BASICMICRO
MOTION CONTROL

RoboClaw Series Brushed DC Motor Controllers

RoboClaw OEM
RoboClaw Solo
RoboClaw 2x7A
RoboClaw 2x15A
RoboClaw 2x30A
RoboClaw 2x45A
RoboClaw 2x45A ST
RoboClaw 2x60A
Roboclaw 2x60HV
Roboclaw 2x200A
Roboclaw 2x300A

User Manual

Firmware 4.3.0 and Newer
User Manual Revision 5.9

Contents

Warnings

Proper installation and operation of the RoboClaw motor controller is critical for both device safety and operator safety. Incorrect wiring can result in permanent damage to the controller and connected equipment. Furthermore, any system involving mechanical movement presents inherent safety risks that require careful consideration of emergency scenarios. Please note the following warnings!

Disconnecting the negative terminal while I/O devices or a USB cable are connected can create ground loops and damage the motor controller or attached devices. Always disconnect the positive lead first when powering down.

"Brushed DC motors act as generators when in motion. If a robot is pushed or allowed to coast, the motors can generate sufficient voltage to intermittently power RoboClaw's logic circuits, creating an unsafe operating state.

RoboClaw requires a stable minimum voltage to operate. Under heavy loads, voltage drops may cause brownouts or erratic behavior. In these situations a separate logic battery is recommended for reliable operation.

Never reverse battery polarity. Doing so will permanently damage the motor controller.

Never disconnect the motors while they are running. Doing so can generate voltage spikes that may damage the motor controller or other connected electronics.

Control Overview

I/O

RoboClaw is compatible with both 5V and 3.3V logic systems. All inputs are protected by internal current limiting and voltage clipping circuits that safely handle signals up to 5V. RoboClaw outputs a 3.3V logic level that is compatible with both 5V and 3.3V systems. This design ensures reliable operation while protecting the I/O circuitry from damage.

Headers

RoboClaw's share the same header and screw terminal pinouts accross models in this user manual. The main control I/O are arranged for easy connectivity to control devices such as R/C controllers. The headers are also arranged to provide easy access to ground and power for supplying power to external controllers. see the specific model of RoboClaw's data sheet for pinout details.

Control Inputs

S1, S2, S3, S4 and S5 are setup for standard servo style headers I/O(except on ST models), +5V and GND. S1 and S2 are the control inputs for serial, analog and RC modes. S3 can be used as a flip switch input when in RC or Analog modes. In serial mode S3, S4 and S5 can be used as emergency stop inputs or as voltage clamp control outputs. When set as E-Stop inputs they are active when pulled low and have internal pullups so they will not accidentally trip when left floating. S4 and S5 can also optionally be used as home signal inputs. The pins closest to the board edge are the I/0s, center pin is the +5V and the inside pins are ground. Some RC receivers have their own supply and will conflict with the RoboClaw's 5v logic supply. It may be necessary to remove the +5V pin from the RC receivers cable in those cases.

Encoder Inputs

EN1 and EN2 are the inputs from the encoders on pin header versions of RoboClaw. 1B, 1A, 2B and 2A are the encoders inputs on screw terminal versions of RoboClaw.

When connecting the encoder make sure the leading channel for the direction of rotation is connected to A. If one encoder is backwards to the other you will have one internal counter counting up and the other counting down. Refer to the data sheet of the encoder you are using for channel direction. Which encoder is used on which motor can be swapped via a software setting.

Logic Battery (LB IN)

The LB-IN input is an optional input intended to provide backup power to the logic section in systems where the main battery may be subject to heavy load conditions. During continuous full-load operation or rapid direction changes, voltage drops may occur on the main battery supply. If the logic section is powered solely from the main battery, these voltage drops can result in logic brownouts. Supplying a secondary logic battery through the LB-IN input prevents brownouts and ensures stable logic operation. Refer to the specific motor controller datasheet for the minimum and maximum supported logic battery voltage ranges.

Encoder Power (+ -)

The pins labeled + and - are the source power pins for encoders. The positive (+) is located at the board edge and supplies +5VDC. The ground (-) pin is near the heatsink. On ST models all power must come from the single 5v screw terminal and the single GND screw terminal

Operational Requirements

Main Power

Refer to the model-specific datasheet to confirm RoboClaw's minimum and maximum input voltage. Standard models typically operate from 6 VDC up to 34 VDC; High-Voltage (HV) variants operate from 10.5 VDC to 60 VDC. Connect power to the terminals labeled "+" (positive) and "–" (negative). Keep battery wiring as short as possible to reduce inductance and minimize voltage spikes.

Reversing main power polarity will cause permanent damage to the controller.

Emergency Stop

The motor controller should be wired with an external contactor, relay or high-amperage mechanical switch on the main power input. This allows the system to remove power quickly in the event of a malfunction or runaway condition. The device used must be rated for the full battery voltage and current.

Safe Disconnect

Only the positive lead should be disconnected. Do not disconnect the negative lead if any other ground-referenced connections, such as USB or signal grounds, are still attached. Disconnecting only the positive lead ensures a safe power-down without creating unwanted return paths through other equipment.

Do not disconnect the battery negative wire if other ground sources are connected, such as USB cable. Damage will occur.

Motor Wiring Guidelines

Motor terminals are labeled M1A / M1B for channel 1, and M2A / M2B for channel 2. In differential-drive systems, reverse the wiring on one motor so both rotate in the "forward" direction when using mixed-drive commands.

Use proper ring terminals for power and motor connections. Crimp securely and torque tight enough to maintain solid contact under load. Keep motor and battery leads as short as possible as long wiring increases inductance and can lead to harmful voltage spikes when current changes abruptly.

Fuses

Protecting the power path with a fuse is important for system safety. A properly chosen fuse helps protect downstream electronics, wiring, and the power source (such as a battery) from excessive current. However, a fuse may not prevent damage to the motor controller itself in all fault conditions. Its primary function is a safety device.

Selecting a fuse for motor systems is not trivial. The fuse must carry the normal running current while tolerating short-duration surges such as startup or stall current, without opening prematurely. Time-delay (slow-blow) or motor-rated fuses are generally recommended for this purpose.

