIGURE 14-1. DB25 and DB15 connectors pin code locations

When used in the RS232 mode, the pins on the controller's DB15 or DB25 connector (depending on the controller model) are mapped as described in the table below

TABLE 14-1. RS232 Signals on DB15 and DB25 connectors §

Pin Number	Input or Output	Signal	Description
2	Output	Data Out	RS232 Data from Controller to PC
3	Input	Data In	RS232 Data In from PC
5	-	Ground	Controller ground

Connector RS485 Pin Assignment

When used in the RS485 mode, the pins on the controller's DB15, DB25 or DB9 connector (depending on the controller model) are mapped as described in each controller's datasheet.

Setting Different Bit Rates

It is possible to set RS232 and RS485 bit rate to lower values. This operation cannot be done while the controller is connected via RS232 or RS485. Beware that once the bit rate is different than the default 115200, it will no longer be able to communicate with the PC utility if serial connection is used. From the Console, send the following commands:

```
^RSBR nn
```

where nn =
0: 115200
1: 57600
2: 38400
3: 19200
4: 9600
10: 230400

Make sure that the controller respond to this command with a +. Check that the value has been accepted by sending ~RSBR.

Send %EESAV from the console to store the new configuration to flash.

Cable configuration

The RS232 connection requires the special cabling as described in Figure 14-2. The 9-pin female connector plugs into the PC (or other microcontroller). The 15-pin or 25-pin male connector plugs into the controller.

It is critical that you do not confuse the connector's pin numbering. The pin numbers on the drawing are based on viewing the connectors from the front. Most connectors brands have pin numbers molded on the plastic.

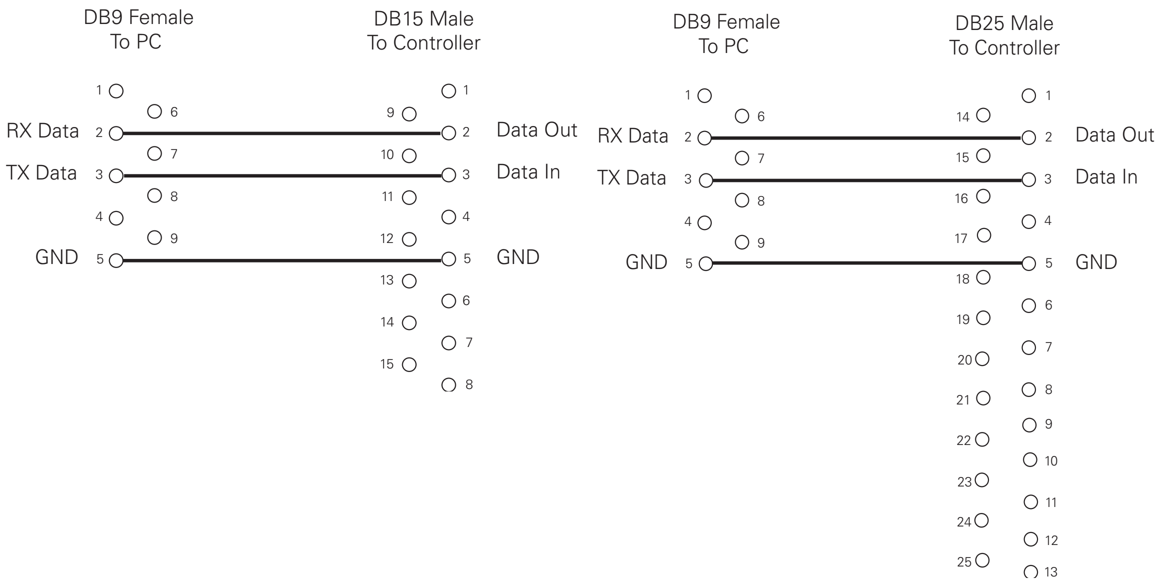


FIGURE 14-2. PC to controller RS 232 cable/connector wiring diagram

The 9 pin to 15 pin cable is provided by Roboteq for controllers with 15 pin connectors.

Controllers with 25 pins connectors are fitted with a USB port that can be used with any USB cables with a type B connector.

Extending the RS232 Cable

RS232 extension cables are available at most computer stores. However, you can easily build one using a 9-pin DB9 male connector, a 9-pin DB9 female connector and any 3-conductor cable. DO NOT USE COMMERCIAL 9-PIN TO 25-PIN CONVERTERS as these do not match the 25-pin pinout of the controller. These components are available at any electronics distributor. A CAT5 network cable is recommended, and cable length may be up to 100' (30m). Figure 14-3 shows the wiring diagram of the extension cable.

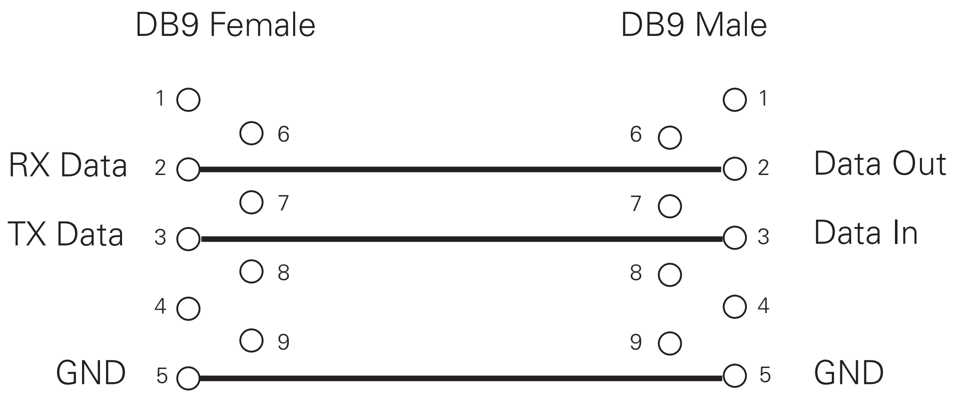


FIGURE 14-3. RS232 extension cable/connector wiring diagram

Connecting to Arduino and other TTL Serial Microcomputers

Arduino and similar microcomputers have a TTL serial port. There are Roboteq controllers supporting RS485, so the connection can be done directly. However, for the rest of the controllers there is a full RS232 serial interface. RS232 has the following differences from TTL serial:

TABLE 14-2. Connecting with TTL devices

	RS232	TTL Serial
Voltage Levels	+10V/-10V	0-3V
Logic level	Inverted	Non-Inverted

A TTL to RS232 adapter must be therefore be used to convert to the Arduino serial interface. Newer Roboteq controller allows the RS232 signal to be non-inverted. Interfacing to Arduino or other TTL Serial interface can therefore be done with just a resistors, and 2 optional diodes as shown in the diagram below:

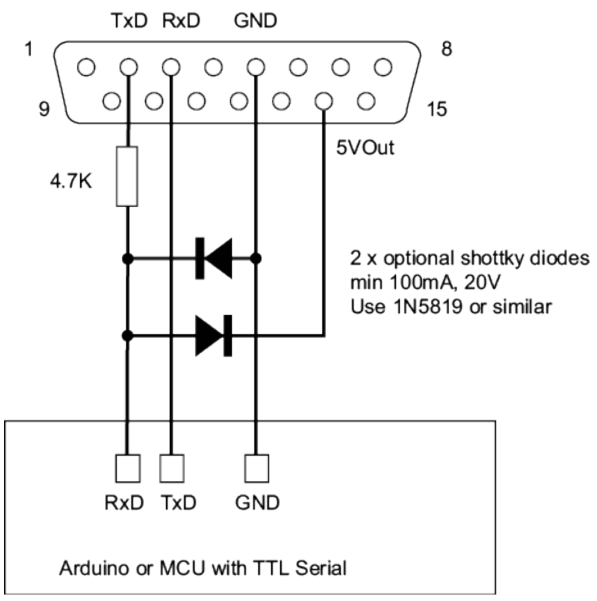


FIGURE 14-4. Simplified TTL to RS232 connection

The data sent from the TTL serial port are 0-3V and can be directly connected to the controller's RS232 input where it will be captured as valid 0-1 levels.

The data at the output of the controller is +/-10V. At the other end of the resistor, the voltage is clamped to around 0-3.3V by the protection diodes that are included in the Arduino MCU. However, to avoid any stress it is highly recommended to insert the 2 diodes shown on the diagram.

To operate, the RS232 output must be set to inverted. This must be done from the Console of the Roborun Utility while connected via USB. This will only work on newer controller models fitted with firmware version 1.6a or more recent.

From the Console, send the following commands:

^RSBR nn

where nn =

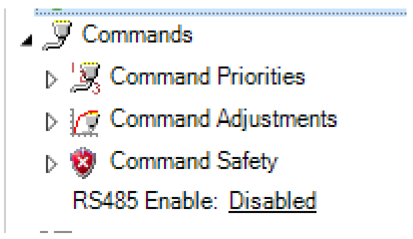
5: 115200 + Inverted RS232
6: 57600 + Inverted RS232
7: 38400 + Inverted RS232
8: 19200 + Inverted RS232
9: 9600 + Inverted RS232

Make sure that the controller respond to this command with a +. Check that the value has been accepted by sending ~RSBR. If a - is replied or if the value is different than the one entered, then the hardware and/or firmware does not support serial inverted and cannot be used with this circuit.

Send %EESAV from the console to store the new configuration to flash.

RS485 Configuration

Consult your controller's datasheet in order to know whether RS485 communication is supported. There are controller models whose pins related to RS485 communication are shared with other functionalities (check the controller's datasheet). For this reason for these controllers only, the RS485 functionality is by default disabled. In order to enable it the RS485 configuration command (RS485 enable) should be set to 1 (^RS485 1).



RS485 Enable: Disabled

FIGURE 14-5. RS485 Enable configuration

USB Configuration

USB is available on all Roboteq controller models and provides a fast and reliable communication method between the controller and the PC. After plugging the USB cable to the controller and the PC, the PC will detect the new hardware, and install the driver. Upon successful installation, the controller will be ready to use.

The controller will appear like another Serial device to the PC. This method was selected because of its simplicity, particularly when writing custom software: opening a COM port and exchanging serial data is a well documented technique in any programming language.

Note that Windows will assign a COM port number that is more or less random. The Roborun PC utility automatically scans all open COM ports and will detect the controller on its own. When writing your own software, you will need to account for this uncertainty in the COM port assignment.

Important Warning

Beware that because of its sophistication, the USB protocol is less likely to recover than RS232 should an electrical disturbance occur. We recommend using USB for configuration and monitoring, and use RS232 for field deployment. Deploy USB based system only after performing extensive testing and verifying that it operates reliably in your particular environment.

TCP Configuration

Controller models that support communication via an Ethernet cable are identified with the letter E, for example, FBL2360E or FBL2360ES. By default, TCP communication is disabled on controllers, so in order to use this feature configuration is needed. All configuration parameters may be accessed under the Roborun+ Configuration tab. All TCP parameters are configurable from the TCP node under the Inputs/Outputs Column. Find more details in section 15, "TCP Communication Commands".

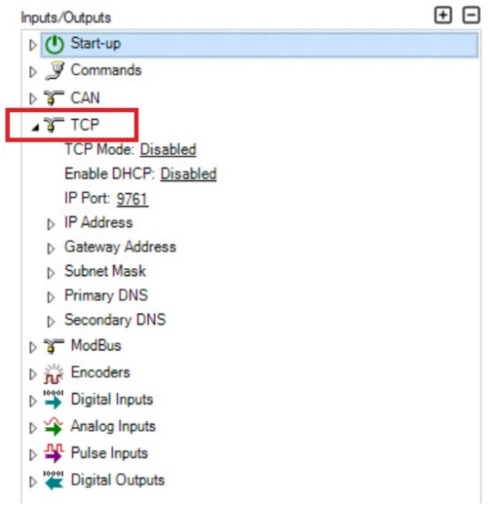

Inputs/Outputs
Start-up
Commands
CAN
TCP
TCP Mode: Disabled
Enable DHCP: Disabled
IP Port: 9761
IP Address
Gateway Address
Subnet Mask
Primary DNS
Secondary DNS
ModBus
Encoders
Digital Inputs
Analog Inputs
Pulse Inputs
Digital Outputs

FIGURE 14-6. TCP configuration

All TCP/IP configuration changes will be applied after restarting the controller. Communication via TCP, Modbus TCP or Modbus TCP over RTU will be available as long as the TCP mode is enabled.

Command Priorities

The controller will respond to commands from one of five possible sources:

- Network or Script Command
- Serial (RS232, RS485, TCP or USB)
- Pulse
- Analog

One, two, three or four (except from Network or Script command, which is always enabled) command modes can be enabled at the same time. When multiple modes are enabled, the controller will select which mode to use based on a user selectable priority scheme. The priority mechanism is described in details in "Input Command Modes and Priorities" on page 79.

Communication Arbitration

Commands may arrive through the RS232, RS485, TCP or the USB port at the same time. They are executed as they arrive in a first come first served manner. Commands that are arriving via USB are replied on USB. Commands arriving via the UART are replied on the UART. Redirection symbol for redirecting outputs to the other port exists (e.g. a command can be made respond on USB even though it arrived on RS232).

Network Commands

Commands arriving via Network have bigger priority than serial commands and will not conflict with motor command arriving via serial, TCP or USB. Network commands are also subject to the serial Watchdog timer. Motors will be stopped and command input will switch according to the Priority table if the Watchdog timer is allowed to timeout.

Script-generated Commands

Commands that are issued from a user script have bigger priority than serial and Network commands and will not conflict with motor command arriving via serial, TCP, USB or the network. Script commands are also subject to the serial Watchdog timer. Motors will be stopped and command input will switch according to the Priority table if the Watchdog timer is allowed to timeout.

Communication Protocol Description

The controller uses a simple communication protocol based on ASCII characters. Commands are not case sensitive. **?a** is the same as **?A**. Commands are terminated by carriage return (Hex 0x0d, '\r').

The underscore '_' character is interpreted by the controller as a carriage return. This alternate character is provided so that multiple commands can be easily concatenated inside a single string. Please note that this feature is exclusive to the RS232 interface and does not apply to RS485. For RS485 carriage return must be used.

All other characters lower than 0x20 (space) have no effect.

Character Echo

The controller will echo back to the PC or Microcontroller every valid character it has received. If no echo is received, one of the following is occurring:

- echo has been disabled
- the controller is Off
- the controller may be defective

Command Acknowledgment

The controller will acknowledge commands in one of the two ways:

For commands that cause a reply, such as a configuration read or a speed or amps queries, the reply to the query must be considered as the command acknowledgment.

For commands where no reply is expected, such as speed setting, the controller will issue a "plus" character (**+**) followed by a Carriage Return after every command as an acknowledgment.

Command Error

If a command or query has been received, but is not recognized or accepted for any reason, the controller will issue a "minus" character (**-**) to indicate the error.

If the controller issues the "**-**" character, it should be assumed that the command was not recognized or lost and that it should be repeated.

Watchdog time-out

For applications demanding the highest operating safety, the controller should be configured to automatically switch to another command mode or to stop the motor (but otherwise remain fully active) if it fails to receive a valid command on its RS232, RS485, TCP, USB or network ports, or from a MicroBasic Script for more than a predefined period.

By default, the watchdog is enabled with a timeout period of 1 second. Timeout period can be changed or the watchdog can be disabled by the user. When the watchdog is enabled and timeout expires, then the controller will go to a Quick Stop (force the motor to stop based on Fault Deceleration configuration command). After that the controller will accept commands from the next source in the priority list. See "Command Priorities" on page 181.

Controller Present Check

The controller will reply with an ASCII ACK character (0x06) anytime it receives a QRY character (0x05). This feature can be used to quickly scan a serial port and detect the presence, absence or disappearance of the controller. The QRY character can be sent at any time (even in the middle of a command) and has no effect at all on the controller's normal operation.

Raw Redirect Mode

In the Raw Redirect mode, received unprocessed data coming from either RS232 or RS485 interfaces, can be read by the user. Likewise, the user can send data with any content towards either RS232 or RS485 interfaces. The data are split into frames. In ASCII mode one frame is the sequence of characters ending with a termination character (\r). In RTU mode one frame is defined based on the Modbus RTU regulations (after specific silent time goes by, see Modbus Manual for more details).

Configuration

Raw Redirect mode is enabled for either RS232 or RS485 interfaces using the Raw Redirect Mode (see ISM - Raw Redirect Mode, in page 313) configuration command. By default ASCII mode is used, however if the respective Modbus Mode (see DMOD – Modbus Mode, in page 311) is set to RTU then the received data will be handled according to the Modbus RTU regulations (See Modbus Manual for more details).

Checking Received Frames

Received frames are first loaded in the 256-byte FIFO buffer. Before a frame can be read, it is necessary to check if any frames are present in the buffer using the ?CD query. The query can be sent from the serial/USB port, or from a MicroBasic script using the getvalue(_CD) function. The query will return the number of frames that are currently pending, and copy the oldest frame into the read buffer, from which it can then be accessed. Sending ?CD again, copies the next frame into the read buffer.

The query usage is as follows:

Syntax: ?CD

Reply: CD=number of frames pending

Reading Received Frames

After a frame has been moved to the read buffer, the frame size and the data can be read with the ?DDT query. The query can be sent from the serial/USB port, or from a MicroBasic script using DDT query or SDT query for ASCII mode. The query usage is as follows:

Syntax: ?DDT [ee]

Reply: DDT=frame size:data0:data1: .... :dataN

Where: ee = frame element

1 = byte size

2-64 = data0 to data62

Examples:

Q: ?DDT

R: DDT=8:82:111:98:111:116:101:113

Q: ?DDT 3

R: DDT=111

Q: ?SDT

R: SDT="Roboteq"

Transmitting Frames

ASCII or RTU data can easily be transmitted using the Send Raw USART Frames Command !CU. This command can be used to define the output port (RS232 or RS485), the frame length and the data, one element at a time. The frame is sent immediately after the frame length is entered, and so it should be entered last.

Syntax: !CU ee nn

Where: ee = frame element

1 = outport (0 for RS232 and 6 for RS485)

2 = frame length

3 to 18 = data0 to data15

nn = value

Examples: !CS 1 0 Enter 0 in outport (RS232)

!CS 3 49 Enter 49 in Data 0

!CS 4 50 Enter 50 in Data 1

!CS 4 51 Enter 51 in Data 2

!CS 2 3 Enter 2 in frame length. Send data frame

SECTION 15

Commands Reference

This section lists all the commands accepted by the controller. Commands are typically sent via the serial (RS232, RS485, TCP or USB) ports (See "Serial (RS232/RS485/TCP/USB) Operation" in Section 14) Except for a few maintenance commands, they can also be issued from within a user script written using the MicroBasic language (See "MicroBasic Scripting Manual").

Commands Types

The controller will accept and recognize four types of commands:

Runtime commands

These start with "!" when called via the serial communication (RS232, RS485, TCP or USB), or using the setcommand() MicroBasic function. These are usually motor or operation commands that will have immediate effect (e.g. to turn on the motor, set a speed or activate digital output). Most of Runtime commands are mapped inside a respective Object Directories in order to be accessed by the network interfaces, CAN/CANOpen, EtherCAT and Profinet (See "CAN/EtherCAT Networking Manual" or "Profinet Networking Manual" respectively) See "Runtime Commands" on page 188 for the full list and description of these commands.

Runtime queries

These start with "**?**" when called via the serial communication (RS232, RS485, TCP or USB), or using the getvalue() Microbasic function. These are used to read operating values at runtime (e.g. read Amps, Volts, power level, counter values). Most of Runtime queries are mapped inside a respective Object Directories in order to be accessed by the network interfaces, CAN/CANOpen, EtherCAT and Profinet (See "CAN/EtherCAT Networking Manual" or "Profinet Networking Manual" respectively). See Runtime commands are commands that can be sent at any time during controller operation and are taken into consideration immediately. Runtime commands start with "!" and are followed by one to three letters. Runtime commands are also used to refresh the watchdog timer to ensure safe communication. Runtime commands can be called from a MicroBasic script using the setcommand() function.

Maintenance commands

These are only available trough serial (RS232, RS485, TCP or USB) and start with "%". They are used for all of the maintenance commands such as (e.g. set the time, save configuration to EEPROM, reset, load default, etc.).

Configuration commands

These start with "~" for read and "^" for write when called via the serial communication (RS232, RS485, TCP or USB), or using the getconfig() and setconfig() MicroBasic functions. They are used to read or configure all the operating parameters of the controller (e.g. set or read amps limit). See "Set/Read Configuration Commands" on page 303 for the full list and description of these commands.

Runtime Commands

Runtime commands are commands that can be sent at any time during controller operation and are taken into consideration immediately. Runtime commands start with "!" and are followed by one to three letters. Runtime commands are also used to refresh the watchdog timer to ensure safe communication. Runtime commands can be called from a MicroBasic script using the setcommand() function.

TABLE 15-1. Runtime Commands

Command	Arguments	Description
AC	Channel Acceleration	Set Acceleration
AX	Channel Acceleration	Next Acceleration
B	VarNbr Value	Set User Boolean Variable
BRK	Channel Value	Brake Override
C	Channel Value	Set Encoder Counters
CB	Channel Value	Set Internal Sensor Counter
CIG	PID Channel Gain	Set Current Integral Gains
CG	Channel Value	Set Motor Command via CAN
CPG	PID Channel Gain	Set Current Proportional Gains
CS	Element Value	CAN Send
CSS	Channel Value	Set SSI Sensor Counter
CU	Element Value	Raw Redirect Send
D0	OutputNbr	Reset Individual Digital Out bits
D1	OutputNbr	Set Individual Digital Out bits
DC	Channel Deceleration	Set Deceleration
DG	PID Channel Gain	Set PID Derivative Gains
DS	Value	Set all Digital Out bits
DX	Channel Value	Next Decceleration
EES	None	Save Configuration in EEPROM
EX	None	Emergency Shutdown
G	Channel Value	Go to Speed or to Relative Position
GIQ	Channel Value	Go to Torque Amps
GID	Channel Value	Go to Flux Amps
H	Channel	Load Home counter
IG	PID Channel Gain	Set PID Integral Gains
MG	None	Emergency Stop Release and Fault Clearance
MS	Channel	Stop in all modes
MSS	Channel	Motor Sensor Setup

Command	Arguments	Description
P	Channel Destination	Go to Absolute Desired Position
PG	PID Channel Gain	Set PID Proportional Gains
PR	Channel Delta	Go to Relative Desired Position
PRX	Channel Delta	Next Go to Relative Desired Position
PX	Channel Delta	Next Go to Absolute Desired Position
QST	Channel Value	Quick Stop
R	[Option]	MicroBasic Run
RST	Channel Value	Resets the drive
S	Channel Value	Set Motor Speed
STT	None	STO Self-Test
SX	Channel Value	Next Velocity
VAR	VarNbr Value	Set User Variable

AC - Set Acceleration

Alias: ACCEL HexCode: 07 CANOpen id: 0x2006

Description:

Set the rate of speed change during acceleration for a motor channel. This command is identical to the MACC configuration command but is provided so that it can be changed rapidly during motor operation. Acceleration value is in 0.1 * RPM per second. When using controllers fitted with encoder, the speed and acceleration value are actual RPMs. Brushless motor controllers use the hall sensor for measuring actual speed and acceleration will also be in actual RPM/s. When using the controller without speed sensor, the acceleration value is relative to the Max RPM configuration parameter, which itself is a user-provided number for the speed normally expected at full power. Assuming that the Max RPM parameter is set to 1000, and acceleration value of 10000 means that the motor will go from 0 to full speed in exactly 1 second, regardless of the actual motor speed. In Closed Loop Torque mode acceleration value is in 10 * miliAmps per second. This command is not applicable if either of the acceleration (MAC) or deceleration (MDEC) configuration value is set to 0.

Syntax Serial: !AC cc nn

Syntax Scripting: setcommand(_AC, cc, nn)
setcommand(_ACCEL, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Acceleration Type: Signed 32-bit
Min: 0 Max: 500000

Where:
cc = Motor channel
nn = Acceleration value in 0.1 * RPM/s

Example:

!AC 1 2000 : Increase Motor 1 speed by 200 RPM every second if speed is measured by encoder
!AC 2 20000 : Time from 0 to full power is 0.5s if no speed sensors are present and Max RPM is set to 1000

AX - Next Acceleration

Alias: NXTACC HexCode: 14 CANOpen id: 0x2012

Description:

This command is used for chaining commands in Position Count mode. It is similar to AC except that it stores an acceleration value in a buffer. This value will become the next acceleration the controller will use and becomes active upon reaching a previous desired position. If omitted, the command will be chained using the last used acceleration value. This command is not applicable if either of the acceleration (MAC) or deceleration (MDEC) configuration value is set to 0.

Syntax Serial: !AX cc nn

Syntax Scripting: setcommand(_AX, cc, nn)
setcommand(_NXTACC, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Acceleration Type: Signed 32-bit

Min: 0 Max: 500000

Where:

cc = Motor channel
nn = Acceleration value in 0.1 * RPM/s

B - Set User Boolean Variable

Alias: BOOL HexCode: 16 CANOpen id: 0x2015

Description:

Set the state of user boolean variables inside the controller. These variables can then be read from within a user MicroBasic script to perform specific actions.

Syntax Serial: !B nn mm

Syntax Scripting: setcommand(_B, nn, mm)
setcommand(_BOOL, nn, mm)

Number of Arguments: 2

Argument 1: VarNbr
Min: 1 Max: Total nbr of Bool Vars

Argument 2: Value Type: Boolean
Min: 0 Max: 1

Where:

nn = Variable number
mm = 0 or 1

Note:

The total number of user variables depends on the controller model and can be found in the product datasheet.

BRK - Brake Override

Alias: - HexCode: AB CANOPEN id: 0x2034

Description:

This command is used to override the automatic engage/release of the brake. The Digital Out action must be configured as "Motor is On".

Syntax Serial: !BRK cc nn

Syntax Scripting: setcommand (_BRK, cc)

Number of Arguments:

Argument 1: PWM Brake Channel
Min: 1 Max: Total Number of Motors

Argument 2: Override Status Type: Unsigned 8-bit
Min: 0 Max: 2 Default: 0

Where:

nn =
0: Auto. Brake is controlled by Motor is On action.
1: Brake Release. Brake is released ignoring the Motor is On action.
2: Brake Engage. Brake is engaged ignoring the Motor is On action.

Example:

!BRK 1 1: will release the PWM brake.

Important Note

This command overrides the automatic brake control. This also means that if the motor is running and the user sends a Force brake command, the brake WILL be engaged.

C - Set Encoder Counters

Alias: SENCNTR HexCode: 04 CANOpen id: 0x2003

Description:

This command loads the encoder counter for the selected motor channel with the value contained in the command argument. Beware that changing the controller value while operating in closed-loop mode can have adverse effects.

Syntax Serial: !C [cc] nn

Syntax Scripting: setcommand(_C, cc, nn)
setcommand(_SENCNTR, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Encoders

Argument 2: Value Type: Signed 32-bit

Min: -2147M Max: +2147M

Where:

cc = Motor channel
nn = Counter value

Example:

!C 2 -1000 : Loads -1000 in encoder counter 2
!C 1 0 : Clears encoder counter 1

CB - Set Internal Sensor Counter

Alias: SBLCNTR HexCode: 05 CANOpen id: 0x2004

Description:

This command loads the Internal Sensor counter with the value contained in the command argument. Beware that changing the controller value while operating in closed-loop mode can have adverse effects.

Syntax Serial: !CB [cc] nn

Syntax Scripting: setcommand(_CB, cc, nn)
setcommand(_SBLCNTR, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Value Type: Signed 32-bit

Min: -2147M Max: +2147M

Where:

cc = Motor channel
nn = Counter value

Example:

!CB 1 -1000 : Loads -1000 in brushless counter 1
!CB 2 0 : Clears brushless counter 2

CIG – Set Current Integral Gains

Alias: - HexCode: A5 CANOpen id: 0x2032

Description:

Sets the Current PI's Integral Gain. The value is set as the gain multiplied by 10^4. On brushless motor controller operating in sinusoidal mode, two gains can be set for each motor channel, in order to control the Flux and Torque current. On DC brushed controllers or in brushless motor controllers when operating in trapezoidal mode the gains for the Torque current are used only.

Syntax Serial: !CIG [cc] nn

Syntax Scripting: setcommand(_CIG, cc, nn)

Number of Arguments: 2

Argument 1: Channel		Type: Unsigned 8-bit
	Min: 1	Max: 2 x Total Number of Motors
Argument 2: Gain	Type: Unsigned 32-bit	
	Min: 0 Max: 2,000,000,000	

Where:

cc (single channel) =

1: Flux Integral Gain

2: Torque Integral Gain

cc (dual channel) =

1: Flux Integral Gain for motor 1

2: Flux Integral Gain for motor 2

3: Torque Integral Gain for motor 1

4: Torque Integral Gain for motor 2

nn: Integral Gain*10.000

Example:

!CIG 1 2300 : will set Flux Integral Gain of Motor 1 to 0.23

CG - Set Motor Command via CAN

Alias: CANGO HexCode: 19 CANOpen id: 0x2000

Description:

This command is identical to the G (GO) command except that it is meant to be used for sending motor commands via CANOpen. See the G command for details.

Syntax Serial: !CG cc nn

Syntax Scripting: setcommand(_CG, cc, nn)
setcommand(_CANGO, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Value Type: Signed 32-bit

Min: -1000 Max: +1000

Where:

cc = Motor channel
nn = Command value

CPG – Set Current Proportional Gains

Alias: - HexCode: A4 CANOpen id: 0x2031

Description:

Sets the Current PI's Proportional Gain. The value is set as the gain multiplied by 10^4. On brushless motor controller operating in sinusoidal mode, two gains can be set for each motor channel, in order to control the Flux and Torque current. On DC brushed controllers or in brushless motor controllers when operating in trapezoidal mode the gains for the Torque current are used only.

Syntax Serial: !CPG [cc] nn

Syntax Scripting: setcommand(_CPG, cc, nn)

Number of Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: 2 x Total Number of Motors

Argument 2: Gain Type: Unsigned 32-bit

Min: 0 Max: 2,000,000,000

Where:

cc (single channel) =

1: Flux Proportional Gain

2: Torque Proportional Gain

cc (dual channel) =

1: Flux Proportional Gain for motor 1

2: Flux Proportional Gain for motor 2

3: Torque Proportional Gain for motor 1

4: Torque Proportional Gain for motor 2

nn: Proportional Gain*10.000

Example:

!CPG 1 2300 : will set Flux Proportional Gain of Motor 1 to 0.23

CS - CAN Send

Alias: CANSEND HexCode: 18 CANOpen id:

Description:

This command is used in CAN-enabled controllers to build and send CAN frames in the RawCAN mode (See RawCAN section in manual). It can be used to enter the header, bytecount, and data, one element at a time. The frame is sent immediately after the bytecount is entered, and so it should be entered last.

Syntax Serial: !CS ee nn

Syntax Scripting: setcommand(_CS, ee, nn)
setcommand(_CANSEND, ee, nn)

Number of Arguments: 2

Argument 1: Element

Min: 1 Max: 10

Argument 2: Value Type: Unsigned 8-bit

Min: 0 Max: 255

Where:

ee =
1 : Header
2 : Bytecount
3 to 10 : Data0 to data7
nn = value

Example:

!CS 1 5 : Enter 5 in header
!CS 3 2 : Enter 2 in data 0
!CS 4 3 : Enter 3 in data 1
!CS 2 2 : Enter 2 in bytecount and send CAN frame

CSS - Set SSI Sensor Counter

Alias: - HexCode: 6C CANOpen id: 0x201F

Description:

This command loads the SSI Sensor counter with the value contained in the command argument. Beware that changing the controller value while operating in closed-loop mode can have adverse effects. This command is not applicable if the respective sensor's use has been set as absolute feedback.

Syntax Serial: !CSS [cc] nn

Syntax Scripting: setcommand(_CSS, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of SSI sensors

Argument 2: Value Type: Signed 32-bit
Min: -2147M Max: +2147M

Where:

cc = SSI sensor channel
nn = Counter value

Example:

!CSS 1 -1000 : Loads -1000 in SSI sensor counter 1
!CSS 2 0 : Clears SSI sensor counter 2

CU - Raw Redirect Send

Alias: - HexCode: 94 CANOpen id: -

Description:

This command is used to build and send Raw frames in serial interfaces (See Raw Redirect section in manual). It can be used to enter the outport (RS232 or RS485), frame length, and data, one element at a time. The frame is sent immediately after the frame length is entered, and so it should be entered last.

Syntax Serial: !CS ee nn

Syntax Scripting: setcommand(_CS, ee, nn)

Number of Arguments: 2

Argument 1: Element

Min: 1 Max: 18

Argument 2: Value Type: Unsigned 8-bit

Min: 0 Max: 255

Where: ee = frame element

1 = outport (0 for RS232 and 6 for RS485)

2 = frame length

3 to 18 = data0 to data15

nn = value

Examples: !CS 1 0 Enter 0 in outport (RS232)

!CS 3 49 Enter 49 in Data 0

!CS 4 50 Enter 50 in Data 1

!CS 4 51 Enter 51 in Data 2

!CS 2 3 Enter 2 in frame length. Send data frame

D0 - Reset Individual Digital Out bits

Alias: DRES HexCode: 09 CANOpen id: 0x200A

Description:

The D0 command will turn off the single digital output selected by the number that follows.

Syntax Serial: !D0 nn

Syntax Scripting: setcommand(_D0, nn)
setcommand(_DRES, nn)

Number of Arguments: 1

Argument 1: OutputNbr Type: Unsigned 8-bit

Min: 1 Max: Total number of Digital Outs

Where:

nn = Output number

Example:

!D0 2 : will deactivate output 2

Note:

Digital Outputs are Open Collector. Activating an outputs will force it to ground. Deactivating an output will cause it to float.

D1 - Set Individual Digital Out bits

Alias: DSET	HexCode: 0A	CANOpen id: 0x2009

Description:

The D1 command will activate the single digital output that is selected by the parameter that follows.

Syntax Serial: !D1 nn

Syntax Scripting: setcommand(_D1, nn)
setcommand(_DSET, nn)

Number of Arguments: 1

Argument 1: OutputNbr Type: Unsigned 8-bit
Min: 1 Max: Total number of Digital Outs

Where:

nn = Output number

Example:

!D1 1 : will activate output 1

Note:

Digital Outputs are Open Collector. Activating an outputs will force it to ground. Deactivating an output will cause it to float.

DC - Set Deceleration

Alias: DECEL	HexCode: 08	CANOpen id: 0x2007

Description:

Set the rate of speed change during decceleration for a motor channel. This command is identical to the MDEC configuration command but is provided so that it can be changed rapidly during motor operation. Decceleration value is in 0.1 * RPM per second. When using controllers fitted with encoder, the speed and decceleration value are actual RPMs. Brushless motor controllers use the hall sensor for measuring actual speed and decceleration will also be in actual RPM/s. When using the controller without speed sensor, the decceleration value is relative to the Max RPM configuration parameter, which itself is a user-provided number for the speed normally expected at full power. Assuming that the Max RPM parameter is set to 1000, and decceleration value of 10000 means that the motor will go from full speed to 0 in exactly 1 second, regardless of the actual motor speed. In Closed Loop Torque mode deceleration value is in 0.1 * miliAmps per second. This command is not applicable if either of the acceleration (MAC) or deceleration (MDEC) configuration value is set to 0.

Syntax Serial: !DC cc nn

Syntax Scripting: setcommand(_DC, cc, nn)
setcommand(_DECEL, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Deceleration Type: Signed 32-bit

Min: 0 Max: 500000

Where:

cc = Motor channel
nn = Deceleration value in 0.1 * RPM/s

Example:

!DC 1 2000 : Reduce Motor 1 speed by 200 RPM every second if speed is measured by encoder
!DC 2 20000 : Time from full power to stop is 0.5s if no speed sensors are present and Max RPM is set to 1000

DG – Set PID Derivative Gains

Alias: - HexCode: A3 CANOpen id: 0x2030

Description:

Sets the PID's Derivative Gain. The value is set as the gain multiplied by 10^6. This value is used for both speed and position derivative gains.

Syntax Serial: !DG [cc] nn

Syntax Scripting: setcommand(_DG, cc, nn)

Number of Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: 2 x Total Number of Motors

Argument 2: Gain Type: Unsigned 32-bit

Min: 0 Max: 2,000,000,000

Where:

cc (single channel) =

1: Speed Derivative Gain

2: Position Derivative Gain

cc (dual channel) =

1: Speed Derivative Gain for motor 1

2: Speed Derivative Gain for motor 2

3: Position Derivative Gain for motor 1

4: Position Derivative Gain for motor 2

nn: Derivative Gain*1.000.000

Example:

!DG 1 10.000 : will set Speed Derivative Gain of Motor 1 to 0.01

DS - Set all Digital Out bits

Alias: DOUT HexCode: 09 CANOpen id: 0x2008

Description:

The D command will turn ON or OFF one or many digital outputs at the same time. The number can be a value from 0 to 255 and binary representation of that number has 1bit affected to its respective output pin.

Syntax Serial: !DS nn

Syntax Scripting: setcommand(_DS, nn)
setcommand(_DOUT, nn)

Number of Arguments: 1

Argument 1: Value Type: Unsigned 8-bit
Min: 0 Max: 255

Where:

nn = Bit pattern to be applied to all output lines at once

Example:

!DS 03 : will activate outputs 1 and 2. All others are off

Note:

Digital Outputs are Open Collector. Activating an outputs will force it to ground. Deactivating an output will cause it to float.

DX - Next Deceleration

Alias: NXTDEC HexCode: 15 CANOpen id: 0x2013

Description:

This command is used for chaining commands in Position Count mode. It is similar to DC except that it stores a decceleration value in a buffer. This value will become the next decceleration the controller will use and becomes active upon reaching a previous desired position. If omitted, the command will be chained using the last used decceleration value. This command is not applicable if either of the acceleration (MAC) or deceleration (MDEC) configuration values is set to 0 (bypass command ramp).

Syntax Serial: !DX cc nn

Syntax Scripting: setcommand(_DX, cc, nn)
setcommand(_NXTDEC, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Value Type: Signed 32-bit

Min: 0 Max: 500000

Where:

cc = Motor channel
nn = Acceleration value

EES - Save Configuration in EEPROM

Alias: EESAV HexCode: 1B CANOpen id: 0x2017

Description:

This command causes any changes to the controller's configuration to be saved to Flash. Saved configurations are then loaded again next time the controller is powered on. This command is a duplication of the EESAV maintenance command. It is provided as a Real-Time command as well in order to make it possible to save configuration changes from within MicroBasic scripts.

Syntax Serial: !EES

Syntax Scripting: setcommand(_EES, 1)
setcommand(_EESAV, 1)

Number of Arguments: 0

Note:

Do not save configuration while motors are running. Saving to EEPROM takes several milliseconds, during which the control loop is suspended.
Number of EEPROM write cycles are limited to around 10000. Saving to EEPROM must be done scarcely.

EX - Emergency Stop

Alias: ESTOP HexCode: 0E CANOpen id: 0x200C

Description:

The EX command will cause the controller to enter an emergency stop in the same way as if hardware emergency stop was detected on an input pin. The emergency stop condition will remain until controller is reset or until the MG release command is received.

Syntax Serial: !EX

Syntax Scripting: setcommand(_EX, cc)
setcommand(_ESTOP, cc)

Number of Arguments: 1

Argument 1: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total Number of channels+1

Where:

cc =
1: All channels
2: Channel 1
3: Channel 2
4: Channel 3

Example:

!EX 3: Will trigger Emergency stop only for channel 2.

G - Go to Speed or to Relative Position

Alias: GO HexCode: 00 CANOpen id: Use CG

Description:

G is the main command for activating the motors. The command is a number ranging 1000 to +1000 so that the controller respond the same way as when commanded using Analog or Pulse, which are also -1000 to +1000 commands. The effect of the command differs from one operating mode to another.

In Open Loop Speed mode the command value is the desired power output level to be applied to the motor.

In Closed Loop Speed mode, the command value is relative to the maximum speed that is stored in the MXRPM configuration parameter.

In Closed Loop Position Relative and in the Closed Loop Tracking mode, the command is the desired relative destination position mode.

The G command has no effect in the Position Count mode.

In Torque mode, the command value is the desired Motor Amps relative to the Amps Limit configuration parameters

Syntax Serial: !G [nn] mm

Syntax Scripting: setcommand(_G, nn, mm)
setcommand(_GO, nn, mm)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Value Type: Signed 32-bit

Min: -1000 Max: 1000

Where:

cc = Motor channel
nn = Command value

Example:

!G 1 500 : In Open Loop Speed mode, applies 50% power to motor channel 1
!G 1 500 : In Closed Loop Speed mode, assuming that 3000 is contained in Max RPM parameter (MXRPM), motor will go to 1500 RPM
!G 1 500 : In Closed Loop Relative or Closed Loop Tracking modes, the motor will move to 75% position of the total -1000 to +1000 motion range
!G 1 500 : In Torque mode, assuming that Amps Limit is 60A, motor power will rise until 30A are measured.

GIQ - Go to Torque Amps

Alias: - HexCode: 7A CANOpen id: -

Description:

GIQ is the command for the torque amps in closed loop torque mode only in sinusoidal mode. In other cases it is void. The value is set in Amps*10. After the motor stops or at power up the target flux amps is set by the TID value.

Syntax Serial: !GIQ [nn] mm

Syntax Scripting: setcommand(_GIQ, nn, mm)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Amps

Type: Signed 32

Min: 0 Max: Max Amps in datasheet

Where:

cc = Motor channel

nn = Amps*10

Example:

!GIQ 1 500 : In Closed Loop Torque mode, applies 50,0A torque amps command.

GID - Go to Flux Amps

Alias: - HexCode: 7B CANOpen id: -

Description:

GID is the command for the flux amps in closed loop torque mode only in sinusoidal mode. In other cases it is void. The value is set in Amps*10. A non-zero value creates field weakening and can be used to achieve higher rotation speed

Syntax Serial: !GID [nn] mm

Syntax Scripting: setcommand(_GID, nn, mm)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Amps

Type: Signed 32

Min: 0 Max: Max Amps in datasheet

Where:

cc = Motor channel

nn = Amps*10

Example:

!GID 1 -200: In Closed Loop Torque mode, applies -20,0A flux amps command (field weakening).

H - Load Home counter

Alias: HOME HexCode: 0D CANOpen id: 0x200B

Description:

This command loads the Home count value into the Encoder, SSI Sensor, or Brushless Counters. The Home count can be any user value and is set using the EHOME, SHOME and BHOME configuration parameters. When SSI sensors are used as absolute encoders (Absolute Feedback) then this command loads to the Home count value the SSI sensor counter. In this case the Home count value is used as offset to the SSI sensor Counter. Beware that loading the counter with the home value while the controller is operating in closed loop can have adverse effects.

Syntax Serial: !H [cc]

Syntax Scripting: setcommand(_H, cc)
setcommand(_HOME, cc)

Number of Arguments: 1

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: Total Number of Encoders

Where:

cc = Motor channel

Example:

!H 1: Loads encoder counter 1, SSI sensor counter 1 and brushless counter 1 with their preset home values.

IG – Set PID Integral Gains

Alias: - HexCode: A2 CANOpen id: 0x202F

Description:

Sets the PID's Integral Gain. The value is set as the gain multiplied by 10^6. This value is used for both speed and position integral gains.