Fuse Selection Guidelines

- Time-Delay vs Fast-Acting: Motors draw high startup and stall currents. Use a time delay (slow-blow) or motor-rated fuse to survive brief surges. Standard fast-acting fuses may open during normal startup.
- Sizing: A fuse should be rated above the motor's normal running current (often 120% or more) to avoid nuisance trips, while still protecting against faults.
- Trip Characteristics (I^2t): Fuses respond non-linearly, large surges clear quickly, smaller overloads take longer. Select a fuse whose I^2t curve matches expected motor surges.
- Temperature Effects: Fuse ratings change with heat. Check datasheets for derating information when operating in warm environments.

Power Supplies

RoboClaw is a regenerative motor controller. When a motor brakes, slows or coasts, it generates energy. When using a power supply, the regenerative energy has nowhere to go. If it is not managed, the voltage can rise and damage the motor controller, the power supply or both. Regenerative energy can be handled in several ways. The two most common are adding a battery in parallel with the power supply, or clamping the excess energy and converting it to heat using a high-watt resistor. See VClamp at Basicmicro.com

Regenerative energy can result in damage to the motor controller if left unmanaged.

Current Reporting

External instruments such as digital multimeters and programmable power supplies typically measure and display RMS or time-averaged current over much longer integration periods (seconds). As a result, their displayed values may differ from the controller's reported current.

The motor controller measures phase current by taking samples during the PWM "on" interval. Each reading is averaged over roughly 50 milliseconds and represents the instantaneous current during that sample period. Minimal filtering is applied to maintain a fast response in the control loop.

Users may apply additional software filtering or averaging to sequential motor current readings, or rely on external instrumentation designed for long integration periods.

Motor Current vs Battery Current

Motor current will not be equal to battery current. When operating below 100% duty, the motor only receives a fraction of the battery voltage during each PWM cycle. To produce the same torque at a lower effective voltage, the motor draws higher current than the battery. The bulk capacitance on the motor controller and the motor windings store and recirculate energy during each PWM cycle. As a result, the battery supplies only the average current, while the motor sees short, higher current pulses. For example:

- 25% duty: Motor ~6V, about 4× battery current
- 50% duty: Motor ~12V, about 2× battery current
- 75% duty: Motor ~18V, about 1.3× battery current

ESD Sensitivity

RoboClaw utilizes a high-performance processor that can be susceptible to electrostatic discharge. The I/O and control signals are protected, but static can still be introduced into unprotected areas, such as the motor terminals, when wiring or handling the controller. Avoid handling the PCB unnecessarily and ensure any static charge is discharged before working with RoboClaw.

Static Sensitive Device. Handle with care!

Environmental Conditions

RoboClaw motor controllers are not rated for direct exposure to moisture, condensation, corrosive agents, or extreme ambient temperatures. It is the end user's responsibility to evaluate the expected operating conditions and provide appropriate protection such as enclosures, ventilation, or environmental controls when required. Proper mitigation will help ensure reliable long-term operation.

Motor Specification

When pairing a motor controller with motors, it is important to know the motor's stall current and continuous current ratings. These values determine how much current the motor will draw under different operating conditions. The stall current defines the maximum current the motor will demand at startup or when it cannot turn, while the continuous current specifies the safe, long-term operating level. Matching these values to the controller ensures that the motor can run reliably without overloading or damaging the controller. Using a controller that is undersized may lead to overheating, shutdowns, or permanent failure.

Stall Current

Stall current is the maximum current a motor will draw when it is not turning. This happens at startup, or if the motor is held in place and unable to rotate. Every time a motor starts, it briefly pulls its stall current until it gets moving. A motor controller must be able to handle this surge of current, even if it only lasts a short time.

Continuous Current

Continuous current is the maximum amount of current a motor can safely draw while running without overheating. This value reflects the motor's normal load-carrying ability during sustained operation. The motor controller continuous current rating should be equal to or greater that this rating. If the motor controller is undersized the system may experience overheating, reduced performance, or shutdown.

Wire Lengths

Keep all motor and battery wire lengths as short as possible to minimize inductance. Excessive wire inductance can generate voltage spikes beyond RoboClaw's rating, leading to component failure.

Run Away

During project development, take precautions to prevent uncontrolled motor operation. Keep robot wheels elevated and clear of all surfaces until development and testing are complete. For embedded motor applications, ensure a readily accessible method of power disconnection from the RoboClaw is available as a fail-safe measure.

Powering External Devices

When powering external devices from RoboClaw ensure the maximum BEC output rating is not exceeded. This can cause RoboClaw to suffer logic brown out which will cause erratic behavior. Some encoders can cause excessive noise being put on the +5VDC rail of the RoboClaw. This excessive noise will cause unpredictable behavior.

Encoders

RoboClaw features dual channel quadrature/absolute decoding capabilities. Signal noise on encoder lines can cause erratic behavior or position errors. Install appropriate filtering circuits, to ensure reliable encoder operation. When connecting encoders, verify that the encoder's directional output matches the motor's rotation direction. Incorrect encoder connections can result in uncontrolled motor operation. Refer to the encoder section of this manual for detailed setup instructions and recommended filtering configurations.

Data Sheets

Please refer to the data sheet for the specific model of RoboClaw being used. The data sheet contains information specific to each model of RoboClaw. This manual is a general overview of RoboClaw usage and does not contain detailed information such as pinouts for each model of RoboClaw.

Example Libraries

Official RoboClaw libraries are available to streamline development and provide reliable communication using the Packet Serial protocol. These libraries handle CRC generation, byte formatting, and packet structure, allowing development with simple, high-level functions instead of low-level serial code.

Libraries are provided for the following platforms:

Arduino (C++) – Includes functions like ForwardM1() and ReadEncM1() for direct motor control and encoder access

C# (.NET for Windows and Mono for Linux)

Python (Raspberry Pi, Linux, macOS, etc.)

Libraries can be downloaded from the BasicMicro Downloads page:
https://www.basicmicro.com/downloads_ep_45-1.html

Source code is also available on GitHub:
https://github.com/basicmicro

Additional tutorials, wiring guides, and usage examples can be found at:
https://resources.basicmicro.com

Getting Started

Initial Setup

RoboClaw offers several control methods, each with specific configuration options. This quick-start guide covers the basic setup process. Most control schemes require minimal configuration, with detailed options covered later in this manual.