Syntax Serial: !IG [cc] nn

Syntax Scripting: setcommand(_IG, cc, nn)

Number of Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: 2 x Total Number of Motors

Argument 2: Gain Type: Unsigned 32-bit

Min: 0 Max: 2,000,000,000

Where:

cc (single channel) =

1: Speed Integral Gain

2: Position Integral Gain

cc (dual channel) =

1: Speed Integral Gain for motor 1

2: Speed Integral Gain for motor 2

3: Position Integral Gain for motor 1

4: Position Integral Gain for motor 2

nn: Integral Gain*1.000.000

Example:

!IG 1 100.000 : will set Speed Integral Gain of Motor 1 to 0.1

MG - Emergency Stop Release and Fault Clearance

Alias: MGO HexCode: 0F CANOpen id: 0x200D

Description:

The MG command will release the emergency stop condition or any other fault and allow the controller to return to normal operation. Always make sure that the fault condition has been cleared before sending this command.

Syntax Serial: !MG

Syntax Scripting: setcommand(_MG, 1)
setcommand(_MGO, 1)

Number of Arguments: 0

MS - Stop in all modes

Alias: MSTOP HexCode: 10 CANOpen id: 0x200E

Description:

The MS command will stop the motor for the specified motor channel.

Syntax Serial: !MS [cc]

Syntax Scripting: setcommand(_MS, cc)
setcommand(_MSTOP, cc)

Number of Arguments: 1

Argument 1: Channel Type: Unsigned 8-bit
Min: 1 Max: Total Number of Motors

Where:

cc = Motor channel

MSS - Motor Sensor Setup

Alias: - HexCode: 62 CANOpen id: 0x202d

Description:

This command is used in order to perform motor and/or sensor setup, in cases of DC brushless motor controllers, when working in Sinusoidal mode. This command will make the motor spin slowly and will configure accordingly the respective fields in order to have a smooth motor spin and alignment between the motor and the sensor directions. During setup no motor command can be applied to the motors. For more details see Section 8.

Syntax Serial: !MSS [cc]

Syntax Scripting: setcommand(_MSS, cc)

Number of Arguments: 1

Argument 1: Channel Type: Unsigned 8-bit
Min: 1 Max: Total Number of Motors

Where:

cc = Motor channel

P - Go to Absolute Desired Position

Alias: MOTPOS HexCode: 02 CANOpen id: 0x2001

Description:

This command is used in the Position Count mode to make the motor move to a specified feedback sensor count value. Take into consideration that in case of DS402 enabled, then this command is not active. POS command should be used instead.

Syntax Serial: !P [cc] nn

Syntax Scripting: setcommand(_P, cc, nn)
setcommand(_MOTPOS, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Destination Type: Signed 32-bit
Min: -2147M Max: +2147M

Where:

cc = Motor channel
nn = Absolute count destination

Example:

!P 1 10000 : make motor go to absolute count value 10000.

PG – Set PID Proportional Gains

Alias: - HexCode: A1 CANOpen id: 0x202E

Description:

Sets the PID's Proportional Gain. The value is set as the gain multiplied by 10^6. This value is used for both speed and position proportional gains.

Syntax Serial: !PG [cc] nn

Syntax Scripting: setcommand(_PG, cc, nn)

Number of Arguments: 2

Argument 1: Channel	Type: Unsigned 8-bit
Min: 1	Max: 2 x Total Number of Motors

Argument 2: Gain Type: Unsigned 32-bit
Min: 0 Max: 2,000,000,000

Where:

cc (single channel) =

1: Speed Proportional Gain

2: Position Proportional Gain

cc (dual channel) =

1: Speed Proportional Gain for motor 1

2: Speed Proportional Gain for motor 2

3: Position Proportional Gain for motor 1

4: Position Proportional Gain for motor 2

nn: Proportional Gain*1.000.000

Example:

!PG 1 100.000 : will set Speed Proportional Gain of Motor 1 to 0.1

PR - Go to Relative Desired Position

Alias: MPOSREL	HexCode: 11	CANOpen id: 0x200F

Description:

This command is used in the Position Count mode to make the motor move to a feedback sensor count position that is relative to its current desired position.

Syntax Serial:	PR [cc] nn

Syntax Scripting:	setcommand(_PR, cc, nn)
setcommand(_MPOSREL, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1	Max: Total Number of Motors

Argument 2: Delta	Type: Signed 32-bit
Min: -2147M	Max: +2147M

Where:

cc = Motor channel
nn = Relative count position

Example:

!PR 1 10000 : while motor is stopped after power up and counter = 0, motor 1 will go to +10000
!PR 2 10000 : while previous command was absolute goto position !P 2 5000, motor will go to +15000

Note:

Beware that counter will rollover at counter values +/-2’147’483’648.

PRX - Next Go to Relative Desired Position

Alias: NXTPOSR HexCode: 13 CANOpen id: 0x2011

Description:

This command is similar to PR except that it stores a relative count value in a buffer. This value becomes active upon reaching a previous desired position and will become the next destination the controller will go to. See Position Command Chaining in manual.

Syntax Serial: !PRX [cc] nn

Syntax Scripting: setcommand(_PRX, cc, nn)
setcommand(_NXTPOSR, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Delta Type: Signed 32-bit
Min: -2147M Max: +2147M

Where:

cc = Motor channel
nn = Relative count position

Example:

!P 1 5000 followed by !PRX 1 -10000 : will cause motor to go to count position 5000 and upon reaching the destination move to position -5000.

PX - Next Go to Absolute Desired Position

Alias: NXTPOS HexCode: 12 CANOpen id: 0x2010

Description:

This command is similar to P except that it stores an absolute count value in a buffer. This value will become the next destination the controller will go to and becomes active upon reaching a previous desired position. See Position Command Chaining in manual.

Syntax Serial: !PX [nn] cc
Syntax Scripting: setcommand(_PX, nn, cc)
setcommand(_NXTPOS, nn, cc)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Delta Type: Signed 32-bit

Min: -2147M Max: +2147M

Where:

cc = Motor channel
nn = Absolute count position

Example:

!P 1 5000 followed by !PX 1 -10000 : will cause motor to go to count position 5000 and upon reaching the destination move to position -10000.

QST - Quick Stop

Alias: HexCode:8D CanOpen id: 0x202C

Description:

With this command the motor speed decelerates to zero according to the fault deceleration speed ramp. This command can be used when the user wants to decelerate the motor speed to zero at all modes. A safe stop flag will be generated during the deceleration and it will be reseted when the motor command will be idle (0 in speed modes or equal to feedback in position modes) and the speed 0.

Please note that Quick Stop will switch the system to Speed mode to decelerate the motor. For proper operation, the Speed mode must be configured with the appropriate PID gains.

Syntax Serial: !QST [cc]

Syntax Scripting: setcommand (_QST, cc)

Number of Arguments: 1

Min:1 Max: Total number of Motors

Where:

cc= Motor channel

Example:

!QST 1: The motor 1 will start decelerate its speed according to deceleration speed ramp.

R - MicroBasic Run

Alias: BRUN HexCode: 0C CANOpen id: 0x2018

Description:

This command is used to start, stop and restart a MicroBasic script if one is loaded in the controller.

Syntax Serial: !R [nn]

Syntax Scripting: setcommand(_R, nn)
setcommand(_BRUN, nn)

Number of Arguments: 1

Argument 1: [Option]	Type: Unsigned 8-bit
Min: None	Max: 2

Where:

nn =
None : Start/resume script
0 : Stop script
1 : Start/resume script
2 : Reinitialize and restart script

S - Set Motor Speed

Alias: MOTVEL HexCode: 03 CANOpen id: 0x2002

Description:

In the Closed-Loop Speed mode, this command will cause the motor to spin at the desired RPM speed. In Closed-Loop Position modes, this commands determines the speed at which the motor will move from one position to the next. It will not actually start the motion.

Syntax Serial: !S [cc] nn

Syntax Scripting: setcommand(_S, cc, nn)
setcommand(_MOTVEL, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Value	Type: Signed 32-bit
Min:-65535	Max: 65535

Where:

cc = Motor channel
nn = Speed value in RPM

Example:

!S 2500 : set motor 1 position velocity to 2500 RPM

STT - STO Self-Test

Alias: - HexCode: 70 CANOpen id: -

Description:

With this command the STO Self-Test process is executed in order to check whether there is a fault. This process is applicable only on motor controllers with STO circuit implemented on their board. The result of the test is taken back using the STT query. Along with the STO self-test an extra test is also taken place in order to detect whether either of the power MOSFETs are shorted (so damaged) or not. This test takes place even if the STO feature is disabled and during power-up. In case of STO fault the Respective STO Fault bit in the Fault Flags is set. The STO fault is triggered when:

- Any of the transistors or other component of the STO circuit is damaged.
- The respective jumper is placed on the board.

In case of MOSFET fault the MOSFail and Estop bits in fault flags and the FETs Off bit in status flags are set. This fault is cleared only when the STT runs again and the MOSFET failure goes away.

Syntax Serial: !STT

Syntax Scripting: setcommand(_STT, 1)

Number of Arguments: 0

SX - Next Velocity

Alias: NXTVEL HexCode: 17 CANOpen id: 0x2014

Description:

This command is used in Position Count mode. It is similar to S except that it stores a velocity value in a buffer. This value will become the next velocity the controller will use and becomes active upon reaching a previous desired position. If omitted, the command will be chained using the last used velocity value. See Position Command Chaining in manual.

Syntax Serial: !SX cc nn

Syntax Scripting: setcommand(_SX, cc, nn)
setcommand(_NXTVEL, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Value Type: Signed 32-bit
Min: -500000 Max: 500000

Where:

cc = Motor channel
nn = Velocity value

VAR - Set User Variable

Alias: VAR HexCode: 06 CANOpen id: 0x2005

Description:

This command is used to set the value of user variables inside the controller. These variables can be then read from within a user MicroBasic script to perform specific actions. The total number of variables depends on the controller model and can be found in the product datasheet. Variables are signed 32-bit integers.

Syntax Serial: !VAR nn mm
Syntax Scripting: setcommand(_VAR, nn, mm)
setcommand(_VAR, nn, mm)

Number of Arguments: 2

Argument 1: VarNbr
Min: 1 Max: Total nbr of User Variables

Argument 2: Value Type: Signed 32-bit
Min: -2147M Max: 2147M

Where:

nn = Variable number
mm = Value

DS402 Runtime Commands

Runtime commands created to support DS402 specification are described below:

TABLE 15-2.

Command	Arguments	Description
CW	Channel Value	Control Word (DS402)
FEW	Element Value	Following Error Window (DS402)
FET	Element Value	Following Error Time Out (DS402)
HMD	Channel Value	Homing Method (DS402)
HSP	Element Value	Homing Speed (DS402)
INT	Element Value	Interpolation Time Period (DS402)
MSL	Element Value	Max Motor Speed (DS402)
PAC	Channel Value	Profile Acceleration (DS402)
PDC	Channel Value	Profile Deceleration (DS402)
PLT	Element Value	Software Position Limit (DS402)
POF	Channel Value	Position Offset (DS402)
POS	Channel Value	Target Position (DS402)
PSP	Channel Value	Profile Velocity (DS402)
ROM	Channel Value	Modes of Operation (DS402)
SPE	Channel Value	Target Velocity (DS402)
SAC	Element Value	Velocity Acceleration (DS402)

Command	Arguments	Description
SDC	Element Value	Velocity Deceleration (DS402)
SPC	Channel Value	Target Profile Velocity (DS402)
SPL	Element Value	Velocity Min/Max Amount (DS402)
TC	Channel Value	Target Torque (DS402)
TOF	Channel Value	Torque Offset (DS402)
TSL	Channel Value	Torque Slope (DS402)
VOF	Channel Value	Velocity Offset (DS402)

CW – Control Word (DS402)

Alias: CW HexCode: 56 CANOpen id: 0x6040

Description:

This command controls the Power Drive System-Finite State Automation (PDS-FSA), which is the state machine as defined by the DS402 standard. Bits 9, 6, 5, and 4 of the ControlWord are operation mode specific. The halt function (bit 8) behavior is operation mode specific. If the bit is 1, the commanded motion shall be interrupted (the motor will start slowing down until it stops), after releasing the halt function, the commanded motion shall be continued if possible, see Table 15-5.

TABLE 15-3. Control Word Mapping

15 … 11	10	9	8	7	6 … 4	3	2	1	0
R	R	OMS	H	FR	OMS	EO	QS	EV	SO
MSB									LSB

R → Reserved, OMS → Operation mode specific, H → Halt, FR → Fault reset,
EO → Enable operation QS → Quick stop, EV → Enable voltage, and SO → Switch on

TABLE 15-4. Command Coding

Command	Bits of the Control Word					Transition
	Bit 7	Bit 3	Bit 2	Bit 1	Bit 0	
Shutdown	0	X	1	1	0	2,6,8
Switch On	0	0	1	1	1	3
Switch On + Enable Operation	0	1	1	1	1	3+4
Disable Voltage	0	X	X	0	X	7,9,10,12
Quick Stop	0	X	0	1	X	7,10,11
Disable Operation	0	0	1	1	1	5
Enable Operation	0	1	1	1	1	4,16
Fault Reset	0->1	X	X	X	X	15

TABLE 15-5. Halt bit (bit 8)

Bit	Value	Definition
8	0	Positioning shall be executed or continued
	1	Axis shall be stopped. Slow down on quick stop ramp (EDEC) and stay in operation enabled

Profile Position Mode

TABLE 15-6. Control Word Mapping in Profile Position Mode

15 10	9	8	7	6	5	4	3 0
see Table 15-3	Reserved	Halt	see Table 15-3	Abs/rel	Change Set Immediately	New Set Point	see Table 15-3
MSB				LSB			

In Profile Position Mode the operation specific bits are mapped in Table 15-7. With bits 4 and 5, user can define when the command for next Position (0x607A - POS) will be processed. Bit 6 defines whether the command is absolute or relative to the current position.

TABLE 15-7. Definition of Bits 4, 5, 6, and 9 in Profile Position Mode

Bit 5	**Bit 4**	**Definition**
1	0->1	Next positioning shall be started immediately
0	0->1	Positioning with the current profile velocity up to the current set-point shall be proceeded and then next positioning shall be applied
Bit	**Value**	**Definition**
6	0	Target position shall be an absolute value
	1	Target position shall be a relative value. Positioning moves shall be performed relative to the preceding (internal absolute) target position

Velocity Mode

TABLE 15-8. Control Word Mapping in Velocity Mode

15 9	8	7	6	5	4	3 0
see Table 15-3	Halt	see Table 15-3	Reference Ramp	Unlock Ramp	Enable Ramp	see Table 15-3
MSB						LSB

In Velocity Mode the operation specific bits are mapped on Table 15-9. With bits 4, 5 and 6, user can configure the available ramp related options as shown in Table 15-9.

TABLE 15-9a Definition of Bits 4, 5, and 6 in Velocity Mode

Bit	**Value**	**Definition**
4	0	Motor shall be halted. Slow down on quick stop ramp (EDEC) and stay in operation enabled
	1	Velocity demand value shall accord with ramp output value
5	0	Ramp output value shall be locked to current output value
	1	Ramp output value shall follow ramp input value
6	0	Ramp input value shall be set to zero
	1	Ramp input value shall accord with ramp reference
Note: Bit 4 has got higher priority than bit 5.		

Homing Mode

TABLE 15-9b. control word mapping in homing mode

15	10	9	8	7	6	5	4	3	0
see Table 15-8			Halt	see Table 15-8	Reserved	Reserved	Start Homing	see Table 15-8	
MSB					LSB				

In Homing Mode, the operation specific bits are mapped in Table 15-9b. With bit 4 the the homing mode will be started and with bit 8 it will be halted.

TABLE 15-9c. Definition of bit 8

Bit	Value	Definition
4	0	Do not start homing procedure
	1	Start or continue homing procedure
8	0	The motion shall be executed or continued
	1	Axis shall be stopped according to the halt option code (605Dh)

Other Modes

Those modes use some bits of the controlword. Table 15-10 shows the structure of the controlword. Table 15-11 defines the values for bit 8 of the controlword.

TABLE 15-10. Controlword for profile torque mode

15 9	8	7	6 4	3 0
See Table 15-3	Halt	See Table 15-3	reserved	See Table 15-3
MSB				LSB

TABLE 15-11. Definition of bit 8

Bit	Value	Definition
8	0	The motion shall be executed or continued
	1	Axis shall be stopped according to the halt option code (605Dh)
Note: At cyclic synchronous modes Halt function is not active		

Syntax Serial: !CW cc nn

Syntax Scripting: SetCommand(_CW, cc, nn)

Arguments: 2

Argument 1: Channel

Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Argument 2: Value

Type: Unsigned 16-bit

Where:

cc = Motor channel

nn = Control word value

FEW - Following Error Window (DS402)

Alias: FEW HexCode:99 CanOpen id: 0x6065

Description:

This command indicates the configured range of the tolerated position values symmetrically to the position demand value. If the position actual value is out of the following error window, a following error occurs. If the value of the following error window is FFFF FFFFh, the following control is disabled. Based on DS402 standard this command is only applicable is position modes.

Syntax Serial: !FEW cc nn

Syntax Scripting: SetCommand(_FEW, cc, nn)

Arguments: 2

Argument 1: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total of motors
Argument 2: Value	Type: Unsigned 32-bit

Where:

cc = Motor channel

nn = Following Error Window (counts)

FET - Following Error Time Out (DS402)

Alias: FET HexCode:9A CanOpen id: 0x6066

Description:

This command shall indicate the configured time for a following error condition, after that the bit 13 of the statusword shall be set to 1. When the following error occurs, the controller goes to quick stop operation mode. The value is given in ms.

Syntax Serial: !FET cc nn

Syntax Scripting: SetCommand(_FET, cc, nn)

Arguments: 2

Argument 1: Channel	Type: Unsigned 8-bit

Min: 1 Max: Total of motors

Argument 2: Value Type: Unsigned 16-bit

Where:

cc = Motor channel

nn = Following Error Time out (ms)

HMD – Homing Method (DS402)

Alias: - HexCode: AD CANOpen id: 0X6098

Description:

This parameter selects the Homing Method that will be used. Supported methods are 17-30 and 35. For more details go to DS402 standard.

Syntax Serial: !HMD cc nn

Syntax Scripting: setcommand(_HMD, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Homing Method number

Type: Unsigned 8-bit

Min: 0 Max: 255

Where:

cc = Motor Channel

nn =Homing Method

Example:

!HMD 1 18: Selects Homing Method 18 for Motor channel 1

HSP – Homing Speed (DS402)

Alias: - HexCode: AE CANOpen id: 0X6099

Description:

This parameter sets the speed that will be used during the homing procedure. Each channel has 2 speed settings. The first is the speed during search for Home switch and the second is the speed during search for Index pulse (currently not supported).

Syntax Serial: !HSP cc nn

Syntax Scripting: setcommand(_HSP, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: 2 * Total Number of Motors

Argument 2: Homing Speed (RPM)

Type: Unsigned 32-bit

Min: 0 Max: 20000

Where:

cc=

1: Homing speed during search for Home switch for ch1

2: Homing speed during search for Index pulse (currently not supported) for ch1

3: Homing speed during search for Home switch for ch2

4: Homing speed during search for Index pulse (currently not supported) for ch2

nn =Homing Speed

Example:

!HSP 1 200: Selects Homing Speed (search for Home Switch) 200 (RPM) for Motor channel 1.

!HSP 3 250: Selects Homing Speed (search for Home Switch) 250 (RPM) for Motor channel 2.

INT - Interpolation Time Period (DS402)

Alias: INT HexCode:9C CanOpen id: 0x60C2

Description:

This command indicates the configured interpolation cycle time. The interpolation time period value is given in (interpolation time base)x10^(interpolation time index) s(seconds). The interpolation time index is dimensionless. The default value of the interpolation time is 0.001 s (1 millisecond, meaning Interpolation Time Base = 1 and Interpolation Time Index = -3).

Syntax Serial: !INT ee nn

Syntax Scripting: SetCommand(_INT, ee, nn)

Arguments: 2

Argument 1: Element Type: Unsigned 8-bit

Min: 1 Max: 2 × Total number of motors

Argument 2: Value Type:	Interpolation Time Base: Unsigned 8 bits
	Min: 0 Max: 255
	Interpolation Time Index: Signed 8 bits
	Min: -3 Max: 7

Where:

ee =

1: nn = Interpolation time base channel 1

2: nn = Interpolation time index channel 1

3: nn = Interpolation time base channel 2

4: nn = Interpolation time index channel 2

…

2 × (m - 1) + 1: nn = Interpolation time base channel m.

2 × (m - 1) + 2: nn = Interpolation time index channel m.

MSL - Max Motor Speed (DS402)

Alias: MSL HexCode: 66 CanOpen id: 0x6080

Description:

This command indicate the configured maximal allowed speed for the motor in either direction. The value is given in rotations per minute (r/min).

Syntax Serial: !MSL cc nn

Syntax Scripting: SetCommand(_MSL, cc, nn)

Arguments: 2

Argument 1: Channel	Type: Unsigned 8-bit
	Min: 1 Max: Total of motors

Argument 2: Value Type: Signed 32-bit

Where:

cc = Motor channel

nn = Speed command in RPM

PAC – Profile Acceleration (DS402)

Alias: PAC HexCode: 5E CANOpen id: 0x6083

Description:

This command is used to set the configured acceleration in 10×RPM/second.

Syntax Serial: !PAC cc nn

Syntax Scripting: SetCommand(_PAC, cc, nn)

Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Argument 2 Value Type: Unsigned 32-bit

Where:

cc = Motor channel

nn = Profile acceleration in 10×RPM/second

PDC – Profile Deceleration (DS402)

Alias: PDC HexCode: 5F CANOpen id: 0x6084

Description:

This command is used to set the configured deceleration in 10×RPM/second.

Syntax Serial: !PDC cc nn

Syntax Scripting: SetCommand(_PDC, cc, nn)

Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Argument 2: Value Type: Unsigned 32-bit

Where:

cc = Motor channel

nn = Profile deceleration in 10×RPM/second

PLT - Software Position Limit (DS402)

Alias: PLT HexCode:9D CANOpen id: 0x607D

Description:

This command indicates the configurated maximal and minimal software position limits. These parameters define the absolute position limits for the position demand value and the position actual value. Every new target has been checked against these limits. To disable the software position limits, the min position limit (1st element) and the max position limit (2nd element) shall be set to 0. The positions limits is given in same position units as the target position.

Syntax Serial: !PLT ee nn

Syntax Scripting: SetCommand(_PLT, ee, nn)

Arguments: 2

Argument 1: Element Type: Unsigned 8-bit

Min: 1 Max: 2 × Total number of motors

Argument 2: Value Type: Unsigned 8-bit or Signed 8-bit

Where:

ee =

1: Minimum Position Limit channel 1

2: Maximum Position Limit channel 1

3: Minimum Position Limit channel 2

4: Maximum Position Limit channel 2

…

2 × (m - 1) + 1: Minimum Position Limit channel m.

2 × (m - 1) + 2: Maximum Position Limit channel m.

where m = number of motor

POF – Position Offset

Alias: - HexCode: B0 CANOpen id: 0x60B0

Description:
This runtime command is used to set an offset in the position command. In relative position modes the allowed offset values are between -1000 and 1000.

Syntax Serial: !POF cc nn

Syntax Scripting: setcommand (_POF, cc, nn)

Number of Arguments: 2

Argument 1: Motor	Type: Unsigned 8-bits Min: 1 Max: Total Number of Motors
Argument 2: Position Offset	Type: Signed 32-bit Min: -2,000,000,000 Max: 2,000,000,000 Default: 0

Where:
cc = Channel
nn = Position counts

Example:
!POF 1 230000: Set motor channel 1 position offset to 230000.

POS – Target Position (DS402)

Alias: POS HexCode: 5C CANOpen id: 0x607A

Description:

This command is used to set the commanded position that the drive should move to in position profile mode using the current settings of motion control parameters such as velocity, acceleration, deceleration, motion profile type etc. The value is interpreted as absolute or relative depending on the abs/rel flag in the Control Word. Take into consideration that this command is active only when DS402 mode is enabled.

Syntax Serial !POS cc nn

Syntax Scripting SetCommand(_POS, cc, nn)

Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Argument 2: Value Type: Signed 32-bit

Where:

cc = Motor channel

nn = Target position

PSP – Profile Velocity (DS402)

Alias: PSP HexCode: 5D CANOpen id: 0x6081

Description:

This command is used to set the velocity in RPM, normally attained at the end of the acceleration ramp during a profiled motion and is valid for both directions of motion.

Syntax Serial: !PSP cc nn

Syntax Scripting: SetCommand(_PSP, cc, nn)

Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Argument 2: Value Type: Unsigned 16-bit

Where:

cc = Motor channel
nn = Profile velocity

ROM – Modes of Operation (DS402)

Alias: ROM HexCode: 5A CANOpen id: 0x6060

Description:
This command configures the modes of operation.

Syntax Serial: !ROM cc nn

Syntax Scripting: SetCommand(_ROM, cc, nn)

Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Argument 2: Value Type: Signed 8-bit

Where

cc = Motor channel
nn = Modes of operation (see the table below)

TABLE 15-12. Operation Modes

Value	Definition	Roboteq Operation Mode
-4[1]	Velocity Mode	Closed Loop Speed Position
-3[1]	Profile Velocity Mode	Closed Loop Speed Position
-2[1]	Profile Position Mode	Closed Loop Position Tracking Mode[2]
-1[1]	Profile Position Mode	Closed Loop Position Relative Mode$_2$
0	No Mode	Open Loop Mode
1	Profile Position Mode	Closed Loop Count Position Mode
2	Velocity Mode	Closed Loop Speed Mode
3	Profile Velocity Mode	Closed Loop Speed Mode
4	Torque Profile Mode	Closed Loop torque Mode

Value	Definition	Roboteq Operation Mode
8	Cyclic Synchronous Position Mode	Closed Loop Count Position Mode
9	Cyclic Synchronous Velocity Mode	Closed Loop Speed Mode
10	Cyclic Synchronous Torque Mode	Closed Loop Torque Mode
[1]Roboteq Specific Modes [2]Not all Profile Position features can be supported with this mode.		

RST – Reset Controller

Alias: - HexCode: 23 CANOpen id: 0x2021

Description:

Initiating this command resets the controller, mirroring the effects of a power cycle: firmware reinitializes, runtime variables clear, and temporary settings are lost. Exercise caution due to these substantial impacts. The command is similar to the %RESET maintenance command.

Syntax Serial: !RST

Syntax Scripting: setcommand(_EX, 1)

setcommand(_ESTOP, 1)

Number of Arguments: 0

S16 –Target Velocity (DS402)

Alias: MOTVEL HexCode: 61 CANOpen id: 0x6042

Description:

Sets the required velocity of the system in RPM. Positive values shall indicate forward direction and negative values shall indicate reverse direction. It is applicable to velocity mode.

Syntax Serial: !S16 cc, nn

Syntax Scripting: SetCommand(_S16, cc, nn)

SetCommand(_MOTVEL, cc, nn)

Arguments: 2

Argument 1: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total number of motors
Argument 2: Value	Type: Signed 16-bit
Min: -500000	Max: 500000

Where:

cc = Motor channel

nn = Target velocity in RPM

SAC – Velocity Acceleration (DS402)

Alias: SAC HexCode: 58 CANOpen id: 0x6048

Description:

This command configures the velocity acceleration.

Syntax Serial: !SAC ee nn

Syntax Scripting: SetCommand(_SAC, ee, nn)

Arguments: 2

Argument 1: Element	Type: Unsigned 8-bit
Min: 1	Max: 2 × Total number of motors
Argument 2: Value	Type: Unsigned 32-bit

Where:

ee =

1: Delta speed in 10×RPM for channel 1

2: Delta time in seconds for channel 1

3: Delta speed in 10×RPM for channel 2

4: Delta time in seconds for channel 2

…

2 × (m - 1) + 1: Delta speed in 10×RPM for channel m.

2 × (m - 1) + 1: Delta time in seconds for channel m.

nn = Delta speed/time

SDC – Velocity Deceleration (DS402)

Alias: SDC HexCode: 59 CANOpen id: 0x6049

Description:

This command configures the velocity deceleration.

Syntax Serial: !SDC ee nn

Syntax Scripting: SetCommand(_SDC, ee, nn)

Arguments: 2

Argument 1: Element	Type: Unsigned 8-bit
Min: 1	Max: 2 × Total number of motors

Argument 2: Value	Type: Unsigned 32-bit

Where:

ee =

1: Delta speed in 10×RPM for channel 1

2: Delta time in seconds for channel 1

3: Delta speed in 10×RPM for channel 2

4: Delta time in seconds for channel 2

…

2 × (m - 1) + 1: Delta speed in 10×RPM for channel m.

2 × (m - 1) + 1: Delta time in seconds for channel m.

nn = Delta speed/time

SPC - Target Profile Velocity (DS402)

Alias: SPC HexCode: 66 CanOpen id: 0x60FF

Description:

Sets the required velocity of the system in RPM. Positive values shall indicate forward direction and negative values shall indicate reverse direction. This command is applicable to profile velocity and cyclic synchronous velocity modes.

Syntax Serial: !SPC cc nn

Syntax Scripting: SetCommand(_SPC, cc, nn)

Arguments: 2

Argument 1: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total of motors
Argument 2: Value	Type: Signed 32-bit

Where:

cc = Motor channel

nn = Speed command in RPM

SPL – Velocity Min/Max Amount (DS402)

Alias: SPL HexCode: 57 CANOpen id: 0x6046

Description:

This command configures the minimum and maximum amount of velocity in RPM. The vl velocity max amount is mapped internally to the vl velocity max positive and vl velocity max negative values. The vl velocity min amount is be mapped internally to the vl velocity min positive and vl velocity min negative values as shown in Figure 15-1. It is applicable to Velocity mode.

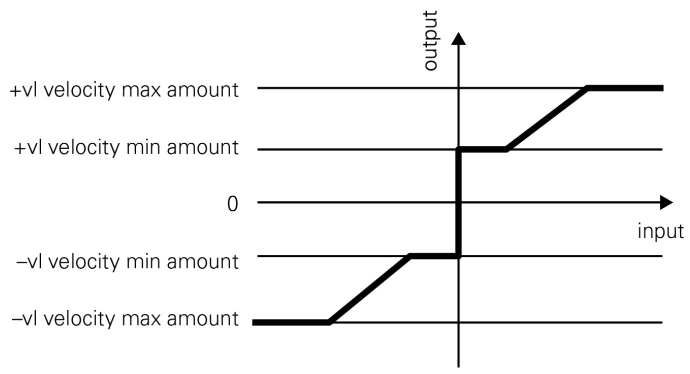


FIGURE 15-1. Velocity Min Max Amount

Syntax Serial: !SPL ee nn

Syntax Scripting: SetCommand(_SPL, ee, nn)

Arguments: 2

Argument 1: Eelment	Type: Unsigned 8-bit
Min: 1	Max: 2 × Total number of motors
Argument 2: Value	Type: Unsigned 32-bit

Where:

ee =

1: Min amount for channel 1
2: Max amount for channel 1
3: Min amount for channel 2
4: Max amount for channel 2
…
2 × (m - 1) + 1: Min amount for channel m.
2 × (m - 1) + 2: Max amount for channel m.
nn = Velocity max/min amount

TC – Target Torque (DS402)

Alias: TC HexCode: 5B CANOpen id: 0x6071

Description:

This command configures the target torque command, applicable when the controller operates in torque mode. Its value is expressed in thousandths of rated torque, which means that a torque command of 1000 will drive the motor up to the rated torque. Beware that in order to have correct results, the configuration commands nominal current (NOMA) and torque constant (TNM) must be set appropriately.

Syntax Serial: !TC cc nn

Syntax Scripting: SetCommand(_TC, cc, nn)

Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Argument 2: Value Type: Signed 16-bit

Where:

cc = Motor channel

nn = Torque input value in thousandths of rated torque

TOF – Torque Offset

Alias: - HexCode: B2 CANOPEN id: 0x60B2

Description:
This runtime command is used to set an offset in the commanded torque. Commanded torque could be either directly from user (in torque mode), or produced from speed or position control loops. The value is in miliNm (Nm * 1000). Beware in order to have correct values make sure to have configured appropriately configuration command TNM (torque constant).

Syntax Serial: !TOF cc nn

Syntax Scripting: setcommand (_TOF, cc, nn)

Number of Arguments: 2

Argument 1: Motor Type: Unsigned 8-bits
Min: 1 Max: Total Number of Motors

Argument 2: Toque Offset Type: Signed 32-bit
Min: -2,000,000,000 Max: 2,000,000,000 Default: 0

Where:

cc = Channel

nn = Torque offset in Nm * 1000.

Example:

!TOF 1 300: Set motor channel 1 torque offset to 0.3 Nm.

TSL – Torque Slope (DS402)

Alias: TSL HexCode: 60 CANOpen id: 0x6087

Description:

This command is used to set the rate of change of command. Beware in order to have correct values make sure to have configured appropriately configuration command TNM.

Syntax Serial: !TSL cc nn

Syntax Scripting: SetCommand(_TSL, cc, nn)

Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit

Min: 1 Max: otal number of motors

Argument 2: Value Type: Unsigned 32-bit

Where:

cc = Motor channel

nn = Torque slope in (miliNm*10)/ sec.

VOF – Velocity Offset

Alias: - HexCode: B1 CANOPEN id: 0x60B1

Description:

This runtime command is used to set an offset in the velocity command. Velocity command could be either directly set from user, in speed mode operation, or produced from position control loop.

.

Syntax Serial: !VOF cc nn

Syntax Scripting: setcommand (_VOF, cc, nn)

Number of Arguments: 2

Argument 1: Motor Type: Unsigned 8-bits

Min: 1 Max: Total Number of Motors

Argument 2: Velocity Offset Type: Signed 32-bit

Min: -2,000,000,000 Max: 2,000,000,000 Default: 0

Where:

cc = Channel

nn = Velocity offset in RPM

Example:

!VOF 1 250: Set motor channel 1 velocity offset to 250 RPM.

Runtime Queries

Runtime queries can be used to read the value of real-time measurements at any time during the controller operation. Real-time queries are very short commands that start with "**?**" followed by one to three letters. In some instances, queries can be sent with or without a numerical parameter.

Without parameter, the controller will reply with the values of all channels. When a numerical parameter is sent, the controller will respond with the value of the channel selected by that parameter.

Example:

Q:**?T**
R: **T=20:30:40**

Q: **?T2**
R: **T=30**

All queries are stored in a history buffer that can be made to automatically recall the past 16 queries at a user-selectable time interval. See "Query History Commands" on page 295.

Routine queries can be sent from within a MicroBasic Script using the getvalue() function.

TABLE 15-13. Runtime Queries

Command	Argument	Description
A	Channel	Read Motor Amps
AI	InputNbr	Read Analog Inputs
AIC	InputNbr	Read Analog Input after Conversion
ANG	Channel	Read Rotor Angle
ASI	Channel	Read Raw Sin/Cos sensor
B	VarNbr	Read User Boolean Variable
BA	Channel	Read Battery Amps
BCR	Channel	Read Internal Sensor Count Relative
BRK	Channel	Read Brake Override status
BMC	SensorValue	Read BMS State Of Charge in AmpHours
BMF	SensorValue	Read BMS status flags
BMS	SensorValue	Read BMS switch states
BS	Channel	Read Internal Sensor Motor Speed in RPM
BSC	SensorNumber	Read Battery State of Charge in percentage
BSR	Channel	Read Internal Sensor Motor Speed as 1/1000 of Max RPM
C	Channel	Read Encoder Counter Absolute
CAN	Element	Read Raw CAN frame
CB	Channel	Read Absolute Internal Sensor Counter
CD	None	Read Raw Redirect Received Frames Count
CEC	Element	CAN Error Counter
CF	None	Read Raw CAN Received Frames Count

TABLE 15-13. Runtime Queries

Command	Argument	Description
CHS	Channel	CAN Consumer Heartbeat Status
CIA	Channel	Read Converted Analog Command
CIG	PID Channel Gain	Read Current Integral Gains
CIP	Channel	Read Internal Pulse Command
CIS	Channel	Read Internal Serial Command
CL	Group	Read RoboCAN Alive Nodes Map
CPG	PID Channel Gain	Read Current Proportional Gains
CR	Channel	Read Encoder Count Relative
CSR	Channel	Read Relative SSI Sensor Counter
CSS	Channel	Read Absolute SSI Sensor Counter
D	None	Read Digital Inputs
DI	InputNbr	Read Individual Digital Inputs
DDT	Element	Read Raw Redirect Received Frame
DG	PID Channel Gain	Read PID Derivative Gains
DO	None	Read Digital Output Status
DPA	Channel	Read Motor DC/Peak Amps
DR	Channel	Read Destination Reached
E	Channel	Read Closed Loop Error
F	Channel	Read Feedback
FC	Channel	Read FOC Angle Adjust
FLW	SensorNumber	Read Flow Sensor Counter
FF	None	Read Fault Flags
FID	None	Read Firmware ID
FIN	None	Read Firmware ID (numerical)
FM	Channel	Read Runtime Status Flag
FS	None	Read Status Flags
HS	Channel	Read Hall Sensor States
ICL	NodeId	Is RoboCAN Node Alive
IG	PID Channel Gain	Read PID Integral Gains
LK	None	Read Lock status
M	Channel	Read Motor Command Applied
MA	AmpsChannel	Read Field Oriented Control Motor Amps
MCB	SensorNumber	Read Magsensor Markers Pattern
MCU	None	Microprocessor Usage
MGD	SensorNumber	Read Magsensor Track Detect
MGM	SensorNumber	Read Magsensor Markers
MGS	SensorNumber	Read Magsensor Status

TABLE 15-13. Runtime Queries

Command	Argument	Description
MGT	Channel	Read Magsensor Track Position
P	Channel	Read Motor Power Output Applied
PG	PID Channel Gain	Read PID Proportional Gains
PHA	CurrentSensorNumber	Read Phase Amps
PI	InputNbr	Read Pulse Inputs
PIC	InputNbr	Read Pulse Input after Conversion
S	Channel	Read Encoder Motor Speed in RPM
SCC	None	Read Script Checksum
SEC	Channel	Read Sensor Errors
SDT	Element	Read Raw Redirect Received Frame as string
SNA	Channel	Read Sensor Angle
SNS	Motor Phase Number	Sense Voltage
SR	Channel	Read Encoder Speed Relative
SS	Channel	Read SSI Sensor Motor Speed in RPM
SSR	Channel	Read SSI Sensor Speed Relative
STT	Fault Indication	STO Self-Test Result
T	SensorNbr	Read Temperature
TM	Element	Read Time
TR	Channel	Read Position Relative Tracking
TRN	None	Read Control Unit type and Controller Model
UID	Element	Read MCU Id
V	SensorNumber	Read Volts
VAR	VarNumber	Read User Integer Variable

A - Read Motor Amps

Alias: MOTAMPS HexCode: 00 CANOpen id: 0x2100

Description:

Measures and reports the motor Amps, in Amps*10, for all operating channels. For brushless controllers this query reports the RMS value. Note that the current flowing through the motors is often higher than this flowing through the battery.

Syntax Serial: ?A [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_A, cc)
result = getvalue(_MOTAMPS, cc)

Reply:

A = aa Type: Signed 16-bit Min: 0

Where:

cc = Motor channel
aa = Amps *10 for each channel

Example:

Q: ?A
R: A=100:200
Q: ?A 2
R: A=200

Note:

Single channel controllers will report a single value. Some power board units measure the Motor Amps and calculate the Battery Amps, while other models measure the Battery Amps and calculate the Motor Amps. The measured Amps is always more precise than the calculated Amps. See controller datasheet to find which Amps is measured by your particular model.

AI - Read Analog Inputs

Alias: ANAIN HexCode: 10 CANOpen id: 0x2146

Description:

Reports the raw value in mV of each of the analog inputs that are enabled. Input that is disabled will report 0. The total number of Analog input channels varies from one controller model to another and can be found in the product datasheet.

Syntax Serial: ?AI [cc]

Argument: InputNbr
Min: 1 Max: Max Number of Analog Inputs

Syntax Scripting: result = getvalue(_AI, cc)
result = getvalue(_ANAIN, cc)

Reply:

AI=nn Type: Signed 16-bit Min: 0 Max: 5300

Where:

cc = Analog Input number
nn = Millivolt for each channel

AIC - Read Analog Input after Conversion

Alias: ANAINC HexCode: 23 CANOpen id: 0x2147

Description:

Returns value of an Analog input after all the adjustments are performed to convert it to a command or feedback value (Min/Max/Center/Deadband/Linearity). If an input is disabled, the query returns 0. The total number of Analog input channels varies from one controller model to another and can be found in the product datasheet.

Syntax Serial:	?AIC [cc]
Argument:	InputNbr Min: 1 Max: Total Number of Analog Inputs

Syntax Scripting:	result = getvalue(_AIC, cc)
	result = getvalue(_ANAINC, cc)

Reply:

AIC=nn Type: Signed 16-bit Min: -1000 Max: 1000

Where:

cc = Analog Input number
nn = Converted analog input value +/-1000 range

ANG - Read Rotor Angle

Alias: ANG HexCode: 42 CANOpen id: 0x2132

Description:

On brushless controller operating in sinusoidal mode, this query returns the real time value of the rotor's electrical angle of brushless motor. This query is useful for verifying troubleshooting sin/cos and SPI/SSI sensors. Angle are reported in 0-511 degrees.

Syntax Serial:	?ANG [cc]
Argument:	Channel Min: 1 Max: Total Number of Motors
Syntax Scripting:	result = getvalue(_ANG, cc)

Reply:

ANG=nn Type: Unsigned 16-bit Min: 0 Max: 511

Where:

cc = Motor channel
nn = Rotor electrical angle

ASI - Read Raw Sin/Cos sensor

Alias: ASI HexCode: 33 CANOpen id:

Description:

Returns real time raw values of ADC connected to sin/cos sensors of each motor or the real time values of the raw data reported by the SSI sensor of the motor. This query is useful for verifying troubleshooting sin/cos sensors and SSI sensors.

Syntax Serial:	?ASI [cc]
Argument:	Channel Min: 1 Max: 2 * Number of Motors

Syntax Scripting: result = getvalue(_ASI, cc)

Reply:

ASI=nn Type: Unsigned 16-bit Min: 0 Max: 65535

Where:

cc =
1 : Sin input 1/SSI input 1
2 : Cos input 1
3 : Sin input 2/SSI input 2
4 : Cos input 2
nn = ADC value

B - Read User Boolean Variable

Alias: BOOL HexCode: 16 CANOpen id: 0x2115

Description:

Read the value of boolean internal variables that can be read and written to/from within a user MicroBasic script. It is used to pass boolean states between user scripts and a microcomputer connected to the controller. The total number of user boolean variables varies from one controller model to another and can be found in the product datasheet.

Syntax Serial: ?B [nn]

Argument: VarNbr
Min: 1 Max: Total Number of Bool Variables

Syntax Scripting: result = getvalue(_B, nn)
result = getvalue(_BOOL, nn)

Reply:

B=bb Type: Boolean Min: 0 Max: 1

Where:

nn = Boolean variable number
bb = 0 or 1 state of the variable

BA - Read Battery Amps

Alias: BATAMPS HexCode: 0C CANOpen id: 0x210C

Description:

Measures and reports the Amps flowing from the battery in Amps * 10. Battery Amps are often lower than motor Amps.