1. Review the Introduction and Hardware Overview sections of this manual.

2. Size the controller with the motor's stall current in mind (startup of the motor or high load), which can be much higher than running current. A RoboClaw can drive a motor whose stall current exceeds the controller's rating only if the system avoids sustained stalls or high load situations. Use sensible acceleration/voltage settings. Otherwise, protection may trip or hardware can be damaged.

3. Before configuring RoboClaw. Make sure a reliable power source is available such as a fully charged battery. See Wiring section of this manual for proper wiring instructions.

4. Configure RoboClaw using Motion Studio software. Motion Studio provides complete access to all configuration options and is required for controller setup.

5. After configuration, refer to the Wiring section of this manual. While a basic wiring diagram is provided for initial testing, implementing the Safety Wiring diagram is strongly recommended for reliable and safe operation.

Encoder Setup

RoboClaw supports multiple encoder types, including quadrature and absolute encoders. Proper tuning is required for reliable motor control. The built-in auto-tune function provides good results for most motor/encoder combinations, but manual adjustments may still be needed for optimal performance. For quadrature encoders, auto-tune typically produces usable values. With absolute (analog) encoders, auto-tune can be used, but the limited resolution and higher susceptibility to noise may prevent reliable results. In those cases, manual tuning is recommended.

1. Attach and wire the encoder to the motor as shown in the encoder wiring diagrams. Verify that the encoder can be powered from a regulated 5 VDC source.

2. Double-check wiring connections before applying power. After confirming wiring, proceed to the Auto Tune function in the Encoder section.

3. Verify encoder signal and power quality using an oscilloscope. RoboClaw's encoder inputs do not include built-in filtering, to maintain compatibility with high-count encoders where signal rise/fall time is critical. Some encoders may introduce noise onto their signal or power lines. If noise is present, external filtering may be required. This can include pull-up resistors, decoupling capacitors, or other conditioning circuitry.

4. Run the Auto Tune function. In most cases it provides acceptable performance, but fine-tuning of PID parameters may be necessary depending on the motor, encoder, and application. For additional tuning assistance, a support request can be opened via the contact section of BasicMicro.com

USB Control

USB Connection

When RoboClaw is connected to a powered USB host, it automatically enables USB communications. USB can be connected in any mode. In non-packet serial modes (except RC mode if no RC signal is present), USB packet serial commands can read status information and modify configuration settings, but motor movement commands are disabled. When operating in packet serial mode with another device (such as an Arduino) connected to S1 and S2 pins, both the device commands and USB packet serial commands will be executed.

USB Power

RoboClaw cannot be powered through the USB connection and requires an external power source to operate. If no external power is connected, it may unintentionally draw power from the USB port, which can cause unstable or erratic behavior. To prevent this, always connect the main power source before connecting via USB.

USB Comport and Baudrate

When connected via USB, the RoboClaw is detected as a CDC (Communications Device Class) virtual COM port.

- On Windows, a driver must be installed. This driver is available for download from our website.
- On Linux and macOS, no manual installation is needed—the operating system will automatically detect the RoboClaw and load the appropriate driver.

Unlike traditional serial ports, the USB CDC virtual COM port does not require a baud rate setting. Communication occurs at the maximum speed supported by both the host and the device, typically around 1 Mbps.

USB Noise Immunity

USB connections have limited noise immunity and are more susceptible to electrical disturbances than other interfaces. For improved stability in noisy environments, a USB cable with ferrite cores at both ends is recommended. This reduces interference and provides more reliable communication. Preconfigured cables are available from the BasicMicro website.

USB Isolation

In setups involving motor controllers, communication cables and power wiring often share ground connections. This can create ground loops, which are closed wiring paths that allow unintended current flow. Ground loops may induce voltage surges, shift reference levels, or drive current through USB or signal lines, potentially damaging both the motor controller and the connected host device.

For added protection, use a USB isolator. A USB isolator breaks the ground path between the controller and the USB host while maintaining data transmission. This prevents the USB cable from becoming an unintended current path, particularly if the negative battery terminal is disconnected while the USB cable is still attached. USB isolator that has been tested to work with RoboClaw is available from the BasicMicro website.

Install Motion Studio

Motion Studio Setup

Download and install the Motion Studio application (Windows 10 or newer required). When launched, Motion Studio will automatically check for updates and verify the presence of the USB driver. If the driver is missing, it will be installed automatically.

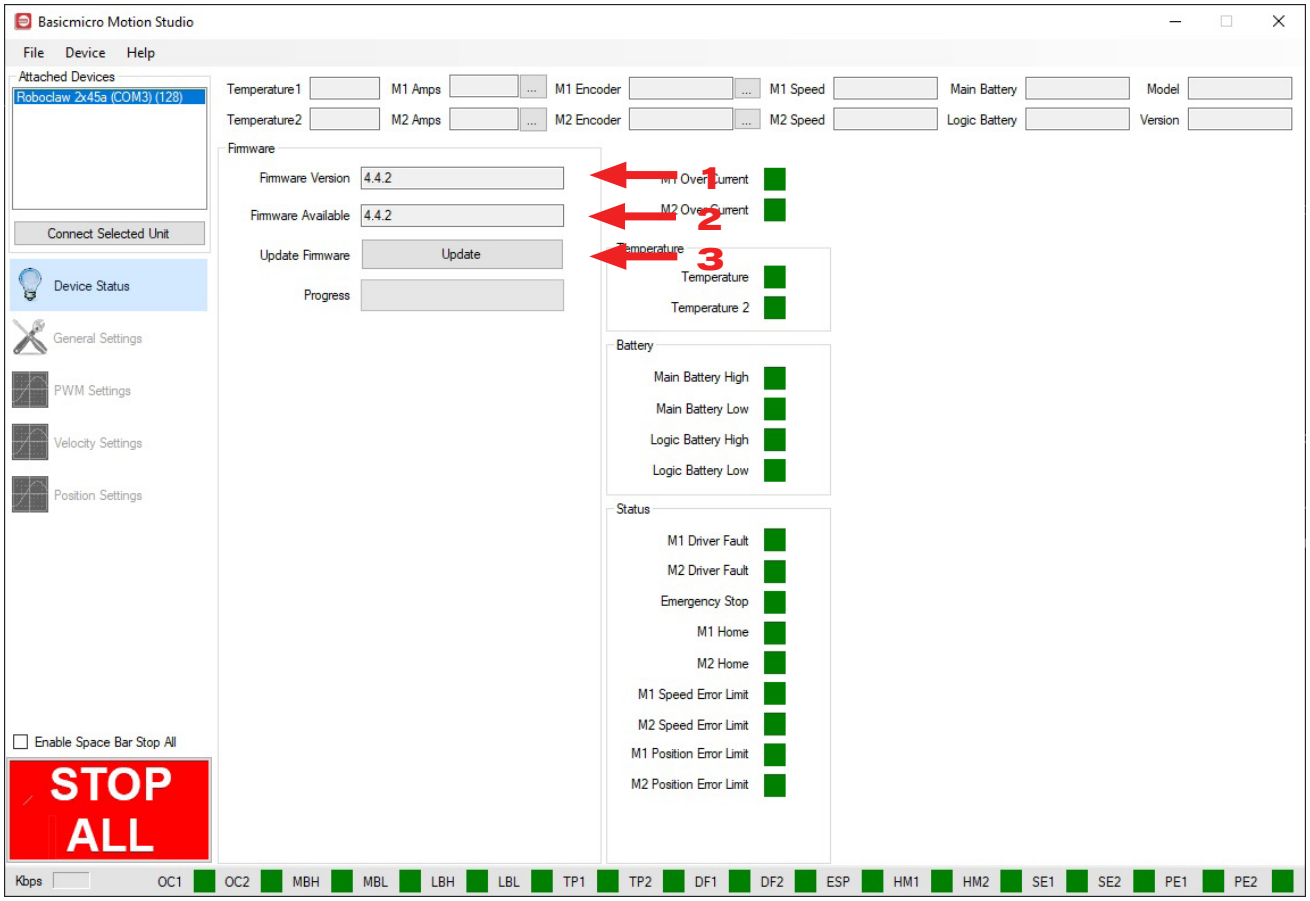
1. Open the Motion Studio application.

2. Connect a reliable power source, such as a fully charged battery, to the motor controller.

3. With Motion Studio open, connect the powered motor controller to your computer via USB.

Firmware Update

Once Motion Studio detects the motor controller, it displays the current firmware version in Firmware Version (1). Each time the software starts, it checks for a newer version of itself, which always includes the latest firmware. If an update is available, it will be downloaded and shown in Firmware Available (2). To install the update, click the Update button. Do not interrupt the process or disconnect power during the update.



1. If a new firmware version is available, click the Update button (3) to begin the update.

2. During the update, the onboard LEDs will flash and the flash memory will be erased. Do not disconnect power during this process, or the motor controller may become unusable.

3. After the update is complete, the motor controller will reset. Click Connect Selected Unit to reconnect.

Motion Studio Overview

Motion Studio

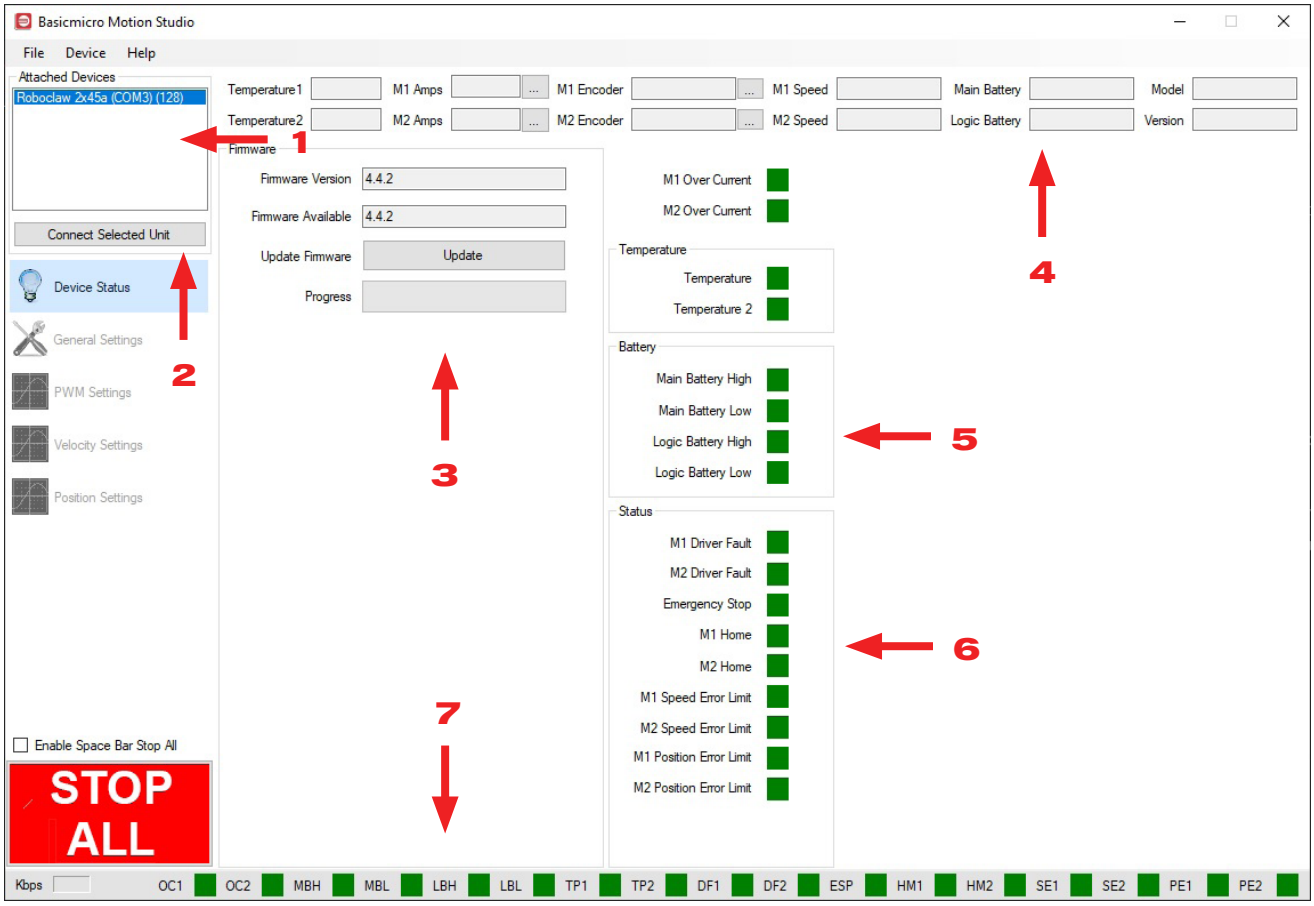
BasicMicro Motion Studio is a software suite for configuring, monitoring, and maintaining RoboClaw motor controllers. It supports all modes and options, and provides real-time control and diagnostics. Motion Studio can be downloaded from https://www.basicmicro.com. Once installed, it automatically checks for the latest version each time it runs.