Syntax Serial: ?BA [cc]

Argument: Channel:
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_BA, cc)
result = getvalue(_BATAMPS, cc)

Reply:

BA=aa Type: Signed 16-bit Min: 0

Where:

cc = Motor channel
aa = Amps *10 for each channel

Example:

Q: ?BA
R: BA=100:200

Note:

Some controller models measure the Motor Amps and Calculate the Battery Amps, while other models measure the Battery Amps and calculate the Motor Amps. The measured Amps is always more precise than the calculated Amps. See controller datasheet to find which Amps is measured by your particular model.

BCR - Read Internal Sensor Count Relative

Alias: BLRCNTR HexCode: 09 CANOpen id: 0x2109

Description:

Returns the amount of Internal sensor (Hall, SinCos, Resolver) counts that have been measured from the last time this query was made. Relative counter read is sometimes easier to work with, compared to full counter reading, as smaller numbers are usually returned. If the query is used via RoboCAN, the query will be refreshed by default every 20ms, reseting the counter. Therefore a second argument has been introduced dictating the refresh rate.

Syntax Serial: ?BCR [cc, nn]

Arguments: 2 or 1

Argument 1:	Channel Min: 1 Max: Total Number of Motors
Argument 2:	Time in millisecond, refresh rate (only if used in RoboCAN).

BCR= nn,cc Type: Signed 32-bit Min: 2147M Max: 2147M

Where:

cc=Motor channel
nn=value

Example:

?BCR 1: Ask for BCR for channel 1

@01?BCR 1 0: Ask from node 1 for BCR for channel 1 only once, the rate value is zero.

@02?BCR 1 100: Ask from node 2 for BCR for channel 1 with rate of 100ms.

BMC - Read BMS State Of Charge in AmpHours

Alias: - HexCode: 4C CANOpen id: 0x2141

Description:

When one or more BMS10X0 are connected to the controller, this query reports the Battery's State Of Charge in AmpHours, which is connected to the respective BMS10X0. If only one BMS10X0 is connected to any pulse input this query will report the data of that device, regardless which pulse input it is connected to. If more than one BMS10X0 is connected to pulse inputs and these inputs are enabled and configured in BMS MultiPWM mode, then the argument following the query is used to select the sensor.

Syntax Serial: ?BMC [cc]

Argument: SensorNumber

Min: None Max: Total Number of Pulse Inputs

Syntax Scripting: result = getvalue(_BMC, cc)

Reply:

BMC=nn Type: Unsigned 8-bit Min: 0 Max: 255

Where:

cc = (When only one sensor enabled)

None or 1 : Current BMS10X0

cc = (When several sensors enabled)

1 : BMS10X0 at pulse input 1

2 : BMS10X0 at pulse input 2

...

p : BMS10X0 at pulse input p

nn = AmpHours (Ah)

BMF - Read BMS status flags

Alias: - HexCode: 4D CANOpen id: 0x2142

Description:

When one or more BMS10X0 are connected to the controller, this query reports the status flags of the respective BMS10X0. If only one BMS10X0 is connected to any pulse input this query will report the data of that device, regardless which pulse input it is connected to. If more than one BMS10X0 is connected to pulse inputs and these inputs are enabled and configured in BMS MultiPWM mode, then the argument following the query is used to select the sensor.

Syntax Serial: ?BMF [cc]

Argument:	SensorNumber	
	Min: None	Max: Total Number of Pulse Inputs

Syntax Scripting: result = getvalue(_BMF, cc)

Reply:

BMF = f1 + f2*2 + f3*4 + ... + fn*2^n-1 Type: Unsigned 8-bit Min: 0 Max: 255

Where:

cc = (When only one sensor enabled)

None or 1 : Current BMS10X0

cc = (When several sensors enabled)

1 : BMS10X0 at pulse input 1

2 : BMS10X0 at pulse input 2

...

p : BMS10X0 at pulse input p

and

f1 = Unsafe Temperature

f2 = Over or Under Voltage Error Set

f3 = Amp Trigger Set

f4 = Over Current Error Set

f5 = Short Load or Inv Charger

f6 = Bad State Of Health

f7 = Config Error

f8 = Internal Fault

BMS - Read BMS switch states

Alias: -	HexCode: 4E	CANOpen id: 0x2143

Description:

When one or more BMS10X0 are connected to the controller, this query reports the switch states of the respective BMS10X0. If only one BMS10X0 is connected to any pulse input this query will report the data of that device, regardless which pulse input it is connected to. If more than one BMS10X0 is connected to pulse inputs and these inputs are enabled and configured in BMS MultiPWM mode, then the argument following the query is used to select the sensor.

Syntax Serial: ?BMC [cc]

Argument:	SensorNumber	
	Min: None	Max: Total Number of Pulse Inputs

Syntax Scripting: result = getvalue(_BMC, cc)

Reply:

BMC = f1 + f2*2 + f3*4 + ... + fn*2^n-1 Type: Unsigned 8-bit Min: 0 Max: 255

Where:

cc = (When only one sensor enabled)

None or 1 : Current BMS10X0

cc = (When several sensors enabled)

1 : BMS10X0 at pulse input 1

2 : BMS10X0 at pulse input 2

...

p : BMS10X0 at pulse input p

and

f1 = Pack Switch

f2 = Load Switch

f3 = Charger Switch

f4 = Brake Resistor / Aux Switch

f5 = Reserved

f6 = Bluetooth Vcc switch

f7 = Reserved

f8 = Reserved

BRK - Read Brake Override Status

Alias: -	HexCode: AB	CANOpen id: 0x2161

Description:

Returns the Brake Override status.

Syntax Serial: ?BRK [cc]

Argument: Channel

Min: 1 Max: Total Number of Motors

Syntax Scripting: getvalue(_BRK, cc)

Reply:

BRK=nn Type: Unsigned 8-bit Min: 0 Max:2

Where:

nn =

0: Auto. Brake is controlled by Motor is On action.

1: Brake Release. Brake is released ignoring the Motor is On action.

2: Brake Engage. Brake is engaged ignoring the Motor is On action.

Example:

?BRK 1: Status of PWM Brake Override for channel 1

BS - Read Internal Sensor Motor Speed in RPM

Alias: BLSPEED HexCode: 0A CANOpen id: 0x210A

Description:

On brushless motor controllers, reports the actual speed measured using the motor's Internal sensors (Hall, SinCos, Resolver) as the actual RPM value. To report RPM accurately, the correct number of motor poles must be loaded in the BPOL configuration parameter.

Syntax Serial: ?BS [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_BS, cc)
result = getvalue(_BLSPEED, cc)

Reply:

BS=nn Type: Signed 32-bit Min: -65535 Max: 65535

Where:

cc = Motor channel
nn = Speed in RPM

BSC - Read BMS State of Charge in percentage

Alias: - HexCode: 50 CANOpen id: 0x213A

Description:

When one or more BMS10X0 are connected to the controller, this query reports the Battery's State Of Charge in percentage, which is connected to the respective BMS10X0. If only one BMS10X0 is connected to any pulse input this query will report the data of that device, regardless which pulse input it is connected to. If more than one BMS10X0 is connected to pulse inputs and these inputs are enabled and configured in BMS MultiPWM mode, then the argument following the query is used to select the sensor.

Syntax Serial: ?BSC [cc]

Argument: SensorNumber

Min: None Max: Total Number of Pulse Inputs

Syntax Scripting: result = getvalue(_BSC, cc)

Reply:

BSC=nn Type: Unsigned 8-bit Min: 0 Max: 255

Where:

cc = (When only one sensor enabled)

None or 1 : Current BMS10X0

cc = (When several sensors enabled)

1 : BMS10X0 at pulse input 1

2 : BMS10X0 at pulse input 2

...

p : BMS10X0 at pulse input p

nn = State Of Charge (%)

BSR - Read Internal Sensor Motor Speed as 1/1000 of Max RPM

Alias: BLRSPEED HexCode: 0B CANOpen id: 0x210B

Description:

On brushless motor controllers, returns the measured motor speed, using the motor's Internal sensors (Hall, Sin/Cos, Resolver), as a ratio of the Max RPM configuration parameter. The result is a value of between 0 and +/-1000. Note that if the motor spins faster than the Max RPM, the return value will exceed 1000. However, a larger value is ignored by the controller for its internal operation. To report an accurate result, the correct number of motor poles must be loaded in the BPOL configuration parameter.

Syntax Serial: ?BSR

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_BSR,)
result = getvalue(_BLRSPEED,)

Reply:

BSR=nn Type: Signed 16-bit Min: -1000 Max: 1000

Where:

nn = Speed relative to max

Example:

Q: ?BSR
R: BSR=500: speed is 50%of the RPM value stored in the Max RPM configuration

C - Read Encoder Counter Absolute

Alias: ABCNTR	HexCode: 04	CANOpen id: 0x2104

Description:

Returns the encoder value as an absolute number. The counter is 32-bit with a range of +/- 2147483648 counts.

Syntax Serial: ?C [cc]

Argument:	Channel Min: 1 Max: Total Number of Encoders

Syntax Scripting:	result = getvalue(_C, cc)
	result = getvalue(_ABCNTR, cc)

Reply:

C=nn	Type: Signed 32-bit	Min: -2147M	Max: 2147M

Where:

cc = Encoder channel number
nn = Absolute counter value

CAN - Read Raw CAN frame

Alias: CAN	HexCode: 27	CANOpen id:

Description:

This query is used in CAN-enabled controllers to read the content of a received CAN frame in the RawCAN mode. Data will be available for reading with this query only after a ?CF query is first used to check how many received frames are pending in the FIFO buffer. When the query is sent without arguments, the controller replies by outputting all elements of the frame separated by colons.

Syntax Serial: ?CAN [ee]

Argument:	Element Min: 1 Max: 10

Syntax Scripting: result = getvalue(_CAN, ee)

Reply:

CAN = dd1:dd2:dd3: ... :dd10	Type: Unsigned 16-bit	Min: 0	Max: 255

Where:

ee = Byte in frame

dd1 = Header

dd2= Bytecount
dd3 to dd10 = Data0 to data7

Example:

Q: ?CAN
R: CAN=5:4:11:12:13:14:0:0:0:0
Q: ?CAN 3
R: CAN=11

CB - Read Absolute Internal Sensor Counter

Alias: BLCNTR	HexCode: 05	CANOpen id: 0x2105

Description:

On brushless motor controllers, returns the running total of Internal sensor (Hall, SinCos, Resolver) transition value as an absolute number. The counter is 32-bit with a range of +/-2147483648 counts.

Syntax Serial:	?CB [cc]
Argument:	Channel Min: 1 Max: Total Number of Motors

Syntax Scripting:	result = getvalue(_CB, cc)
	result = getvalue(_BLCNTR, cc)

Reply:

CB=nn	Type: Signed 32-bit	Min: -2147M	Max: 2147M

Where:

cc = Motor channel
nn = Absolute counter value

CD - Read Raw Redirect Received Frames Count

Alias: CD	HexCode: 8E	CANOpen id: -

Description:

This query is used to read the number of received Raw Redirect frames pending in the FIFO buffer and copies the oldest frame into the read buffer, from which it can then be accessed using the ?DDT or ?SDT queries. Sending ?CD again, copies the next frame into the read buffer.

Syntax Serial: ?CD

Argument: None

Syntax Scripting: result = getvalue(_CD, 1)

Reply:

CD=nn Type: Unsigned 8-bit Min: 0 Max: 255

Where:

nn = Number of frames in receive queue

CEC – CAN Error Counter

HexCode: AC CANOPEN id:

Description:

Returns the CAN error counters. It contains Transmit error counter, Receive Error counter, Last error code and Bus-off counter (number of buss-offs since boot)

- TEC: Increases by 1 or 8 according to the error and frame until it reaches 256. Then the counter is reset, and the controller enters bus-off state.
- REC: Increases by 1 or 8 according to the error and frame until it reaches 127. Then the counter is reset, and the controller enters bus-off state.
- LEC: 000: No Error

  001: Stuff Error

  010: Form Error

  011: Acknowledgment Error

  100: Bit recessive Error

  101: Bit dominant Error

  110: CRC Error

  111: Set by software
- BUS-OFF: Every time there is a bus-off this counter increases by one. This value is reset on boot.

Syntax Serial: ?CEC [cc]

Argument: Error counter/status

Syntax Scripting: getvalue(_CEC, cc)

cc=

1: TEC

2: REC

3:LEC

4: Bus-off Counter

Reply:

CEC=nn

Type: Unsigned 16-bit

Min: 0 Max:2

Where:

nn =

Returns the error counter according to cc.

Example:

?CEC 1: Transmit error counter?

CEC=123 – Current number of Transmit errors is 123.

CF - Read Raw CAN Received Frames Count

Alias: CF HexCode: 28 CANOpen id:

Description:

This query is used to read the number of received CAN frames pending in the FIFO buffer and copies the oldest frame into the read buffer, from which it can then be accessed using the ?CAN query. Sending ?CF again, copies the next frame into the read buffer. The controller can buffer up to 16 CAN frames.

Syntax Serial: ?CF

Argument: None

Syntax Scripting: result = getvalue(_CF, 1)

Reply:

CF=nn Type: Unsigned 8-bit Min: 0 Max: 16

Where:

nn = Number of frames in receive queue

CHS - CAN Consumer Heartbeat Status

Alias: - HexCode: 94 CANOpen id: -

Description:

Returns the the status of the respective consumer heartbeat channel. With CANOpen enabled, there are 4 slots in order to monitor heartbeats. Their status can be checked with this query.

Syntax Serial: ?CHS [cc]

Argument: Heartbeat Channel

Min: 1 Max: 4

Syntax Scripting: result = getvalue(_CHS, cc)

Reply:

CIA=nn Type: Unsigned 8-bit Min: 0 Max: 127

Where:

cc = Heartbeat channel

nn = Heartbeat Status:

0: Not configured

1: Pending (waiting for the first heartbeat message from node)

2: Active (node is sending heartbeat regularly)

127: Inactive (node stopped sending heartbeat)

CIA - Read Converted Analog Command

Alias: CMDANA HexCode: 1A CANOpen id: 0x2117

Description:

Returns the motor command value that is computed from the Analog inputs whether or not the command is actually applied to the motor. The Analog inputs must be configured as Motor Command. This query can be used, for example, to read the command joystick from within a MicroBasic script or from an external microcomputer, even though the controller may be currently responding to Serial or Pulse command because of a higher priority setting. The returned value is the raw Analog input value with all the adjustments performed to convert it to a command (Min/Max/Center/Deadband/Linearity).

Syntax Serial: ?CIA [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_CIA, cc)
result = getvalue(_CMDANA, cc)

Reply:

CIA=nn Type: Signed 32-bit Min: -1000 Max: 1000

Where:

cc = Motor channel
nn = Command value in +/-1000 range

CIG – Read Current Integral Gains

Alias: - HexCode: A5 CANOpen id: 0x2160

Description:

Reads the Current Integral Gains. The value is read as the gain multiplied by 10^4. This value is used for both flux and torque Integral gains.

Syntax Serial: ?CIG [cc]

Syntax Scripting: getvalue(_CIG, cc)

Number of Arguments: 1

Argument 1: Channel		Type: Unsigned 8-bit
	Min: 1	Max: 2 x Total Number of Motors

Reply:

CIG = aa Type: Unsigned 32-bit Min: 0 Max: 2,000,000,000

Where:

cc (single channel) =

1: Flux Integral Gain

2: Torque Integral Gain

cc (dual channel) =

1: Flux Integral Gain for motor 1

2: Flux Integral Gain for motor 2

3: Torque Integral Gain for motor 1

4: Torque Integral Gain for motor 2

aa: Integral Gain*10.000

CIP - Read Internal Pulse Command

Alias: CMDPLS HexCode: 1B CANOpen id: 0x2118

Description:

Returns the motor command value that is computed from the Pulse inputs whether or not the command is actually applied to the motor. The Pulse input must be configured as Motor Command. This query can be used, for example, to read the command joystick from within a MicroBasic script or from an external microcomputer, even though the controller may be currently responding to Serial or Analog command because of a higher priority setting. The returned value is the raw Pulse input value with all the adjustments performed to convert it to a command (Min/Max/Center/Deadband/Linearity).

Syntax Serial: ?CIP [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_CIP, cc)
result = getvalue(_CMDPLS, cc)

Reply:

CIP=nn Type: Signed 32-bit Min: -1000 Max: 1000

Where:

cc = Motor channel
nn = Command value in +/-1000 range

CIS - Read Internal Serial Command

Alias: CMDSER HexCode: 19 CANOpen id: 0x2116

Description:

Returns the motor command value that is issued from the serial input or from a MicroBasic script whether or not the command is actually applied to the motor. This query can be used, for example, to read from an external microcomputer the command generated inside MicroBasic script, even though the controller may be currently responding to a Pulse or Analog command because of a higher priority setting.

Syntax Serial: ?CIS [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_CIS, cc)
result = getvalue(_CMDSER, cc)

Reply:

CIS=nn Type: Signed 32-bit Min: -1000 Max: 1000

Where:

cc = channel
nn = command value in +/-1000 range

CL - Read RoboCAN Alive Nodes Map

Alias: CALIVE HexCode: 26 CANOpen id:

Description:

With CL it is possible to see which nodes in a RoboCAN are alive and what type of device is present at each node. A complete state of the network is represented in sixteen 32-bit numbers. Within each 32-bit word are 8 groups of 4-bits. The 4-bits contain the node information. E.g. bits 0-3 of first number is for node 0, bits 8-11 of first number is for node 2, bits 4-7 of second number is for node 5 and bits 12-15 of fourth number is for node 11, etc.

Syntax Serial: ?CL nn

Argument: Group
Min: 1 Max: 16

Syntax Scripting: result = getvalue(_CL, nn)
result = getvalue(_CALIVE, nn)

Reply:

CL=mm Type: Unsigned 32-bit Min: 0 Max: 4194M

Where:

nn =

1 : nodes 0-3

2 : nodes 4-7

...

...

15 : nodes 120-123

16 : nodes 124-127

mm = 4 words of 4 bits. Each 4-bit word:

0b0000 : Inactive node

0b0001 : Active motor controller

0b0011 : Active magsensor

0b0101 : Active RIOX

0b0111 : Active BMS

0b1001 : Active OTS

0b1011 : Active FLW

CPG – Read Current Proportional Gains

Alias: - HexCode: A4 CANOpen id: 0x215F

Description:

Reads the Current Proportional Gains. The value is read as the gain multiplied by 10^4. This value is used for both flux and torque proportional gains.

Syntax Serial: ?CPG [cc]

Syntax Scripting: getvalue(_CPG, cc)

Number of Arguments: 1

Argument 1: Channel		Type: Unsigned 8-bit
	Min: 1	Max: 2 x Total Number of Motors

Reply:

CPG = aa Type: Unsigned 32-bit Min: 0 Max: 2,000,000,000

Where:

cc (single channel) =

	1: Flux Proportional Gain
	2: Torque Proportional Gain

cc (dual channel) =

	1: Flux Proportional Gain for motor 1
	2: Flux Proportional Gain for motor 2
	3: Torque Proportional Gain for motor 1
	4: Torque Proportional Gain for motor 2

aa: Proportional Gain*10.000

CR - Read Encoder Count Relative

Alias: RELCNTR HexCode: 08 CANOpen id: 0x2108

Description:

Returns the amount of counts that have been measured from the last time this query was made. Relative counter read is sometimes easier to work with, compared to full counter reading, as smaller numbers are usually returned.

Syntax Serial: ?CR [cc]

Argument: Channel
Min: 1 Max: Total Number of Encoders

Syntax Scripting: result = getvalue(_CR, cc)
result = getvalue(_RELCNTR, cc)

Reply:

CR=nn Type: Signed 32-bit Min: -2147M Max: 2147

Where:

cc = Motor channel
nn = Counts since last read using ?CR

CSR - Read Relative SSI Sensor Counter

Alias: - HexCode: 6D CANOpen id: 0x213F

Description:

Returns the amount of counts that have been measured from the last time this query was made. Relative counter read is sometimes easier to work with, compared to full counter reading, as smaller numbers are usually returned.

Syntax Serial: ?CSR [cc]

Argument:	Channel	
	Min: 1	Max: Total Number of SSI Encoders

Syntax Scripting: result = getvalue(_CSR, cc)

Reply:

CSR=nn Type: Signed 32-bit Min: -2147M Max: 2147

Where:

cc = SSI sensor channel

nn = Counts since last read using ?CSR

CSS - Read Absolute SSI Sensor Counter

Alias: - HexCode: 6E CANOpen id: 0x213E

Description:

Returns the SSI encoder value as an absolute number. The counter is 32-bit with a range of +/- 2147483648 counts.

Syntax Serial: ?CSS [cc]

Argument:	Channel	
	Min: 1	Max: Total Number of SSI Encoders

Syntax Scripting: result = getvalue(_CSS, cc)

Reply:

CSS=nn Type: Signed 32-bit Min: -2147M Max: 2147

Where:

cc = SSI sensor channel

nn = Absolute counter value

D - Read Digital Inputs

Alias: DIGIN HexCode: 0E CANOpen id: 0x210E

Description:

Reports the status of each of the available digital inputs. The query response is a single digital number which must be converted to binary and gives the status of each of the inputs. The total number of Digital input channels varies from one controller model to another and can be found in the product datasheet.

Syntax Serial: ?D

Argument: None

Syntax Scripting: result = getvalue(_D, 1)
result = getvalue(_DIGIN, 1)

Reply:

D=nn Type: Unsigned 32-bit

Where:

nn = b1 + b2*2 + b3*4 + ... +bn*2^n-1

Example:

Q: ?D
R: D=17 : Inputs 1 and 5 active, all others inactive

DDT - Read Raw Redirect Received Frame

Alias: DDT HexCode: 8F CANOpen id: -

Description:

This query is used in Raw Redirect mode to read the content of a received Raw Redirect frame. Data will be available for reading with this query only after a ?CD query is first used to check how many received frames are pending in the FIFO buffer. When the query is sent without arguments, the controller replies by outputting all elements of the frame separated by colons.

Syntax Serial: ?DDT [ee]

Argument: Element
Min: 1 Max: 64

Syntax Scripting: result = getvalue(_DDT, ee)

Reply:

DDT = dd1:dd2:dd3: ... :dd64 Type: Unsigned 8-bit Min: 0 Max: 255

Where:

ee = Byte in frame

dd1 = byte size

dd2 to dd64 = data0 to data62

Examples: Q: ?DDT

R: DDT=8:82:111:98:111:116:101:113

Q: ?DDT 3

R: DDT=111

DG – Read PID Derivative Gains

Alias: - HexCode: A3 CANOpen id: 0x215E

Description:

Reads the PID's Derivative Gains. The value is read as the gain multiplied by 10^6. This value is used for both speed and position Derivative gains.

Syntax Serial: ?DG [cc]

Syntax Scripting: getvalue(_DG, cc)

Number of Arguments: 1

Argument 1: Channel		Type: Unsigned 8-bit
	Min: 1	Max: 2 x Total Number of Motors

Reply:

DG = aa Type: Unsigned 32-bit Min: 0 Max: 2,000,000,000

Where:

cc (single channel) =

1: Speed Derivative Gain

2: Position Derivative Gain

cc (dual channel) =

1: Speed Derivative Gain for motor 1

2: Speed Derivative Gain for motor 2

3: Position Derivative Gain for motor 1

4: Position Derivative Gain for motor 2

aa: Derivative Gain*1.000.000

DI - Read Individual Digital Inputs

Alias: DIN HexCode: 0F CANOpen id: 0x2145

Description:

Reports the status of an individual Digital Input. The query response is a boolean value (0 or 1). The total number of Digital input channels varies from one controller model to another and can be found in the product datasheet.

Syntax Serial: ?DI [cc]

Argument: InputNbr
Min: 1 Max: Total Number of Digital Inputs

Syntax Scripting: result = getvalue(_DI, cc)
result = getvalue(_DIN, cc)

Reply:

DI=nn Type: Boolean Min: 0 Max: 1

Where:

cc = Digital Input number
nn = 0 or 1 state for each input

Example:

Q: ?DI
R: DI=1:0:1:0:1:0
Q: ?DI 1
R: DI=0

DO - Read Digital Output Status

Alias: DIGOUT HexCode: 17 CANOpen id: 0x2113

Description:

Reads the actual state of all digital outputs. The response to that query is a single number which must be converted into binary in order to read the status of the individual output bits. When querying an individual output, the reply is 0 or 1 depending on its status. The total number of Digital output channels varies from one controller model to another and can be found in the product datasheet.

Syntax Serial: ?DO

Argument: None

Syntax Scripting: result = getvalue(_DO, 1)
result = getvalue(_DIGOUT, 1)

Reply:

DO=nn Type: Unsigned 16-bit Min: 0 Max: 65536

Where:

nn = d1 + d2*2 + d3*4 + ... + dn * 2^n-1

Example:

Q: ?DO
R: DO=17 : Outputs 1 and 5 active, all others inactive

DPA - Read DC/Peak Amps

Alias: - HexCode: 6E CANOpen id: -

Description:

Applicable only for brushless controllers. Measures and reports the Peak Amps , in Amps*10, for all operating channels.

Syntax Serial: ?DPA [cc]

Argument:	Channel
	Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_DPA, cc)

Reply:

DPA = aa Type: Signed 16-bit Min: 0

Where:

cc = Motor channel

aa = Amps *10 for each channel

Example:

Q: ?DPA

R: DPA=100:200

Q: ?DPA 2

R: DPA=200

DR - Read Destination Reached

Alias: DREACHED HexCode: 22 CANOpen id: 0x211B

Description:

This query is used when chaining commands in Position Count mode, to detect that a destination has been reached and that the next destination values that were loaded in the buffer have become active. The Destination Reached bit is latched and is cleared once it has been read.

Syntax Serial: ?DR [cc]

Argument:	Channel
	Min: 1 Max: Total Number of Motors

Syntax Scripting:	result = getvalue(_DR, cc)
	result = getvalue(_DREACHED, cc)

Reply:

DR=nn Type: Unsigned 8-bit Min: 0 Max: 1

Where:

cc = Motor channel
nn =
0 : Not yet reached
1 : Reached

E - Read Closed Loop Error

Alias: LPERR HexCode: 18 CANOpen id: 0x2114

Description:

In closed-loop modes, returns the difference between the desired speed or position and the measured feedback. This query can be used to detect when the motor has reached the desired speed or position. In open loop mode, this query returns 0.

Syntax Serial: ?E [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_E, cc)
result = getvalue(_LPERR, cc)

Reply:

E=nn Type: Signed 32-bit Min: -2147M Max: 2147M

Where:

cc = Motor channel
nn = Error value

F - Read Feedback

Alias: FEEDBK HexCode: 13 CANOpen id: 0x2110

Description:

Reports the value of the feedback sensors that are associated to each of the channels in closed-loop modes. The feedback source can be Encoder, Analog or Pulse. Selecting the feedback source is done using the encoder, pulse or analog configuration parameters. This query is useful for verifying that the correct feedback source is used by the channel in the closed-loop mode and that its value is in range with expectations.

Syntax Serial: ?F [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_F, cc)
result = getvalue(_FEEDBK, cc)

Reply:

F=nn	Type: Signed 32-bit	Min: -2147M	Max: 2147M

Where:

cc = Motor channel
nn = Feedback values

FC - Read FOC Angle Adjust

Alias: FC HexCode: 47 CANOpen id: 0x2135

Description:

Read in real time the angle correction that is currently applied by the Field Oriented algorithm in order achieve optimal performance.

Syntax Serial: ?FC [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_FC, cc)

Reply:

FC = nn	Type: Signed 16-bit	Min: -512	Max: 512

Where:

cc = Motor channel
nn = Angle correction

FLW - Read Flow Sensor Counter

Alias: - HexCode: 7B CANOpen id: 0x214A

Description:

When one or more FLW100 are connected to the controller, this query reports the count measurements of X and Y axis in mm*10 of the respective FLW100. If only one FLW100 is connected to any pulse input this query will report the data of that device, regardless which pulse input it is connected to. If more than one FLW100 is connected to pulse inputs and these inputs are enabled and configured in FlowSensor MultiPWM mode, then the argument following the query is used to select the sensor.

Syntax Serial: ?FLW [cc]

Argument: SensorNumber
Min: 1 Max: 2*Total Number of Pulse Inputs

Syntax Scripting: result = getvalue(_FLW, cc)

Reply:

FLW=nn Type: Signed 32-bit Min: -2147M Max: 2147M

Where:

cc = (When only one sensor enabled)

1 : X Counter

2: Y Counter

cc = (When several sensors enabled)

1 : X Counter of sensor at pulse input 1

2 : Y Counter of sensor at pulse input 1

3 : X Counter of sensor at pulse input 2

4 : Y Counter of sensor at pulse input 2

...

((p-1)*2)+1 : X Counter of sensor at pulse input p

((p-1)*2)+2 : Y Counter of sensor at pulse input p

nn = Distance in mm*10.

FF - Read Fault Flags

Alias: FLTFLAG HexCode: 15 CANOpen id: 0x2112

Description:

Reports the status of the controller fault conditions that can occur during operation. The response to that query is a single number which must be converted into binary in order to evaluate each of the individual status bits that compose it.

Syntax Serial: ?FF

Argument: None

Syntax Scripting: result = getvalue(_FF, 1)
result = getvalue(_FLTFLAG, 1)

Reply:

FS = f1 + f2*2 + f3*4 + ... + fn*2^n-1 Type: Unsigned 16-bit Min: 0 Max: 65535

Where:

TABLE 15-14. FF - Read Fault

Fault bit	Fault	Activated when
f1	OverHeat	The measured temperatures go beyond the configured limit. For more details see chapter “Temperature-Based Protection” at section 7.
f2	OverVolt	The measured voltages go beyond the configured limit. For more details see chapter “Overvoltage Protection” at section 7.

Fault bit	Fault	Activated when
f3	UnderVolt	The measured voltages go below the configured limit. For more details see chapter "Undervoltage Protection" at section 7.
f4	Short	A possible short has been detected. For more details see chapter Short Circuit Protection at section 7.
f5	EStop	The EX command has been sent or the emergency stop action has been triggered. For more details see chapter "Digital Inputs Configurations and Uses" at section 4 and chapter "EX – Emergency Stop" at section 15
f6	Motor/Sensor	There is a sensor fault during runtime or motor sensor setup. For more details see chapter "Sensor Error Detection" at section 8.
f7	MOSFail	Either of the power MOSFETs is detected as damaged. For more details see "Motor Deactivation in Case of Output Stage Hardware Failure" at section 2.
f8	DefConfig	Default configuration is loaded at startup. This could be due to either corruption in flash memory or by powering up a brand-new controller.
f9	STO Fault	The STO circuit's self-test routine returns a fault (it happens also when either of the STO inputs is low, while the other is high). For more details see "Motor Deactivation in Case of Output Stage Hardware Failure" at section 2.

Example:

Q: ?FF

R: FF=2 : Overvoltage fault

FID - Read Firmware ID

Alias: FID HexCode: 1E CANOpen id:

Description:

This query will report a string with the date and identification of the firmware revision of the controller.

Syntax Serial: ?FID

Argument: None

Syntax Scripting: result = getvalue(_FID, 1)

Reply:

FID=ss Type: String

Where:

ss = Firmware ID string

Example:

Q: ?FID
R: FID=Roboteq v1.6 RCB500 05/01/2016

FIN - Read Firmware ID (numerical)

Alias: - HexCode: 3F CANOpen id: 0x2137

Description:

This query will report the version and the date of the firmware revision of the controller. If only element 1 is queried then the version format will be 4 characters in which:

- 1st character will be the major version
- 2nd character will be the minor version
- 3rd character will be the version variation (a, b, etc.)
- 4th character will be the firmware status which is used for technical support reasons.

Syntax Serial: ?FIN [ee]

Argument:	Element	
	Min: None	Max: 4

Syntax Scripting: result = getvalue(_FIN, ee)

Reply:

FID = nn Type: Unsigned 16-bit Min: 0

Where:

ee = Firmware Version Element

1: Version

2: Month

3: Day

4: Year

nn = Value.

FM - Read Runtime Status Flag

Alias: MOTFLAG HexCode: 30 CANOpen id: 0x2122

Description:

Report the runtime status of each motor. The response to that query is a single number which must be converted into binary in order to evaluate each of the individual status bits that compose it.

Syntax Serial: ?FM [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_FM, cc)
result = getvalue(_MOTFLAG, cc)

Reply:

FM = f1 + f2*2 + f3*4 + ... + fn*2n-1

Type: Unsigned 16-bit Min: 0 Max: 255

Where:

cc = Motor channel

TABLE 15-15. FM - Read Runtime Status Flag

Status bit	Fault	Activated when
f1	AmpLim	The current is limited to the configured Current limit (ALIM), since the command requires higher current or I2T protection is triggered. For more details see chapters "Current Limiting" and "I2T Protection" at section 7.
f2	Stall	The stall detection method is triggered. For more details see chapter "Stall Detection" at section 8.
f3	Loop Error	The loop error detection method is triggered. For more details see chapter "Closed Loop Error Protection" at section 7.
f4	Quick Stop	The QST command has been sent or the Quick Stop action has been triggered. For more details see chapter "Digital Inputs Configurations and Uses" at section 4 and chapter "QST – Quick Stop" at section 15
f5	FwdLimit	The Forward limit switch action has been triggered. For more details see chapter "Digital Inputs Configurations and Uses" at section 4.
f6	RevLimit	The Reverse limit switch action has been triggered. For more details see chapter "Digital Inputs Configurations and Uses" at section 4.
f7	AmpTrig	Amps trigger threshold has been reached. For more details see "ATRIG - Amps Trigger Level" at section 15.
f8	FETs Off	Power MOSFETs have been floated due to a specific error. For more details see "Motor Deactivation in Normal Operation" at section 2.

Example:

Q: ?FM 1
R: FM=6 : Motor 1 is stalled and loop error detected

Note:

f2, f3 and f4 are cleared when the next idle motor command is given (0 in case of speed modes, equal to feedback in case of position modes). When f5 or f6 are on, the motor can only be commanded to go in the reverse direction.

FS - Read Status Flags

Alias: STFLAG HexCode: 14 CANOpen id: 0x2111

Description:

Report the state of status flags used by the controller to indicate a number of internal conditions during normal operation. The response to this query is the single number for all status flags. The status of individual flags is read by converting this number to binary and look at various bits of that number.

Syntax Serial: ?FS

Argument: None

Syntax Scripting: result = getvalue(_FS, 1)
result = getvalue(_STFLAG, 1)

Reply:

FS = f1 + f2*2 + f3*4 + ... + fn*2^n-1 Type: Unsigned 16-bit Min: 0 Max: 65535

Where:

TABLE 15-16. FS - Read Status Flags

Status bit	Fault	Activated when
f1	Serial	The motor command is given via a serial interface (RS232, RS485, TCP, USB). For more details see chapter Input Command Modes and Priorities at section 6.
f2	Pulse	The motor command is given via a Pulse Input (R/C radio, PWM, Frequency). For more details see chapter Input Command Modes and Priorities at section 6
f3	Analog	The motor command is given via a Analog Input. For more details see chapter Input Command Modes and Priorities at section 6.
f4	FETs Off	All the power MOSFETs of the controller are floated.
f5	Stall	There is stall error in either of the motor channels.
f6	At Limit	There is forward or reverse limit triggered in either of the motor channels.
f7	STO	Both STO inputs are grounded. In that case STO is triggered and the power circuit is deactivated.
f8	RunScript	A script is running.
f9	Setup	The motor/sensor setup process is executed.

HS - Read Hall Sensor States

Alias: HSENSE HexCode: 31 CANOpen id: 0x2123

Description:

Reports that status of the hall sensor inputs. This function is mostly useful for trouble-shooting. When no sensors are connected, all inputs are pulled high and the value 7 will be replied. For 60 degrees spaced Hall sensors, 0-1- 3-4- 6-7 are valid combinations, while 2 and 5 are invalid combinations. For 120 degrees spaced sensors, 1-2- 3-4- 5-6 are valid combinations, while 0 and 7 are invalid combinations. In normal conditions, valid values should appear at one time or the other as the motor shaft is rotated.

Syntax Serial: ?HS [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_HS, cc)
result = getvalue(_HSENSE, cc)

Reply:

HS= ha + 2*hb + 4*hc Type: Unsigned 8-bit Min: 0 Max: 7

Where:

cc = channel
ha = hall sensor A
hb = hall sensor B
hc = hall sensor C
Example:
Q: ?HS 1
R: HS=5 : sensors A and C are high, sensor B is low

Note:

Function not available on HBLxxxxx products

ICL - Is RoboCAN Node Alive

Alias: ICL HexCode: 46 CANOpen id: 0x2134

Description:

This query is used to determine if specific RoboCAN node is alive on CAN bus.

Syntax Serial: ?ICL cc

Argument: NodeId
Min: 1 Max: 127

Syntax Scripting: result = getvalue(_ICL, cc)

Reply:

ICL=nn Type: Unsigned 8-bit Min: 0 Max: 1

Where:

cc = Node Id
nn =
0 : Not present
1 : Alive

IG – Read PID Integral Gains

Alias: - HexCode: A2 CANOpen id: 0x215D

Description:

Reads the PID's Integral Gains. The value is read as the gain multiplied by 10^6. This value is used for both speed and position integral gains.

Syntax Serial: ?IG [cc]

Syntax Scripting: getvalue(_IG, cc)

Number of Arguments: 1

Argument 1: Channel		Type: Unsigned 8-bit
	Min: 1	Max: 2 x Total Number of Motors

Reply:

IG = aa Type: Unsigned 32-bit Min: 0 Max: 2,000,000,000

Where:

cc (single channel) =

1: Speed Integral Gain

2: Position Integral Gain

cc (dual channel) =

1: Speed Integral Gain for motor 1

2: Speed Integral Gain for motor 2

3: Position Integral Gain for motor 1

4: Position Integral Gain for motor 2

aa: Integral Gain*1.000.000

LK - Read Lock status

Alias: LOCKED HexCode: 1D CANOpen id: 0x2124

Description:

Returns the status of the lock flag. If the configuration is locked, then it will not be possible to read any configuration parameters until the lock is removed or until the parameters are reset to factory default. This feature is useful to protect the controller configuration from being copied by unauthorized people.

Syntax Serial: ?LK

Argument: None

Syntax Scripting: result = getvalue(_LK, 1)
result = getvalue(_LOCKED, 1)

Reply:

LK=ff	Type: Unsigned 8-bit	Min: 0	Max: 1

Where:

ff =
0 : unlocked
1 : locked

M - Read Motor Command Applied

Alias: MOTCMD HexCode: 01 CANOpen id: 0x2101

Description:

Reports the command value that is being used by the controller. The number that is reported will be depending on which mode is selected at the time. The choice of one command mode vs. another is based on the command priority mechanism. In the Serial mode, the reported value will be the command that is entered in via the RS232, RS485, TCP or USB port and to which an optional exponential correction is applied. In the Analog and Pulse modes, this query will report the Analog or Pulse input after it is being converted using the min, max, center, deadband, and linearity corrections. This query is useful for viewing which command is actually being used and the effect of the correction that is being applied to the raw input.

Syntax Serial: ?M [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_M, cc)
result = getvalue(_MOTCMD, cc)

Reply:

M=nn	Type: Signed 32-bit	Min: -2147M	Max: 2147M

Where:

cc = Motor channel
nn = Command value used for each motor. 0 to +/-1000 range

Example:

Q: ?M
R: M=800:-1000
Q: ?M 1 R:
M=800

MA - Read Field Oriented Control Motor Amps

Alias: MEMS HexCode: 25 CANOpen id: 0x211C

Description:

On brushless motor controllers operating in sinusoidal mode, this query returns the Torque (also known as Quadrature or Iq) current, and the Flux (also known as Direct, or Id) current. Current is reported in Amps x 10.

Syntax Serial:	?MA nn
Argument:	AmpsChannel
	Min: 1 Max: 2 * Total Number of Motors
Syntax Scripting:	result = getvalue(_MA, nn)
	result = getvalue(_MEMS, nn)

Reply:

MA=mm Type: Signed 16-bit

Where:

nn =
1 : Flux Amps 1 (Id)
2 : Torque Amps 1 (Iq)
3 : Flux Amps 2 (Id)
4 : Torque Amps 2 (Iq)
mm = Amps * 10

MCB - Read Magsensor Markers Pattern

Alias: - HexCode: A8 CANOpen id: -

Description:

When one or more MGS1600 Magnetic Guide Sensors are connected to the controller, this query reports whether there is marker pattern detected. If no pattern is detected, the output will be 0. If only one sensor is connected to any pulse input, no argument is needed for this query. If more than one sensor is connected to pulse inputs and these inputs are enabled and configured in Magsensor MultiPWM mode, then the argument following the query is used to select the sensor.

Syntax Serial: ?MCB [cc]

Argument: SensorNumber

Min: None Max: Total Number of Pulse Inputs

Syntax Scripting: result = getvalue(_MCB, cc)

Reply:

MCB = nn Type: Unsigned 8-bit Min: 0 Max: 255

Where:

cc = (When only one sensor enabled)

None or 1 : Current sensor

cc = (When several sensors enabled)

1 : Sensor at pulse input 1

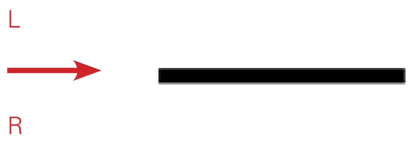
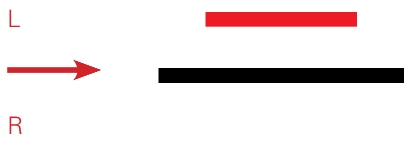
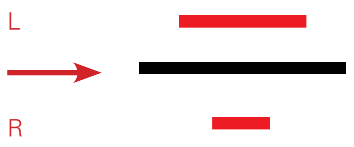
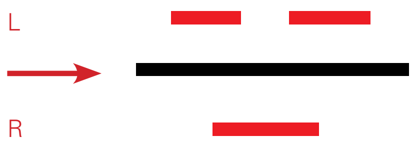
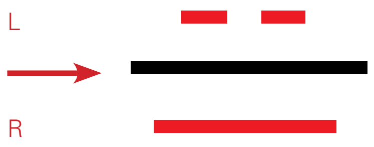
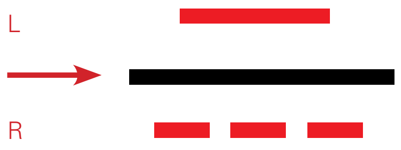
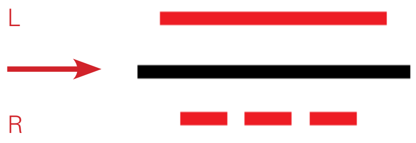
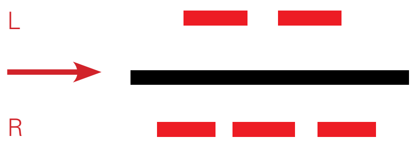
2 : Sensor at pulse input 2

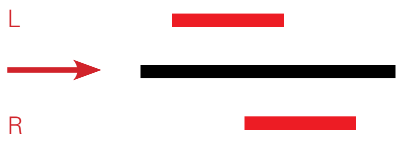
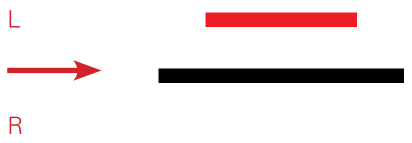
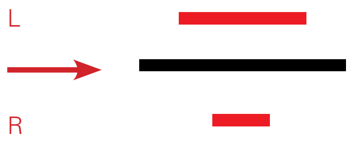
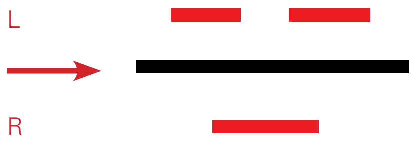
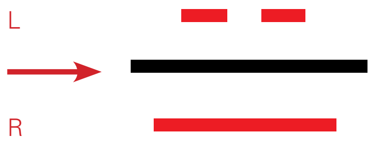
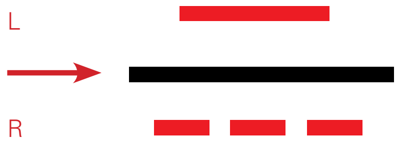
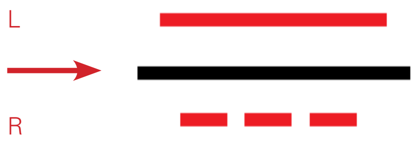
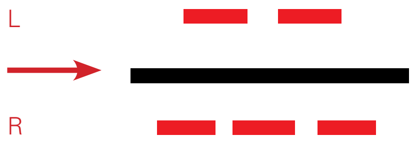
...

p : Sensor at pulse input p

nn =see Table below

TABLE 15-17. Marker Pattern Number

Marker Pattern Number	**Markers Black: Tape Red: Marker**	**Left Right Transitions**
0		L: 0 R: 0
1		L: 1 R: 0
2		L: 111 R: 010
3		L: 01110 R: 11011
4		L: 11111 R: 01010
5		L: 0111110 R: 1101011
6		L: 1111111 R: 0101010
7		L: 011101110 R: 110111011

Marker Pattern Number	Markers Black: Tape Red: Marker	Left Right Transitions
8		L: 011 R: 110
9		L: 0 R: 1
10		L: 010 R: 111
11		L: 11011 R: 01110
12		L: 01010 R: 11111
13		L: 1101011 R: 0111110
14		L: 0101010 R: 1111111
15		L: 110111011 R: 011101110

MCU - Microprocessor Usage

Alias: - HexCode: A7 CANOpen id: -

Description:

Reports an indicative value of the Microprocessor usage of the product. The value is in percentage out of 100. This value will help user evaluate the effect of the use of scripting and communication traffic.