Connection

When Motion Studio launches, the first screen displays a list of detected RoboClaw units (1). If more than one is shown, select the desired unit from the list, then click Connect Selected Unit (2) to establish a connection.

Once connected, the firmware section (3) will display the current firmware version and check for any available updates. If a newer version is available, click Update Firmware to begin the update process. A progress bar will indicate update status.

Status indicators (4, 5, 6, 7) show key RoboClaw conditions. Green indicates normal operation, yellow signals a warning, and red indicates a fault or error.



Device Status

Once a RoboClaw is connected, the screen switches to the Device Status view. The settings panel (1) becomes active, allowing configuration of the connected device. Monitored parameter fields (2) and status indicators (3, 4) update in real time to reflect the current state of the RoboClaw.

The Stop All button (5) is also enabled upon connection. A checkbox allows this function to be triggered using the spacebar—providing a quick and safe way to stop all motor activity while using Motion Studio.

Device Status Screen Layout

Label	Function	Description
1	Settings Selection	Used to select which settings or testing screen is currently displayed.
2	Monitored Parameters	Displays continuously updated status parameters.
3	Status Indicators	Displays current warnings and faults.
4	Status Indicators	Displays abbreviated status of warnings and faults. Visible at all times.
5	Stop All	Stops all motion. Can activate from keyboard space bar.

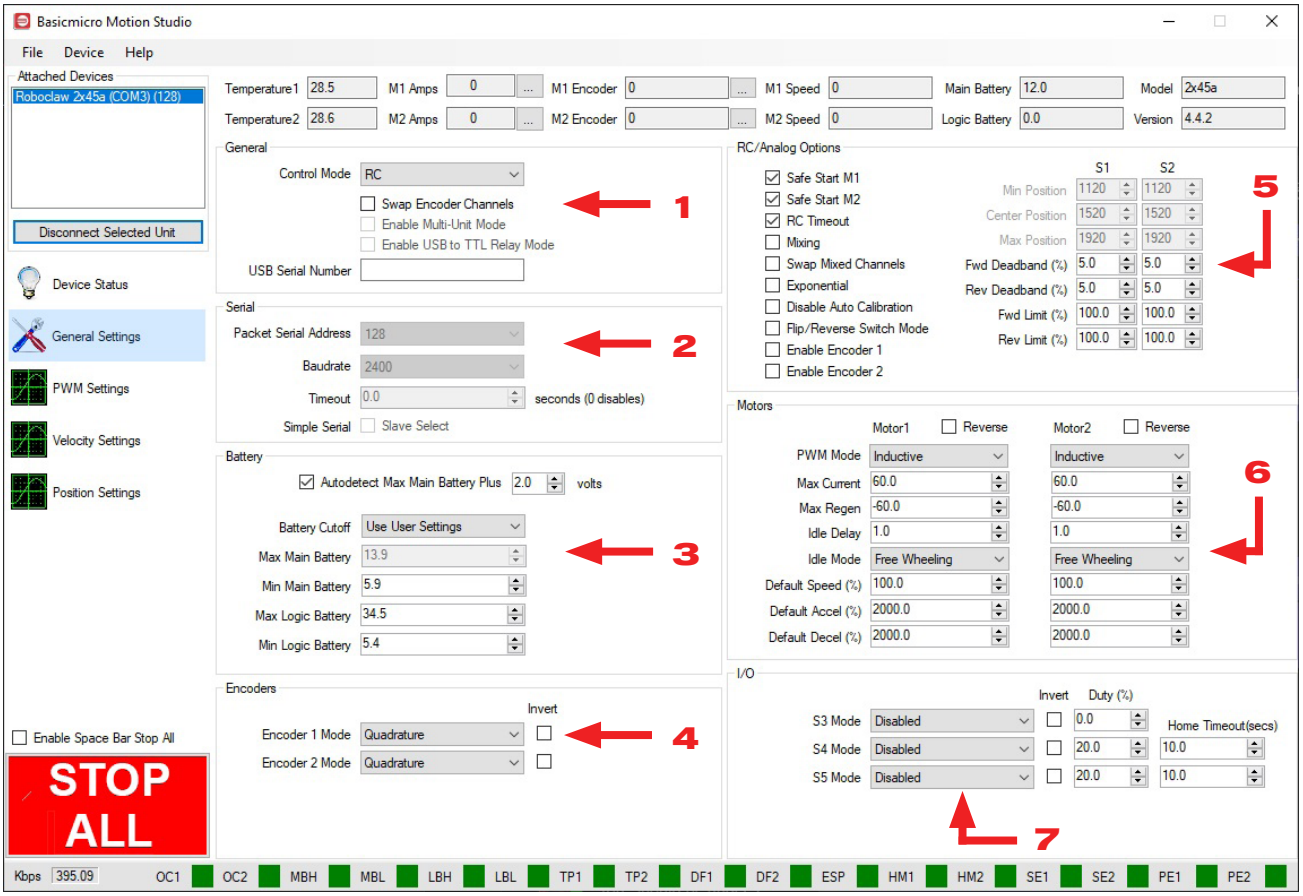
Status Indicator (4)

The status indicators at the bottom of the screen provide a quick overview of any active errors, warnings, or faults reported by the connected RoboClaw. These serve as a summary view of the full status indicators shown on the Device Status screen

Label	Description
OC1	Motor 1 over current.
OC2	Motor 2 over current.
MBH	Main battery over voltage.
MBL	Main battery under voltage.
LBH	Logic battery over voltage.
LBL	Logic battery under voltage.
TP1	Temperature 1
TP2	Optional temperature 2 on some RoboClaw models.
DF1	Motor driver 1 fault.
DF2	Motor driver 2 fault.
ESP	Emergency stop. When active.
HM1	Motor 1 homed or limit switch active. When option in use.
HM2	Motor 2 homed or limit switch active. When option in use.
SE1	Speed error on channel 1. Active when encoders are in use.
SE2	Speed error on channel 2. Active when encoders are in use.
PE1	Position error on channel 1. Active when encoders are in use.
PE2	Postion error on channel 2. Active when encoders are in use.