Syntax Serial: ?MCU

Syntax Scripting: result = getvalue(_MCU, 1)

Reply:

MCU = nn Type: Unsigned 8-bit Min: 0 Max: 100

Where:

nn =Percentage of the microprocessor usage.

MGD - Read Magsensor Track Detect

Alias: MGDET HexCode: 29 CANOpen id: 0x211D

Description:

When one or more MGS1600 Magnetic Guide Sensors are connected to the controller, this query reports whether a magnetic tape is within the detection range of the sensor. If no tape is detected, the output will be 0. If only one sensor is connected to any pulse input, no argument is needed for this query. If more than one sensor is connected to pulse inputs and these inputs are enabled and configured in Magsensor MultiPWM mode, then the argument following the query is used to select the sensor.

Syntax Serial: ?MGD [cc]

Argument: SensorNumber
Min: None Max: Total Number of Pulse Inputs

Syntax Scripting: result = getvalue(_MGD, cc)
result = getvalue(_MGDET, cc)

Reply:

MGD=nn Type: Unsigned 8-bit Min: 0 Max: 1

Where:

cc = (When only one sensor enabled)
None or 1 : Current sensor
cc = (When several sensors enabled)
1 : Sensor at pulse input 1
2 : Sensor at pulse input 2
...
p : Sensor at pulse input p
nn =

0 : No track detected
1 : Track detected

MGM - Read Magsensor Markers

Alias: MGMRKR HexCode: 2B CANOpen id: 0x211F

Description:

When one or more MGS1600 Magnetic Guide Sensors are connected to the controller, this query reports whether left or right markers are present under sensor.If only one sensor is connected to any pulse input this query will report the data of that sensor, regardless which pulse input it is connected to. If more than one sensor is connected to pulse inputs and these inputs are enabled and configured in Magsensor MultiPWM mode, then the argument following the query is used to select the sensor.

Syntax Serial: ?MGM [cc]

Argument: SensorNumber
Min: 1 Max: 2 * Total Number of Pulse Inputs

Syntax Scripting: result = getvalue(_MGM, cc)
result = getvalue(_MGMRKR, cc)

Reply:

MGM=mm Type: Unsigned 8-bit Min: 0 Max: 1

Where:

cc = (When only one sensor enabled)
1 : Left Marker
2 : Right Marker
cc = (When several sensors enabled)
1 : Left Marker of sensor at pulse input 1
2 : Right Marker of sensor at pulse input 1
3 : Left Marker of sensor at pulse input 2
4 : Right Marker of sensor at pulse input 2
...
((p-1)* 2)+1 : Left Marker of sensor at pulse input p
((p-1)* 2)+2 : Right Marker of sensor at pulse input p
nn =
0 : No marker detected
1 : Marker detected

MGS - Read Magsensor Status

Alias: MGSTATUS HexCode: 2C CANOpen id: 0x2120

Description:

When one or more MGS1600 Magnetic Guide Sensors are connected to the controller, this query reports the state of the sensor. If only one sensor is connected to any pulse input, no argument is needed for this query. If more than one sensor is connected to pulse inputs and these inputs are enabled and configured in Magsensor MultiPWM mode, then the argument following the query is used to select the sensor.

Syntax Serial:	?MGS
Argument:	SensorNumber Min: None Max: Total Number of Pulse Inputs

Syntax Scripting:	result = getvalue(_MGS,)
	result = getvalue(_MGSTATUS,)

Reply:

MGS=f1 + f2*2 + f3*4 + ... + fn*2n-1 Type: Unsigned 16-bit

Where:

cc = (When only one sensor enabled)
None or 1 : Current sensor
cc = (When only several sensors enabled)
1 : Sensor at pulse input 1
2 : Sensor at pulse input 2

...

p : Sensor at pulse input p
f1 : Tape cross
f2 : Tape detect
f3 : Left marker present
f4 : Right marker present
f9 : Sensor active

MGT - Read Magsensor Track Position

Alias: MGTRACK HexCode: 2A CANOpen id: 0x211E

Description:

When one or more MGS1600 Magnetic Guide Sensors are connected to the controller, this query reports the position of the tracks detected under the sensor. If only one sensor is connected to any pulse input, the argument following the query selects which track to read. If more than one sensor is connected to pulse inputs and these inputs are enabled and configured in Magsensor MultiPWM mode, then the argument following the query is used to select the sensor. The reported position of the magnetic track in millimeters, using the center of the sensor as the 0 reference.

Syntax Serial:	?MGT cc
Argument:	Channel Min: 1 Max: 3 * Total Number of Pulse Inputs

Syntax Scripting:	result = getvalue(_MGT, cc)
	result = getvalue(_MGTRACK, cc)

Reply:

MGM = nn Type: Signed 16-bit

Where:

cc = (When only one sensor enabled)
1 : Left Track
2 : Right Track
3 : Active Track

cc = (When several sensors enabled)
1 : Left Track of sensor at pulse input 1
2 : Right Track of sensor at pulse input 1
3 : Active Track of sensor at pulse input 1
4 : Left Track of sensor at pulse input 2
5 : Right Track of sensor at pulse input 2
6 : Active Track of sensor at pulse input 2

...

((p-1)* 3)+1 : Left Track of sensor at pulse input p
((p-1)* 3)+2 : Right Track of sensor at pulse input p
((p-1)* 3)+3 : Active Track of sensor at pulse input p
nn = position in millimeters

P - Read Motor Power Output Applied

Alias: MOTPWR HexCode: 02 CANOpen id: 0x2102

Description:

Reports the actual PWM level that is being applied to the motor at the power output stage. This value takes into account all the internal corrections and any limiting resulting from temperature or over current. A value of 1000 equals 100% PWM. The equivalent motor phase to phase voltage amplitude is the battery voltage * PWM level.

Syntax Serial: ?P [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_P, cc)
result = getvalue(_MOTPWR, cc)

Reply:

P=nn Type: Signed 16-bit Min: -1000 Max: 1000

Where:

cc = Motor channel
nn = 0 to +/-1000 power level

Example:

Q: ?P 1
R: P=800

PG – Read PID Proportional Gains

Alias: - HexCode: A1 CANOpen id: 0x215C

Description:

Reads the PID's Proportional Gains. The value is read as the gain multiplied by 10^6. This value is used for both speed and position proportional gains.

Syntax Serial: ?PG [cc]

Syntax Scripting: getvalue(_PG, cc)

Number of Arguments: 1

Argument 1: Channel		Type: Unsigned 8-bit
	Min: 1	Max: 2 x Total Number of Motors

Reply:

PG = aa Type: Unsigned 32-bit Min: 0 Max: 2,000,000,000

Where:

cc (single channel) =

1: Speed Proportional Gain

2: Position Proportional Gain

cc (dual channel) =

1: Speed Proportional Gain for motor 1

2: Speed Proportional Gain for motor 2

3: Position Proportional Gain for motor 1

4: Position Proportional Gain for motor 2

aa: Proportional Gain*1.000.000

PHA - Read Phase Amps

Alias: - HexCode: 49 CANOpen id: -

Description:

Measures and reports instant motor phase Amps, in Amps*10, for all current sensors located in the motor phases. Applicable only for brushless and AC Induction motor controllers.

Syntax Serial: ?PHA [cc]

Argument:	Channel	
	Min: 1	Max: Total Number Of Current Sensors

Syntax Scripting: result = getvalue(_PHA, cc)

Reply:

PHA = aa Type: Signed 16-bit Min: -32767 Max: 32767

Where:

cc = Current Sensor

aa = Amps*10

PI - Read Pulse Inputs

Alias: PLSIN HexCode: 11 CANOpen id: 0x2148

Description:

Reports the value of each of the enabled pulse input captures. The value is the raw number in microseconds when configured in Pulse Width mode. In Frequency mode, the returned value is in Hertz. In Duty Cycle mode, the reported value ranges between 0 and 4095 when the pulse duty cycle is 0% and 100% respectively. In Pulse Count mode, the reported value in the number of pulses as detected. This counter only increments. In order to reset that counter the pulse capture mode needs to be set back to disabled and then again to Pulse Count.

Syntax Serial:	?PI [cc]
Argument:	InputNbr Min: 1 Max: Total Number of Pulse Input

Syntax Scripting:	result = getvalue(_PI, cc)
	result = getvalue(_PLSIN, cc)

Reply:

PI=nn Type: Unsigned 16-bit Min: 0 Max: 65536

Where:

cc = Pulse capture input number

nn = Value

Note:

The total number of Pulse input channels varies from one controller model to another and can be found in the product datasheet.

PIC - Read Pulse Input after Conversion

Alias: PLSINC HexCode: 24 CANOpen id: 0x2149

Description:

Returns value of a Pulse input after all the adjustments were performed to convert it to a command or feedback value (Min/Max/Center/Deadband/Linearity). If an input is disabled, the query returns 0.

Syntax Serial:	?PIC [cc]
Argument:	InputNbr Min: 1 Max: Total Number of Pulse Input

Syntax Scripting: result = getvalue(_PIC, cc)
result = getvalue(_PLSINC, cc)

Reply:

PIC=nn Type: Signed 16-bit Min: -1000 Max: 1000

Where:

cc = Pulse input number
nn = Converted input value to +/-1000 range

S - Read Encoder Motor Speed in RPM

Alias: ABSPEED HexCode: 03 CANOpen id: 0x2103

Description:

Reports the actual speed measured by the encoders as the actual RPM value. To report RPM accurately, the correct Pulses per Revolution (PPR) must be stored in the encoder configuration.

Syntax Serial: ?S [cc]

Argument: Channel
Min: 1 Max: Total Number of Encoders

Syntax Scripting: result = getvalue(_S, cc)
result = getvalue(_ABSPEED, cc)

Reply:

S = nn Type: Signed 32-bit Min: -65535 Max: 65535

Where:

cc =Motor channel
nn = Speed in RPM

SCC - Read Script Checksum

Alias: SCC HexCode: 45 CANOpen id: 0x2133

Description:

Scans the script storage memory and computes a checksum number that is unique to each script. If not script is loaded the query outputs the value 0xFFFFFFFF. Since a stored script cannot be read out, this query is useful for determining if the correct version of a given script is loaded.

Syntax Serial: ?SCC

Argument: None

Syntax Scripting: result = getvalue(_SCC, 1)

Reply:

SCC = nn Type: Unsigned 32-bit

Where:

nn = Checksum number

SDT - Read Raw Redirect Received Frame as string

Alias: SDT HexCode: 90 CANOpen id: -

Description:

This query is used in Raw Redirect mode to read the content of a received Raw Redirect frame in string format. Data will be available for reading, with this query, only after a ?CD query is first used to check how many received frames are pending in the FIFO buffer.

Syntax Serial: ?SDT

Argument: None

Reply:

SDT=ss Type: String

Where:

ss = ASCII string

Example:

Q: ?SDT

R: SDT=Roboteq

SEC - Read Sensor Errors

Alias: SEC HexCode: 8D CANOpen id:

Description:

This query is used to read the quality of the sensor used for commutation.

- In case of Hall Trapezoidal or Hall Sinusoidal it will return the number of the out of sequence hall states.
- In case of Hall+Encoder Sinusoidal it will return the differrence between the electrical angle estimated out of the hall sensors and the electrical angle estimated out of the encoder sensor. The value is in degrees.
- In case of SinCos Sinusoidal or Resolver Sinusoidal it will return the result of (sin^2(x)+cos^2(x)) * 100. The returned value is indicative of the quality of the sin/cos signal. The closer the value is to 100 the better is the quality of the signal.
- In case of any other sensor used in sinusoidal mode it is not applicable.

Syntax Serial: ?SEC[cc]

Argument: Channel

Min:1 Max:Total Number of Motors

Syntax Scripting: result=getvalue(_SEC, cc)

Reply:

SEC=nn Type: unsigned 8-bit Min: 0 Max:

Where:

cc = Motor channel

nn = Number of sensor errors in case of of Hall trapezoidal or Hall Sinusoidal

Angle difference in degrees in case of Hall+Encoder Sinusoidal

Signal Quality in case of SinCos Sinusoidal or Resolver Sinusoidal.

SNA - Read Sensor Angle

Alias: - HexCode: 79 CANOpen id: -

Description:

On brushless controller operating in sinusoidal mode, this query returns the real time value of the rotor's angle sensor of brushless motor. This query is useful for verifying troubleshooting sin/cos and SPI/SSI sensors. Angle are reported in 0-511 degrees.

Syntax Serial: ?SNA [cc]

Argument: Channel

Min: 1 Max: Total Number Of Motors

Syntax Scripting: result = getvalue(_SNA, cc)

Reply:

SNA = aa Type: Unsigned 16-bit Min: 0 Max: 511

Where:

cc = Motor Channel

aa = Sensor Angle

SNS – Sense Voltage

Alias: - HexCode: 71 CANOpen id: 0x2172

Description:

It reads the instantaneous voltage of a specified phase of the motor and reports the value in millivolts.

Syntax Serial: ?SNS [cc]

Syntax Scripting: result = GetValue(_SNS, cc)

Argument: Motor Phase Number

Min: 1(U Phase of Channel 1) Max: 6 (W Phase of Channel 2)

Reply:

RMP=nn	Type: Signed 32-bit	Min: -65535	Max: 65535

Where:

cc = Motor Phase Number
nn = Phase voltage value in mV.

SR - Read Encoder Speed Relative

Alias: RELSPEED HexCode: 07 CANOpen id: 0x2107

Description:

Returns the measured motor speed as a ratio of the Max RPM (MXRPM) configuration parameter. The result is a value of between 0 and +/1000. As an example, if the Max RPM is set at 3000 inside the encoder configuration parameter and the motor spins at 1500 RPM, then the returned value to this query will be 500, which is 50% of the 3000 max. Note that if the motor spins faster than the Max RPM, the returned value will exceed 1000. However, a larger value is ignored by the controller for its internal operation.

Syntax Serial: ?SR [cc]

Argument: Channel
Min: 1 Max: Total Number of Encoders

Syntax Scripting: result = getvalue(_SR, cc)
result = getvalue(_RELSPEED, cc)

Reply:

SR = nn	Type: Signed 16-bit	Min: -1000	Max: 1000

Where:

cc = Motor channel
nn = Speed relative to max

SS - Read SSI Sensor Motor Speed in RPM

Alias: - HexCode: 6A CANOpen id: 0x213C

Description:

Reports the actual speed measured by the SSI sensors as the actual RPM value. To report RPM accurately, the correct Counter number of bits (SLEN) must be stored in the encoder configuration.

Syntax Serial: ?SS [cc]

Argument:	Channel	
	Min: 1	Max: Total Number of SSI sensors

Syntax Scripting: result = getvalue(_SS, cc)

Reply:

SS = aa Type: Signed 32-bit Min: -65535 Max: 65535

Where:

cc = Motor channel

aa = Speed in RPM.

SSR - Read SSI Sensor Speed Relative

Alias: - HexCode: 6B CANOpen id: 0x213D

Description:

Returns the measured motor speed as a ratio of the Max RPM (MXRPM) configuration parameter. The result is a value of between 0 and +/1000. As an example, if the Max RPM is set at 3000 inside the encoder configuration parameter and the motor spins at 1500 RPM, then the returned value to this query will be 500, which is 50% of the 3000 max. Note that if the motor spins faster than the Max RPM, the returned value will exceed 1000. However, a larger value is ignored by the controller for its internal operation.

Syntax Serial: ?SSR [cc]

Argument:	Channel	
	Min: 1	Max: Total Number of SSI sensors

Syntax Scripting: result = getvalue(_SSR, cc)

Reply:

SSR = aa Type: Signed 16-bit Min: -1000 Max: 1000

Where:

cc = Motor channel

aa = Speed relative to max.

STT - STO Self-Test Result

Alias: - HexCode: 70 CANOpen id: -

Description:

Returns the result of the latest executed STO Self-Test process. This process is applicable only on motor controllers with STO circuit implemented on their board. If the result is not successful the Respective STO Fault bit in the Fault Flags is set. The fault is triggered when:

- Any of the transistors or other component of the STO circuit is damaged.
- Any of the Power MOSFETs is damaged.
- The respective jumper is placed on the board.

Syntax Serial: ?STT xx

Argument: Fault Indication

Min: 1 (STO Result) Max: 2 (MOSFET Result)

Syntax Scripting: result = getvalue(_STT, xx)

Reply:

If xx = 1 (STO Result)

STT=ff Type Singed 32-bit Min: -1 Max: 4

Where ff=

-1: The test is in process

0: Test successful

1: STO1 failed the test

2: STO2 failed the test

3: Test failed using input values

5: Any of the Power MOSFETs is damaged.

If xx = 2 (MOSFET Result)

STT = f1 + f2*2 + f3*4 + ... + fn*2^n-1 Type: Signed 32-bit Min: 0 Max: 4095

Where:

f1 = Top MOSFET of U1 phase is damaged

f2 = Bottom MOSFET of U1 phase is damaged

f3 = Top MOSFET of V1 phase is damaged

f4 = Bottom MOSFET of V1 phase is damaged

f5 = Top MOSFET of W1 phase is damaged

f6 = Bottom MOSFET of W1 phase is damaged

f7 = Top MOSFET of U2 phase is damaged

f8 = Bottom MOSFET of U2 phase is damaged

f9 = Top MOSFET of V2 phase is damaged

f10 = Bottom MOSFET of V2 phase is damaged

f11 = Top MOSFET of W2 phase is damaged

f12 = Bottom MOSFET of W2 phase is damaged

T - Read Temperature

Alias: TEMP HexCode: 12 CANOpen id: 0x210F

Description:

Reports the temperature at each of the Heatsink sides and on the internal MCU silicon chip. The reported value is in degrees C with a one degree resolution.

Syntax Serial: ?T [cc]

Argument: SensorNbr
Min: 1 Max: 2*(Total Number of Motors) + 1

Syntax Scripting: result = getvalue(_T, cc)
result = getvalue(_TEMP, cc)

Reply:

T= cc Type: Signed 16-bit Min: -40 Max: 1000

Where:

cc =

For Single Channel Controllers:

1 : MCU temperature

2 : Heatsink Temperature

3: Motor Temperature

For dual or triple channel controllers*

1 : MCU temperature

2 : Channel 1 Heatsink Temperature

3 : Channel 2 Heatsink Temperature

4: Channel 1 Motor Temperature

5: Channel 2 Motor Temperature

6: Channel 3 Motor Temperature (if applicable)

tt = temperature in degrees

*Applicable for single channel versions of dual channel controllers.

Note:

On some controller models, additional temperature values may reported. These are measured at different points and not documented. You may safely ignore this extra data. Other controller models only have one heatsink temperature sensor and therefore only report one value in addition to the Internal IC temperature.

TM - Read Time

Alias: TIME HexCode: 1C CANOpen id: 0x2119

Description:

Reports the value of the time counter in controller models equipped with Real-Time clocks with internal or external battery backup. On older controller models, time is counted in a 32-bit counter that keeps track the total number of seconds, and that can be converted into a full day and time value using external calculation. On newer models, the time is kept in multiple registers for seconds, minutes, hours (24h format), dayofmonth, month, year in full.

Syntax Serial: ?TM [ee]

Argument: Element
Min: None Max: 6

Syntax Scripting: result = getvalue(_TM, ee)
result = getvalue(_TIME, ee)

Reply:

TM = nn Type: Unsigned 32-bit Min: 0

Where:

ee = date element in new controller model
1 : Seconds
2 : Minutes
3 : Hours (24h format)
4 : Dayofmonth
5 : Month
6 : Year in full
nn = Value

TR - Read Position Relative Tracking

Alias: TRACK HexCode: 20 CANOpen id: 0x2125

Description:

Reads the real-time value of the expected motor position in the position tracking closed loop mode and in speed position.

Syntax Serial: ?TR [cc]

Argument: Channel
Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_TR, cc)
result = getvalue(_TRACK, cc)

Reply:

TR=nn Type: Signed 32-bit Min: -2147M Max: 2147M

Where:

cc = Motor channel
nn = Position

TRN - Read Control Unit type and Controller Model

Alias: TRN HexCode: 1F CANOpen id:

Description:

Reports two strings identifying the Control Unit type and the Controller Model type. This query is useful for adapting the user software application to the controller model that is attached to the computer.

Syntax Serial: ?TRN

Argument: None

Syntax Scripting: result = getvalue(_TRN, 1)

Reply:

TRN=ss Type: String

Where:

ss = Control Unit Id String:Controller Model Id String

Example:

Q: ?TRN
R:TRN=RCB500:HDC2460

UID - Read MCU Id

Alias: UID HexCode: 32 CANOpen id:

Description:

Reports MCU specific information. This query is useful for determining the type of MCU: 100 = STM32F10X, 300 = STM32F30X. The query also produces a unique Id number that is stored on the MCU silicon.

Syntax Serial: ?UID [ee]

Argument: Element
Min: 1 Max: 5

Syntax Scripting: result = getvalue(_UID, ee)

Reply:

UID = nn	Type: Unsigned 32-bit	Min: 1	Max: 4294M

Where:

ee = Data element
1 : MCU type
2 : MCU Device Id
3-5 : MCU Unique ID
nn = value

V - Read Volts

Alias: VOLTS HexCode: 0D CANOpen id: 0x210D

Description:

Reports the voltages measured inside the controller at three locations: the main battery voltage, the internal voltage at the motor driver stage, and the voltage that is available on the 5V output on the DSUB 15 or 25 front connector. For safe operation, the driver stage voltage must be above 12V. The 5V output will typically show the controller's internal regulated 5V minus the drop of a diode that is used for protection and will be in the 4.7V range. The battery voltage is monitored for detecting the undervoltage or overvoltage conditions.

Syntax Serial: ?V [ee]

Argument: SensorNumber
Min: 1 Max: 3

Syntax Scripting: result = getvalue(_V, ee)
result = getvalue(_VOLTS, ee)

Reply:

V = nn Type: Unsigned 16-bit

Where:

ee =
1 : Internal volts
2 : Battery volts
3 : 5V output
nn = Volts * 10 for internal and battery volts. Milivolts for 5V output

Example:

```
Q: ?V
R:V=135:246:4730
Q: ?V 3
R:V=4730
```

VAR - Read User Integer Variable

Alias: VAR	HexCode: 06	CANOpen id: 0x2106

Description:

Read the value of dedicated 32-bit internal variables that can be read and written to/from within a user MicroBasic script. It is used to pass 32-bit signed number between user scripts and a microcomputer connected to the controller. The total number of user integer variables varies from one controller model to another and can be found in the product datasheet.

Syntax Serial: ?VAR [ee]

Argument: VarNumber
Min: 1 Max: Total Number of User Variables

Syntax Scripting: result = getvalue(_VAR, ee)

Reply:

VAR=nn	Type: Signed 32-bit	Min: -2147M	Max: 2147M

Where:

ee = Variable number
nn = Value

SL - Read Slip Frequency

Alias: SL	HexCode: 48	CANOpen id: 0x2136

Description:

This query is only used in AC Induction boards. Read the value of the Slip Frequency between the rotor and the stator of an AC Induction motor.

Syntax Serial: ?SL [cc]

Argument: VarNumber

Min: 1 Max: Total Number of Motors

Syntax Scripting: result = getvalue(_SL, cc)

Reply:

SL=nn	Type: Signed 16-bit	Min: -32768	Max: 32768

Where:

cc = Motor channel

nn = Slip Frequency in Hertz * 10

DS402 Runtime Queries

Runtime queries created to support DS402 specification are described below:

TABLE 15-18.

Command	Arguments	Description
AOM	Channel	Modes of Operation Display (DS402)
CW	Channel	Control Word (DS402)
SPE	Channel	Velocity Actual Value (DS402)
FEW	Element Value	Following Error Window (DS402)
FET	Element Value	Following Error Time Out (DS402)
HMD	Channel Value	Homing Method (DS402)
HSP	Element Value	Homing Speed (DS402)
INT	Element Value	Interpolation Time Period (DS402)
MSL	Element Value	Max Motor Speed (DS402)
PAC	Channel	Profile Acceleration (DS402)
PDC	Channel	Profile Deceleration (DS402)
PLT	Element Value	Software Position Limit (DS402)
POF	Channel	Position Offset (DS402)
POS	Channel	Target Position (DS402)
PSP	Channel	Profile Velocity (DS402)
PST	Channel	Position Actual Value (DS402)
RMP	Channel	Velocity Demand (DS402)
ROM	Channel	Modes of Operation (DS402)
S16	Channel	Target Velocity (DS402)
SAC	Element	Velocity Acceleration (DS402)
SDC	Element	Velocity Deceleration (DS402)
SDM	None	Supported Drive Modes (DS402)
SPC	Channel Value	Target Profile Velocity (DS402)
SPL	Element	Velocity Min/Max Amount (DS402)
SW	Channel	Status Word (DS402)
TC	Channel	Target Torque (DS402)
TOF	Channel	Torque Offset (DS402)
TRQ	Channel	Torque Actual Value (DS402)
TSL	Channel	Torque Slope (DS402)
VDV	Channel Value	Velocity Demand Value (DS402)
VSA	Channel Value	Velocity Sensor Actual Value (DS402)
VDV	Channel	Velocity Demand (DS402)
VNM	None	Version Number (DS402)
VOF	Channel	Velocity Offset (DS402)

AOM – Modes of Operation Display (DS402)

Alias: AOM HexCode: 63 CANOpen id: 0x6061

Description:

Read the actual operation mode.

Syntax Serial: ?AOM [cc]

Reply: AOM=nn

Syntax Scripting: nn = GetValue(_AOM, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Signed 8-bit

Where:

cc = Motor channel
nn = Actual operation mode (see the table below).

TABLE 15-19. Operation Modes

Value	Definition	Roboteq Operation Mode
-4[1]	Velocity Mode	Closed Loop Speed Position
-3[1]	Profile Velocity Mode	Closed Loop Speed Position
-2[1]	Profile Position Mode	Closed Loop Position Tracking Mode[2]
-1[1]	Profile Position Mode	Closed Loop Position Relative Mode$_2$
0	No Mode	Open Loop Mode
1	Profile Position Mode	Closed Loop Count Position Mode
2	Velocity Mode	Closed Loop Speed Mode
3	Profile Velocity Mode	Closed Loop Speed Mode
4	Torque Profile Mode	Closed Loop torque Mode
8	Cyclic Synchronous Position Mode	Closed Loop Count Position Mode
9	Cyclic Synchronous Velocity Mode	Closed Loop Speed Mode
10	Cyclic Synchronous Torque Mode	Closed Loop Torque Mode
[1]Roboteq Specific Modes [2]Not all Profile Position features can be supported with this mode.		

CW – Control Word (DS402)

Alias: CW HexCode: 56 CANOpen id: 0x6040

Description:

Read the value of the control word.

Syntax Serial: ?CW [cc]

Reply: CW=nn

Syntax Scripting: nn = GetValue(_CW, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Unsigned 16-bit

Where:

cc = Motor channel
nn = Control word value

SPE – Velocity Actual Value (DS402)

Alias: SPE HexCode: 96 CANOpen id: 0x6044, and 0x606C

Description:

Reads the velocity actual value in RPM.

Syntax Serial: ?SPE [cc]

Reply: SPE=nn

Syntax Scripting: nn = GetValue(_SPE, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Signed 32-bit

Where:

cc = Motor channel
nn = Velocity actual value

FEW - Following Error Window (DS402)

Alias: FEW HexCode:99 CanOpen id: 0x6065

Description:

Read the configured following error window for the position mode in counts. If the value is FFFF FFFFh, the following control is disabled.

Syntax Serial: ?FEW [cc]
Reply: FEW=nn

Syntax Scripting: nn=SetCommand(_FEW, cc)

Argument: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total number of motors
Result: Value	Type Unsigned 32-bit

Where:

cc= Motor Channel

nn = Following error window in counts

FET - Following Error Time Out (DS402)

Alias: FET HexCode:9A CanOpen id: 0x6066

Description:

Read the configured following error time out for the position mode in milliseconds.

Syntax Serial: ?FET[cc]

Reply: FET=nn

Syntax Scripting: nn = SetCommand(_FET, cc)

Argument: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total number of motors
Result: Value	Type: Unsigned 16-bit

Where:

cc = Motor channel

nn= Following error time out

HMD – Homing Method (DS402)

Alias: - HexCode: AD CANOpen id: 0X6098

Description:

Read the Homing Method that is used.

Syntax Serial: ?HMD [ee]

Reply: HMD = nn

Syntax Scripting: nn=getvalue(_HMD, ee)

Argument : Channel Unsigned 8-bit

Min: 1 Max: Total Number of Motors

Result: Homing Method number Type: Unsigned 8-bit

Min: 0 Max: 255

HSP – Homing Speed (DS402)

Alias: - HexCode: AE CANOpen id: 0X6099

Description:

Read the speed that will be used during the homing procedure. Each channel has 2 speed settings. The first is the speed during search for Home switch and the second is the speed during search for Index pulse (currently not supported).

Syntax Serial: ?HSP [ee]

Reply: HSP = nn

Syntax Scripting: nn=getvalue(_HSP, ee)

Argument : Channel Unsigned 8-bit

Min: 1 Max: Total Number of Motors

Result: Homing Speed (RPM) Type: Unsigned 32-bit

Min: 0 Max: 20000

Where:

ee=

1: Homing speed during search for Home switch for ch1

2: Homing speed during search for Index pulse (currently not supported) for ch1

3: Homing speed during search for Home switch for ch2

4: Homing speed during search for Index pulse (currently not supported) for ch2

INT - Interpolation Time Period (DS402)

Alias: INT HexCode: 9C CanOpen id: 0x60C2

Description:

Read the parameters for the Interpolation cycle time. The interpolation time base is the element 1 and the interpolation time index is element 2. The interpolation time value comes out of the following formula:

<Interpolation Time(seconds)> = <Interpolation Time Base> x 10^<Interpolation Time Index>

Syntax Serial: ?INT [ee]

Reply: INT=nn

Syntax Scripting: nn=GetValue(_INT,ee)

Argument: Element	Type:Unsigned 8-bit
Min: 1	Max: 2xTotal number of motors
Result: Value	Type: element 1: Unsigned 8-bit
	element 2: Signed 8-bit

Where

ee=

1: Interpolation time base for channel 1

2: Interpolation time index for channel 1

3: Interpolation time base for channel 2

4: Interpolation time index for channel 2

...

2x(m-1)+1: Interpolation time base for channel m

2x(m-1)+2: Interpolation time index for channel m

MSL - Max Motor Speed (DS402)

Alias: MSL HexCode: 9B CanOpen id: 0x6080

Description:

Read the configured maximum motor speed in profile position, profile velocity, profile torque, cyclic synchronous velocity, cyclic synchronous torque and cyclic synchronous position modes.

Syntax Serial: ?MSL[cc]

Reply: MSL=nn

Syntax Scripting: nn = GetValue(_MSL, cc)

Argument: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total number of motors
Result: Value	Type: Unsigned 32-bit

PAC – Profile Acceleration (DS402)

Alias: PAC HexCode: 5E CANOpen id: 0x6083

Description:

Read the configured acceleration in 10×RPM/second.

Syntax Serial: ?PAC [cc]

Reply: PAC=nn

Syntax Scripting: nn = GetValue(_PAC, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Unsigned 32-bit

Where:

cc = Motor channel
nn = Profile acceleration in 10×RPM/second

PDC – Profile Deceleration (DS402)

Alias: PDC HexCode: 5F CANOpen id: 0x6084

Description:

Read the configured deceleration in 10×RPM/second.

Syntax Serial: ?PDC [cc]

Reply: PDC=nn

Syntax Scripting: nn = GetValue(_PDC, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Unsigned 32-bit

Where:

cc = Motor channel
nn = Profile deceleration in 10×RPM/second

PLT - Software Position Limit (DS402)

Alias: PLT HexCode: 9D CanOpen id: 0x607D

Description:

Read the position limits.

Syntax Serial: ?PLT [ee]

Reply: PLT=nn

Syntax Scripting: nn = GetValue(_PLT, cc)

Argument: element	Type: Unsigned 8-bit
Min: 1	Max: 2xTotal number of motors

Result: Value	Type: element 1: Unsigned 8-bit
	element 2: Signed 8-bit

Where

ee=

1: Software position min limit for channel 1

2: Software position max limit for channel 1

3: Software position min limit for channel 2

4: Software position max limit for channel 2

...

2x(m-1)+1: Software position min limit for channel m

2x(m-1)+2: Software position max limit for channel m

POF - Position Offset (DS402)

Alias: POF HexCode: B0 CANOpen id: 0x60B0

Description:
Read the position offset.

Syntax Serial: ?POF [cc]

Reply: POF = nn

Syntax Scripting: result = getvalue(_POF, cc)

Argument: Channel		Type: Unsigned 8-bit
	Min: 1	Max: Total Number of motors

Result: Counts   Type: Signed 32-bit.

PST - Position Actual Value

Alias: PST   HexCode: 95   CanOpen id: 0x6064

Description:

Read the actual value of the position sensor.

Syntax Serial: ?PST[cc]

Reply: PST=nn

Syntax Scripting: nn = GetValue(_PST, cc)

Argument: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total number of motors
Result: Value	Type: Singed 32-bit

POS – Target Position (DS402)

Alias: POS HexCode: 5C CANOpen id: 0x607A

Description:

Read the configured target position.

Syntax Serial: ?POS [cc]

Reply: POS=nn

Syntax Scripting: nn = GetValue(_POS, cc)

Argument: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total number of motors
Result: Value	Type: Signed 32-bit

Where:

cc = Motor channel

nn = Target position

PSP – Profile Velocity (DS402)

Alias: PSP HexCode: 5D CANOpen id: 0x6081

Description:

Read the configured velocity in RPM.

Syntax Serial: ?PSP [cc]

Reply: PSP=nn

Syntax Scripting: nn = GetValue(_PSP, cc)

Argument: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total number of motors

Result: Value Type: Unsigned 16-bit

Where:

cc = Motor channel

nn = Profile velocity

RMP – VL Velocity Demand (DS402)

Alias: RMP HexCode: 62 CANOpen id: 0x6043

Description:

Read the instantaneous velocity in RPM generated by the ramp function. Positive values shall indicate forward direction and negative values shall indicate reverse direction.

Syntax Serial: ?RMP [cc]

Reply: RMP=nn

Syntax Scripting: nn = GetValue(_RMP, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Signed 32-bit

Where:

cc = Motor channel

nn = Velocity in RPM

ROM – Modes of Operation (DS402)

Alias: ROM HexCode: 5A CANOpen id: 0x6060

Description:

Read the configured modes of operation.

Syntax Serial: ?ROM [cc]

Reply: ROM=nn

Syntax Scripting: nn = GetValue(_ROM, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Signed 8-bit

Where:

cc = Motor channel

nn = Modes of operation

S16 – Target Velocity (DS402)

Alias: MOTVEL HexCode: 03 CANOpen id: 0x6042

Description:

Read the target velocity in RPM for velocity mode. Positive values shall indicate forward direction and negative values shall indicate reverse direction.

Syntax Serial: ?S [cc]

Reply: S=nn

Syntax Scripting: nn = GetValue(_S, cc)

nn = GetValue(_MOTVEL, cc)

Argument: Channel: Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Signed 16-bit

Min: -500000 Max: 500000

Where:

cc = Motor channel
nn = Target velocity in RPM

SAC – Velocity Acceleration (DS402)

Alias: SAC HexCode: 58 CANOpen id: 0x6048

Description:

Read the configured velocity acceleration.

Syntax Serial: ?SAC [ee]

Reply: SAC=nn

Syntax Scripting: nn = GetValue(_SAC, ee)

Argument: Element Type: Unsigned 8-bit

Min: 1 Max: 2 × Total number of motors

Result: Value Type: Unsigned 32-bit

Where:

ee =

1: Delta speed in 10×RPM for channel 1

2: Delta time in seconds for channel 1

3: Delta speed in 10×RPM for channel 2

4: Delta time in seconds for channel 2

…

2 × (m - 1) + 1: Delta speed in 10×RPM for channel m.

2 × (m - 1) + 1: Delta time in seconds for channel m.

nn = Delta speed/time

SDC – Velocity Deceleration (DS402)

Alias: SDC HexCode: 59 CANOpen id: 0x6049

Description:

Read the configured velocity deceleration.

Syntax Serial: ?SDC [ee]

Reply: SDC=nn

Syntax Scripting: nn = GetValue(_SDC, ee)

Argument: Element Type: Unsigned 8-bit

Min: 1 Max: 2 × Total number of motors

Result: Value Type: Unsigned 32-bit

Where:

ee =

1: Delta speed in 10×RPM for channel 1

2: Delta time in seconds for channel 1

3: Delta speed in 10×RPM for channel 2

4: Delta time in seconds for channel 2

…

2 × (m - 1) + 1: Delta speed in 10×RPM for channel m.

2 × (m - 1) + 1: Delta time in seconds for channel m.

nn = Delta speed/time

SDM – Supported Drive Modes (DS402)

Alias: SDM HexCode: 64 CANOpen id: 0x6502

Description:

Read the supported drive modes. Roboteq controllers support the following modes:

- Profile Position Mode (PP).
- Velocity Mode (VL).
- Profile Velocity Mode (PV).
- Torque Mode (TQ).

Syntax Serial: ?SDM

Reply: SDM=nn

Syntax Scripting: nn = GetValue(_TSL)

Result: Value Type: Unsigned 32-bit

Where:

nn = Supported drive modes

SPL – Velocity Min/Max Amount (DS402)

Alias: SPL HexCode: 57 CANOpen id: 0x6046

Description:
Read the configured minimum and maximum amount of velocity in RPM.

Syntax Serial: ?SPL [ee]

Reply: SPL=nn

Syntax Scripting: nn = GetValue(_SPL, ee)

Argument: Element Type: Unsigned 8-bit

Min: 1 Max: 2 × Total number of motors

Result: Value Type: Unsigned 32-bit

Where:

ee =
1: Min amount for channel 1
2: Max amount for channel 1
3: Min amount for channel 2
4: Max amount for channel 2
…
2 × (m - 1) + 1: Min amount for channel m.
2 × (m - 1) + 2: Max amount for channel m.
nn = Velocity max/min amount

SW – Status Word (DS402)

Alias: SW HexCode: 61 CANOpen id: 0x6041

Description:
Read the status of the PDS FSA.

TABLE 15-20. Status Word Mapping

15 14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
NU	OMS	ILA	TR	RM	MS	W	SOD	QS	VE	F	OE	SO	RTSO
MSB													LSB
NU → Not Used, OMS → Operation mode specific, ILA → Internal limit active TR → Target reached, RM → Remote, W → Warning, SOD → Switch on disabled, QS → Quick stop, VE → Voltage enabled, F → Fault, OE → Operation Enabled, SO → Switch on RTSO → Ready to switch on.													

If bit 4 (voltage enabled) of the status word is always 1. If bit 5 (quick stop) of the status word is 0, this shall indicate that the PDS is reacting on a quick stop request (quick stop mode is always 2). Bit 7 (warning) is always 0. Bit 9 (remote) of the status word is always 1 when the control is done only via the DS402 commands and queries. In case of any command triggerred by any other interface or action by any trigger input (diigtal inputs, sensor limits, etc.), then this bit goes to 0. If bit 10 (target reached) of the status word is 1, this shall indicate that the PDS has reached the set-point. Bit 10 shall also be set to 1, if the operation mode has been changed. The change of a target value by software shall alter this bit. If halt occurred and the PDS has halted then bit 10 shall be set to 1, too. If the same internal value is commanded then bit 10 shall not alter, if bit 10 is supported (see Table 15-22). If bit 11 (internal limit active) of the status word is 1, this shall indicate that, current limit has been reached or the motor command is out of limits.

TABLE 15-21. State Coding

Status Word	PDS FSA state
xxxx xxxx x0xx 0000_b	Not ready to switch on
xxxx xxxx x1xx 0000_b	Switch on disabled
xxxx xxxx x01x 0001_b	Ready to switch on
xxxx xxxx x01x 0011_b	Switched on
xxxx xxxx x01x 0111_b	Operation enabled
xxxx xxxx x00x 0111_b	Quick stop active
xxxx xxxx x0xx 1111_b	Fault reaction active
xxxx xxxx x0xx 1000_b	Fault

TABLE 15-22. Definition of Bit 10

Bit	Value	Definition
10	0	Halt (bit 8 in control word) = 0: Speed or Position Target not reached Halt (bit 8 in control word) = 1: Axis decelerates
	1	Halt (bit 8 in control word) = 0: Speed or Position Target reached Halt (bit 8 in control word) = 1: Velocity of axis is 0
Note: In Roboteq controllers, Halt operation mode is 2. Slow down on slow down ramp and stay in operation enable.		

Profile Position Mode

TABLE 15-23. Status Word Mapping in Profile Position Mode

15–14	13	12	11	10	9–0
see Table 15-20	Not Used	Set-Point Acknowledge	see Table 15-20	Target Reached	see Table 15-20
MSB					LSB

In Profile Position Mode the operation specific bits are mapped in Table 15-23. Status Word Mapping in Profile Position Mode with bits 10 and 12 user can acknowledge the status of the controller as shown in Table 10 and Table 12. Bit 13 is always 0.

Profile Torque Mode

The profile torque mode uses some bits of the statusword for mode specific purposes. Table 15-24 shows the structure of the status word. Target torque reached is defined table 15-25.