General Settings

The general settings screen can be used to configure RoboClaw. This includes modes, mode options and monitored parameters. For detailed explanations see the Configuration with Ion Studio section of this manual.



Configuration Options

Each control mode includes several configuration options. Some options may appear grayed out, indicating they are not available for the selected mode. After making changes, settings must be saved and the RoboClaw must be reset for them to take effect. To save, select Save Settings from the File menu.

Label	Function	Description
1	Setup	Main confirguration options and main control mode selection drop down.
2	Serial	Settings for serial modes. Set packet address, baudrate and slave select.
3	Battery	Voltage setting options for main battery and logic batteries.
4	Encoders	Set encoder type. Invert signal option.
5	RC/Analog Options	Configure RC and Ananlog control options.
6	Motors	PWM Mode, Motor current, accel and deccel settings.
7	I/O	Set encoder input type. Set S3, S4 and S5 configuration options. Enabling output pins on certain models of RoboClaw.

General

Label	Function	Description
1	Control Mode	Selects the active control mode.
1	Swaper Encoder Channeles	Swaps Encoder Channels between motors.
1	Enable Multi-Unit Mode	Sets S2 pin to open drain. Allows multiple Roboclaws to be controlled from a single serial port.
1	Enable USB to TTL Relay Mode	Allows USB serial commands to be passed through the master RoboClaw via the S1 (RX) and S2 (TX) pins, enabling multiple RoboClaws to be networked and controlled from a single USB-connected master unit.
1	USB Serial Number	Add a serial number up to 18 characters. Saved to the motor controller.

Serial

Label	Function	Description
2	Packet Serial Address	Sets the RoboClaw's address for packet serial mode, enabling control of multiple units from a single serial port using unique addresses.
2	Baudrate	Sets the serial communication speed.
2	Timeout	Sets communication timeout (0–25.5 seconds). Motors stop if no command is received within the specified period.
2	Simple Serial	Enables slave select for Simple Serial mode. Set pin S2 high to enable the attached RoboClaw. Pull S2 low and all commands will be ignored.

Battery

Label	Function	Description
3	Autodetect Max Main Battery Plus	Detects max main battery voltage with offset. Auto detect requires a properly charged battery.
3	Battery Cutoff	Selects battery cutoff preset based on LiPo cell count. Overrides user-defined voltage limits unless 'Use User Settings' is selected.
3	Max Main Battery	Sets maximum main battery voltage. The motor controller will decelerate to prevent regenerative voltage from exceeding this limit.
3	Min Main Battery	Sets minimum main battery voltage. The motor controller enters freewheel mode if voltage drops below this limit to protect the battery.
3	Max Logic Battery	Sets maximum logic battery voltage. Triggers a motor control error if exceeded to protect internal circuitry.
3	Min Logic Battery	Sets maximum logic battery voltage. Triggers a motor control error if exceeded to protect internal circuitry.

Encoders

Label	Function	Description
4	Encoder 1 Mode	Selects encoder type and direction for M1.
4	Encoder 2 Mode	Selects encoder type and direction for M2.

RC / Analog Options

Label	Function	Description
5	Safe Start M1	Enables Safe Start. Input must go to center before Motor will start.
5	Safe Start M2	Enables Safe Start. Input must go to center before Motor will start.
5	RC Timeout	Stops motors if input signal is lost for over 100 ms. Not available in Analog Mode.
5	Analog Error Detect	Stops motors if analog input reads 0 or 2047, indicating a failed potentiometer.
5	Mixing	Enables mixing of S1 and S2 inputs for differential drive control. S1 sets forward/reverse speed; S2 controls turning. Similar to RC car steering. Disable for independent tank-style control.
5	Swap Mixed Channels	Swaps M1 and M2 roles in mixing.
5	Exponential	Adds low-speed control sensitivity.
5	Disable Auto Calibration	Disables auto calibrate. Allows slow MCU to send R/C pulses at lower than normal R/C rates.
5	Flip/Reverse Switch Mode	Enables Flip/Reverse mode via S3. When activated, motor directions are reversed (useful if a robot is flipped). Active-low by default unless the pin is inverted. S3 must be set to Flip/Revese switch in its drop down menu.
5	Enable Encoder 1	Enables Encoder 1 for use in RC or Analog mode. Controls motor speed or position based on selected PID mode. Speed is scaled using the QPPS value; position is limited by set min and max position values.
5	Enable Encoder 2	Enables Encoder 2 for use in RC or Analog mode. Controls motor speed or position based on selected PID mode. Speed is scaled using the QPPS value; position is limited by set min and max position values.
5	Min Position	Sets the minimum input value when Auto Calibration is disabled. (RC: µs, Analog: 0–2V mapped to 0–2047)
5	Center Position	Sets the center input value when Auto Calibration is disabled. (RC: µs, Analog: 0–2V mapped to 0–2047)
5	Max Position	Sets the maximum input value when Auto Calibration is disabled. (RC: µs, Analog: 0–2V mapped to 0–2047)
5	Fwd Deadband (%)	Sets the forward input range near neutral that is ignored (treated as zero throttle).
5	Rev Deadband (%)	Sets the reverse input range near neutral that is ignored (treated as zero throttle).
5	Fwd Limit (%)	Sets the maximum allowed forward output by capping the PWM duty cycle to less than 100%.
5	Rev Limit (%)	Sets the maximum allowed reverse output by capping the PWM duty cycle to less than 100%.