TABLE 15-24. Statusword for profile torque mode

15–14	13–12	11	10	9–0
see Table 15-20	reserved	see Table 15-20	Target reached	see Table 15-20
MSB				LSB

TABLE 15-25. Definition of bit 10

Bit	Value	Definition
10	0	Halt (bit 8 in controlword) = 0: Target torque not reached Halt (bit 8 in controlword) = 1: Axis decelerates
	1	Halt (bit 8 in controlword) = 0: Target torque reached Halt (bit 8 in controlword) = 1: Velocity of axis is 0

Velocity Mode

The Velocity mode uses some bits of the statusword for mode specific purposes. Table 15-26 shows the structure of the status word.

TABLE 15-26. Statusword for velocity mode

15–14	13–12	11	10	9–0
see Table 15-20	reserved (0)	see Table 15-20	reserved (0)	see Table 15-20
MSB				LSB

Cyclic Synchronous Position Mode

The cyclic synchronous position mode uses three bits of the statusword for mode-specific purposes. Table 15-27 shows the structure of the statusword. Table 15-28 defines the values for bit 10, 12, and 13 of the statusword.

TABLE 15-27. Statusword for profile cyclic synchronous position mode

15 14	13	12	11	10	9 0
see Table 15-20	Following error	Drive follows the command value	see Table 15-20	reserved	see Table 15-20
MSB					LSB

TABLE 15-28. Definition of bit 10, bit 12, and bit 13

Bit	Value	Definition
10	0	Reserved
	1	Reserved
12	0	Drive does not follow the command value – Target position ignored
	1	Drive follows the command value – Target position used as input to position control loop
13	0	No following error
	1	Following error

Cyclic Synchronous Velocity Mode and Cyclic Synchronous Torque Mode

The Cyclic synchronous velocity and Cyclic synchronous torque mode use some bits of statusword. Table 15-29 shows the structure of the statusword. Table 15-30) defines the values for bit 10, 12, and 13 of the statusword.

TABLE 15-29. Statusword for profile cyclic synchronous velocity mode

15 14	13	12	11	10	9 0
see Table 15-20	reserved	Drive follows the command value	see Table 15-20	reserved	see Table 15-20
MSB					LSB

TABLE 15-30. Definition of bit 10, bit 12, and bit 13

Bit	Value	Definition
10	0	Reserved
	1	Reserved
12	0	Target velocity or torque ignored
	1	Target velocity or torque used as input to velocity or torque control loop.
13	0	Reserved
	1	Reserved

Profile Velocity Mode

TABLE 15-31. Status Word Mapping in Profile Velocity Mode

15 – 14	13	12	11	10	9 – 0
see Table 15-20	Not Used	Speed	see Table 15-20	Target Reached	see Table 15-20
MSB					LSB

In Profile Velocity Mode the operation specific bits are mapped in Table 15-31. With bits 10 and 12 user can acknowledge the status of the controller as shown in Table 15-32 and Table 15-33. Bit 13 is always 0.

TABLE 15-32. Definition of bit 12 in Profile Velocity Mode

Bit	Value	Definition
12	0	Speed is not equal 0
	1	Speed is equal 0

TABLE 15-33. Definition of Bit 12 in Profile Position Mode

Bit	Value	Definition
12	0	Previous set-point already processed, waiting for new set-point
	1	Previous set-point still in process, set-point overwriting shall be accepted

Syntax Serial: ?SW [cc]

Reply: SW=nn

Syntax Scripting: nn = GetValue(_SW, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Unsigned 16-bit

Where:

cc = Motor channel

nn = Status word value

TC – Target Torque (DS402)

Alias: TC HexCode: 5B CANOpen id: 0x6071

Description:

Read the configured target torque in per thousand of rated torque, when the controller is in torque mode. Beware that in order to have correct results, the configuration commands nominal current (NOMA) and torque constant (TNM) must be set appropriately.

Syntax Serial: ?TC [cc]

Reply: TC=nn

Syntax Scripting: nn = GetValue(_TC, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Signed 16-bit

Where:

cc = Motor channel
nn = Torque input value in 100×Nm

TOF - Torque Offset (DS402)

Alias: TOF HexCode: B2 CANOpen id: 0x60B2

Description:

Read the torque offset added to the commanded torque. Commanded torque could be either directly from user (in torque mode), or produced from speed or position control loops. Beware in order to have correct values make sure to have configured appropriately configuration command TNM.

Syntax Serial: ?TOF [cc]

Reply: TOF = nn

Syntax Scripting: result = getvalue(_TOF, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total Number of motors

Result: miliNm Type: Signed 32-bit.

TRQ – Target Torque (DS402)

Alias: TRQ HexCode: 7A CANOpen id: 0x6077

Description:

Read the actual torque in 100×Nm. Beware in order to have correct values make sure to have configured appropriately configuration command TNM.

Syntax Serial: ?TRQ [cc]

Reply: TRQ=nn

Syntax Scripting: nn = GetValue(_TRQ, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Signed 16-bit

Where:

cc = Motor channel

nn = Actual torque 100×Nm

TSL – Torque Slope (DS402)

Alias: TSL HexCode: 60 CANOpen id: 0x6087

Description:

Read the configured rate of change of torque command. Beware in order to have correct values make sure to have configured appropriately configuration command TNM.

Syntax Serial: !TSL [cc]

Reply: TSL=nn

Syntax Scripting: nn = GetValue(_TSL, cc)

Argument: Channel Type: Unsigned 8-bit

Min: 1 Max: Total number of motors

Result: Value Type: Unsigned 32-bit

Where:

cc = Motor channel

nn = Torque slope

VDV – Velocity Demand (DS402)

Alias: - HexCode: 97 CANOpen id: 0x606B

Description:

It reads the instantaneous velocity demand in RPM. In Speed mode, it coincides with the RPM command. In Position modes, provided cascaded control is enabled, the command is the output of the Position loop that is used as input in the Speed loop.

Syntax Serial: ?VDV [cc]

Argument: Channel	Type: Unsigned 8-bit
Min: 1	Max: Total number of motors

Syntax Scripting: nn = GetValue(_VDV, cc)

Reply: RMP=nn	Type: Signed 32-bit	Min: -65535	Max: 65535

Where:

cc = Motor channel
nn = Velocity demand in RPM

VNM – Version Number (DS402)

Alias: VNM HexCode: 65 CANOpen id: 0x67FE

Description:

Read the version number of the CiA 402 profile.

Syntax Serial: !VNM

Reply: VNM=nn

Syntax Scripting: nn = GetValue(_TSL)

Result: Value Type: Unsigned 32-bit

Where:

nn = Version number

VOF - Velocity Offset (DS402)

Alias: VOF HexCode: B1 CANOpen id: 0x60B1

Description:

Read the velocity offset added to velocity command. Velocity command could be either directly set from user, in speed mode operation, or produced from position control loop.

Syntax Serial: ?VOF [cc]

Reply: VOF = nn

Syntax Scripting: result = getvalue(_VOF, cc)

Argument: Channel	Type: Unsigned 8-bit	
	Min: 1	Max: Total Number of motors
Result: RPM	Type: Signed 32-bit.	

Query History Commands

Every time a Real Time Query is received and executed, it is stored in a history buffer from which it can be recalled. The buffer will store up to 16 queries. If more than 16 queries are received, the new one will be added to the history buffer while the firsts are removed in order to fit the 16 query buffer.

Queries can then be called from the history buffer using manual commands, or automatically, at user selected intervals. This feature is very useful for monitoring and telemetry.

Additionally, the history buffer can be loaded with a set of user selected queries at power on so that the controller can automatically issue operating values immediately after power up. See "TELS - Telemetry String" configuration command for details on how to set up the startup Telemetry string. "Another feature is the streams. In this case the data can be printed after a prefix and separated with a delimiter. In order to enable a stream, the special character "/" needs to be typed in front of the first query.

A command set is provided for managing the history buffer. These special commands start with a "**#**" character.

TABLE 15-34. Query History Commands

Command	Description
#	Send the next value. Stop automatic sending
# C	Clear buffer history
# nn	Start automatic sending
# xx nn	Start automatic sending for specific stream
/"<prefix>","<delimiter>"?Q cc	Create data streams
//?	Dump the streams' prefixes and delimiters

# - Send Next History Item / Stop Automatic Sending

A **#** alone will call and execute the next query in the buffer. If the controller was in the process of automatically sending queries from the buffer, then receiving a # will cause the sending to stop.

When a query is executed from the history buffer, the controller will only display the query result (e.g. A=10:20). It will not display the query itself.

Syntax:

#

Reply: **QQ**

Where:

QQ = is reply to query in the buffer.

# C - Clear Buffer History

This command will clear the history buffer of all queries that may be stored in it. If the controller was in the process of automatically sending queries from the buffer, then receiving this command will also cause the sending to stop

Syntax:

# C

Reply: None

# nn - Start Automatic Sending

This command will initiate the automatic retrieving and execution of queries from the history buffer. The number that follows the command is the time in milliseconds between repetition. A single query is fetched and executed at each time interval.

Syntax:

# nn

Reply: **QQ** at every nn time intervals

Where:

QQ = is reply to query in the buffer.
nn = time in ms

Range: **nn** = 1 to 32000ms

# xx nn - Start automatic sending for specific stream

Using this syntax one can set the refresh rate of each stream. This is used only in case of using streams.

Syntax:

# xx nn

Reply: The respective stream at every nn ms.

Where:

xx = the stream.
nn = time in ms.

/?Q cc - Create data streams

Using this syntax the next queries that are going to be sent will be printed after a prefix and separated by a delimiter. The default prefix is none and the default delimiter is tab. There can be created up to 3 streams and can have different refresh rates.

Syntax:

/"<prefix>","<delimiter>"?Q cc

Reply: The regular reply of the query ?Q cc.

Where:

Q = is reply the query symbol.

cc = motor channel

prefix = the prefix for the stream

delimiter = the delimiter for the stream

For example, if one wants a log of the Motor Power(P), Motor Amps(A) and Encoder Speed(S) of channel 1, with refresh rate 50ms, prefix "d=" and delimiter ':' they should type:

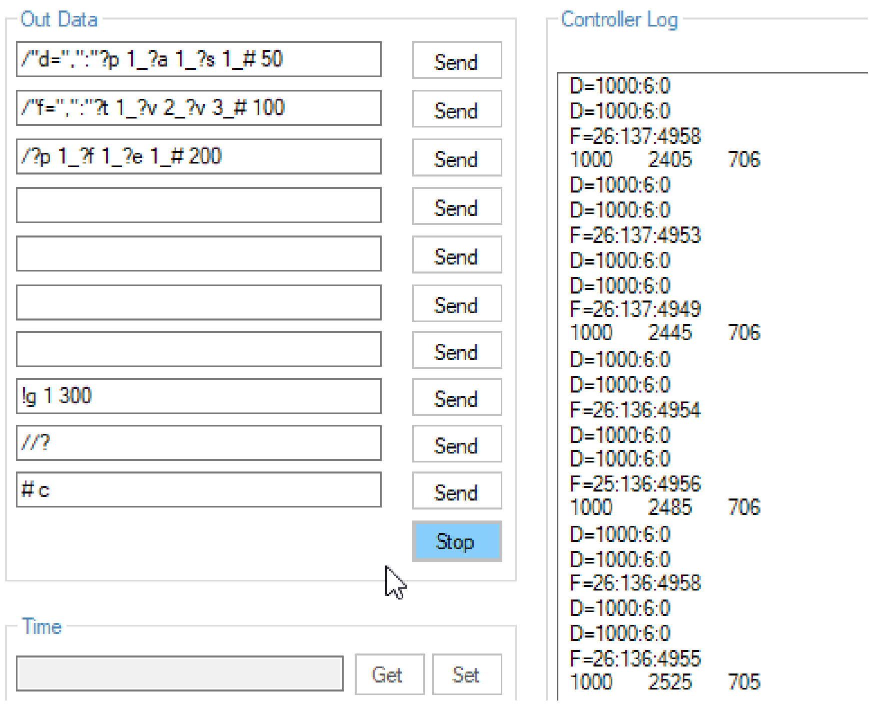
/"d=","":"?p 1_?a 1_?s 1_# 50

Two more stream examples are shown below:

/"f=","":"?t 1_?v 2_?v 3_# 100

/?p 1_?f 1_?e 1_# 200

The outcome of these three streams are shown in the image below:



```
/"d=",":"?p 1_?a 1_?s 1_# 50
/"f=",":"?t 1_?v 2_?v 3_# 100
/?p 1_?f 1_?e 1_# 200
!g 1 300
//?
# c
```

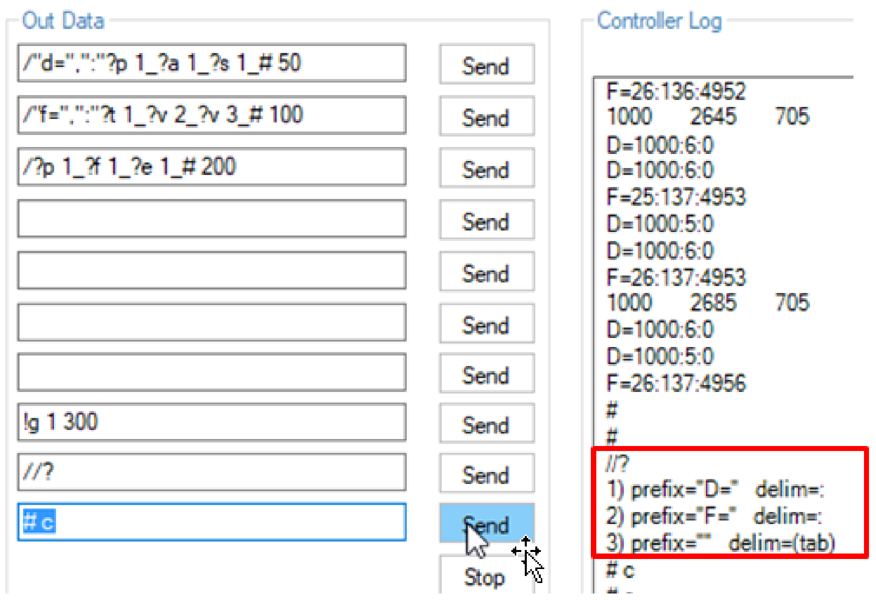
```
D=1000:6:0
D=1000:6:0
F=26:137:4958
1000    2405    706
D=1000:6:0
D=1000:6:0
F=26:137:4953
D=1000:6:0
D=1000:6:0
F=26:137:4949
1000    2445    706
D=1000:6:0
D=1000:6:0
F=26:136:4954
D=1000:6:0
D=1000:6:0
F=25:136:4956
1000    2485    706
D=1000:6:0
D=1000:6:0
F=26:136:4958
D=1000:6:0
D=1000:6:0
F=26:136:4955
1000    2525    705
```

FIGURE 15-2. Data streams

In order to clear the streams # c can be sent and then, the legacy history method can be used. In order to stop the streams but not delete them send #. then using command # xx nn one can restart the streams with nn refresh rate.

//? - Dump the streams' prefixes and delimiters

Using this syntax one can see in each of the 3 streams which prefix and delimiter is set. In that case one can know which stream respects to each stream number.



```
/"d=",":"?p 1_?a 1_?s 1_# 50
/"f=",":"?t 1_?v 2_?v 3_# 100
/?p 1_?f 1_?e 1_# 200
!g 1 300
//?
# c
```

```
F=26:136:4952
1000    2645    705
D=1000:6:0
D=1000:6:0
F=25:137:4953
D=1000:5:0
D=1000:6:0
F=26:137:4953
1000    2685    705
D=1000:6:0
D=1000:5:0
F=26:137:4956
#
#
//?
1) prefix="D="   delim=:
2) prefix="F="   delim=:
3) prefix=""   delim=(tab)
# c
```

FIGURE 15-3. Data streams prefixes

Maintenance Commands

This section contains a few commands that are used occasionally to perform maintenance functions.

TABLE 15-35. Maintenance Commands

Command	Arguments	Description
CLMOD	None	Motor/Sensor Setup
CLSAV	Key	Save calibrations to Flash
DFU	Key	Update Firmware via USB/CANOpen
EELD	None	Load Parameters from EEPROM
EELOG	None	Dump Flash Log Data
EERST	Key	Reset Factory Defaults
EESAV	None	Save Configuration in EEPROM
ERASE	None	Erase Flash Log Data
LK	Key	Lock Configuration Access
RESET	Key	Reset Controller
SLD	Key	Script Load
STIME	Time	Set Time
UK	Key	Unlock Configuration Access

CLMOD – Motor/Sensor Setup

Argument: None

Description:

This command is used in order to perform motor and/or sensor setup, and will configure accordingly the respective fields in order to have a smooth motor spin and alignment between the motor and the sensor directions. During setup no motor command can be applied to the motors. For more details see Section 8.

Note: The motor will spin.

Syntax: %CLMOD nn

Where:

0: Exit/Stop Setup Mode.

2: Setup channel 1.

3: Setup channel 2.

CLSAV - Save calibrations to Flash

Argument: Key

Description:

Saves changes to calibration to Flash. Calibration parameters are stored permanently until new values are stored. This command must be used with care and must be followed by a 9-digit safety key to prevent accidental use.

Syntax:
%CLSAV safetykey

Where:
safetykey = 321654987

DFU - Update Firmware via USB/CANOpen

Argument: Key

Description:

Firmware update can be performed via the RS232 port via USB or via CANOpen. When done via USB or CANOpen, the DFU command is used to cause the controller to enter in the firmware upgrade mode. This command must be used with care and must be followed by a 9-digit safety key to prevent accidental use. Once the controller has received the DFU command, it will no longer respond to the PC utility and no longer be visible on the PC. When this mode is entered, you must either launch Roborun+ DFU Loader, in case of USB, or Roborun+ Firmware Loader, in case of CANOpen. In case of CANOpen the default parameters are used which are Node ID 126 and Bit Rate 125Kb/s.

Syntax:
%DFU safetykey

Where:
safetykey = 321654987

EELD - Load Parameters from EEPROM

Argument: None

Description:

This command reloads the configuration that are saved in EEPROM back into RAM and activates these settings.

Syntax:
%EELD

EELOG - Dump Flash Log Data

Argument: None

Description:

This command is used in order to Dump the Flash Log Data. Each data entry is printed in a row with each value to be separated with a tab. The values, that are going to be printed, are the following:

- timestamp (milisseconds since power-up),
- maximum absolute motor current since the time of the previous entry,
- maximum battery voltage since the time of the previous entry,
- maximum heatsink temperature since the time of the previous entry and
- Fault Flags that have caused the entry.

Each entry is saved to flash at every new fault or every 10 minute when a fault is active.

Note: Flash Log Data are not working when any of the motor channels are configured in resolver sinusoidal mode.

Syntax: %EELOG

EERST - Reset Factory Defaults

Argument: Key

Description:

The EERST command will reload the controller's RAM and EEPROM with the factory default configuration. Beware that this command may cause the controller to no longer work in your application since all your configurations will be erased back to factory defaults. This command must be used with care and must be followed by a 9-digit safety key to prevent accidental use.

Syntax:
%EERST safetykey

Where:
safetykey = 321654987

EESAV - Save Configuration in EEPROM

Argument: None

Description:

Controller configuration that have been changed using any Configuration Command can then be saved in EEPROM. Once in EEPROM, it will be loaded automatically in the controller every time the unit is powered on. If the EESAV command is not called after changing a configuration, the configuration will remain in RAM and active only until the controller is turned off. When powered on again, the previous configuration that was in the EEPROM is loaded. This command uses no parameters.

Syntax:
%EESAV

ERASE - Erase Flash Log Data

Argument:None

Description:

This command is used in order to Erase the Flash Log Data.

Syntax: %ERASE

LK - Lock Configuration Access

Argument: Key

Description:

This command is followed by any user-selected secret 32-bit number. After receiving it, the controller will lock the configuration and store the key inside the controller, in area which cannot be accessed. Once locked, the controller will no longer respond to configuration reads. However, it is still possible to store or to set new configurations.

Syntax:
%LK secretkey

Where:
secretkey = 32-bit number (1 to 4294967296)

RESET - Reset Controller

Argument: Key

Description:

This command will cause the controller to reset similarly as if it was powered OFF and ON. This command must be used with care and must be followed by a 9-digit safety key to prevent accidental reset.

Syntax:
%RESET safetykey

Where:
safetykey = 321654987

SLD - Script Load

Argument: Key

Description:

After receiving this command, the controller will enter the script loading mode. It will reply with HLD and stand ready to accept script bytecodes in intel Hex Format. The exact download and data format is described in the MicroBasic section of the manual. The timeout for the communication is 3 seconds. After the timeout expires, the controller will return "-".

Syntax:
%SLD

STIME - Set Time

Argument: Hours Mins Secs

Description:

This command sets the time inside the controller's clock that is available in some controller models. The clock circuit will then keep track of time as long as the clock remains under power. On older controller models, the clock is a single 32-bit counter in which the number of seconds from a preset day and time is stored (for example 02/01/00 at 3:00). On newer model, the clock contains 6 registers for seconds, dates, minuted, hours, dayofmonth, month, year. The command syntax will be different for each of these models.

Syntax:
%STIME nn : Older models%STIME ee nn : Newer models

Where:
Older models:nn = number of secondsNewer modelsee = 1: Seconds2: Minutes3: Mours (24h format)4: Dayofmonth5: Month6: Year in fullnn = Value

UK - Unlock Configuration Access

Argument: Key

Description:

This command will release the lock and make the configuration readable again. The command must be followed by the secret key which will be matched by the controller internally against the key that was entered with the LK command to lock the controller. If the keys match, the configuration is unlocked and can be read.

Syntax:
%UK secretkey

Where:
secretkey = 32-bit number (1 to 4294967296)

Set/Read Configuration Commands

These commands are used to set or read all the operating parameters needed by the controller for its operation. Parameters are loaded from EEPROM into RAM, from where they are and then used every time the controller is powered up or restarted.

Important Notices

The total number of configuration parameters is very large. To simplify the configuration process and avoid errors, it is highly recommended to use the RoborunPlus PC utility to read and set configuration.
Some configuration parameters may be absent depending on the presence or absence of the related feature on a particular controller model.

Setting Configurations

The general format for setting a parameter is the "**^**" character followed by the command name followed by parameter(s) for that command. These will set the parameter in the controller's RAM and this parameter becomes immediately active for use. The parameter can also be permanently saved in EEPROM by sending the **%EESAV** maintenance command.

Some parameters have a unique value that applies to the controller in general. For example, overvoltage or undervoltage. These configuration commands are therefore followed by a single parameter:

^UVL 100 : Sets Undervoltage limit to 10.0V
^OVL 400 : Sets Overvoltage limit to 40.0V

Other parameters have multiple value, with typically one value applying to a different channel. Multiple value parameters are numbered from 1 to n. For example, Amps limit for a motor channel or the configuration of an analog input channel.

^ALIM 1 250 : Sets Amps limit for channel 1 to 25.0A
^AMIN 4 2000 : Sets low range of analog input 4 to 2000

Using 0 as the first parameter value will cause all elements to be loaded with the same content.

^ADB 0 10 : Sets the deadband of all analog inputs to 10%

Important Notice

Saving configuration into EEPROM can take up to 20ms per parameter. The controller will suspend the loop processing during this time, potentially affecting the controller operation. Avoid saving configuration to EEPROM during motor operation.

Reading Configurations

Configuration parameters are read by issuing the "**~**" character followed by the command name and with an optional channel number parameter. If no parameter is sent, the controller will give the value of all channels. If a channel number is sent, the controller will give the value of the selected channel.

The reply to parameter read command is the command name followed by "**=**" followed by the parameter value. When the reply contains multiple values, then the different values are separated by "**:**". The list below describes every configuration command of the controller. For Example:

~ALIM : Read Amps limit for all channels

Reply: **ALIM= 750:650**
~ALIM 2: Read Amps limit for channel 2

Reply: **ALIM= 650**

Configuration parameters can be read from within a MicroBasic script using the getconfig() function. The setconfig() function is used to load a new value in a configuration parameter.

Important Warning

Configuration commands can be issued at any time during controller operation. Beware that some configuration parameters can alter the motor behavior. Change configurations with care. Whenever possible, change configurations while the motors are stopped.

Configuration Read Protection

The controller may be locked to prevent the configuration parameters to be read. Given the large number of possible configurations, this feature provides effective system-level copy protection. The controller will reply to configuration read requests only if the read protection is unlocked. If locked, the controller will respond a "**-**" character.

General Configuration and Safety

The commands in this group are used to configure the controller's general and safety settings.

TABLE 15-36. General and Safety Configurations

Command	Arguments	Description
ACS	Enable	Analog Center Command to Start
AMS	Enable	Analog keep within Guard Bands
BEE	Address Value	User Storage in Battery Backed RAM
BRUN	Enable	Script Auto-Start
CLIN	Channel Linearity	Command Linearity
CPRI	Level Command	Command Priorities
DFC	Channel Value	Default Command value
DMOD	Mode	Modbus Mode
ECHOF	OffOn	Enable/Disable Serial Echo
EE	Address Data	User-Defined Values
FLCL	None	Automatic Fault Clearance
ISM	None	Raw Redirect Mode
MDAL	Option	Modbus Data Alignment
MNOD	ID	Modbus Slave ID

Command	Arguments	Description
PMS	Enable	Pulse keep within Min & Max Safety
RS485	Enable	Enable RS485
RSBR	BitRate	Set RS232/RS485 baudrate
RWD	Timeout	Serial Data Watchdog
SCRO	Port	Select Print output port for scripting
STO	Enable	Safe Torque Off
TELS	String	Telemetry string

ACS - Analog Center Command to Start

HexCode: 0B CANOpen id: 0x300B

Description:

This parameter enables the analog safety that requires that the input be at zero or centered before it can be considered as good. This safety is useful when operating with a joystick and requires that the joystick be centered at power up before motors can be made to run. On mutli-channel controllers, this configuration acts on all analog command inputs, meaning that all joysticks must be centered before any one becomes active.

Syntax Serial: ^ACS nn
~ACS

Syntax Scripting: setconfig(_ACS, nn)

Number of Arguments: 1

Argument 1: Enable

Type: Unsigned 8-bit
Min: 0 Max: 1
Default: 1

Where:
nn =
0: Safety disabled
1: Safety enabled

AMS - Analog keep within Guard Bands

HexCode: 0C CANOpen id: 0x300C

Description:

This configuration is used to make sure that the analog input command is always within a user preset minimum and maximum safe value. It is useful to detect, for example, that the wire connection to a command potentiometer is broken. If the safety is enabled and the input is outside the safe range, the Analog input command will be considered invalid. The controller will then apply a motor command based on the priority logic.

Syntax Serial: ^AMS nn
~AMS

Syntax Scripting: setconfig(_AMS, nn)

Number of Arguments: 1

Argument 1: Enable

	Type: Unsigned 8-bit	
	Min: 0	Max: 1
	Default: 1 = Enabled	

Where:
nn =
0: Disabled
1: Enabled

BEE - User Storage in Battery Backed RAM

HexCode: 64 CANOpen id: 0x3064

Description:

Store and retrieve user data in battery backed RAM. Storage is quasi permanent, limited only by the on-board battery (usually several years) . Unlike storage in Flash using the EE configuration commands, there are no limits in the amount or frequency of read and write cycles with BEE. This feature is only available on selected models, see product datasheet. Battery must be installed in the controller for storage to be possible.

Syntax Serial: ^BEE aa dd
~BEE aa

Syntax Scripting: setconfig(_BEE, aa, dd)

Number of Arguments: 2

Argument 1: Address

	Min: 1	Max: Total Number of BEE

Argument 2: Value

	Type: Signed 16-bit	
	Min: -32768	Max: 32767
	Default: 0	

Where:

aa = Address

dd = Data

Example:

^BEE 1 555 : Store value 555 in Battery Backed RAM location 1
~BEE 1: Read data from RAM location 1

BRUN - Script Auto-Start

HexCode: 48 CANOpen id: 0x3048

Description:

This parameter is used to enable or disable the automatic MicroBasic script execution when the controller powers up. When enabled, the controller checks that a valid script is present in Flash and will start its execution 2 seconds after the controller has become active. The 2 seconds wait time can be circumvented by putting 2 in the command argument. However, this must be done only on scripts that are known to be bug-free. A crashing script will cause the controller to continuously reboot with little means to recover.

Syntax Serial: ^BRUN nn
~BRUN

Syntax Scripting: setconfig(_BRUN, nn)

Number of Arguments: 1

Argument 1: Enable

Type: Unsigned 8-bit
Min: 0 Max: 2
Default: 0 = Disabled

Where:

nn =
0: Disabled
1: Enabled after 2 seconds
2: Enabled immediately

CLIN - Command Linearity

HexCode: 0D CANOpen id: 0x300D

Description:

This parameter is used for applying an exponential or a logarithmic transformation on the command input, regardless of its source (serial, pulse or analog). There are 3 exponential and 3 logarithmic choices. Exponential correction make the commands change less at the beginning and become stronger at the end of the command input range. The logarithmic correction will have a stronger effect near the start and lesser effect near the end. The linear selection causes no change to the input. A linearity transform is also available for all analog and pulse inputs. Both can be enabled although in most cases, it is best to use the Command Linearity parameter for modifying command profiles.

Syntax Serial: ^CLIN cc nn
~CLIN [cc]

Syntax Scripting: setconfig(_CLIN, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Linearity

	Type: Unsigned 8-bit	
	Min: 0	Max: 7
	Default: 0 = Linear	

Where:

cc = Motor channel
nn =
0: Linear (no change)
1: Exp weak
2: Exp medium
3: Exp strong
4: Log weak
5: Log medium
6: Log strong

Example:

^CLIN 1 1 : Sets linearity for channel 1 to exponential weak

CPRI - Command Priorities

HexCode: 07 CANOpen id: 0x3007

Description:

This parameter contains up to 3 variables and is used to set which type of command the controller will respond in priority and in which order. The first item is the third priority, the second item is the fourth priority, and the third item is the fifth priority. The first priority belongs to the script mode and the second priority belongs to the CAN mode. Each priority item is then one of the three command modes: Serial, Analog or RC Pulse. See Command Priorities in the User Manual. Default priority orders are: 1-Serial, 2-Pulse, 3-None.

Syntax Serial: ^CPRI pp nn
~CPRI [pp]

Syntax Scripting: setconfig(_CPRI, pp, nn)

Number of Arguments: 2

Argument 1: Level

	Min: 1	Max: 3 or 4
	Default: See description	

Argument 2: Command

	Type: Unsigned 8-bit	
	Min: 0	Max: 2 or 3
	Default: See description	

Where:

pp = Priority rank
nn =
0: Serial
1: RC

2: Analog
3: None
4: None

Example:

^CPRI 1 2 : Set Analog as first priority
~CPRI 2 : Read what command mode is second priority

Note:

USB, RS232, RS485 and TCP commands share the "Serial" type. When serial commands come from different Serial source, they are executed in the order received.

DFC - Default Command value

HexCode: 0E CANOpen id: 0x300E

Description:

The default command values are the command applied to the motor when no valid command is fed to the controller. Value 1001 causes no change in position at power up until a new position command is received.

Syntax Serial: ^DFC cc nn
~DFC [cc]

Syntax Scripting: setconfig(_DFC, cc, nn)

Number of Arguments: 2

Argument 1: Channel		
	Min: 1	Max: Total Number of Motors
Argument 2: Value		
	Type: Signed 16-bit	
	Min: -1000	Max: 1001
	Default: 0	

Where:

cc : Motor channel

nn : Command value

Example:

^DFC 1 500 : Sets motor command to 500 when no command source are detected
^DFC 2 1001 : Motor takes present position as destination after power up. Motor doesn't move.

DMOD – Modbus Mode

HexCode: A1 CANOpen id: 0x30A1

Description:

Configure this parameter in order to enable Modbus and the desired mode.

Syntax Serial: ^DMOD nn
~DMOD

Syntax Scripting: setconfig(_DMOD, nn)

Number of Arguments: 1

Argument 1: Modbus Mode
Type: Unsigned 8-bit

Min: 0 Max: 4

Default: 0

Where:

nn =

0: Off

1: TCP

2: RTU over TCP

3: RS232 ASCII

4: RS485 ASCII

5: RS232 RTU

6: RS485 RTU

Example:

^DMOD 3: Enable Modbus RS232 ASCII mode.

ECHOF - Enable/Disable Serial Echo

HexCode: 09 CANOpen id: 0x3009

Description:

This command is used to disable/enable the echo on the RS232, RS485, TCP or USB port. By default, the controller will echo everything that enters the serial communication port. By setting ECHOF to 1, commands are no longer being echoed. The controller will only reply to queries and the acknowledgements to commands can be seen.

Syntax Serial: ^ECHOF nn
~ECHOF

Syntax Scripting: setconfig(_ECHOF, nn)

Number of Arguments: 1

Argument 1: OffOn
Type: Unsigned 8-bit
Min: 0 Max: 1
Default: 0 = Echo on

Where:

nn =
0: Echo is enabled
1: Echo is disabled

Example:

^ECHOF 1 : Disable echo

EE - User-Defined Values

HexCode: 00 CANOpen id: 0x3000

Description:

Read and write user-defined values that can be permanently stored in Flash. Storage area size is typically 32 x 16-bit words but can vary from one product to the other. The command alters data contained in a RAM area. The %EESAV Maintenance Command, or !EES Real Time Command must be used to copy the RAM array to Flash. The Flash is copied to RAM every time the device powers up.

Syntax Serial: ^EE aa dd
~EE aa

Syntax Scripting: setconfig(_EE, aa, dd)

Number of Arguments: 2

Argument 1: Address

Min: 1 Max: Total Number of Storage words

Argument 2: Data

Type: Signed 16-bit
Min: -32768 Max: +32767
Default: 0

Where:

aa = Address
dd = Data

Example:

^EE 1 555 : Store value 555 in RAM location 1
%EESAV or !EES : Copy data from temporary RAM to Flash
~EE 1 : Read data from RAM location 1

Note:

See product datasheet to know the total available EE storage.

Do not transfer to Flash with %EESAV or !EES at high frequency as the number of write cycles to Flash are limited to around 10000.

Avoid transferring to Flash while the product is performing critical operation.

Write to address locations 1 and up. Writing at address 0 will fill all RAM location with the value.

FLCL – Automatic Fault Clearance

HexCode: FE CANOpen Id: 0x30FE

Description:

This parameter controls the automatic clearance of the following faults: Overvoltage, Undervoltage and Overtemperature. If disabled, the triggered faults will not be cleared if the fault condition is restored unless a respective command comes (see MG runtime command). So, when e.g. battery volts go above overvoltage limit and overvoltage fault gets triggered, it will not be cleared automatically when the battery volts go below the overvoltage minus the overvoltage hysteresis.

Syntax Serial: ^FLCL nn

~FLCL

Syntax Scripting: setconfig(_FLCL, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 0 Max: 1 Default: 1

Where:

nn =

0: Disabled

1: Enabled

Example:

^FLCL 0: Faults will be cleared only when the MG runtime command is sent.

ISM - Raw Redirect Mode

HexCode: E9 CANOpen id: 0x30E9

Description:

Configure this parameter in order to enable Raw Redirect Mode and the interface to be used.

Syntax Serial: ^ISM nn

~ISM

Syntax Scripting: setconfig(_ISM, nn)

Number of Arguments: 1

Argument 1: Raw Redirect Mode

Type: Unsigned 8-bit

Min: 0 Max: 2

Default: 0

Where:

nn =

0: Off

2: RS232

4: RS485

Example:

^ISM 1: Enable ModbusRaw Redirect mode on RS232 interface.

MDAL – Modbus Data Alignment

HexCode: CA CANOpen id: 0x30CA

Description:

Configure this parameter in order to set the alignment of the Modbus byte frame. This option depends on what the Modbus master supports.

Syntax Serial: ^MDAL nn

~MDAL

Syntax Scripting: setconfig(_MDAL, nn)

Number of Arguments: 1

Argument 1: Modbus Data Alignment

Type: Unsigned 8-bit

Min: 0 Max: 3

Default: 0

Where:
nn =
0: High Word/High Byte
1: High Word/Low Byte
2: Low Word/High Byte
3: Low Word/Low Byte

Example:

^MDAL 2: Configure Modbus Data Alignment to Low Word/High Byte.

MNOD – Modbus Slave ID

HexCode: A2 CANOpen id: 0x30A2

Description:

Configure this parameter in order to set Modbus Slave Node ID of the controller. In that way this controller wil be distinguished inside a Modbus network.

Syntax Serial: ^MNOD nn

~MNOD

Syntax Scripting: setconfig(_MNOD, nn)

Number of Arguments: 1

Argument 1: Modbus Node ID

Type: Unsigned 8-bit

Min: 0 Max: 127

Default: 1

Where:
nn = Node ID

Example:

^MNOD 3: Configure Modbus Node ID to 3.

PMS - Pulse keep within Min & Max Safety

HexCode: F4 CANOpen id: 0x30F4

Description:

This configuration is used to make sure that the pulse input command is always within a user preset minimum and maximum safe value. It is useful to detect, for example, that the wire connection is broken. If the safety is enabled and the input is outside the safe range, the Pulse input command will be considered invalid.

The controller will then apply a motor command based on the priority logic.

Syntax Serial: ^PMS nn

~PMS

Syntax Scripting: setconfig(_PMS, nn)

Number of Arguments: 1

Argument 1: Enable

Type: Unsigned 8-bit

Min: 0 Max: 1

Default: 1 = Enabled

Where:

nn =

0: Disabled

1: Enabled

RSBR - Set RS232/RS485 baudrate

HexCode: 0A CANOpen id: 0x300A

Description:

Sets the serial communication bit rate of the RS232 and RS485 ports. Choices are one of five most common bit rates. On selected products, the port output can be inverted to allow a simplified connection to devices that have TTL serial ports instead of full RS232 (not applicable for RS485 port).

Syntax Serial: ^RSBR nn
~RSBR

Syntax Scripting: setconfig(_RSBR, nn)

Number of Arguments: 1

Argument 1: BitRate
Type: Unsigned 8-bit
Min: 0 Max: 4 or 9
Default: 0 = 115200

Where:
nn =
0: 115200
1: 57600
2: 38400
3:19200
4: 9600
5: 115200 + Inverted RS232
6: 57600 + Inverted RS232
7: 38400 + Inverted RS232
8: 19200 + Inverted RS232
9: 9600 + Inverted RS232
10: 230600

Example:

^RSBR 3 : sets baud rate at 19200

Note:

This configuration can only be changed while connected via USB or via scripting. After the baud rate has been changed, it will not be possible to communicate with the Roborun PC utility using the serial port until the rate is changed back to 115200. Slow bit rates may result in data loss if more characters are sent than can be handled. The inverted mode is only available on selected products.

RS485 - Enable RS485

HexCode: D0 CANOpen id: 0x30D0

Description:

Configure this parameter in order to enable the RS485 communication. This feature is applicable only on motor controllers, where RS485 pins are shared with other features. In these controllers the default value is Disabled. In any other controller that support RS485 with dedicated pins the default value is Enabled.

Syntax Serial: ^RS485 nn

~ RS485

Syntax Scripting: setconfig(_RS485, nn)

Number of Arguments: 1

Argument 1: Enable RS485

Type: Unsigned 8-bit

Min: 0 Max: 1

Default: 0

Where:
nn = Enable RS485
0: Disabled.
1: Enabled.

Example:

^RS485 1: Enable RS485.

RWD - Serial Data Watchdog

HexCode: 08 CANOpen id: 0x3008

Description:

This is the Serial Commands watchdog timeout parameter. It is used to detect when the controller is no longer receiving commands and switch to the next priority level. Any Real-time Command arriving from RS232, RS485, TCP, USB, CAN or Microbasic Scripting, The watchdog value is a number in ms (1000 = 1s). The watchdog function can be disabled by setting this value to 0. The watchdog will only detect the loss of real time commands, as shown in section 6. All other traffic on the serial port will not refresh the watchdog timer. As soon as a valid command is received, motor operation will resume.

Syntax Serial: ^RWD nn
~RWD

Syntax Scripting: setconfig(_RWD, nn)

Number of Arguments: 1

Argument 1: Timeout

	Type: Unsigned 16-bit	
	Min: 0	Max: 65000
	Default: 1000 = 1s	

Where:

nn = Timeout value in ms

Example:

^RWD 2000 : Set watchdog to 2s
^RWD 0 : Disable watchdog

SCRO - Select Print output port for scripting

HexCode: 5E CANOpen id: 0x305E

Description:

Selects which port the print statement sends data to. When 0, the last port which received a valid character will be the one the script outputs to.

Syntax Serial: ^SCRO nn
~SCRO

Syntax Scripting: setconfig(_SCRO, nn)

Number of Arguments: 1

Argument 1: Port

	Type: Unsigned 8-bit	
	Min: 0	Max: 4
	Default: 0 = Last used	

Where:

nn =
0: Last used
1: Serial
2: USB
3: RS485 if applicable,
4: TCP if applicable

STO – STO Enable

HexCode: CF CANOpen id: 0x30CF

Description:

Configure this parameter in order to enable the STO functionality. For boards which have the respective circuit the respective jumper needs to be removed (see Chapter **Safe Torque-Off (STO)**, in Section2).

Syntax Serial: ^STO nn

~STO

Syntax Scripting: setconfig(_STO, nn)

Number of Arguments: 1

Argument 1: STO Status

Type: Unsigned 8-bit

Min: 0 Max: 1

Default: 0

Where:

nn = STO status

0: Disabled.
1: Enabled.

Example:

^STO 1: Enable STO functionality.

TELS - Telemetry String

HexCode: 47 CANOpen id: 0x3047

Description:

This parameter command lets you enter the telemetry string that will be used when the controller starts up. The string is entered as a series of queries characters between a beginning and an ending quote. Queries must be separated by ":" colon characters. Upon the power up, the controller will load the query history buffer and it will automatically start executing commands and queries based on the information in this string. Strings up to 48 characters long can be stored in this parameter.

Syntax Serial: ^TELS "string"
~TELS

Syntax Scripting:

Number of Arguments: 1

Argument 1: Telemetry
Type: String
Min: "" Max: 48 characters string
Default: "" = Empty string

Where:

string = string of ASCII characters between quotes

Example:

^TELS "?A:?V:?T:# 200" = Controller will issue Amps, Volts and temperature information automatically upon power up at 200ms intervals.

Analog, Digital, Pulse IO Configurations

These parameters configure the operating mode and how the inputs and outputs work.

TABLE 15-37. Input/Output Configurations

Command	Arguments	Description
ACTR	InputNbr Center	Analog Input Center (0)
ADB	InputNbr Deadband	Analog Input Deadband
AINA	InputNbr Use	Analog Input Use
ALIN	InputNbr Linearity	Analog Input Linearity
AMAX	InputNbr Max	Analog Input Max
AMAXA	InputNbr Action	Analog Input Action at Max
AMIN	InputNbr Min	Analog Input Min
AMINA	InputNbr Action	Analog Input Action at Min
AMOD	InputNbr Mode	Analog Conversion Type
APOL	InputNbr Polarity	Analog Input Conversion Polarity
AUXV	None	Digital Output High Side Drive Voltage Level
DINA	InputNbr Action	Digital Input Action
DINL	ActiveLevels	Digital Input Active Level
DOA	OutputNbr Action	Digital Output Action
DOL	ActiveLevels	Digital Outputs Active Level
DOT	OutputNbr Action	Digital Output Type
ENCO	None	Encoder Output Enable
PCTR	InputNbr Center	Pulse Input Center
PDB	InputNbr Deadband	Pulse Input Deadband
PINA	InputNbr Use	Pulse Input Use
PLIN	InputNbr Linearity	Pulse Input Linearity
PMAX	InputNbr Max	Pulse Input Max
PMAXA	InputNbr Action	Pulse Input Action at Max
PMIN	InputNbr Min	Pulse Input Min
PMINA	InputNbr Action	Pulse Input Action at Min
PMOD	InputNbr Mode	Pulse Input Capture Type
PPOL	InputNbr Polarity	Pulse Input Capture Polarity

ACTR - Analog Input Center (0)

HexCode: 16 CANOpen id: 0x3016

Description:

This parameter is the measured voltage on input that will be considered as the center or the 0 value. The min, max and center are useful to set the range of a joystick or of a feedback sensor. Internally to the controller, commands and feedback values are converted to 1000, 0, +1000.

Syntax Serial: ^ACTR cc nn
~ACTR [cc]

Syntax Scripting: setconfig(_ACTR, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr	Min: 1	Max: Total Number of Analog Inputs
Argument 2: Center	Type: Unsigned 16-bit	
	Min: 0	Max: 10000
	Default: 2500 mV	

Where:

cc = Analog input channel

nn = 0 to 10000mV

Example:

^ACTR 3 2000 : Set Analog Input 3 Center to 2000mV

Note:

Center value must always be a number greater of equal to Min, and smaller or equal to Max
Make the center value the same as the min value in order to produce a converted output range that is positive only (0 to +1000)

ADB - Analog Input Deadband

HexCode: 17 CANOpen id: 0x3017

Description:

This parameter selects the range of movement change near the center that should be considered as a 0 command. This value is a percentage from 0 to 50% and is useful to allow some movement of a joystick around its center position without change at the converted output.

Syntax Serial: ^ADB cc nn
~ADB [cc]

Syntax Scripting: setconfig(_ADB, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr	Min: 1	Max: Total Number of Analog Inputs
Argument 2: Deadband	Type: Unsigned 8-bit	
	Min: 0	Max: 50
	Default: 5 = 5%	

Where:

cc = Analog input channel
nn = Deadband in %

Example:

^ADB 6 10 : Sets Deadband for channel 6 at 10%

Note:

Deadband is not used when input is used as feedback

AINA - Analog Input Use

HexCode: 19 CANOpen id: 0x3019

Description:

This parameter selects whether an input should be used as a command feedback or left unused. When selecting command or feedback, it is also possible to select which channel this command or feedback should act on. Feedback can be position feedback if potentiometer is used or speed feedback if tachometer is used. Motor Temperature reads the thermistor and calculates the temperature according to the Motor Thermistor parameters (R25 and B25). Embedded in the parameter is the motor channel to which the command or feedback should apply.

Syntax Serial: ^AINA cc (nn + mm)
~AINA [cc]

Syntax Scripting: setconfig(_AINA, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr		
	Min: 1	Max: Total Number of Analog Inputs
Argument 2: Use		
	Type: Unsigned 8-bit	
	Min: 0	Max: 255
	Default: 0 = No action	

Where:

cc = Analog input channel
nn =
0: No action
1: Command
2: Feedback
4: Motor Temperature

mm =
mot1*16 + mot2*32 + mot3*64

Example:

^AINA 1 17: Sets Analog channel 1 as command for motor 1. I.e. 17 = 1 (command) +16 (motor 1)

ALIN - Analog Input Linearity

HexCode: 18 CANOpen id: 0x3018

Description:

This parameter is used for applying an exponential or a logarithmic transformation on an analog input. There are 3 exponential and 3 logarithmic choices. Exponential correction will make the commands change less at the beginning and become stronger at the end of the joystick movement. The logarithmic correction will have a stronger effect near the start and lesser effect near the end. The linear selection causes no change to the input.

Syntax Serial: ^ALIN cc nn
~ALIN [cc]

Syntax Scripting: setconfig(_ALIN, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Analog Inputs

Argument 2: Linearity

Type: Unsigned 8-bit
Min: 0 Max: 6
Default: 0 = Linear

Where:

cc = Analog input channel
nn =
0: Linear (no change)
1: Exp weak
2: Exp medium
3: Exp strong
4: Log weak
5: Log medium
6: Log strong

Example:

^ALIN 1 1 : Sets linearity for channel 1 to exp weak

AMAX - Analog Input Max

HexCode: 15 CANOpen id: 0x3015

Description:

This parameter sets the voltage that will be considered as the maximum command value. The min, max and center are useful to set the range of a joystick or of a feedback sensor. Internally to the controller, commands and feedback values are converted to -1000, 0, +1000.

Syntax Serial: ^AMAX cc nn
~AMAX [cc]

Syntax Scripting: setconfig(_AMAX, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr		
	Min: 1	Max: Total Number of Analog Inputs
Argument 2: Max		
	Type: Unsigned 16-bit	
	Min: 0	Max: 10000
	Default: 4900 mV	

Where:

cc = Analog input channel
nn = 0 to 10000mV

Example:

^AMAX 4 4500 : Set Analog Input 4 Max range to 4500mV

Note:

Analog input can capture voltage up to around 5.2V. Setting the Analog maximum above 5200 mV, means the conversion will never be able to reach +1000

AMAXA - Analog Input Action at Max

HexCode: 1B CANOpen id: 0x301B

Description:

This parameter selects what action should be taken if the maximum value that is defined in AMAX is reached. The list of action is the same as these of digital inputs. For example, this feature can be used to create soft limit switches, in which case the motor can be made to stop if the feedback sensor in a position mode has reached a maximum value.

Syntax Serial: ^AMAXA cc (aa + mm)
~AMAXA [cc]

Syntax Scripting: setconfig(_AMAXA, cc, aa)

Number of Arguments: 2

Argument 1: InputNbr		
	Min: 1	Max: Total Number of Analog Inputs
Argument 2: Action		
	Type: Unsigned 8-bit	
	Min: 0	Max: 255
	Default: 0 = No action	

Where:

cc = Analog input channel
aa =
0: No action
1: Quick stop

2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value
mm = mot1*16 + mot2*32 + mot3*64

Example:

^AMAXA 3 33 : Stops motor 2. I.e. 33 = 1 (Quick stop) + 32 (motor2)

AMIN - Analog Input Min

HexCode: 14 CANOpen id: 0x3014

Description:

This parameter sets the raw value on the input that will be considered as the minimum command value. The min, max and center are useful to set the range of a joystick or of a feedback sensor. Internally to the controller, commands and feedback values are converted to -1000, 0, +1000.

Syntax Serial: ^AMIN cc nn
~AMIN [cc]

Syntax Scripting: setconfig(_AMIN, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Analog Inputs

Argument 2: Min

Type: Unsigned 16-bit
Min: 0 Max: 10000
Default: 100 mV

Where:

cc = Analog input channel
nn = 0 to 10000mV

Example:

^AMIN 5 250 : Set Analog Input 5 Min to 250mV

Note:

Analog input can capture voltage up to around 5.2V. Setting the Analog minimum between 5200 and 10000 mV means the conversion will always return 0

AMINA - Analog Input Action at Min

HexCode: 1A CANOpen id: 0x301A

Description:

This parameter selects what action should be taken if the minimum value that is defined in AMIN is reached. The list of action is the same as these of the DINA configuration command. For example, this feature can be used to create soft limit switches, in which case the motor can be made to stop if the feedback sensor in a position mode has reached a minimum value.

Syntax Serial: ^AMINA cc (aa + mm)
~AMINA [cc]

Syntax Scripting: setconfig(_AMINA, cc, aa)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Analog Inputs

Argument 2: Action

Type: Unsigned 8-bit
Min: 0 Max: 255
Default: 0 = No action

Where:

cc = Analog input channel
aa =
0: No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value
mm = mot1*16 + mot2*32 + mot3*64

Example:

^AMINA 2 33 : Stops motor 2. I.e. 33 = 1 (Quick stop) + 32 (motor2)

AMOD - Analog Conversion Type

HexCode: 13 CANOpen id: 0x3013

Description:

This parameter is used to enable/disable an analog input pin. When enabled, it can be made to measure an absolute voltage from 0 to 5V, or a relative voltage that takes the 5V output on the connector as the 5V reference. The absolute mode is preferred whenever measuring a voltage generated by an outside device or sensor. The relative mode is the mode to use when a sensor or a potentiometer is powered using the controller's 5V output of the controller. Using the relative mode gives a correct sensor reading even though the 5V output is imprecise.

Syntax Serial: ^AMOD cc nn
~AMOD [cc]

Syntax Scripting: setconfig(_AMOD, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Analog Inputs

Argument 2: Mode
Type: Unsigned 8-bit
Min: 0 Max: 2
Default: 0 = Disabled

Where:
cc = Analog input channel
nn =
0: Disabled
1: Absolute
2: Relative

Example:

^AMOD 1 1 : Analog input 1 enabled in absolute mode

APOL - Analog Input Conversion Polarity

HexCode: 1C CANOpen id: 0x301C

Description:

Inverts the analog capture polarity value after conversion. When this configuration bit is cleared, the pulse capture is converted into a -1000 to +1000 command or feedback value. When set, the converted range is inverted to +1000 to -1000.

Syntax Serial: ^APOL cc nn
~APOL [cc]

Syntax Scripting: setconfig(_APOL, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr
Min: 1 Max: Total Number of Analog Inputs

Argument 2: Polarity
Type: Unsigned 8-bit
Min: 0 Max: 1
Default: 0 = Non inverted

Where:

cc = Analog input channel

nn =
0: Not inverted
1: Inverted

AUXV - Digital Output High Side Drive Voltage Level

HexCode: 10E CANOpen id: 0x310E

Description:

This parameter is used in order to select the voltage level of the High Side Digital Outputs. This feature is only available on selected models, see product datasheet.

Syntax Serial: ^AUXV nn
~AUXV

Syntax Scripting: setconfig(_AUXV, nn)

Number of Arguments: 1

Argument 1: Voltage Level
Type: Unsigned 8-bit
Min: 0 Max: 1

Where:

nn =

0: 5 Volts

1: 24 Volts

Example:

^AUXV 1 : Set the voltage level of the high side digital outputs to 24 Volts.

DINA - Digital Input Action

HexCode: 0F CANOpen id: 0x300F

Description:

This parameter sets the action that is triggered when a given input pin is activated. The action list includes: limit switch for a selectable motor and direction, use as a deadman switch, emergency stop, Quick stop or invert direction. Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^DINA cc (aa + [mm])
~DINA [cc]

Syntax Scripting: setconfig(_DINA, cc, aa)

Number of Arguments: 2

Argument 1: InputNbr

Min: 0 Max: Total Number of Digital Inputs

Argument 2: Action

Type: Unsigned 8-bit
Min: 0 Max: 255
Default: 0 = No actions

Where:

cc = Input channel number

aa =
0: No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value
9: Soft STO

mm = mot1*16 + mot2*32 + mot3*64

Example:

^DINA 1 33 : Input 1 as Quick stop for Motor 2. I.e. 33 = 1 (Quick stop) + 32 (Motor2)

DINL - Digital Input Active Level

HexCode: 10 CANOpen id: 0x3010

Description:

This parameter is used to set the active level for each Digital input. An input can be made to be active high or active low. Active high means that pulling it to a voltage will trigger an action. Active low means pulling it to ground will trigger an action. This parameter is a single number for all inputs.

Syntax Serial: ^DINL cc aa
~DINL [cc]

Syntax Scripting: setconfig(_DINL, cc, aa)

Number of Arguments: 1

Argument 1: ActiveLevels

Type: Unsigned 32-bit

Min: 0 Max: 2 ^ Total Number of Digital Inputs

Default: 0 = All Active high

Where:
cc = Digital input number

aa=
0: Active High
1: Active Low

Example:

^DINL 2 1 : Sets digital input 2 to active low

DOA - Digital Output Action

HexCode: 11 CANOpen id: 0x3011

Description:

This configuration parameter will set what will trigger a given output pin. The parameter is a number in a list of possible triggers: when one or several motors are on, when one or several motors are reversed, when an Overvoltage condition is detected or when an Over-temperature condition is detected. Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^DOA nn (a + cc)
~DOA [nn]

Syntax Scripting: setconfig(_DOA, nn, (a+cc))

Number of Arguments: 2

Argument 1: OutputNbr
Type: Unsigned 32-bit
Min: 1 Max: Total Number of Digital Outputs

Argument 2: Action
Min: 0
Default: See Note

Where:

cc =

16: channel 1

32: channel 2

64: channel 3

aa =
0: Never
1: Motor on
2: Motor reversed
3: Overvoltage
4: Overtemperature
5: Mirror status LED
6: No MOSFET failure

Example:

^DOA 1 19: Output 1 is active when Overvoltage is observed. on channel 1
^DOA 2 33: Output 2 is active when motor of channel 2 is on.

Note:

Typical default configuration is Digital outputs 1 (2) are active when motor is on. Digital output 2 (3) when no MOSFET failure is detected.
To activate an output via serial command or from a Microbasic script, set that output to Never

DOL - Digital Outputs Active Level

HexCode: 12 CANOpen id: 0x3012

Description:

This parameter configures whether an output should be set to ON or to OFF when it is activated.

Syntax Serial: ^DOL cc aa
~DOL

Syntax Scripting: setconfig(_DOL, cc, aa)

Number of Arguments: 2

Argument 1: OutputNbr

Type: Unsigned 32-bit

Min: 1 Max: Total Number of Digital Outputs

Argument 2: ActiveLevels
Type: Unsigned 32-bit

Min: 0 Max: 2 ^ Total Number of Digital Outputs

Default: 0 = All active high

Where:

cc = Digital input number
aa=
0: On when active
1: Off when active

DOT - Digital Output Type

HexCode: 10D CANOpen id: 0x310D

Description:

This parameter is used in order to select the type of the digital outputs. This feature is only available on selected models that support high side type, see product datasheet. For the other models all the digital outputs are open drain.

Syntax Serial: ^DOT cc aa
~DOT

Syntax Scripting: setconfig(_DOT, cc, aa)

Number of Arguments: 2

Argument 1: OutputNbr

Type: Unsigned 32-bit

Min: 1 Max: Total Number of Digital Outputs

Argument 2: Type

Type: Unsigned 8-bit

Min: 0 Max: 1

Where:

nn =

0: Open Drain

1: High Side Driver

Example:

^DOT 2 1: Set the type of digital output 2 to High Side Driver.

ENCO - Encoder Output Enable

HexCode: 10F CANOpen id: 0x310F

Description:

This parameter is used in enable the encoder output of the motor sensor. This feature is only available on selected models, see product datasheet.

Syntax Serial: ^ENCO nn

~ENCO

Syntax Scripting: setconfig(_ENCO, nn)

Number of Arguments: 1

Argument 1:

Type: Unsigned 8-bit

Min: 0 Max: 1

Where:

nn =

0: Encoder Output Disabled

1: Encoder Output Enabled

Example:

^ENCO 1 : Enable the encoder output.

PCTR - Pulse Input Center

HexCode: 20 CANOpen id: 0x3020

Description:

This defines the raw value of the measured pulse that would be considered as the 0 value inside the controller. The default value is 1500 which is the center position of the pulse in the RC radio mode.

Syntax Serial: ^PCTR cc nn
~PCTR [cc]

Syntax Scripting: setconfig(_PCTR, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr
Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Center
Type: Unsigned 16-bit
Min: 0 Max: 65536
Default: 1500us

Where:

cc = Pulse input number
nn = 0 to 65536us

PDB - Pulse Input Deadband

HexCode: 21 CANOpen id: 0x3021

Description:

This sets the deadband value for the pulse capture. It is defined as the percent number from 0 to 50% and defines the amount of movement from joystick or sensor around the center position before its converted value begins to change.

Syntax Serial: ^PDB cc nn
~PDB [cc]

Syntax Scripting: setconfig(_PDB, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Deadband

Type: Unsigned 8-bit
Min: 0 Max: 50
Default: 5 = 5%

Where:
cc = Pulse input number
nn = Deadband in %

Note:

Deadband is not used when input is used as feedback

PINA - Pulse Input Use

HexCode: 23 CANOpen id: 0x3023

Description:

This parameter selects whether an input should be used as a command feedback, position feedback or left unused. Embedded in the parameter is the motor channel that this command or feedback should act on. Feedback can be position feedback if potentiometer is used or speed feedback if tachometer is used.

Syntax Serial: ^PINA cc (nn + mm)
~PINA [cc]

Syntax Scripting: setconfig(_PINA, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Use

Type: Unsigned 8-bit
Min: 0 Max: 255
Default: See note

Where:

cc = Pulse input number
nn =
0: No action
1: Command

2: Feedback
mm =
mot1*16 + mot2*32 + mot3*64

Example:

^AINA 1 17: Sets Pulse input 1 as command for motor 1. I.e. 17 = 1 (command) +16 (motor 1)

PLIN - Pulse Input Linearity

HexCode: 22 CANOpen id: 0x3022

Description:

This parameter is used for applying an exponential or a logarithmic transformation on a pulse input. There are 3 exponential and 3 logarithmic choices. Exponential correction will make the commands change less at the beginning and become stronger at the end of the joystick movement. The logarithmic correction will have a stronger effect near the start and lesser effect near the end. The linear selection causes no change to the input.

Syntax Serial: ^PLIN cc nn
~PLIN [cc]

Syntax Scripting: setconfig(_PLIN, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr
Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Linearity
Type: Unsigned 8-bit
Min: 0 Max: 6
Default: 0 = Linear

Where:

cc = Pulse input number
nn =
0: Linear (no change)
1: Exp weak
2: Exp medium
3: Exp strong
4: Log weak
5: Log medium
6: Log strong

PMAX - Pulse Input Max

HexCode: 1F CANOpen id: 0x301F

Description:

This parameter defines the raw pulse measurement number that would be considered as the +1000 internal value to the controller. By default, it is set to 2000 which is the max pulse width of an RC radio pulse.

Syntax Serial: ^PMAX cc nn
~PMAX [cc]

Syntax Scripting: setconfig(_PMAX, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Max

Type: Unsigned 16-bit
Min: 0 Max: 65536
Default: 2000

Where:

cc = Pulse input number
nn = 0 to 65536us

PMAXA - Pulse Input Action at Max

HexCode: 25 CANOpen id: 0x3025

Description:

This parameter configures the action to take when the max value that is defined in PMAX is reached. The list of action is the same as in the DINA digital input action list. Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^PMAXA cc (aa + mm)
~PMAXA [cc]

Syntax Scripting: setconfig(_PMAXA, cc, aa)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Action

Type: Unsigned 8-bit
Min: 0 Max: 255
Default: 0 = No action

Where:

cc = Pulse input number
aa =
0: No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value

mm = mot1*16 + mot2*32 + mot3*64

PMIN - Pulse Input Min

HexCode: 1E CANOpen id: 0x301E

Description:

This sets the raw value of the pulse capture that would be considered as the -1000 internal value to the controller. The value is in number of microseconds (1000 = 1ms). The default value is 1000 microseconds which is the typical minimum value on an RC radio pulse.

Syntax Serial: ^PMIN cc nn
~PMIN [cc]

Syntax Scripting: setconfig(_PMIN, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Min

Type: Unsigned 16-bit
Min: 0 Max: 65536
Default: 1000

Where:

cc = Pulse input number
nn = 0 to 65536us

PMINA - Pulse Input Action at Min

HexCode: 24 CANOpen id: 0x3024

Description:

This parameter selects what action should be taken if the minimum value that is defined in PMIN is reached. The list of action is the same as these of the DINA digital input actions. Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^PMINA cc (aa + mm)
~PMINA [cc]

Syntax Scripting: setconfig(_PMINA, cc, aa)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Action

Type: Unsigned 8-bit
Min: 0 Max: 255
Default: 0 = No action

Where:

cc = Pulse input number
aa =
0: No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value

mm = mot1*16 + mot2*32 + mot3*64

PMOD - Pulse Input Capture Type

HexCode: 1D CANOpen id: 0x301D

Description:

This parameter is used to enable/disable the pulse input and select its operating mode, which can be: pulse with measurement, frequency or duty cycle. Inputs can be measured with a high precision over a large range of time or frequency. An input will be processed and converted to a command or a feedback value in the range of -1000 to +1000 for use by the controller internally.

Syntax Serial: ^PMOD cc nn
~PMOD [cc]

Syntax Scripting: setconfig(_PMOD, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Mode

Type: Unsigned 8-bit
Min: 0 Max: 4
Default: See note

Where:
cc = Pulse input number
nn =
0: Disabled
1: Pulse width
2: Frequency
3: Duty cycle
4: Magsensor

5: BMS
6: Pulse Count
7: Flow Sensor

Example:

^PMOD 4 4 : Sets Pulse input 4 in Multi-PWM for Robteq's MGS1600 magnetic guide sensor

Note:

Pulse width is designed for capturing RC radio commands. Pulse width must be beween 500us and 3000us, and repeat rate 50Hz or higher
On some products, enabling a pulse input will cause a an offset voltage to be present when that same input is read as analog

PPOL - Pulse Input Capture Polarity

HexCode: 26 CANOpen id: 0x3026

Description:

Inverts the pulse capture value after conversion. When this configuration bit is cleared, the pulse capture is converted into a -1000 to +1000 command or feedback value. When set, the converted range is inverted to +1000 to -1000. Center value must always be a number greater of equal to Min, and smaller or equal to Max. Make the center value the same as the min value in order to produce a converted output range that is positive only (0 to +1000)

Syntax Serial: ^PPOL cc nn
~PPOL

Syntax Scripting: setconfig(_PPOL, cc, nn)

Number of Arguments: 2

Argument 1: InputNbr

Min: 1 Max: Total Number of Pulse Inputs

Argument 2: Polarity

Type: Unsigned 8-bit
Min: 0 Max: 1
Default: 0 = Non inverted

Where:

cc = Pulse input number
nn =
0: Not inverted
1: Inverted

Motor Configurations

This section covers the various configuration parameter applying to motor operations.

TABLE 15-38. Motor Configurations

Command	Arguments	Description
ALIM	Channel Limit	Amps Limit
ATGA	Channel Action	Amps Trigger Action
ATGD	Channel Delay	Amps Trigger Delay
ATRIG	Channel Level	Amps Trigger Level
B25	Channel Coefficient	Thermistor Temperature Coefficient β_{25}.
BKD	Delay	Brake Delay
BPR	Channel Value	Bypass Trajectory/Ramp
BRV	Channel Voltage	Brake Release Voltage
BHV	Channel Voltage	Brake Hold Voltage
BDT	Channel Delay	Brake Delay Time
BLFB	Channel Sensor	Closed Loop Feedback Sensor
BLSTD	Channel Mode	Stall Detection
BR	Channel Value	Mechanical System Rotating Friction Coefficient
CLERD	Channel Mode	Close Loop Error Detection
EDEC	Channel Deceleration	Motor Fault Deceleration Rate
EHL	Channel Value	Encoder Max Limit
EHLA	Channel Action	Encoder Action at Max
EHOME	Channel Value	Encoder Home Count
ELL	Channel Value	Encoder Min Limit
ELLA	Channel Action	Encoder Action at Min
EMOD	Channel Use	Encoder Usage
EPPR	Channel Value	Encoder Pulse/Rev Value
FET	Channel Mode	Loop Error Time
FEW	Channel Mode	Loop Error Limit
ICAP	Channel Cap	PID Integrator Limit
JR	Channel Value	Mechanical System Inertia
KDG	PID Channel Gain	PID Derivative Gain
KIG	PID Channel Gain	PID Integral Gain
KPG	PID Channel Gain	PID Proportional Gain
LPFB	Channel Value	Speed feedback low pass filter bandwidth
MAC	Channel Acceleration	Motor Acceleration Rate
MCLE	Channel Value	SSI Multi-turn Counter number of bits
MDEC	Channel Deceleration	Motor Deceleration Rate
MLX	InputNbr Value	Molex Input
MMOD	Channel Mode	Operating Mode
MNRPM	Channel RPM	Min Speed RPM Value

Command	Arguments	Description
MSTA	Channel Value	SSI Multi-turn Counter start bit position
MVEL	Channel Velocity	Position Mode Velocity
MXMD	Mode	Mixed Mode
MXPF	Channel MaxPower	Motor Max Power Forward
MXPR	Channel MaxPower	Motor Max Power Reverse
MXRPM	Channel RPM	Max Speed RPM
MXTRN	Channel Turns	Position Turns Min to Max
NOMA	Channel Current	Nomiinal Current
OVH	Voltage	Overvoltage hysteresis
OVL	Voltage	Overvoltage Limit
OTL	Temperature	Over Temperature Limit
R25	Channel Resistance	Thermistor Resistance at 25^{o}C
SCLE	Channel Value	SSI Counter number of bits
SCLK	Level	SSI Clock Frequency
SED	Channel Value	Sensor Error Detection
SFTS	Channel Mode	Safety Switch Connected
SHL	Channel Value	SSI Sensor Max Limit
SHLA	Channel Action	SSI Sensor Action at Max
SHOME	Channel Value	SSI Sensor Home Count
SLEN	Channel Value	SSI frame number of bits
SLL	Channel Value	SSI Sensor Min Limit
SLLA	Channel Action	SSI Sensor Action at Min
SMOD	Channel Use	SSI Sensor Usage
SSTA	Channel Value	SSI Counter start bit position
THLD	Level	Short Circuit Detection Sensitivity
TNM	Channel Constant	Motor Torque Constant
TPAL	Channel Time	Time for Amps Limit
UVL	Voltage	Undervoltage Limit

ALIM - Amps Limit

HexCode: 2A CANOpen id: 0x302A

Description:

This is the maximum Amps that the controller will be allowed to deliver to a motor regardless the load of that motor. The value is entered in Amps multiplied by 10. The value is the Amps that are measured at the motor and not the Amps measured from a battery. When the motor draws current that is above that limit, the controller will automatically reduce the output power until the current drops below that limit. For brushless controllers this value is considered to be in RMS. This value is also used for the calculation of the I2T limit.

Syntax Serial: ^ALIM cc nn
~ALIM [cc]

Syntax Scripting: setconfig(_ALIM, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Motors

Argument 2: Limit

	Type: Unsigned 16-bit	
	Min: 10	Max: Max Amps in datasheet
	Default: See note	

Where:

cc = Motor channel
nn = Amps *10

Example:

^ALIM1 455: Set Amp limit for Motor 1 to 45.5A

Note:

Default value is typically set to the controller's max amps as defined in the datasheet

ATGA - Amps Trigger Action

HexCode: 2C CANOpen id: 0x302C

Description:

This parameter sets what action to take when the Amps trigger is activated. The list is the same as in the DINA digital input actions. Typical use for that feature is as a limit switch when, for example, a motor reaches an end and enters stall condition, the current will rise, and that current increase can be detected and the motor be made to stop until the direction is reversed. Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^ATGA cc (aa + mm)
~ATGA [cc]

Syntax Scripting: setconfig(_ATGA, cc, aa + mm)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Motors

Argument 2: Action

	Type: Unsigned 8-bit	
	Min: 10	Max: 255

Default: 0 = No action

Where:

cc = Motor channel
aa =
0 : No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value

mm = mot1*16 + mot2*32 + mot3*64

ATGD - Amps Trigger Delay

HexCode: 2D CANOpen id: 0x302D

Description:

This parameter contains the time in milliseconds during which the Amps Trigger Level (ATRIG) must be exceeded before the Amps Trigger Action (ATGA) is called. This parameter is used to prevent Amps Trigger Actions to be taken in case of short duration spikes.

Syntax Serial: ^ATGD cc nn
~ATGD [cc]

Syntax Scripting: setconfig(_ATGD, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Delay
Type: Unsigned 16-bit
Min: 0 Max: 10000
Default: 500ms

Where:

cc = Motor channel
nn = Delay in ms

Example:

^ATGD 1 1000: Action will be triggered if motor Amps exceeds the value set with ATGL for more than 1000ms

ATRIG - Amps Trigger Level

HexCode: 2B CANOpen id: 0x302B

Description:

This parameter lets you select Amps threshold value that will trigger an action. This threshold must be set to be below the ALIM Amps limit. When that threshold is reached, then list of action can be selected using the ATGA parameter.

Syntax Serial: ^ATRIG cc nn
~ATRIG [cc]

Syntax Scripting: setconfig(_ATRIG, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Level
Type: Unsigned 16-bit
Min: 10 Max: Max Amps in datasheet
Default: Max Amps rating in datasheet

Where:

cc = Motor channel
nn = Amps *10

Example:

^ATRIG2 550: Set Amps Trigger to 55.0A

B25 - Thermistor Temperature Coefficient β25

HexCode: 106 CANOpen id: 0x3106

Description:

Set the temperature coefficient of the thermistor attached to the motor. This value can be derived from the thermistor datasheet either directly or after making some calculations as stated in chapter Connecting External Thermistor to Analog Inputs, SECTION 3.

Syntax Serial: ^B25 cc nn
~B25 cc

Syntax scripting: setconfig(_B25,cc,nn)

Argument 1: Channel Type: Unsigned 8-bit
Min: 1 Max: Total Number of Motors

Argument 2: Coefficient Type: Signed 32-bit
Default: 0

Where:

cc=Motor channel

nn=Coefficient in Kelvin (K)

Example:

^B25 1 3100. Set β25 at 3100.

BKD - Brake Delay

HexCode: 01 CANOpen id: 0x3001

Description:

Set the delay in milliseconds from the time a motor stops and the time an output connected to a brake solenoid will be released. Applies to any Digital Ouput(s) that is configured as motor is on. Delay value applies to all motors in multi-channel products.

Syntax Serial: ^BKD nn
~BKD

Syntax Scripting: setconfig(_BKD, nn)

Number of Arguments: 1

Argument 1: Delay

Type: Unsigned 16-bit	
Min: 0	Max: 65536
Default: 250 = 250ms	

Where:

nn = Delay in milliseconds

Example:

^BKD 1 1000 : Causes the digital output to go off, and therefore activate the brake, 1.0s after motor stops being energized

BPR - Bypass Trajectory/Ramp

HexCode: F3 CANOpen id: 0x30F3

Description:

This parameter is configured in order to be able to bypass the ramp of the command or the trajectory in case of position mode. It is applicable mainly in torque mode (in case it requires high transitions) or in DS402 cyclic sync modes. It is noted that the bypass ramp of the command selection is not supported in the case of close loop speed position operating mode (command remain to zero until bypass ramp is disabled).

Syntax Serial: ^BPR cc nn
~BPR [cc]

Syntax Scripting: setconfig(_BPR, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Bypass Status

Type: Unsigned 8-bit

Min: 0 Max: 1

Default: 0

Where:

cc = Channel

nn = Bypass Status

0: Disabled

1: Enabled

BRV - Brake Release Voltage

HexCode: E6 CANOpen id: 0x30E6

Description:

This parameter is used to select the Voltage level with which the brake solenoid connected will be released. It is applicable only on products that support PWM brake connection. For more details see product's datasheet. This voltage level is translated in PWM duty cycle applied on Battery volts. If the value is higher than the battery volts, then the value of the battery volts is applied.

Syntax Serial: ^BRV cc nn

~BRV

Syntax Scripting: setconfig(_BRV, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Voltage

Type: Unsigned 16-bit

Min:0 Max: 1000

Default: 120 = 12Volts

Where:

cc = Motor channel

nn = Volts*10

Example:

^BRV 1 200: Sets the Release voltage level of the brake of motor channel 1 to 20Volts.

BHV - Brake Hold Voltage

HexCode: E7 CANOpen id: 0x30E7

Description:

This parameter is used to select the Voltage level with which the brake solenoid connected will be hold released. It is applicable only on products that support PWM brake connection. For more details see product's datasheet. This voltage level is translated in PWM duty cycle applied on Battery volts. If the value is higher than the battery volts, then the half value of the battery volts is applied.

Syntax Serial: ^BHV cc nn

~BHV

Syntax Scripting: setconfig(_BHV, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Voltage

Type: Unsigned 16-bit

Min:0 Max: 1000

Default: 80 = 8Volts

Where:

cc = Motor channel

nn = Volts*10

Example:

^BHV 2 50: Sets the Hold Voltage level of the brake of motor channel 2 to 5Volts.

BDT - Brake Delay Time

HexCode: E8 CANOpen id: 0x30E8

Description:

This parameter is used to select the time period for which the Brake Release Voltage is applied. After this time expires then the Brake Hold Voltage will be applied in order to consume less power. It is applicable only on products that support PWM brake connection. For more details see product's datasheet. If the value is set to 0 then only the Brake Hold Voltage will be applied.

Syntax Serial: ^BDT cc nn

~BDT

Syntax Scripting: setconfig(_BDT, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Delay

Type: Unsigned 16-bit

Min:0 Max: 1000

Default: 500 = 500ms

Where:

cc = Motor channel

nn = Delay in miliseconds

Example:

^BDT 1 200: Sets the Delay Time of the brake of motor channel 1 to 200ms.

BLFB - Closed loop Feedback Sensor

HexCode: 3B CANOpen id: 0x303B

Description:

This parameter is used to select which feedback sensor will be used to measure speed or position. On brushless motors system equipped with optical encoders and/or SSI sensors, this parameter lets you select either of these two sensors or the brushless sensors (i.e. Hall, Sin/Cos, or Resolver), as the source of speed or position feedback. Encoders provide higher precision capture and should be preferred whenever possible. SSI sensors provide absolute feedback and are preferred in position modes (e.g. steering). The choice "Other" is also used to select pulse or analog feedback in some position modes.

Syntax Serial: ^BLFB cc nn
~BLFB

Syntax Scripting: setconfig(_BLFB, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Sensor

Type: Unsigned 8-bit
Min: 0 Max: 1
Default: 0 = Other Sensor

Where:

cc = Motor channel
nn =

0: Other feedback (Encoders, SSI sensors, Analog or RC sensors)
1: Brushless sensor feedback (Hall, Sin/Cos, Resolver)

BLSTD - Stall Detection

HexCode: 3A CANOpen id: 0x303A

Description:

This parameter controls the stall detection for brushless motors, brushed motors in closed loop speed mode and induction motors. If no motion is sensed (i.e. counter remains unchanged) for a preset amount of time while the power applied is above a given threshold, a stall condition is detected and the power to the motor is cut until the next idle motor command is given (0 in case of speed modes, equal to feedback in case of position modes). This parameter allows three combinations of time & power sensitivities. The setting also applies when encoders are used in closed loop speed or closed loop speed position modes on brushed or brushless motors.

Syntax Serial: ^BLSTD cc nn
~BLSTD [cc]

Syntax Scripting: setconfig(_BLSTD, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Mode
Type: Unsigned 8-bit
Min: 0 Max: 3
Default: 2 = 500ms at 25% Power

Where:

cc = Motor channel
nn =
0: Disabled
1: 250ms at 10% Power
2: 500ms at 25% Power
3: 1000ms at 50% Power

Example:

^BLSTD 2: Motor will stop if applied power is higher than 10% and no motion is detected for more than 250ms

BR - Mechanical System Rotating Friction Coefficient

HexCode: 10A CANOpen Id: 0x310A

Description:

This parameter defines the mechanical system rotating friction coefficient, which is utilized at acceleration feedforward control. When this value is zero, it means that no acceleration feedforward control is implemented.

Syntax Serial: ^BR cc nn
~BR [cc]

Syntax Scripting: setconfig(_BR, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Type: Unsigned 8-bit

Min: 1 Max: Total Number motors

Argument 2: Value

Type: Unsigned 32-bit

Min: 0 Max: 99,999,000

Where:

cc = Motor channel

nn = Mechanical System rotating friction coefficient (Nm/(rad/sec)) * 10000000)

Example:

^BR 1 22500: Configure the mechanical system inertia at 2.25 mNm/(rad/sec).

CLERD - Close Loop Error Detection

HexCode: 38 CANOpen id: 0x3038

Description:

This parameter is used to detect large tracking errors due to mechanical or sensor failures, and shut down the motor in case of problem in closed loop speed or position modes. The detection mechanism looks for the size of the tracking error and the duration the error is present. This parameter allows four combinations of time & error level. This parameter is not compatible with Closed Loop Speed Position mode.

Syntax Serial: ^CLERD cc nn
~CLERD

Syntax Scripting: setconfig(_CLERD, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motor Channels

Argument 2: Mode

Type: Unsigned 8-bit

Min: 0 Max: 4

Default: 4

Where:

cc = Motor channel

nn =

0: Detection disabled

1: 250ms at Error > 100

2: 500ms at Error > 250

3: 1000ms at Error > 500

4: Custom (see FEW and FET for configuration)

Example:

^CLERD 2: Motor will stop if command - feedback is greater than 100 for more than 250ms

Note:

Disabling the loop error can lead to runaway or other dangerous conditions in case of sensor failure

EDEC - Motor Fault Deceleration Rate

HexCode: E9 CANOpen id: 0x30E9

Description:

Set the rate of speed change during deceleration for a motor channel, when a fault takes place. The faults under which this rate will be used are, Quick Stop, Deadman Switch and Closed Loop Error. Fault Decceleration value is in 0.1*RPM per second. When using controllers fitted with encoder, the speed and deceleration value are actual RPMs. Brushless motor controllers use the hall sensor for measuring actual speed and acceleration will also be in actual RPM/s.

Syntax Serial: ^EDEC cc nn ~EDEC [cc]

Syntax Scripting: setconfig(_EDEC, cc, nn)

Number of Arguments: 2

Argument 1: Channel Type: Unsigned 8-bit Min: 1 Max: Total Number of Motor Channels
Argument 2: Deceleration Type: Signed 32-bit

Min: 0 Max: 300000 Default: 10000 = 1000.0 RPM/s

Where:

cc = Motor channel

nn = Deceleration time in 0.1 RPM per second

EHL - Encoder Max Limit

HexCode: 4C CANOpen id: 0x304C

Description:

Defines a maximum count value at which the controller will trigger an action when the counter goes above that number. This feature is useful for setting up virtual or soft limit switches .This value, together with the Low Count Limit, are also used in the position mode to determine the travel range when commanding the controller with a relative position command. In this case, the High Limit Count is the desired position when a command of 1000 is received.

Syntax Serial: ^EHL cc nn
~EHL [cc]

Syntax Scripting: setconfig(_EHL, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1	Max: Total Number of Encoders

Argument 2: Value

Type: Signed 32-bit	
Min: -2147M	Max: 2147M
Default: +20000	

Where:

cc = Encoder channel
nn = Counter value

EHLA - Encoder Action at Max

HexCode: 4E CANOpen id: 0x304E

Description:

This parameter lets you select what kind of action should be taken when the high limit count is reached on the encoder. The list of action is the same as in the DINA digital input action list Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^EHLA cc nn
~EHLA [cc]

Syntax Scripting: setconfig(_EHLA, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1	Max: Total Number of Encoders

Argument 2: Action

	Type: Unsigned 8-bit	
	Min: 0	Max: 255
	Default: 0 = No action	

Where:

cc = Encoder channel
aa =
0: No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value

mm = mot1*16 + mot2*32 + mot3*64

EHOME - Encoder Home Count

HexCode: 4F CANOpen id: 0x304F

Description:

Contains a value that will be loaded in the selected encoder counter when a home switch is detected, or when a Home command is received from the serial/USB, or issued from a MicroBasic script.

Syntax Serial: ^EHOME cc nn
~EHOME [cc]

Syntax Scripting: setconfig(_EHOME, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Encoders

Argument 2: Value

	Type: Signed 32-bit	
	Min: -2147M	Max: 2147M
	Default: 0	

Where:

cc = Encoder channel
nn = Counter value to be loaded

ELL - Encoder Min Limit

HexCode: 4B CANOpen id: 0x304B

Description:

Defines a minimum count value at which the controller will trigger an action when the counter dips below that number. This feature is useful for setting up virtual or soft limit switches. This value, together with the High Count Limit, are also used in the position mode to determine the travel range when commanding the controller with a relative position command. In this case, the Low Limit Count is the desired position when a command of -1000 is received.

Syntax Serial: ^ELL cc nn
~ELL [cc]

Syntax Scripting: setconfig(_ELL, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Encoders

Argument 2: Value

	Type: Signed 32-bit	
	Min: -2147M	Max: 2147M
	Default: -20000	

Where:

cc = Encoder channel
nn = Counter value

Example:

^ELL 1 -100000 : Set encoder 1 low limit to minus 100000

ELLA - Encoder Action at Min

HexCode: 4D CANOpen id: 0x304D

Description:

This parameter lets you select what kind of action should be taken when the low limit count is reached on the encoder. The list of action is the same as in the DINA digital input action list Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^ELLA cc (aa + mm)
~ELLA [cc]

Syntax Scripting: setconfig(_ELLA, cc, aa)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Encoders

Argument 2: Action

	Type: Unsigned 8-bit	
	Min: 0	Max: 255
	Default: 0 = No action	

Where:

cc = Encoder channel
aa =
0: No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value

mm = mot1*16 + mot2*32 + mot3*64

EMOD - Encoder Usage

HexCode: 49 CANOpen id: 0x3049

Description:

This parameter defines what use the encoder is for. The encoder can be used to set command or to provide feedback (speed or position feedback). The use of encoder as feedback devices is the most common. Embedded in the parameter is the motor to which the encoder is associated.

Syntax Serial: ^EMOD cc (aa + mm)
~EMOD [cc]

Syntax Scripting: setconfig(_EMOD, cc, aa)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Encoders

Argument 2: Use

	Type: Unsigned 8-bit	
	Min: 0	Max: 255
	Default: 0 = Unused	

Where:

cc = Encoder channel
aa =
0: Unused
1: Command

2: Feedback
mm =
mot1*16 + mot2*32 + mot3*64

Example:

^EMOD 1 18 = Encoder used as feedback for channel 1

EPPR - Encoder Pulse/Rev Value

HexCode: 4A CANOpen id: 0x304A

Description:

This parameter will set the pulse per revolution of the encoder that is attached to the controller. The PPR is the number of pulses that is issued by the encoder when making a full turn. For each pulse there will be 4 counts which means that the total number of a counter increments inside the controller will be 4x the PPR value. Make sure not to confuse the Pulse Per Revolution and the Count Per Revolution when setting up this parameter. Entering a negative number will invert the counter and the measured speed polarity.

Syntax Serial: ^EPPR cc nn
~EPPR [cc]

Syntax Scripting: setconfig(_EPPR, cc, nn)

Number of Arguments: 2

Argument 1: Channel		
	Min: 1	Max: Total Number of Encoders
Argument 2: Value		
	Type: Signed 16-bit	
	Min: -32768	Max: 32767
	Default: 100	

Where:

cc = Encoder channel
nn = PPR value

Example:

^EPPR 2 200 : Sets PPR for encoder 2 to 200

FET – Loop Error Time

HexCode: FD CANOpen id: 0x30FD

Description:

This parameter is used in Close Loop Error Detection and sets the time in the Custom setting of CLERD.

Syntax Serial: ^FET cc nn
~FET

Syntax Scripting: setconfig(_FET, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motor Channels

Argument 2: Mode

Type: Unsigned 32-bit

Min: 0 Max:

Default: 500

Where:

cc = Motor channel

nn = value in units

Example:

^FET 2 500: Motor 2 will stop if command - feedback is greater than set units for more than 500 ms.

FEW – Loop Error Limit

HexCode: FC CANOpen id: 0x30FC

Description:

This parameter is used is Close Loop Error Detection and sets the unit number in the Custom setting of the CLERD.