Motors

Label	Function	Description
6	Motor1 / Motor2 Reverse	Reverses motor channel direction.
6	PWM Mode	Selects PWM drive method.
6	Max Current	Sets maximum motor current per Channel, up to the RoboClaw's rated peak. Actual limit decreases with temperature. From peak at 25 °C to continuous at 85 °C, then ramps down to 0 by 100 °C
6	Max Regen	ets the maximum regenerative current allowed during braking or deceleration. Use a negative value to limit current flowing back into the battery

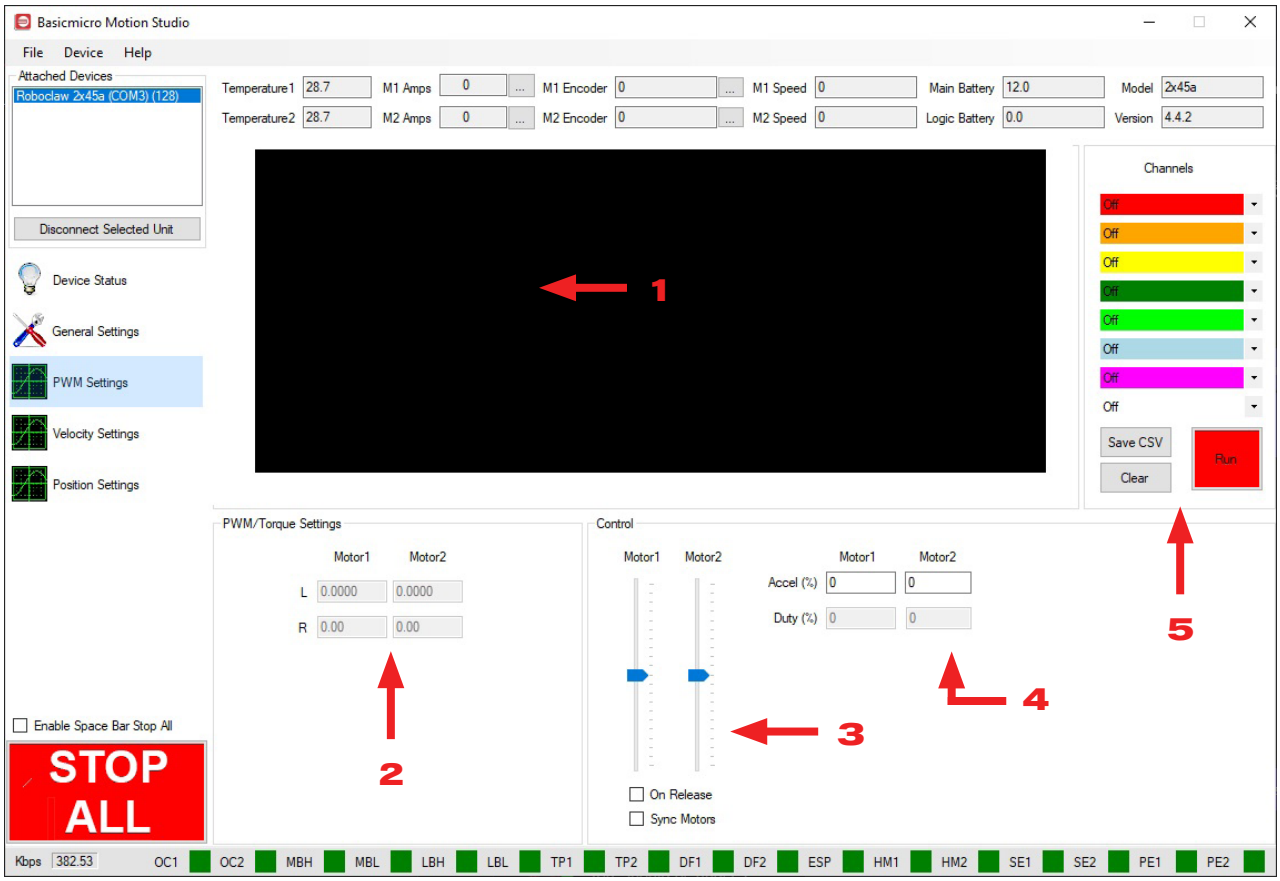
Label	Function	Description
6	Idle Delay	Sets delay before switching to freewheel mode when motor duty cycle reaches zero. Applies only when Idle Mode is set to Freewheel. Holds braking state for the specified time before releasing. Braking mode does not time out.
6	Idle Mode	Selects idle behavior: Free Wheeling or Electric Braking. Free Wheelling removes all power from the motors. Electric Braking shorts the motor.
6	Default Speed (%)	Sets maximum speed of a position movement for RC/Analog/PacketSerial Position commands that have no speed argument. Encoder required.
6	Default Accel (%)	Default acceleration for RC, Analog, or Packet Serial commands. Overridden by commands that include accel or accel/decel arguments.
6	Default Decel (%)	Default deceleration for RC, Analog, or Packet Serial commands. Overridden by commands that include accel or accel/decel arguments.

I/O

Label	Function	Description
7	S3 Mode	Assigns alternate function to input pin S3. See Auxiliary Pins section for details.
7	S4 Mode	Assigns alternate function to input pin S4. See Auxiliary Pins section for details.
7	S5 Mode	Assigns alternate function to input pin S5. See Auxiliary Pins section for details.
7	Invert	Sets input signal to active-high (default is active-low).
7	Duty (%)	Sets PWM duty for applicable alternate functions.
7	Home Timeout (secs)	Sets timeout for applicable alternate functions.

PWM Settings

The PWM settings screen is used to control RoboClaw for testing. Slider are provided to control each motor channel. This screen can also be used to determine the QPPS of attached encoders.



Graph

Label	Function	Description
1	Grid	Displays channel data with 100mS update rate and one second horizontal divisions.

PWM/Torque Settings

Label	Function	Description
2	L	MCP only. Motor Inductance in Henries.
2	R	MCP only. Motor resistance in Ohms.

Control

Label	Function	Description
3	Motor 1	Controls motor 1 duty percentage forward and reverse.
3	Motor 2	Controls motor 2 duty percentage forward and reverse.
3	On Release	When enabled, the command is sent only after the slider is released.
3	Sync Motors	When enabled, Motor 1 and Motor 2 sliders move together, sending the same command to both motors.