Syntax Serial: ^FEW cc nn

~FEW

Syntax Scripting: setconfig(_FEW, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motor Channels

Argument 2: Mode

Type: Unsigned 16-bit

Min: 0 Max:

Default: 250

Where:

cc = Motor channel

nn = value in units

Example:

^FEW 2 100: Motor 2 will stop if command - feedback is greater than 100.

ICAP - PID Integrator Limit

HexCode: 32 CANOpen id: 0x3032

Description:

This parameter is the integral cap as a percentage. This parameter will limit maximum level of the Integral factor in the PID. It is particularly useful in position systems with long travel movement, and where the integral factor would otherwise become very large because of the extended time the integral would allow to accumulate. This parameter can be used to dampen the effect of the integral parameter without reducing the gain. This parameter may adversely affect system performance in closed loop speed mode as the Integrator must be allowed to reach high values in order for good speed control.

Syntax Serial: ^ICAP cc nn
~ICAP [cc]

Syntax Scripting: setconfig(_ICAP, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Motors
Argument 2: Cap	Type: Unsigned 8-bit	
	Min: 1	Max: 100
	Default: 100%	

Where:

cc = Motor channel
nn = Integral cap in %

JR - Mechanical System Inertia

HexCode: 109 CANOpen Id: 0x3109

Description:

This parameter defines the mechanical system inertia, which is utilized at acceleration feedforward control. When this value is zero, it means that no acceleration feedforward control is implemented.

Syntax Serial: ^JR cc nn

~JR [cc]

Syntax Scripting: setconfig(_JR, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Type: Unsigned 8-bit

Min: 1 Max: Total Number motors

Argument 2: Value

Type: Unsigned 32-bit

Min: 0 Max: 99,999,000

Where:

cc = Motor channel

nn = Mechanical System inertia load (kg*m2 * 10000000)

Example:

^JR 1 18500: Configure the mechanical system inertia at 18.5 kg*cm2.

KDG - PID Derivative Gain

HexCode: F2 CANOpen id: 0x30F2

Description:

Sets the PID's Derivative Gain. The value is set as the gain multiplied by 10^6. This value is used for both speed and position Derivative gains.

Syntax Serial: ^KDG cc nn
~KDG [cc]

Syntax Scripting: setconfig(_KDG, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: 2 * Total Number of Motors

Argument 2: Gain Type: Unsigned 32-bit

Min: 0 Max: 2,000,000,000 Default: 0

Where:

cc (single channel) =

1: Speed Derivative Gain
2: Position Derivative Gain

cc (dual channel) =

1: Speed Derivative Gain for motor 1
2: Speed Derivative Gain for motor 2
3: Position Derivative Gain for motor 1
4: Position Derivative Gain for motor 2

nn = Derivative Gain *1,000,000

Example:

^KDG 1 1500000: Set motor channel 1 Speed Derivative Gain to 1.5.

KIG - PID Integral Gain

HexCode: F1 CANOpen id: 0x30F1

Description:

Sets the PID's Integral Gain. The value is set as the gain multiplied by 10^6. This value is used for both speed and position integral gains.

Syntax Serial: ^KIG cc nn
~KIG

Syntax Scripting: setconfig(_KIG, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: 2 * Total Number of Motors

Argument 2: Gain Type: Unsigned 32-bit
Min: 0 Max: 2,000,000,000 Default: 0

Where:

cc (single channel) =
1: Speed Integral Gain
2: Position Integral Gain

cc (dual channel) =
1: Speed Integral Gain for motor 1
2: Speed Integral Gain for motor 2
3: Position Integral Gain for motor 1
4: Position Integral Gain for motor 2

nn = Integral Gain *1,000,000

Example:

^KIG 1 1500000: Set motor channel 1 Speed Integral Gain to 1.5.

KPG - PID Proportional Gain

HexCode: F0 CANOpen id: 0x30F0

Description:

Sets the PID's Proportional Gain. The value is set as the gain multiplied by 10^6. This value is used for both speed and position proportional gains.

Syntax Serial: ^KPG cc nn
~KPG

Syntax Scripting: setconfig(_KPG, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: 2 * Total Number of Motors

Argument 2: Gain

Type: Unsigned 32-bit		
Min: 0	Max: 2,000,000,000	Default: 0

Where:

cc (single channel) =

1: Speed Proportional Gain
2: Position Proportional Gain

cc (dual channel) =

1: Speed Proportional Gain for motor 1
2: Speed Proportional Gain for motor 2
3: Position Proportional Gain for motor 1
4: Position Proportional Gain for motor 2

nn = Proportional Gain *1,000,000

Example:

^KPG 1 1500000: Set motor channel 1 Speed Proportional Gain to 1.5.

LPFB - Speed feedback low pass filter bandwidth

HexCode: 112 CANOpen id: 0x3112

Description:

Defines the low pass filter bandwidth applied at speed feedback measurement. Please note that the filter applies to all feedback sensors except for Hall sensors.

Syntax Serial: ^LPFB cc nn
~LPFB [cc]

Syntax Scripting: setconfig(_LPFB, cc, nn)

Number of Arguments: 2

Argument 1: Channel Min: 1 Max: Total Number of Motors

Argument 2: Low pass filter bandwidth (Hz)

Type: Unsigned 8-bit

Min: 0 Max: 150

Default: 45

where:

cc = Motor channel

nn = Low Pass Filter Bandwidth (Hz)

Example:

^LPFB 1 30 : Set the speed feedback low pass filter bandwidth equal to 30 Hz.

MAC - Motor Acceleration Rate

HexCode: 33 CANOpen id: 0x3033

Description:

Set the rate of speed change during acceleration for a motor channel. This command is identical to the AC realtime command. Acceleration value is in 0.1*RPM per second. When using controllers fitted with encoder, the speed and acceleration value are actual RPMs. Brushless motor controllers use the internal sensor (Hall, Sin/Cos or Resolver) for measuring actual speed and acceleration will also be in actual RPM/s. When using the controller without speed sensor, the acceleration value is relative to the Max RPM configuration parameter, which itself is a user-provide number for the speed normally expected speed at full power. Assuming that the Max RPM parameter is set to 1000, and acceleration value of 10000 means that the motor will go from 0 to full speed in exactly 1 second, regardless of the actual motor speed. If value is set to 0 then the command ramp is by-passed.

Syntax Serial: ^MAC cc nn
~MAC [cc]

Syntax Scripting: setconfig(_MAC, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1	Max: Total Number of Motors

Argument 2: Acceleration

Type: Signed 32-bit	
Min: 0	Max: 300000
Default: 10000 = 1000.0 RPM/s	

Where:
cc = Motor channel
nn = Acceleration time in 0.1 RPM per seconds

Note: In Closed Loop Torque Mode the value is translated in miliAmps/sec

MCLE - SSI Multi-turn Counter number of bits

HexCode: 103 CANOpen Id: 0x3103

Description:

This parameters sets the number of bits of the multi-turn counter of the SSI sensor.

Syntax Serial: ^MCLE cc nn

~MCLE [cc]

Syntax Scripting: setconfig(_MCLSE, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Type: Unsigned 8-bit

Min: 1 Max: Total Number of SSI sensors

Argument 2: Value

Type: Unsigned 8-bit

Min: 0 Max: 32

Where:

cc = SSI Sensor channel

nn = Multi-turn Counter number of bits

MDEC - Motor Deceleration Rate

HexCode: 34 CANOpen id: 0x3034

Description:

Set the rate of speed change during deceleration for a motor channel. This command is identical to the DC realtime command. Decceleration value is in 0.1*RPM per second. When using controllers fitted with encoder, the speed and deceleration value are actual RPMs. Brushless motor controllers use the internal sensor (Hall, Sin/Cos or Resolver) for measuring actual speed and decceleration will also be in actual RPM/s. When using the controller without speed sensor, the deceleration value is relative to the Max RPM configuration parameter, which itself is a user-provide number for the speed normally expected speed at full power. Assuming that the Max RPM parameter is set to 1000, and deceleration value of 10000 means that the motor will go from full speed to 0 1 second, regardless of the actual motor speed. If value is set to 0 then the command ramp is by-passed.

Syntax Serial: ^MDEC cc nn

~MDEC [cc]

Syntax Scripting: setconfig(_MDEC, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Type: Unsigned 8-bit

Min: 1 Max: Total Number of Motor Channels

Argument 2: Deceleration

Type: Signed 32-bit

Min: 0 Max: 300000

Default: 10000 = 1000.0 RPM/s

Where:

cc = Motor channel

nn = Deceleration time in 0.1 RPM per second

Note: In Closed Loop Torque Mode the value is translated in miliAmps/sec.

MLX - Molex Input

HexCode: D5 CANOpen id: 0x30D5

Description:

Configure this parameter in order to change the sensors that will be connected to the molex connector. In brushless controllers if SSI sensors are used then hall sensor inputs are re-mapped to the digital input pins, as dictated in the controller model's datasheet. In order to have these pins functional as hall inputs pull-up resistors should be added externally. In brushed controllers if SSI sensors are used then encoder inputs are re-mapped to the DB16 or DB25 pins as per datasheet.

Syntax Serial: ^MLX cc nn

~ MLX [cc]

Syntax Scripting: setconfig(_MLX, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Molex Input

Type: Unsigned 8-bit

Min: 0 Max: 2

Default: 0

Where:
cc = Motor channel
nn = Hall Input
0: Hall Sensors (not for Brushed Controllers).
1: Encoders (not for Brushless controllers).
2: SSI Encoders

Example:

^MLX 2: Configure Molex to have SSI sensors connected.

MDIR - Motor Direction

HexCode: 77 CANOpen id: 0x3077

Description:

This parameter lets you set the motor direction to inverted or direct.

Syntax Serial: ^MDIR cc nn

Where:

cc = Motor Channel

nn = 0: Not inverted

1: Inverted

Syntax Scripting: setconfig(_MDIR, cc, nn)

MMOD - Operating Mode

HexCode: 27 CANOpen id: 0x3027

Description:

This parameter lets you select the operating mode for that channel. See manual for description of each mode.

```
Syntax Serial: ^MMOD cc nn
               ~MMOD [cc]
```

Syntax Scripting: setconfig(_MMOD, cc, nn)

Number of Arguments: 2

Argument 1: Channel		
	Min: 1	Max: Total Number of Motors
Argument 2: Mode		
	Type: Unsigned 8-bit	
	Min: 0	Max: 6
	Default: 0 = Open loop	

Where:

cc = Motor channel
nn =
0: Open-loop
1: Closed-loop speed
2: Closed-loop position relative
3: Closed-loop count position
4: Closed-loop position tracking
5: Closed-loop torque
6: Closed-loop speed position

Example:

^MMOD 2 : Select Closed loop position relative

MNRPM - Min Speed RPM

HexCode: CB CANOpen id: 0x30CB

Description:

This parameter contains the minimum speed that can be commanded for a motor in closed loop speed or closed loop speed position modes. See Figure 15-1 for a more detailed description.

Syntax Serial: ^MNRPM cc nn

~MNRPM [cc]

Syntax Scripting: setconfig(_MNRPM, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Speed (RPM)

Type: Unsigned 16-bit

Min: 0 Max: 65535

Default: 0

Where:

cc = Channel

nn = Min Speed RPM

MSTA - SSI Multi-turn Counter start bit position

HexCode: 102 CANOpen Id: 0x3102

Description:

This parameter sets the position of the first bit of the multi-turn counter inside the SSI frame.

Syntax Serial: ^MSTA cc nn

~MSTA [cc]

Syntax Scripting: setconfig(_MSTA, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Type: Unsigned 8-bit	
	Min: 1	Max: Total Number of SSI sensors

Argument 2: Value

	Type: Unsigned 8-bit	
	Min: 0	Max: 32

Where:

cc = SSI Sensor channel

nn = Multi-turn Counter's first bit position

MVEL - Position Mode Velocity

HexCode: 35 CANOpen id: 0x3035

Description:

This parameter is the speed at which the motor moves while in position mode. Values are in RPMs. To change velocity while the controller is in operation, use the !S runtime command.

Syntax Serial: ^MVEL cc nn

~MVEL [cc]

Syntax Scripting: setconfig(_MVEL, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Motors

Argument 2: Velocity

	Type: Signed 32-bit	
	Min: 0	Max: 30000
	Default: 1000 RPM	

Where:

cc = Motor channel
nn = Velocity value in RPM

MXMD - Mixed Mode

HexCode: 05 CANOpen id: 0x3005

Description:

Selects the mixed mode operation. It is applicable to dual channel controllers and serves to operate the two channels in mixed mode for tank-like steering. There are 3 possible values for this parameter for selecting separate or one of the two possible mixed mode algorithms. Details of each mixed mode is described in Section 7, chapter "Mixed Mode Select.

Syntax Serial: ^MXMD nn
~MXMD

Syntax Scripting: setconfig(_MXMD, nn)

Number of Arguments: 1

Argument 1: Mode
Type: Unsigned 8-bit
Min: 0 Max: 2
Default: 0 = Separate

Where:
nn =
0: Separate
1: Mode 1
2: Mode 2

Example:

^MXMD 0 : Set mode to separate

MXPF - Motor Max Power Forward

HexCode: 28 CANOpen id: 0x3028

Description:

This parameter lets you select the scaling factor for the PWM output, in the forward direction, as a percentage value. This feature is used to connect motors with voltage rating that is less than the battery voltage. For example, using a factor of 50% it is possible to connect a 12V motor onto a 24V system, in which case the motor will never see more than 12V at its input even when the maximum power is applied.

Syntax Serial: ^MXPF cc nn
~MXPF [cc]

Syntax Scripting: setconfig(_MXPF, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: MaxPower

	Type: Unsigned 8-bit	
	Min: 25	Max: 100
	Default: 100%	

Where:

cc = Motor channel
nn = Max Power

MXPR - Motor Max Power Reverse

HexCode: 29 CANOpen id: 0x3029

Description:

This parameter lets you select the scaling factor for the PWM output, in the reverse direction, as a percentage value. This feature is used to connect motors with voltage rating that is less than the battery voltage. For example, using a factor of 50% it is possible to connect a 12V motor onto a 24V system, in which case the motor will never see more than 12V at its input even when the maximum power is applied.

Syntax Serial: ^MXPR cc nn
~MXPR [cc]

Syntax Scripting: setconfig(_MXPR, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Motors

Argument 2: MaxPower

	Type: Unsigned 8-bit	
	Min: 25	Max: 100
	Default: 100%	

Where:

cc = Motor channel
nn = Max Power

MXRPM - Max Speed RPM

HexCode: 36 CANOpen id: 0x3036

Description:

Commands sent via analog, pulse or the !G command only range between -1000 to +1000. The Max RPM parameter lets you select which actual speed, in RPM, will be considered the speed to reach when a +1000 command is sent. In open loop, this parameter is used together with the acceleration and deceleration settings in order to figure the ramping time from 0 to full power.

Syntax Serial: ^MXRPM cc nn
~MXRPM cc

Syntax Scripting: setconfig(_MXRPM, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Motors

Argument 2: Speed (RPM)

	Type: Unsigned 16-bit	
	Min: 10	Max: 65535
	Default: 1000 RPM	

Where:

cc = Motor channel
nn = Max Speed RPM

MXTRN - Position Turns Min to Max

HexCode: 37 CANOpen id: 0x3037

Description:

The parameter indicates the number of rotor rotations from the sensor's user-defined minimum to maximum value. It is utilized in Closed Loop Relative and Tracking Position modes for measuring the rotor's speed. When using an analog or pulse feedback sensor, the user must manually set the MSTRN value. However, with other sensors such as AB encoders, SSI, or Hall sensors, the firmware automatically calculates this value.

For more details, refer to the 'Closed Loop Relative and Tracking Position Modes' section.

Parameter calculation indicating table:

Sensor	Calculation	Mode
Pulse or Analog	Measure the number of rotor turns from -1000 to +1000 feedback value and set it to the MXTRN parameter	User
AB Encoder	MXTRN = (EHL-ELL)*100/(EPPR*4)	Automatic
Hall	MXTRN = (BHL-BLL)*100/(BPOL*6)	Automatic
SSI	MXTRN = ((SHL-SLL)*100)/(SCPR*SPOL)	Automatic

Syntax Serial: ^MXTRN cc nn
~MXTRN [cc]

Syntax Scripting: setconfig(_MXTRN, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Motors

Argument 2: Turns

	Type: Signed 32-bit	
	Min: 10	Max: 100000
	Default: 10000 = 1000.0 turns	

Where:

cc = Motor channel
nn = Number of turns * 10

Example:

^MXTRN 1 2000: Set max turns for motor 1 to 200.0 turns

NOMA - Nominal Current

HexCode: FA CANOpen id: 0x30FA

Description:

Set the nominal current of the motor. This is the current that the motor can draw continuously. This value is used for the I2T protection. For more details check chapter "I2T Protection in SECTION 7.

Syntax Serial: ^NOMA cc nn
~NOMA cc

Syntax scripting: setconfig(_NOMA,cc,nn)

Argument 1: Channel	Type: Unsigned 8-bit Min: 1 Max: Total Number of Motors
Argument 2: Current	Type: Unsigned 32-bit Default: see note.

Where:

cc=Motor channel

nn=Current in Amps*10

Example:

^NOMA 1 100. Set nominal current at 10A.

Note:

Default value is typically set the controller's continuous amps as defined in the datasheet.

OVH - Overvoltage hysteresis

HexCode: 42 CANOpen id: 0x3042

Description:

This voltage gets subtracted to the overvoltage limit to set the voltage at which the overvoltage condition will be cleared. For instance, if the overvoltage limit is set to 500 (50.0) and the hysteresis is set to 50 (5.0V), the MOSFETs will turn off when the voltage reaches 50V and will remain off until the voltage drops under 45V

Syntax Serial: ^OVH nn
~OVH

Syntax Scripting: setconfig(_OVH, nn)

Number of Arguments: 1

Argument 1: Voltage

Type: Unsigned 8-bit
Min: 0 Max: 255 = 25.5V
Default: 50 = 5.0V

Where:

nn = Volts *10

Example:

^OVH 45 : Sets hysteresis at 4.5V

Note:

Make sure that overvoltage limit minus hysteresis is above the supply voltage.

OVL - Overvoltage Limit

HexCode: 02 CANOpen id: 0x3002

Description:

Sets the voltage level at which the controller must turn off its power stage and signal an Overvoltage condition. Value is in volts multiplied by 10 (e.g. 450 = 45.0V) . The power stage will turn back on when voltage dips below the Overvoltage Clearing threshold that is set with the OVH configuration command.

Syntax Serial: ^OVL nn
~OVL

Syntax Scripting: setconfig(_OVL, nn)

Number of Arguments: 1

Argument 1: Voltage
Type: Unsigned 16-bit
Min: 100 = 10.0V Max: See Product Datasheet
Default: See Product Datasheet

Where:

nn = Volts *10

Example:

^OVL 400 : Set Overvoltage limit to 40.0V

Note:

On firmware versions 1.5 and earlier, no hysteresis exists and the overvoltage condition is cleared as soon as the voltage dips below the overvoltage limit.

OTL - Over Temperature Limit

HexCode: D1 CANOpen id: 0x30D1

Description:

Sets the Heatsink and Motor Temperature level at which the controller must turn off its power stage and signal an OverHeat condition. Value is in Celsius degrees . When temperature reaches 5 degrees below the limit, the controller starts to decrease the maximum applied power (duty cycle), by 20% for each additional degree. The power stage will turn back on gradually when heat sink temperature dips below the above limit. See Section 7, chapter "Temperature-Based Protection" for more details.

Syntax Scripting: setconfig(_OTL, cc, nn)

Number of Arguments: 2

Argument 1: Temperature Sensor	Type: Unsigned 8-bit	Min: 1	Max: 3
Argument 2: Temperature	Type: Unsigned 16-bit	Min: 0	Max: 85 for Heatsink temperature and 120 for motor temperatures

Where:

cc =

1: OverTemperature Limit for Heatsink Temperature

2: OverTemperature Limit for Channel 1 Motor Temperature

3: OverTemperature Limit for Channel 2 Motor Temperature

nn = Temperature in Celsius degrees

Example:

^OTL 1 75: Set Over temperature limit for Heatsink Temperature to 75 Celsius degrees.

R25 - Thermistor Resistance at 25^{o}C

HexCode: 105 CANOpen id: 0x3105

Description:

Set the resistance of the thermistor at 25oC. This value can be derived from the thermistor datasheet directly. Check chapter Connecting External Thermistor to Analog Inputs, SECTION 3 for more details.

Syntax Serial: ^R25 cc nn

~R25 cc

Syntax scripting: setconfig(_R25,cc,nn)

Argument 1: Channel	Type: Unsigned 8-bit Min: 1 Max: Total Number of Motors
Argument 2: Resistance	Type: Signed 32-bit Default: 0

Where:

cc=Motor channel

nn=Resistance in Ohm (Ω)

Example:

^R25 1 1000. Set Resistance of the thermistor at 25oC at 1000Ohm.

SCLE - SSI Counter number of bits

HexCode: 101 CANOpen Id: 0x3101

Description:

Set the number of bits of the angle counter of the SSI sensor.

Syntax Serial: ^SCLE cc nn
~SCLE [cc]

Syntax Scripting: setconfig(_SCLE, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Type: Unsigned 8-bit
Min: 1 Max: Total Number of SSI sensors

Argument 2: Value
Type: Unsigned 8-bit
Min: 0 Max: 32

Where:

cc = SSI Sensor channel

nn = Angle Counter number of bits

SCLK - SSI Clock Speed

HexCode: 107 CANOpen Id: 0x3107

Description:

This parameter will set the frequency of the SSI clock. In case of controllers supporting more than one SSI sensors, this parameter affects all of them. The necessity of modifying that parameter exists based on the specifications of the sensor. Sensors with longer data frame will require quicker clock period, in order to get the full frame within 62,5us, that is the feedback update period.

Syntax Serial: ^SCLK nn

~SCLK

Syntax Scripting: setconfig(_SCLK, nn)

Number of Arguments: 1

Argument 1: Value

Type: Unsigned 8-bit

Min: 0	Max: 4	Default: 0

Where:

nn =

0 for 680kHz

1 for 1360kHz

2 for 2720kHz

3 for 5440kHz

4 for 10800kHz

Example:

^SCLK 1: Sets SSI clock to 1360kHz.

SED - Sensor Error Detection

HexCode: E4 CANOpen id: 0x30E4

Description:

This parameter sets the sensor error detection level. There are three levels of the sensitivity, Disabled, Tolerant and Strict. If it is configured as disabled then the function is inactive. when the selected value is Tolerant then the fault will be activated after 5 detected sensor errors (it is reseted when motor command is zero or the direction is reversed). If it is configured as Strict then the fault will be generated after the first sensor error. For more details check chapter “Sensor Error Detection” in SECTION 8.

Syntax Serial: ^SED cc nn

~SED [cc]

Syntax Scripting: setconfig(_SED, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Type: Unsigned 8-bit

Min: 1 Max: Total Number of Brush-less Motor

Argument 2: Mode

Default: 2 = Strict

Where

cc= Number of brush-less motor

nn=

0: Disable

1: Tolerant

2: Strict

SFTS - Safety Switch Connected

HexCode: 114 CANOpen Id: 0x3114

Description:

This parameter is set when a Safety Brake Switch is connected between the UVW connectors of the respective controller channel and the motor. Having the Safety Brake Switch connected there is no need to perform the MOSFET health test since it is performed by it.

Syntax Serial: ^SFTS cc nn

~SFTS [cc]

Syntax Scripting: setconfig(_SFTS, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Type: Unsigned 8-bit

Min: 1 Max: Total Number of motors

Argument 2: Value

Type: Unsigned 8-bit

Min: 0 Max: 1

Where:

cc = motor channel

nn =

0: Not connected

1: Connected

SHL - SSI Sensor Max Limit

HexCode: D8 CANOpen id: 0x30D8

Description:

Defines a maximum count value at which the controller will trigger an action when the SSI counter goes above that number. This feature is useful for setting up virtual or soft limit switches. This value, together with the SSI Sensor Low Count Limit, are also used in the position mode to determine the travel range when commanding the controller with a relative position command. In this case, the SSI Sensor High Limit Count is the desired position when a command of 1000 is received.

Syntax Serial: ^SHL cc nn

~SHL [cc]

Syntax Scripting: setconfig(_SHL, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of SSI Sensors

Argument 2: Value

Type: Signed 32-bit

Min: -2147M Max: 2147M

Default: +20000

Where:

cc = SSI Sensor channel

nn = Counter value

SHLA - SSI Sensor Action at Max

HexCode: DA CANOpen id: 0x30DA

Description:

This parameter lets you select what kind of action should be taken when the high limit count is reached on the SSI Sensor. The list of action is the same as in the DINA digital input action list Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^SHLA cc nn

~SHLA [cc]

Syntax Scripting: setconfig(_SHLA, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of SSI Sensors

Argument 2: Action

Type: Unsigned 8-bit
Min: 0 Max: 255
Default: 0 = No action

Where:
cc = SSI Sensor channel
aa =
0: No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value
mm = mot1*16 + mot2*32 + mot3*48

SHOME - SSI Sensor Home Count

HexCode: DB CANOpen id: 0x30DB

Description:

Contains a value that will be loaded in the selected SSI counter when a home switch is detected, or when a Home command is received from the serial/USB, or issued from a MicroBasic script. When the SSI sensor is used as absolute sensor (Absolute feedback), this value will hold an offset with which the SSI sensor counter is subtracted.

Syntax Serial: ^SHOME cc nn
~SHOME [cc]

Syntax Scripting: setconfig(_SHOME, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of SSI Sensors

Argument 2: Value

Type: Signed 32-bit
Min: -2147M Max: 2147M
Default: 0

Where:
cc = SSI Sensor channel
nn = Counter value to be loaded, or Counter offset.

SLEN - SSI sensor's frame total number of bits

HexCode: FF CANOpen Id: 0x30FF

Description:

This parameters sets the total number of bits of the SSI sensor's frame. By setting a negative number it changes the polarity of the angle counter.

Syntax Serial: ^SLEN cc nn
~SLEN [cc]

Syntax Scripting: setconfig(_SLEN, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Type: Unsigned 8-bit
Min: 1 Max: Total Number of SSI sensors

Argument 2: Value
Type: Signed 8-bit
Min: -47 Max: 47

Where:
cc = SSI Sensor channel
nn = SSI frame's number of bits

SLL - SSI Sensor Min Limit

HexCode: D7 CANOpen id: 0x30D7

Description:

Defines a minimum count value at which the controller will trigger an action when the counter dips below that number. This feature is useful for setting up virtual or soft limit switches. This value, together with the SSI Sensor High Count Limit, are also used in the position mode to determine the travel range when commanding the controller with a relative position command. In this case, the SSI Sensor Low Limit Count is the desired position when a command of -1000 is received.

Syntax Serial: ^SELL cc nn
~SLL [cc]

Syntax Scripting: setconfig(_SLL, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Encoders

Argument 2: Value

Type: Signed 32-bit

Min: -2147M Max: 2147M
Default: -20000

Where:
cc = SSI Sensor channel
nn = SSI Counter value

Example:

^SLL 1 -100000 : Set SSI Sensor 1 low limit to minus 100000

SLLA - SSI Sensor Action at Min

HexCode: D9 CANOpen id: 0x30D9

Description:

This parameter lets you select what kind of action should be taken when the low limit count is reached on the SSI Sensor. The list of action is the same as in the DINA digital input action list Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^SLLA cc (aa + mm)

~SLLA [cc]

Syntax Scripting: setconfig(_SLLA, cc, aa)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of SSI Sensors

Argument 2: Action

Type: Unsigned 8-bit
Min: 0 Max: 255
Default: 0 = No action

Where:
cc = SSI Sensorchannel
aa =
0: No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script

8: Load counter with home value
mm = mot1*16 + mot2*32 + mot3*64

SMOD - SSI Sensor Usage

HexCode: D6 CANOpen id: 0x30D6

Description:

This parameter defines what use the SSI Sensor is for. The encoder can be used to set command or to provide feedback (speed or position feedback). The use of SSI Sensor as feedback devices is the most common. If absolute Feedback option is set then the feedback will back the absolute value of the SSI Sensor, which is useful for Closed Loop Position Modes. Embedded in the parameter is the motor to which the SSI Sensor is associated.

Syntax Serial: ^SMOD cc (aa + mm)
^SMOD [cc]

Syntax Scripting: setconfig(_SMOD, cc, aa)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of SSI Sensors

Argument 2: Use
Type: Unsigned 8-bit
Min: 0 Max: 255
Default: 0 = Unused

Where:

cc = SSI Sensor channel
aa =
0: Unused
1: Command
2: Feedback
3: Absolute Feedback
mm = mot1*16 + mot2*32 + mot3*64

Example:

^SMOD 1 19 = Encoder used as absolute feedback for channel 1

SSTA - SSI Counter start bit position

HexCode: 100 CANOpen Id: 0x3100

Description:

This parameter sets the position of the first bit of the angle counter inside the SSI frame.

Syntax Serial: ^SSTA cc nn

~SSTA [cc]

Syntax Scripting: setconfig(_SSTA, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Type: Unsigned 8-bit

Min: 1 Max: Total Number of SSI sensors

Argument 2: Value

Type: Unsigned 8-bit

Min: 0 Max: 32

Where:

cc = SSI Sensor channel

nn = Angle Counter's first bit position

THLD - Short Circuit Detection Sensitivity

HexCode: 04 CANOpen id: 0x3004

Description:

This configuration parameter sets the level for the short circuit detection sensitivity. There are 3 sensitivity levels from 0 to 2.

- If set as High Sensitivity (0, default), then it works as an over-current threshold detector. The current threshold is defined at each product's datasheet. So if current goes above that threshold the short fault is triggered.
- If set as Medium Sensitivity (1) then when the current goes beyond the current threshold, the controller deactivates the MOSFETs for 5ms and then recovers. If that happens more than 3 times in a 128ms period then short fault is triggered.
- If set as Low Sensitivity (2) then when the current goes beyond the current threshold, the controller deactivates the MOSFETs for 5ms and then recovers. If that happens more than 7 times in a 128ms period then short fault is triggered.

Syntax Serial: ^THLD nn

~THLD

Syntax Scripting: setconfig(_THLD, nn)

Number of Arguments: 1

Argument 1: Threshold Type: Unsigned 8-bit Min: 0 Max: 2 Default: 0 = High Sensitivity

Where:

nn =

0: High sensitivity

1: Medium sensitivity

2: Low sensitivity

Example:

^THLD 1 : Set short circuit detection sensitivity to medium.

TNM - Motor Torque Constant

HexCode: DF CANOpen id: 0x30DF

Description:

This configuration parameter sets the motor torque constant. It is a value with which we can convert the peak motor current (A) to torque (NM) and vice versa. This value is in miliNm/Amps and is usually referred in the motor datasheets. The conversion is used by the TC and TSL commands and TRQ query.

Note: In the motor torque constant, the peak amplitude of motor current is considered, not the RMS motor current value.

Syntax Serial: ^TNM cc nn

~TNM cc

Syntax Scripting: setconfig(_TNM, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Torque Constant Type: Signed 32-bit

Min: 0 Default: 1000 = 1Nm/Amps

Where:

cc = Motor channel
nn = Motor Torque Constant (miliNm/Amps)

Example:

^TNM 1 1523: Set torque constant for motor 1 to 1.523 Nm/Amps.

TPAL - Time for Amps Limit

HexCode: FB CANOpen id: 0x30FB

Description:

Set the maximum time that motor can handle the maximum current as configured in the ALIM configuration command. This value is used for the I2T protection. For more details check chapter "I2T Protection in SECTION 7. If set to 0 I2T protection is disabled.

Syntax Serial: ^TPAL cc nn

~TPAL cc

Syntax scripting: setconfig(_TPAL,cc,nn)

Argument 1: Channel	Type: Unsigned 8-bit Min: 1 Max: Total Number of Motors
Argument 2: Current	Type: Unsigned 32-bit Default: 0

Where:

cc=Motor channel
nn=Time in seconds

Example:

^TPAL 1 5. Set maximum time the motor can handle ALIM Amps at 5 seconds.

UVL - Undervoltage Limit

HexCode: 03 CANOpen id: 0x3003

Description:

Sets the voltage below which the controller will turn off its power stage. The voltage is entered as a desired voltage value multiplied by 10. Undervoltage condition is cleared as soon as voltage rises above the limit.

Syntax Serial: ^UVL nn

~UVL

Syntax Scripting: setconfig(_UVL, nn)

Number of Arguments: 1

Argument 1: Voltage

Type: Unsigned 16-bit
Min: 50 = 5.0V Max: Max Voltage in Product
Datasheet
Default: 50 = 5.0V

Where:

nn = Volts *10

Example:

^UVL 100 : Set undervoltage limit to 10.0 V

Brushless Specific Commands

TABLE 15-39. Brushless Specific Commands

Command	Arguments	Description
BADJ	Channel Angle	Brushless Angle Zero Adjust
BADV	Channel Value	Brushless timing angle adjust
BFBK	Channel Sensor	Brushless Sinusoidal Angle Sensor
BHL	Channel Value	Brushless Internal Sensor Max Limit
BHLA	Channel Action	Brushless Internal Sensor Action at Max
BHOME	Channel Value	Brushless Internal Sensor Home Count
BLL	Channel Value	Brushless Internal Sensor Min Limit
BLLA	Channel Action	Brushless Internal Sensor Action at Min
BMOD	Channel Mode	Brushless Switching Mode
BPOL	Channel NbrPoles	Number of pole pairs
BZPW	Channel Amps	Brushless Reference Seek Power
HPO	InputNbr Value	Hall Sensor Position Type
FWVR	Channel Value	Field Weakening Voltage Ratio
HSAT	InputNbr Value	Hall Sensor Angle Table
HSM	InputNbr Value	Hall Sensor Map
KIF	PID Channel Gain	Current PID Integral Gain
KPF	PID Channel Gain	Current PID Proportional Gain
LD	Channel Value	D-axis motor Inductance
LQ	Channel Value	Q-axis motor Inductance
PSA	Channel Angle	Phase Shift Angle
MXPW	Channel Value	Max Output Power at Constant Power Region
RS	Channel Value	Motor Stator Resistance
SPOL	Channel Poles	SinCos/SSI Sensor Pole Pairs
SWD	InputNbr Value	Swap Windings
TID	Channel Amps	FOC Target Id
VK	Channel Value	Motor Voltage constant
ZSMA	Channel Value	Cos Amplitude
ZSMC	InputNbr Value	SinCos Calibration

BADJ - Brushless Angle Zero Adjust

HexCode: 60 CANOpen id: 0x3060

Description:

When being in sinusoidal mode and Sin/Cos, Resolver or SSI feedback sensors are used or in Hall+Encoder Trapezoidal mode, this configuration command stores results of automatic zero degrees angle search after performing the Motor/Sensor Setup (%clmod 2/3 or

!mss 1/2). The angle represents the mechanical offset between the sensor's zero position and the rotor's zero position. The value is in electrical degrees ranging from 0 to 511 for a full electrical turn. The value can then be fine tuned manually.

Syntax Serial: ^BADJ cc nn
~BADJ [cc]

Syntax Scripting: setconfig(_BADJ, cc, nn)

Number of Arguments:

Argument 1: Channel

	Min: 1	Max: Total Number of Motors

Argument 2: Angle

	Type: Signed 16-bit	
	Min: -511	Max: 511
	Default: 0	

Where:

cc = Motor channel
nn = Angle

Example:

^BADJ 1 220 : Manually set the zero to 220 degrees

BADV - Brushless timing angle adjust

HexCode: 61 CANOpen id: 0x3061

Description:

When operating in sinusoidal mode, this parameter shifts by number of degrees to the 3 phases rotating magnetic field. This value works symmetrically to produce the same results in both rotation direction. The value is in electrical degrees ranging from 0 to 511 for a full electrical turn.

Syntax Serial: ^BADV cc nn
~BADV [cc]

Syntax Scripting: setconfig(_BADV, cc, nn)

Number of Arguments: 2

Argument 1: Channel

	Min: 1	Max: Total Number of Motors

Argument 2: Value

	Type: Signed 16-bit
Min: -511	Max: 511
Default: 0	

Where:

cc = Motor channel
nn = Angle

Example:

^BADV 1 20 : Advance motor 1 timing by 20 degrees

BFBK - Brushless Sinusoidal Angle Sensor

HexCode: 63 CANOpen id: 0x3063

Description:

Selects the type of rotor angle sensor to be used for brushless motors when operated in sinusoidal mode.

Syntax Serial: ^BFBK cc nn
~BFBK [cc]

Syntax Scripting: setconfig(_BFBK, cc)

Number of Arguments:

Argument 1: Channel		
	Min: 1	Max: Total Number of Motors
Argument 2: Sensor		
	Type: Unsigned 8-bit	
	Min: 0	Max: 4
	Default: 0 = Encoder	

Where:

cc = Motor channel
nn =
0: Encoder
1: Hall
2: Hall + Encoder
3: SPI/SSI sensor
4: Sin/Cos sensor
5: Resolver

BHL - Brushless Internal Sensor Max Limit

HexCode: 3E CANOpen id: 0x303E

Description:

This parameter allows you to define a minimum Internal Sensor count value at which the controller will trigger an action when the counter rises above that number. This feature is useful for setting up virtual or soft limit switches. This value, together with the respective Min Limit, are also used in the position mode to determine the travel range when commanding the controller with a relative position command. In this case, the Max Limit is the desired position when a command of 1000 is received.

Syntax Serial: ^BHL cc nn
~BHL [cc]

Syntax Scripting: setconfig(_BHL, cc, nn)

Number of Arguments: 2

Argument 1: Channel		
	Min: 1	Max: Total Number of Motors
Argument 2: Value		
	Type: Signed 32-bit	
	Min: -2147M	Max: +2147M
	Default: 20000	

Where:

cc = Motor channel
nn = Counter value

Example:

^BHL 10000 : Set brushless Internal Sensor Max Limit to 10000 counts

Note:

Counter is not an absolute position. A homing sequence is necessary to set a reference position.

BHLA - Brushless Internal Sensor Action at Max

HexCode: 40 CANOpen id: 0x3040

Description:

This parameter lets you select what kind of action should be taken when the max limit is reached on the Internal Sensor counter. The list of action is the same as in the DINA digital input action list. Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^BHLA cc nn
~BHLA [cc]

Syntax Scripting: setconfig(_BHLA, cc, nn)

Number of Arguments: 2

Argument 1: Channel		
	Min: 1	Max: Total Number of Motor
Argument 2: Action		
	Type: Unsigned 8-bit	
	Min: 0	Max: 255
	Default: 0 = No action	

Where:

cc = Motor channel

aa =
0 : No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value

mm = mot1*16 + mot2*32 + mot3*64

Example:

^BHLA 1 36 : Forward limit switch for motor 2 (4 + 32)

BHOME - Brushless Internal Sensor Home Count

HexCode: 3C CANOpen id: 0x303C

Description:

This parameter contains a value that will be loaded in the internal sensor counter when a home switch is detected, or when a Home command is received from the serial/ USB, or issued from a MicroBasic script.

Syntax Serial: ^BHOME cc nn
~BHOME [cc]

Syntax Scripting: setconfig(_BHOME, cc, nn)

Number of Arguments: 2

Argument 1: Channel		
	Min: 1	Max: Total Number of Motors
Argument 2: Value		
	Type: Signed 32-bit	
	Min: -2147M	Max: +2147M
	Default: 0	

Where:

cc = Motor channel
nn = Counter value to be loaded

Example:

^BHOME 1 10000 : load Internal Sensor counter with 10000 when Home command is received

Note:

Change counter only while in open loop if brushless counter is used for speed or position feedback

BLL - Brushless Internal Sensor Min Limit

HexCode: 3D CANOpen id: 0x303D

Description:

This parameter defines a minimum Internal Sensor count value at which the controller will trigger an action when the counter dips below that number. This feature is useful for setting up virtual or soft limit switches. This value, together with the respective Max Limit, are also used in the position mode to determine the travel range when commanding the controller with a relative position command. In this case, the Min Limit is the desired position when a command of -1000 is received.

Syntax Serial: ^BLL cc nn
~BLL [cc]

Syntax Scripting: setconfig(_BLL, cc, nn)

Number of Arguments: 2

Argument 1: Channel		
	Min: 1	Max: Total Number of Motors
Argument 2: Value		
	Type: Signed 32-bit	
	Min: -2147M	Max: +2147M
	Default: -20000	

Where:

cc = Motor channel
nn = Counter value

Example:

^BLL 1 -10000 : Set motor 1 Internal Sensor min limit to -10000 counts

Note:

Counter is not an absolute position. A homing sequence is necessary to set a reference position.

BLLA - Brushless Internal Sensor Action at Min

HexCode: 3F CANOpen id: 0x303F

Description:

This parameter lets you select what kind of action should be taken when the min limit is reached on the Internal Sensor counter. The list of action is the same as in the DINA digital input action list. Embedded in the parameter is the motor channel(s) to which the action should apply.

Syntax Serial: ^BLLA cc aa
~BLLA [cc]

Syntax Scripting: setconfig(_BLLA, cc, aa)

Number of Arguments: 2

Argument 1: Channel		
	Min: 1	Max: Total Number of Motors
Argument 2: Action		
	Type: Unsigned 8-bit	
	Min: 0	Max: 255
	Default: 0 = No action	

Where:

cc = Motor channel
aa =
0: No action
1: Quick stop
2: Emergency stop
3: Motor stop
4: Forward limit switch
5: Reverse limit switch
6: Invert direction
7: Run MicroBasic script
8: Load counter with home value

mm = mot1*16 + mot2*32 + mot3*64

Example:

^BLLA 1 37 : Reverse limit switch for motor 2 (5 + 32)

BMOD - Brushless Switching Mode

HexCode: 5F CANOpen id: 0x305F

Description:

Selects the switching mode when controlling brushless motors. Additional settings apply for each mode.

Syntax Serial: ^BMOD cc nn
~BMOD [cc]

Syntax Scripting: setconfig(_BMOD, cc, nn)

Number of Arguments: 2

Argument 1: Channel		
	Min: 1	Max: Total Number of Motors

Argument 2: Mode

Type: Unsigned 8-bit	
Min: 0	Max: 2
Default: 0 = Trapezoidal	

Where:

cc = Motor channel

nn =
0: Trapezoidal
1: Sinusoidal

Note:

After changing this setting, the motor will perform a reference searched when selecting sinusoidal mode with encoder feedback.

BPOL - Number of Pole Pairs

HexCode: 39 CANOpen id: 0x3039

Description:

This parameter is used to define the number of pole pairs of the brushless motor connected to the controller. This value is used to convert the hall sensor transition counts into actual RPM and number of motor turns. Entering a negative number will invert the counter and the measured speed polarity.

```
Syntax Serial: ^BPOL cc nn
               ~BPOL
```

Syntax Scripting: setconfig(_BPOL, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1	Max: Total Number of Motors

Argument 2: Number of Pole Pairs

Type: Signed 8-bit	
Min: -127	Max: +127
Default: 2	

Where:

cc = Motor channel

nn = Number of pole pairs

BZPW - Brushless Reference Seek Power

HexCode: 62 CANOpen id: 0x3062

Description:

Sets the level of Amps to be applied to the motor coils during Motor/Sensor Setup. Motor/Sensor Setup is automatically initiated every time the controller is powered up when sinusoidal with encoder feedback is selected. Motor/Sensor Setup in necessary to be performed only once in the other cases.

Syntax Serial: ^BZPW cc nn
~BZPW [cc]

Syntax Scripting: setconfig(_BZPW, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Amps

Type: Unsigned 16-bit
Min: 0
Default: 50 = 5.0A

Where:

cc = Motor channel
nn = Amps x 10

FWVR - Field Weakening Voltage Ratio

Hex Code: F5 CANOpen id: 0x30F5

Description:

This parameter defines the percentage of the maximum power (stator voltage) to be regulated by the automatic field weakening control. Value equal to 1000 (100%) means that no field weakening control will be applied.

Syntax Serial: ^FWVR cc nn
~FWVR[cc]

Syntax Scripting: setconfig(_FWVR, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Field Weakening Voltage Ratio

Type: Unsigned 16-bit

Min: 750 Max: 1000

Default: 1000

where:

cc=Motor channel

nn=Field Weakening Voltage ratio from 750 (75%) to 1000 (100%) range

Example:

^FWVR 1 900: Configure the voltage level for field weakening control at 90% of the maximum motor output voltage (Power) applied.

HPO - Hall Sensor Position Type

HexCode: A5 CANOpen id: 0x30A5

Description:

This parameter selects the Hall sensor spacing in the motor. Practically all brushless motors use Hall sensors with 120 degrees spacing. Some motors have Hall sensors with 60 degrees. Change this parameter only if the motor's manual clearly specifies 60 degrees spacing.

Syntax Serial: ^HPO cc nn

~HPO [cc]

Syntax Scripting: setconfig(_HPO, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Hall Position

Type: Unsigned 8-bit

Min: 0 Max: 1

Default: 0

Where:

cc = Motor channel

nn = Hall Sensor Position

0: 120degree

1: 60degree

Example:

^HPO 1 1: Configure that the Hall Sensor of motor 1 are spaced by 60 degrees.

HSAT - Hall Sensor Angle Table

HexCode: C2 CANOpen id: 0x30C2

Description:

This parameter is calculated automatically during the motor sensor setup when Hall Sinusoidal or Hall+Encoder Sinusoidal is configured. It represents the electrical angle (range 0-512) at each hall transition at each direction.

Syntax Serial: ^HSAT cc nn

~HSAT [cc]

Syntax Scripting: setconfig(_HSAT, cc, nn)

Number of Arguments: 2

Argument 1: Input Value

Min: 1 Max: Total Number of Motors * 12

Argument 2: Electrical Angle

Type: Unsigned 16-bit

Min: 0 Max: 511

Where:

cc = Input Value

1: Hall Transition 5-1 for motor channel 1

2: Hall Transition 3-2 for motor channel 1

3: Hall Transition 1-3 for motor channel 1

4: Hall Transition 6-4 for motor channel 1

5: Hall Transition 4-5 for motor channel 1

6: Hall Transition 2-6 for motor channel 1

7: Hall Transition 3-1 for motor channel 1

8: Hall Transition 6-2 for motor channel 1

9: Hall Transition 2-3 for motor channel 1

10: Hall Transition 5-4 for motor channel 1

11: Hall Transition 1-5 for motor channel 1

12: Hall Transition 4-6 for motor channel 1

13: Hall Transition 5-1 for motor channel 2

...

24: Hall Transition 4-6 for motor channel 2 nn = Electrical Angle

Example:

^HSAT 1 452: Set the electrical angle 452 for the hall transition 5-1 for motor channel 1.

HSM - Hall Sensor Map

HexCode: A3 CANOpen id: 0x30A3

Description:

Configure this parameter to match the ABC hall sensor cable pattern with the UVW motor windings wire pattern connected to the controller. For each hall sensor cable order and motor wire order, there are 6 combinations, one of which will make the motor spin smoothly and efficiently in both directions. Try each of the 6 available values of HSM (0-5) and retain the one that will make the motor spin in both directions while drawing the same low current. Applicable only in Hall Trapezoidal mode.

Syntax Serial: ^HSM cc nn

~ HSM [cc]

Syntax Scripting: setconfig(_HSM, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Hall Sensor Map

Type: Unsigned 8-bit

Min: 0 Max: 5

Default: 0

Where:

cc = Motor channel

nn = Motor's Hall Sensor Map

Example:

^HSM 1 1: Set Hall Sensor Map for motor 1 to value 1.

KIF - Current PID Integral Gain

HexCode: 8E CANOpen id: 0x308E

Description:

Sets the Current PID's Integral Gain. The value is set as the gain multiplied by 10^4. On brushless motor controller operating in sinusoidal mode or ACIM motor controllers, two gains can be set for each motor channel, in order to control the Flux and Torque current.

On DC brushed controllers or in brushless motor controllers when operating in trapezoidal mode the gains for the Torque current are used only.

Syntax Serial: ^KIF cc nn
~KIF [cc]

Syntax Scripting: setconfig(_KIF, cc)

Number of Arguments:

Argument 1: AmpsChannel
Min: 1 Max: 2 * Total Number of Motors

Argument 2: Gain Type: Unsigned 32-bit

Min: 0 Max: 2,000,000,000 Default: 0

Where:
cc (single channel) =
1: Flux Integral Gain
2: Torque Integral Gain
cc (dual channel) =
1: Flux Integral Gain for motor 1
2: Flux Integral Gain for motor 2
3: Torque Integral Gain for motor 1
4: Torque Integral Gain for motor 2
nn = Gain * 10,000

Example:

^KIF 1 2300: Set motor channel 1 Flux Integral Gain to 0.23.

KPF - Current PID Proportional Gain

HexCode: 8D CANOpen id: 0x308D

Description:

Sets the Current PID's Proportional Gain. The value is set as the gain multiplied by 10^4. On brushless motor controller operating in sinusoidal mode, two gains can be set for each motor channel, in order to control the Flux and Torque current. On DC brushed controllers or in brushless motor controllers when operating in trapezoidal mode the gains for the Torque current are used only.

Syntax Serial: ^KPF cc nn
~KPF [cc]

Syntax Scripting: setconfig(_KPF, cc)

Number of Arguments:

Argument 1: AmpsChannel
Min: 1 Max: 2 * Total Number of Motors

Argument 2: Gain Type: Unsigned 32-bit

Min: 0 Max: 2,000,000,000 Default: 0

Where:

cc (single channel) =
1: Flux Proportional Gain
2: Torque Proportional Gain
cc (dual channel) =
1: Flux Proportional Gain for motor 1
2: Flux Proportional Gain for motor 2
3: Torque Proportional Gain for motor 1
4: Torque Proportional Gain for motor 2
nn = Gain * 10,000

Example:

^KPF 4 2300: Set motor channel 2 Torque Proportional Gain to 0.23.

LD - Motor d-axis Inductance

HexCode: EC CANOpen id: 0x30EC

Description:

Set the d-axis motor inductance. This configuration command is necessary for IPM motor operation, decoupling control and field weakening feedforward function If value is set to 0 then the motor operates as brushless dc motor with Id reference command set, by default, to 0.

Syntax Serial: ^LD cc nn
~LD [cc]

Syntax Scripting: setconfig(_LD, cc, nn)

Number of Arguments: 2

Argument 1: Channel Min: 1 Max: Total Number of Motors

Argument 2: D-axis Motor Inductance

Type: Unsigned 16-bit

Min: 0 Max: 65535

Default: 0

Where:

cc = Motor channel

nn = D-axis motor inductance in Henry * 1,000,000

Example:

^LD 1 750 : Set the d-axis motor inductance to 0,000750 H = 750 uH

LQ - Motor q-axis Inductance

HexCode: ED CANOpen id: 0x30ED

Description:

Set the q-axis motor inductance. This configuration command is necessary for IPM motor operation, decoupling control and field weakening feedforward function. If value is set to 0 then the motor operates as brushless dc motor with Id reference command set, by default, to 0.

Syntax Serial: ^LQ cc nn

~LQ [cc]

Syntax Scripting: setconfig(_LQ, cc, nn)

Number of Arguments: 2

Argument 1: Channel Min: 1 Max: Total Number of Motors

Argument 2: Q-axis Motor Inductance

Type: Unsigned 16-bit

Min: 0 Max: 65535

Default: 0

Where:

cc = Motor channel

nn = Q-axis motor inductance in Henry * 1,000,000

Example:

^LQ 1 950 : Set the q-axis motor inductance to 0,000950 H = 950 uH

MXPW - Maximum Motor Output Power at Constant Power

Hex Code: F6 CANOpen id: 0x30F6

Description:

This parameter defines the maximum allowed motor output power (P[W] = speed [rad/s]* torque[Nm]) during the motor's operation, covering both constant torque and constant power regions. The motor drive will regulate the motor current based on the configured torque constant (Kt [Nm/A rms]), maximum power parameters, and motor angular speed (ω[rad/s]) as follows:

I_max [A rms] = P_max [W] / ω [rad/s] * Kt [Nm/A rms]

The value should be set according to the datasheet parameter. If the maximum power is not directly provided, it can be derived from the motor's torque-speed curve, by mutiply-ing the maximum torque with the maximum speed at the maximum power point.

Syntax Serial: ^MXPW cc nn

~MXPW[cc]

Syntax Scripting: setconfig(_MXPW, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Maximum Motor Output Power

Type: Unsigned 32-bit

Min: 0 Max: Dependent on each controller power ratings

Default: Dependent on each controller power ratings

where:

cc=Motor channel

nn=Maximum Motor Output Power (W) * 10

Example:

^MXPW 1 15000: Configure the maximum motor output power at 1500 W.

PSA - Phase Shift Angle

HexCode: E1 CANOpen id: 0x30E1

Description:

When being in sinusoidal mode and Sin/Cos or Resolver feedback sensors are used, this configuration command defines the Phase Shift Angle between the Sin and Cos signals. The value is in degrees ranging from 0 to 511. If the sensor is a regular Sin/Cos or Resolver Sin/Cos sensor, the phase shift angle is 90° degrees, so we need to configure 90*512/360 = 128. If not then the sensor manufacturer should inform about the phase shift angle.

Syntax Serial: ^PSA cc nn

~PSA [cc]

Syntax Scripting: setconfig(_PSA, cc, nn)

Number of Arguments:

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Angle

Type: Signed 16-bit

Min: -511 Max: 511

Default: 128

Where:

cc = Motor channel
nn = Angle

Example:

^PSA 1 171 : Set the phase shift angle to 171*360/512 = 120^o degrees.

RS - Motor Stator Resistance

HexCode: EB CANOpen id: 0x30EB

Description:

Set the motor phase resistance. This value is used only in motor characterization Roborun+ utility tool for FOC gains calculation.

Syntax Serial: ^RS cc nn

~RS [cc]

Syntax Scripting: setconfig(_RS, cc, nn)

Number of Arguments: 2

Argument 1: Channel Min: 1 Max: Total Number of Motors

Argument 2: Motor phase resistance

Type: Unsigned 16-bit

Min: 0 Max: 65535

Default: 0

Where:

cc = Motor channel

nn = Motor phase resistance in Ohm * 1,000

Example:

^RS 1 500 : Set the motor phase resistance to 0,5 Ohm = 500 mOhm

SPOL - SinCos/SSI Sensor Pole Pairs

HexCode: 41 CANOpen id: 0x3041

Description:

Number of pole pairs of the Sin/Cos, Resolver or SSI angle sensor used in sinusoidal mode with brushless motors.

Syntax Serial: ^SPOL cc nn
~SPOL [cc]

Syntax Scripting: setconfig(_SPOL, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Number

Type: Unsigned 8-bit
Min: 1 Max: 255
Default: 1

Where:

cc = Motor channel
nn = Number of pole pairs

SWD - Swap Windings

HexCode: A4 CANOpen id: 0x30A4

Description:

The concept of swap windings (can also be termed as sensor polarity) is a relative relationship between sensor direction and stator magnetic field rotational direction. During calibration if angle, reported by the sensor, is changing in same direction as angle commanded to the stator then Swap Windings should be "None", else "Swapped".

In case of Hall + Encoder, two different Swap Windings are needed, one for hall and one for encoder. So, instead we use 2 bits of same variable Bit 1 for Hall and Bit 0 for Encoder. The following table shows the truth table for swap windings

TABLE 15-40. SWD values

SWD Value	BIT 1	BIT 0	Swap for Hall (Bit 1)	Swap for Encoder (Bit 0)	Meaning
0	0	0	NO	NO	None
1	0	1	NO	YES	Swapped
2	1	0	YES	NO	Hall only Swapped
3	1	1	YES	YES	Hall+Encoder Swapped

Syntax Serial: ^SWD cc nn

~ SWD [cc]

Syntax Scripting: setconfig(_SWD, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Swap Windings

Type: Unsigned 8-bit

Min: 0 Max: 3

Default: 0

Where:

cc = Motor channel

nn = Motor's Swap Windings

0: None, Angle up-counting for clockwise direction

1: Swapped, Angle down-counting for clockwise direction

2:Hall only Swapped, Angle up-counting for clockwise direction for encoder and Angle down-counting for clockwise direction for Hall sensor.

3:Hall+Encoder Swapped, Angle down-counting for clockwise direction for encoder and Angle down-counting for clockwise direction for Hall sensor.

Example:

^SWD 1 1: Set angle down-counting for clockwise direction for motor 1.

TID - FOC Target Id

HexCode: 8F CANOpen id: 0x308F

Description:

In brushless motors operating in sinusoidal mode, this command sets the desired Flux (also known as Direct Current Id) for Field Oriented Control. This value must be 0 in typical application. A non-zero value creates field weakening and can be used to achieve higher rotation speed.

Syntax Serial: ^TID cc nn
~TID [cc]

Syntax Scripting: setconfig(_TID, cc, nn)

Number of Arguments: 2

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Amps
Type: Signed 32-bit
Min: 0
Default: 0

Where:

cc = Motor Channel
nn = Amps * 10

VK - Motor Voltage constant

HexCode: EE CANOpen id: 0x30EE

Description:

Set the motor voltage constant. This parameter is necessary for IPM motor operation, decoupling control and field weakening feedforward function. This constant considers the peak amplitude of induced phase to phase motor back-emf per 1000 rpm mechanical speed. It is typically included in motor manufacturer's datasheet.

Syntax Serial: ^VK cc nn
~VK [cc]

Syntax Scripting: setconfig(_VK, cc, nn)

Number of Arguments: 2

Argument 1: Channel Min: 1 Max: Total Number of Motors

Argument 2: Motor voltage constant

Type: Unsigned 32-bit

Min: 0 Max: 2000000

Default: 0

Where:

cc = Motor channel

nn = Motor voltage constant in V/krpm * 1000

Example:

^VK 1 14000 : Set the motor voltage constant to 14 V/krpm

ZSMA - Cos Amplitude

HexCode: E5 CANOpen id: 0x30E5

Description:

This parameter contains the amplitude of the Cosine signal (SinCos sensor), which along with the values of ZSMC is used in order to calculate the signal quality (see SEC - Read Sensor Errors).

Syntax Serial: ^ZSMA cc nn

~ZSMA [cc]

Syntax Scripting: setconfig(_ZSMA, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Value

Type: Signed 16-bit

Where:

cc = Channel

nn = Calibration Value

Example:

^ZSMA 2 1800: Set the amplitude of the Cos signal of the SinCos sensor of motor 2 to 1800.

ZSMC - SinCos Calibration

HexCode: 46 CANOpen id: 0x3046

Description:

This parameter contains Sin/Cos calibration values that are captured after the execution of the Motor/Sensor Setup. Values are not to be altered manually. When non-zero values are returned after querying ZSMC, this indicates that the setup has been successfuly completed at one time or another.

Syntax Serial: ^ZSMC cc nn
~ZSMC [cc]

Syntax Scripting: setconfig(_ZSMC, cc)

Number of Arguments:

Argument 1: InputNbr
Min: 1 Max: 6

Argument 2: Value
Type: Signed 16-bit

Where:

cc =
1: Sin Zero Point for motor 1
2: Cos Zero Point for motor 1
3: SinCos Ratio for motor 1
4: Sin Zero Point for motor 2
5: Cos Zero Point for motor 2
6: SinCos Ratio for motor 2

nn = Calibration value

AC Induction Specific Commands

TABLE 15-41. AC Induction Specific Commands

Command	Arguments	Description
BFBK	Channel Mode	AC Induction Operating Mode
ILM	Inductance	Mutual Inductance
ILLR	Inductance	Rotor Leakage Inductance
IRR	Resistance	Rotor Resistance
MPW	MotorPower	Minimum Power
MXS	SlipFrequency	Optimal Slip Frequency
RFC	Channel Amps	Rotor Flux Current
VPH	Channel Ratio	AC Induction Volts per Hertz

BFBK - AC Induction Operating Mode

HexCode: 63 CANOpen id: 0x3063

Description:

For AC Induction motors this parameter selects the operating mode.

Syntax Serial: ^BFBK cc nn
~BFBK [cc]

Syntax Scripting: setconfig(_BFBK, cc)

Number of Arguments:

Argument 1: Channel
Min: 1 Max: Total Number of Motors

Argument 2: Sensor

Type: Unsigned 8-bit

Min: 0 Max: 4

Default: 0 = Volts Per Hertz

Where:

cc = Motor channel

nn =

0: Volts Per Hertz

1: Constant Slip Speed

2: FOC Torque

3: FOC Speed

ILM - Mutual Inductance

HexCode: 9B CANOpen id: 0x309B

Description:

This parameter is only used for AC Induction controllers when operating in FOC mode and contains motor's mutual inductance (coupled to both stator and rotor).

Syntax Serial: ^ILM cc nn

~ ILM [cc]

Syntax Scripting: setconfig(_ILM, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Mutual Inductance

Type: Unsigned 32-bit

Min: 0 Max: 10000

Default: 10

Where:

cc = Motor channel

nn = Motor's Mutual Inductance in μH.

Example:

^ILM 1 961: Set Mutual Inductance of motor 1 to value 961μH.

ILLR - Rotor Leakage Inductance

HexCode: 9A CANOpen id: 0x309A

Description:

This parameter is only used for AC Induction controllers when operating in FOC mode and contains the rotor's per phase leakage inductance of the motor. This value can be obtained from the motor supplier.

Syntax Serial: ^ILLR cc nn

~ ILLR [cc]

Syntax Scripting: setconfig(_ILLR, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Rotor Leakage Inductance

Type: Unsigned 32-bit

Min: 0 Max: 10000

Default: 10

Where:

cc = Motor channel

nn = Motor's Rotor Leakage Inductance in µH.

Example:

^RFC 1 67: Set Rotor Leakage Inductance of motor 1 to value 67µH.

IRR - Rotor Resistance

HexCode: 99 CANOpen id: 0x3099

Description:

This parameter is only used for AC Induction controller when operating in FOC mode and contains the resistance per phase of the rotor. This value can be obtained from the motor supplier.

Syntax Serial: ^IRR cc nn

~ IRR [cc]

Syntax Scripting: setconfig(_IRR, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Rotor Resistance

Type: Unsigned 32-bit

Min: 1 Max: 500000

Default: 20000

Where:

cc = Motor channel

nn = Motor's Rotor Resistance in micro-ohm.

Example:

^IRR 1 24500: Set Rotor Resistance of motor 1 to value 24500µΩ.

MPW - Minimum Power

HexCode: 97 CANOpen id: 0x3097

Description:

This parameter is only used for AC Induction controllers when operating in Volts per Hertz mode. It defines a minimum PWM output value so that there is always a minimal of rotor flux to create induction.

Syntax Serial: ^MPW cc nn

~MPW [cc]

Syntax Scripting: setconfig(_MPW, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Minimum Power

Type: Unsigned 16-bit

Min: 0 Max: 1000

Default: 100

Where:

cc = Motor channel

nn = Motor's Minimum Power in % of PWM Level

Example:

^MPW 1 200: Set Minimum Power for motor 1 to value 20.0% PWM Level.

MXS - Optimal Slip Frequency

HexCode: 96 CANOpen id: 0x3096

Description:

This parameter is only used for AC Induction controllers. The optimal slip is the value that the controller will work to maintain while operating in Constant Slip mode.

Syntax Serial: ^MXS cc nn

~MXS [cc]

Syntax Scripting: setconfig(_MXS, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Optimal Slip Frequency

Type: Unsigned 16-bit

Min: 0 Max: 65535

Default: 50

Where:

cc = Motor channel

nn = Motor's Optimal Slip Frequency in Hertz * 10

Example:

^MXS 1 60: Set Optimal Slip for motor 1 to value 6.0Hz

RFC - Rotor Flux Current

HexCode: 98 CANOpen id: 0x3098

Description:

This parameter is only used for AC Induction controller. This value is the stator current component (Id) for rotor flux production when FOC modes are selected.

Syntax Serial: ^RFC cc nn

~ RFC [cc]

Syntax Scripting: setconfig(_RFC, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: Rotor Flux Current

Type: Unsigned 16-bit

Min: 0 Max: Max Amps in datasheet

Default: 10

Where:

cc = Motor channel

nn = Motor's Rotor Flux Current in Amps * 10

Example:

^RFC 1 50: Set Rotor Flux Current for motor 1 to value 5A.

VPH - AC Induction Volts per Hertz

HexCode: 95 CANOpen id: 0x3095

Description:

This parameter is only used for AC Induction controllers. Each motor has as specification a Volts per Hertz value with which the frequency can be determined when specific voltage is applied.

Syntax Serial: ^VPH cc nn

~VPH [cc]

Syntax Scripting: setconfig(_VPH, cc, nn)

Number of Arguments: 2

Argument 1: Channel

Min: 1 Max: Total Number of Motors

Argument 2: VoltsPerHz

Type: Unsigned 16-bit

Min: 0 Max: 65535

Default: 20000

Where:

cc = Motor channel

nn = Motor's Volts per Hertz * 1000

Example:

^VPH 1 200: Set Volts per Hertx to value 0.200

CAN/EtherCAT Communication Commands

This section describes all the configuration parameters uses for CANbus operation.

TABLE 15-42. CANbus Commands

Command	Arguments	Description
CAS	Rate	CANOpen Auto start
CBR	BitRate	CAN Bit Rate
CEN	Mode	CAN Mode
CGT	Time	CANOpen Guard Time
CHB	HeartBeat	CAN Heartbeat
CHBT	Time	Consumer Heartbeat Time
CHLA	Action	CAN Consumer Heartbeat Lost Action
CLSN	Address	CAN Listen Node ID
CNOD	Address	CAN Node ID
CSRT	Rate	MiniCAN SendRate
CTPS	TPDOnbr Rate	CANOpen TPDO SendRate
CTT	None	CANOpen Transmission Type
ECAT	None	EtherCAT Enable Mode
ECT	None	EtherCAT Cycle Time
EDID	None	EtherCAT Explicit Device ID
FSA	None	DS402 FSA
RPDC	RPDOnbr Rate	CANOpen RPDO COB-ID
RPDM	RPDO Item	CANOpen RPDO Mapping
TPDC	TPDOnbr Rate	CANOpen TPDO COB-ID
TPDM	TPDO Item	CANOpen TPDO Mapping

CAS - CANOpen Auto start

HexCode: 5A CANOpen id: 0x305A

Description:

Determines if device is an active CANOpen node at power up. When set, node is in operational state at power up without the need to receive a start command.

Syntax Serial: ^CAS nn
~CAS

Syntax Scripting: setconfig(_CAS, nn)

Number of Arguments: 1

Argument 1: Rate
Type: Unsigned 8-bit
Min: 0 Max: 1
Default: 0 = Off

Where:
nn =
0: Device is in pre-operational state at power-up.
1: Device is in operational state at power up.

CBR - CAN Bit Rate

HexCode: 58 CANOpen id: 0x3058

Description:

Sets the CAN bus bit rate.

Syntax Serial: ^CBR nn
~CBR

Syntax Scripting: setconfig(_CBR, nn)

Number of Arguments: 1

Argument 1: BitRate
Type: Unsigned 8-bit
Min: 0 Max: 5
Default: 3 = 250K

Where:

nn =
0: 1000K
1: 800K
2: 500K
3: 250K
4: 125K

CEN - CAN Mode

HexCode: 56 CANOpen id: 0x3056

Description:

Enables CAN and selects the CAN protocol.

Syntax Serial: ^CEN nn
~CEN

Syntax Scripting: setconfig(_CEN, nn)

Number of Arguments: 1

Argument 1: Mode
Type: Unsigned 8-bit
Min: 0 Max: 5

Where:

nn =
0: Disabled
1: CANOpen
2: MiniCAN
3: RawCAN
4: RoboCAN
5: MiniJ1939

CGT – CANOpen Guard Time

HexCode: 9D CANOpen Object: 100Ch, 100Dh, 0x309D

Description:

Configures the guard time and life time factor for CANOpen node guarding protocol.

Syntax Serial ^CGT opt val
~CGT [opt]

Syntax Scripting SetConfig(_CGT, opt, val)
val = GetConfig(opt)

Arguments 2

Argument 1: Option [1: guard time, 2: life time factor]

Argument 2: Value

Type: Unsigned 16-bit

Min: 0

Max: 65535 for guard time, 255 for life time factor

Example:

^CGT 1 1000_^CGT 2 5. Set guard time to 5000 ms (5 * 1000).

CHB - CAN Heartbeat

HexCode: 59 CANOpen id: 0x3059

Description:

Sets the rate in milliseconds at which the controller will send a heartbeat frame on the CAN bus. Heartbeat is sent when either MiniCAN, RawCAN, CANOpen are selected. A dedicated, non-user-alterable Heartbeat frame is sent when RoboCAN is selected.

Syntax Serial: ^CHB nn

Syntax Scripting: setconfig(_CHB, nn)

Number of Arguments: 1

Argument 1: HeartBeat

Type: Unsigned 16-bit
Min: 0 Max: 65536
Default: 100ms

Where:

nn = Heartbeat rate in ms

CHBT – Consumer Heartbeat Time

HexCode: 9E CANOpen Object: 1016h, 0x309E

Description:

Configures consumer heartbeat time. If the heartbeat time is 0 or the node-ID is 0 or greater than 127 the corresponding object is not considered. The heartbeat time shall be given in multiples of 1ms.

An attempt to configure several heartbeat times unequal 0 for the same node-ID the controller will respond with SDO abort code.

Syntax Serial ^CHBT cc nn
~CHBT [cc]

Syntax Scripting SetConfig(_CHBT, cc, nn)

Number of Arguments: 2

Argument 1: Index

Type: Unsigned 8-bit

Min: 1 Max: 4, as controller supports monitoring up to 4 nodes.

Argument 2: Value

Type: Unsigned 32-bit

Min: 0 Max: 0x7FFFFF

Where:

cc=Monitoring Node

nn=Node in most significant 4 bytes and Hearbeat Time in less significant 4 bytes

Example:

^CHBT 1 133072 Monitor node 2 with heartbeat time of 2000ms (0x07D0). 133072d = 0207D0h.

CHLA - CAN Consumer Heartbeat Lost Action

HexCode: EF CANOpen id: 0x30EF

Description:

This configuration is used in order to configure the action to be applied on motors once the consumer heartbeat gets lost. If this happens then the command watchdog will expire. Apart from that it can be configured whether Quick stop or emergency will be applied additionally to the motor.

Syntax Serial: ^CHLA nn

~CHLA

Syntax Scripting: setconfig(_CHLA, nn)

Number of Arguments: 1

Argument 1: Action

Type: Unsigned 8-bit

Min: 0 Max: 2

Default: 0 = No Action

Where:

nn =

0: No Action

1: Quick Stop

2: Emergency Stop

CLSN - CAN Listen Node ID

HexCode: 5B CANOpen id: 0x305B

Description:

In RawCAN and MiniCAN mode, this parameter filters the incoming frames in order to capture only these originating from a given node address. In RawCAN, entering 0 disables the filter and will cause all incoming frames to be captured.

Syntax Serial: ^CLSN nn
~CSLN

Syntax Scripting: setconfig(_CLSN, nn)

Number of Arguments: 1

Argument 1: Address

Type: Unsigned 8-bit	
Min: 0	Max: 127
Default: Product dependent	

Where:

nn =
0: Listen to all nodes (RawCAN only)
1-127: Capture frames from specific node id only

CNOD - CAN Node ID

HexCode: 57 CANOpen id: 0x3057

Description:

Stores the product's ID on the CAN bus.

Syntax Serial: ^CNOD nn
~CNOD

Syntax Scripting: setconfig(_CNOD, nn)

Number of Arguments: 1

Argument 1: Address

Type: Unsigned 8-bit	
Min: 0	Max: 127
Default: See datasheet	

Where:

nn = Node address

CSRT - MiniCAN SendRate

HexCode: 5C CANOpen id: 0x305C

Description:

Rate, in ms, at which MiniCAN frames are sent.

Syntax Serial: ^CSRT nn
~CSRT

Syntax Scripting: setconfig(_CSRT, nn)

Number of Arguments: 1

Argument 1: Rate

	Type: Unsigned 8-bit	
	Min: 0	Max: 65536
	Default: 100ms	

Where:

nn = Rate in ms. No frames sent. if value is 0

CTPS - CANOpen TPDO SendRate

HexCode: 5D CANOpen id: 0x305D

Description:

Sets the send rate for each of the 8 TPDOs when CANOpen is enabled.

Syntax Serial: ^CTPS nn mm

Syntax Scripting: setconfig(_CTPS, nn, mm)

Number of Arguments: 2

Argument 1: TPDOnbr

	Min: 1	Max: 8

Argument 2: Rate

	Type: Unsingned 16-bit	
	Min: 0	Max: 65536
	Default: 0 = Off	

Where:

nn = TPDO number, 1 to 8
mm = Rate in ms

Note:

If mm = 0, the TPDO is not transmitted

CTT – CANOpen Transmission Type

HexCode: 70 CANOpen id: 0x3070

Description:

Sets the transmission type of the respective TPDOs.

Syntax Serial: ^CTT nn mm

Syntax Scripting: setconfig(_CTT, nn, mm)

Number of Arguments: 2

Argument 1: TPDOnbr

Min: 1 Max: 8

Argument 2: Type: Unsigned 8-bit

Min: 0 Max:255

Where:

nn = TPDO number, 1 to 8
mm = Transmission Type

ECAT - EtherCAT Enable Mode

HexCode: F7 CANOpen Object: 0x30F7

Description:

Enables EtherCAT and selects operating mode.

Syntax Serial ^ECAT nn
~ECAT

Syntax Scripting SetConfig(_ECAT, nn)

Number of Arguments: 1

Argument 1: Mode

Type: Unsigned 8-bit

Min: 0 Max: 3

Default: 0

Where:

nn =

0: Disabled

1: Polling

2: Sync Manager

3: Distributed Clock

ECT - EtherCAT Cycle Time

HexCode: F8 CANOpen Object: 0x30F8

Description:

It is applied to the Sync Manager and Distributed Clocks modes and should be used for configuring Sync Manager and DC mode (synchronous with SYNC0 event). For Sync Manager it sets the slave' s cycle time (PDI ISR). For DC mode (synchronous with SYNC0 event) it sets the cycle time and the SYNC0 event cycle time (SYNC0 event). Values are in millisecond (ms).

Syntax Serial ^ECT nn

~ECT

Syntax Scripting SetConfig(_ECT, nn)

Number of Arguments: 1

Argument 1: Time

Type: Unsigned 16-bit

Min: 0 Max: 4294

Default: 4

Where:

nn = Cycle time (ms)

EDID - EtherCAT Explicit Device ID

HexCode: 104 CANOpen Object: 0x3104

Description:

Configures the explicit slave device identification. It is useful for distinguishing identical devices in the same network.

Syntax Serial ^EDID nn

~EDID

Syntax Scripting SetConfig(_EDID, nn)

Number of Arguments: 1

Argument 1: ID

Type: Unsigned 16-bit

Min: 0 Max: 65535

Default: 5

Where:

nn = Explicit Slave Device ID

FSA – DS402 PDS Finite State Automation Enable

HexCode: CC CANOpen id: 0x30CC

Description:

Enables or disables the PDS Finite State Automation (FSA), as dictated in DS402 specification.

Syntax Serial: ^FSA nn

~FSA

Syntax Scripting: setconfig(_FSA, nn)

Number of Arguments: 1

Argument 1: Mode

Type: Unsigned 8-bit
Min: 0 Max:1
Default: 0 = Off

Where:

nn =
0: FSA is inactive.
1: FSA is active.

RPDC - CANOpen RPDO COB-ID

HexCode: 10B CANOpen id: 0x310B

Description:

Sets the COB-ID of the respective RPDOs. If it is 0 then the default value for each RPDO is set:

RPDO1: 0x200 + Node ID

RPDO2: 0x300 + Node ID

RPDO3: 0x400 + Node ID

RPDO4: 0x500 + Node ID

RPDO5-8: disabled

Syntax Serial: ^RPDC nn mm

Syntax Scripting: setconfig(_RPDC, nn, mm)

Number of Arguments: 2

Argument 1: RPDO Item	Type:Unsigned 8-bit	Min: 1 Max:8
Argument 2: COB-ID	Type:Unsigned 32-bit	

Where:

nn = RPDO number, 1 to 8

mm = COB-ID

RPDM - CANOpen RPDO Mapping

HexCode: 9F CANOpen id: 0x309F

Description:

Sets the mapping of the respective RPDOs. Each value represents one mapped item and there are 8 items fr each RPDO. The value holds the index, the subindex and the length of the item.

Syntax Serial: ^RPDM nn mm

Syntax Scripting: setconfig(_RPDM, nn, mm)

Number of Arguments: 2

Argument 1: RPDO Item Type:Unsigned 8-bit Min: 1 Max:64

Argument 2: Mapped Item Type:Unsigned 32-bit

Where:

nn=

1: Mapped item 1 for RPDO 1

2: Mapped item 2 for RPDO 1

...

8: Mapped item 8 for RPDO 1

9: Mapped item 1 for RPDO 2

...

16: Mapped item 8 for RPDO 2

17: Mapped item 1 for RPDO 3

...

64: Mapped item 8 for RPDO 8

mm= (index << 16) + (subindex << 8) + length.

Example: Map Set motor command for channel 2 as first map item of RPDO 2.

Object with index 0x2000, subindex 0x2 and length 4 bytes

mm = (0x2000 << 16) + (0x2 << 8) + 4 = 0x20000204 = 536871428

^RPDM 10 536871428

TPDC - CANOpen TPDO COB-ID

HexCode: 10C CANOpen id: 0x310C

Description:

Sets the COB-ID of the respective TPDOs. If it is 0 then the default value for each RPDO is set:

TPDO1: 0x180 + Node ID

TPDO2: 0x280 + Node ID

TPDO3: 0x380 + Node ID

TPDO4: 0x480 + Node ID

TPDO5-8: disabled

Syntax Serial: ^TPDC nn mm

Syntax Scripting: setconfig(_TPDC, nn, mm)

Number of Arguments: 2

Argument 1: TPDO Item	Type:Unsigned 8-bit	Min: 1	Max:8
Argument 2: COB-ID	Type:Unsigned 32-bit		

TPDM - CANOpen TPDO Mapping

HexCode: A0 CANOpen id: 0x30A0

Description:

Sets the mapping of the respective TPDOs. Each value represents one mapped item and there are 8 items fr each TPDO. The value holds the index, the subindex and the length of the item.

Syntax Serial: ^TPDM nn mm

Syntax Scripting: setconfig(_TPDM, nn, mm)

Number of Arguments: 2

Argument 1: TPDO Item	Type:Unsigned 8-bit	Min: 1	Max:64
Argument 2: Mapped Item	Type:Unsigned 32-bit		

Where:

nn=

1: Mapped item 1 for TPDO 1

2: Mapped item 2 for TPDO 1

...

8: Mapped item 8 for TPDO 1

9: Mapped item 1 for TPDO 2

...

16: Mapped item 8 for TPDO 2

17: Mapped item 1 for TPDO 3

...

64: Mapped item 8 for TPDO 8

mm= (index << 16) + (subindex << 8) + length.

Example: Map Read Encoder Motor Speed for channel 1 as second map item of TPDO 1.

Object with index 0x2103, subindex 0x1 and length 4 bytes

mm = (0x2103 << 16) + (0x1 << 8) + 4 = 0x21030104 = 553844996

^TPDM 2 553844996

TCP Communication Commands

This section describes all the configuration parameters uses for TCP operation.

TABLE 15-43. TCP Commands

Command	Arguments	Description
DHCP	Enable	Enable DHCP
GWA	IP Octet	Gateway Address
IPA	IP Octet	IP Address
IPP	None	IP Port
PDNS	IP Octet	Primary DNS
SBM	IP Octet	Subnet Mask
SDNS	IP Octet	Secondary DNS
WMOD	Mode	TCP Mode

DHCP - Enable DHCP

HexCode: 6F CANOpen id: 0x306F

Description:

Configure this parameter in order to enable the DHCP. The default value for DHCP service is disabled. When DHCP is Disabled the user configured IP address is used by the controller to access the network. By enabling DHCP service, the controller uses the IP address provided by the DHCP server.

Syntax Serial: ^DHCP nn

~ DHCP

Syntax Scripting: setconfig(_DHCP, nn)

Number of Arguments: 1

Argument 1: Enable DHCP

Type: Unsigned 8-bit
Min: 0 Max: 1
Default: 0

Where:

nn = Enable DHCP
0: Disabled.
1: Enabled.

Example:

^DHCP 1: Enable DHCP.

GWA - Gateway Address

HexCode: 69 CANOpen id: 0x3069

Description:

Configure this parameter in order to set the Gateway Address of your controller's network. Gateway Address option includes 4 values representing each octet in the IP address v4 format. The default Gateway Address value is 192.168.1.1.

Syntax Serial: ^GWA cc nn

~GWA

Syntax Scripting: setconfig(_GWA, cc, nn)

Number of Arguments: 2

Argument 1: IP Octet
Type: Unsigned 8-bit
Min: 1 Max: 4

Argument 2: Gateway Address

Type: Unsigned 8-bit
Min: 0 Max: 255
Default:

TABLE 15-44. GWA default values

Octet	Value
1	192
2	168
3	1
4	1

Where:

cc = octet
nn = octet value

Example:

^GWA 1 192_^GWA 2 168_^GWA 3 2_^GWA 4 1: Set Gateway Address 192.168.2.1.

IPA - IP Address

HexCode: 68 CANOpen id: 0x3068

Description:

Configure this parameter in order to set the IP Address. IP Address option includes 4 values representing each octet in the IP address v4 format. The default IP address, if DHCP is disabled, is 192.168.1.20.

Syntax Serial: ^IPA cc nn

~IPA

Syntax Scripting: setconfig(_IPA, cc, nn)

Number of Arguments: 2

Argument 1: IP Octet

Type: Unsigned 8-bit
Min: 1 Max: 4

Argument 2: IP Address

Type: Unsigned 8-bit
Min: 0 Max: 255
Default:

TABLE 15-45. IPA default values

Octet	Value
1	192
2	168
3	1
4	20

Where:

cc = octet
nn = octet value

Example:

^IPA 1 192_^IPA 2 168_^IPA 3 1_^IPA 4 100: Set IP Address 192.168.1.100.

IPP - IP Port

HexCode: 6B CANOpen id: 0x306B

Description:

Configure this parameter in order to set the IP Port. Default IP Port value is 9761. The IP address combined with the IP Port value are used to connect to the controller.

Syntax Serial: ^IPP nn

~IPP

Syntax Scripting: setconfig(_IPP, nn)

Number of Arguments: 1

Argument 1: IP Port
Type: Unsigned 16-bit
Min: 0 Max: 65535
Default: 9761

Where:

nn = IP Port

Example:

^IPP 1300: Set IP Port 1300.

PDNS - Primary DNS

HexCode: 6D CANOpen id: 0x306D

Description:
Configure this parameter in order to set the address of the primary DNS server. Primary DNS option includes 4 values representing each octet in the IP address v4 format. Primary DNS server default address is 192.168.1.1.

Syntax Serial: ^PDNS cc nn

~PDNS

Syntax Scripting: setconfig(_PDNS, cc, nn)

Number of Arguments: 2

Argument 1: IP Octet

Type: Unsigned 8-bit
Min: 1 Max: 4

Argument 2: Primary DNS

Type: Unsigned 8-bit
Min: 0 Max: 255
Default:

TABLE 15-46. PDNS default values

Octet	Value
1	192
2	168
3	1
4	1

Where:

cc = octet
nn = octet value

Example:

^PDNS 1 192_^PDNS 2 168_^PDNS 3 2_^PDNS 4 1: Set Primary DNS 192.168.2.1.

SBM - Subnet Mask

HexCode: 6A CANOpen id: 0x306A

Description:

Configure this parameter to set the Subnet Mask to define the range of IP addresses that can be used in your network. Subnet Mask option includes 4 values representing each octet in the IP address v4 format. Devices within the same sub-network can communicate directly. Using the default subnet mask, all devices with the first 3 bytes identical are located in the same sub-network and almost 256 (not all 256 values may be used as a host IP) unique host devices may be used in the network. Subnet Mask option includes 4 values representing each octet in the IP address. The default Subnet Mask value is 255.255.255.0.

Syntax Serial: ^SBM cc nn

~SBM

Syntax Scripting: setconfig(_SBM, cc, nn)

Number of Arguments: 2

Argument 1: IP Octet
Type: Unsigned 8-bit
Min: 1 Max: 4

Argument 2: Subnet Mask

Type: Unsigned 8-bit
Min: 0 Max: 255
Default:

TABLE 15-47. SBM default values

Octet	Value
1	255
2	255
3	255
4	0

Where:

cc = octet
nn = octet value

Example:

^SBM 1 255_^SBM 2 255_^SBM 3 254_^SBM 4 0: Set Gateway Address 255.255.254.0.

SDNS - Primary DNS

HexCode: 6E CANOpen id: 0x306E

Description:

Configure this parameter to set the address of the secondary DNS server. Secondary DNS option includes 4 values representing each octet in the IP address v4 format. Secondary DNS server default address is 0.0.0.0. By setting the secondary DNS server to 0.0.0.0 then automatically a secondary DNS server address is assigned. Since the secondary server address is a backup, there is no need to be configured, unless necessary.

Syntax Serial: ^SDNS cc nn
~SDNS

Syntax Scripting: setconfig(_SDNS, cc, nn)

Number of Arguments: 2

Argument 1: IP Octet
Type: Unsigned 8-bit
Min: 1 Max: 4

Argument 2: Primary DNS
Type: Unsigned 8-bit
Min: 0 Max: 255
Default:

TABLE 15-48. SDNS default values

Octet	Value
1	192
2	168
3	1
4	1

Where:

cc = octet
nn = octet value

Example:

^SDNS 1 192_^SDNS 2 168_^SDNS 3 2_^SDNS 4 1: Set Secondary DNS 192.168.2.1.

WMOD - TCP Mode

HexCode: 67 CANOpen id: 0x3067

Description:

Configure this parameter to enable the TCP functionality. When the TCP mode is set as Disabled the Ethernet port is idle and no data packets are being transmitted or received. To communicate via TCP/IP this parameter must be set to Enabled. For communicating via Modbus TCP or Modbus TCP over RTU, TCP Mode must be set to Enabled.

Syntax Serial: ^WMOD nn

~WMOD

Syntax Scripting: setconfig(_WMOD, nn)

Number of Arguments: 1

Argument 1: TCP Mode
Type: Unsigned 8-bit
Min: 0 Max: 1
Default: 0

Where:

nn = TCP Mode
0: Disabled.
1: Enabled.

Example:

^WMOD 1: Enable TCP functionality.