Control - Motor 1 / Motor 2

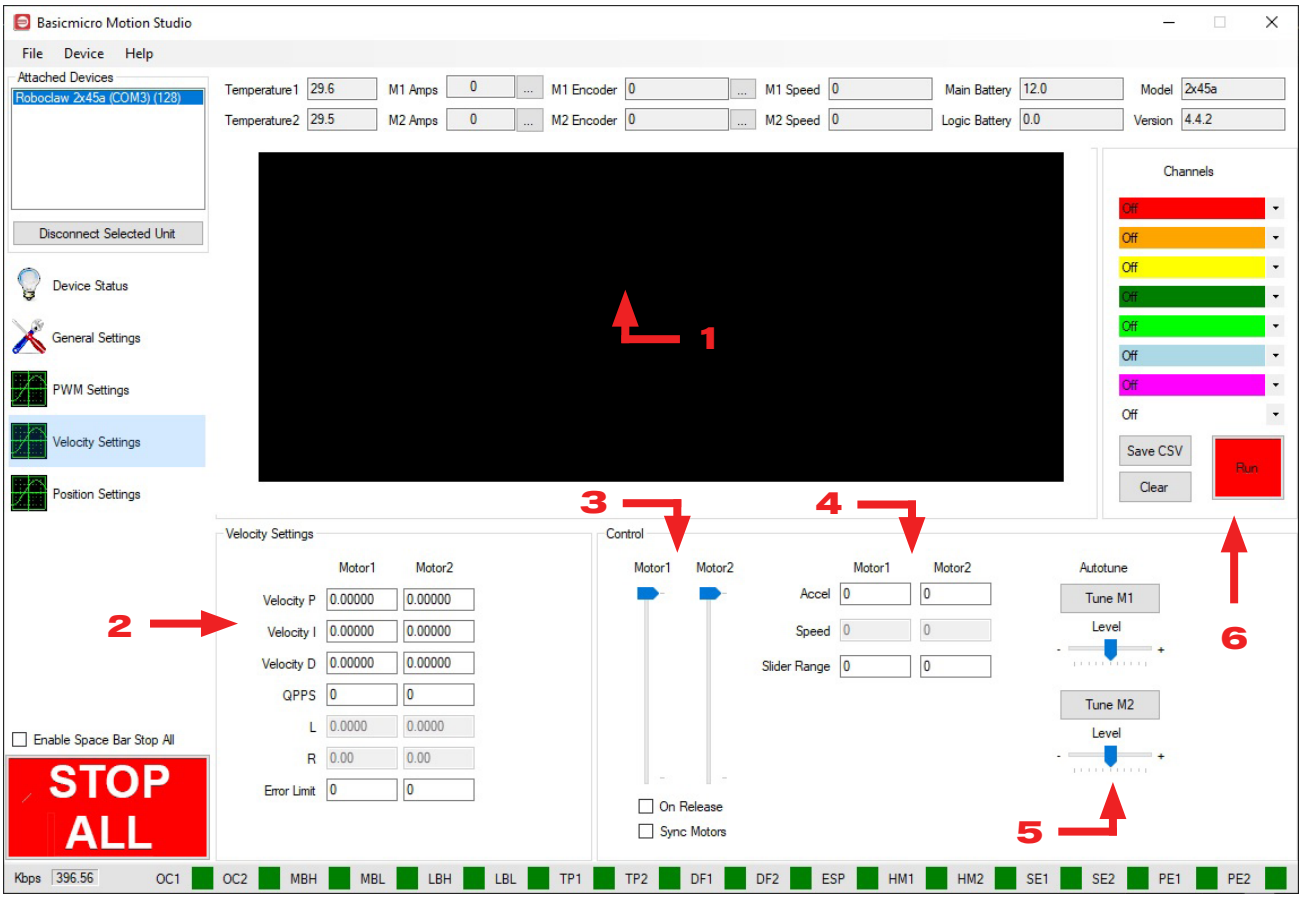
Label	Function	Description
4	Accel	Acceleration rate used when moving the sliders.
4	Duty	Displays the numberic value of the motor slider in 10ths of a Percent (0 to +/- 1000).

Graph Channels

Label	Function	Description
5	Scale	Sets vertical scale to fit the range of the specified Channel.
5	Channels	Select the data to plot for a given channel. The selected value will be graphed in the color shown. Available channel options: • Setpoint – The commanded duty, speed, or position (depends on the active screen) • PWM – Actual motor PWM (duty cycle) • Speed – Current encoder speed (1-second average) • |Speed – Current encoder speed (instantaneous) • Speed Error – Current velocity PID error • Position – Current encoder position • Position Error – Current position PID error • Current – Current motor amps • Temperature – Motor/controller temperature (°F) • Main Battery – Main battery voltage • Logic Battery – Logic supply voltageTemperature • Main Battery Voltage • Logic Battery Voltage
5	Save CSV	Save the collected data to a CSV file.
5	Clear	Clears channels graphed line.

Velocity Settings

The Velocity settings screen is used to set the encoder and PID settings for speed control. The screen is also used for testing and plotting.



Graph

Label	Function	Description
1	Grid	Displays channel data with 100mS update rate and one second horizontal divisions.

Velocity Settings

Label	Function	Description
2	Velocity P	Proportional setting for PID.
2	Velocity I	Integral setting for PID.
2	Velocity D	Differential setting for PID.
2	QPPS	Maximum speed of motor using encoder counts per second.
2	L	MCP only. Motor Inductance in Henries.
2	R	MCP only. Motor resistance in Ohms.
2	Error Limit	The absolute speed PID error value that, when exceeded, triggers a velocity error condition.

Control

Label	Function	Description
3	Motor 1	Motor 1 velocity control (0 to +/- maximum motor speed).
3	Motor 2	Motor 2 velocity control (0 to +/- maximum motor speed).
3	On Release	Will not update new speed until the slider is released.
3	Sync Motors	Synchronises Motor 1 and Motor 2 Sliders.

Control - Motor 1 / Motor 2

Label	Function	Description
4	Accel	Acceleration in encoder counts/sec² (0 uses default)
4	Speed	Current speed in encoder counts/sec (read-only)
4	Slider Range	Sets the maximum speed range for the control sliders. By default, the full forward-to-reverse range is used. Lowering the range makes it easier to control high-speed motors.

Control - Autotune

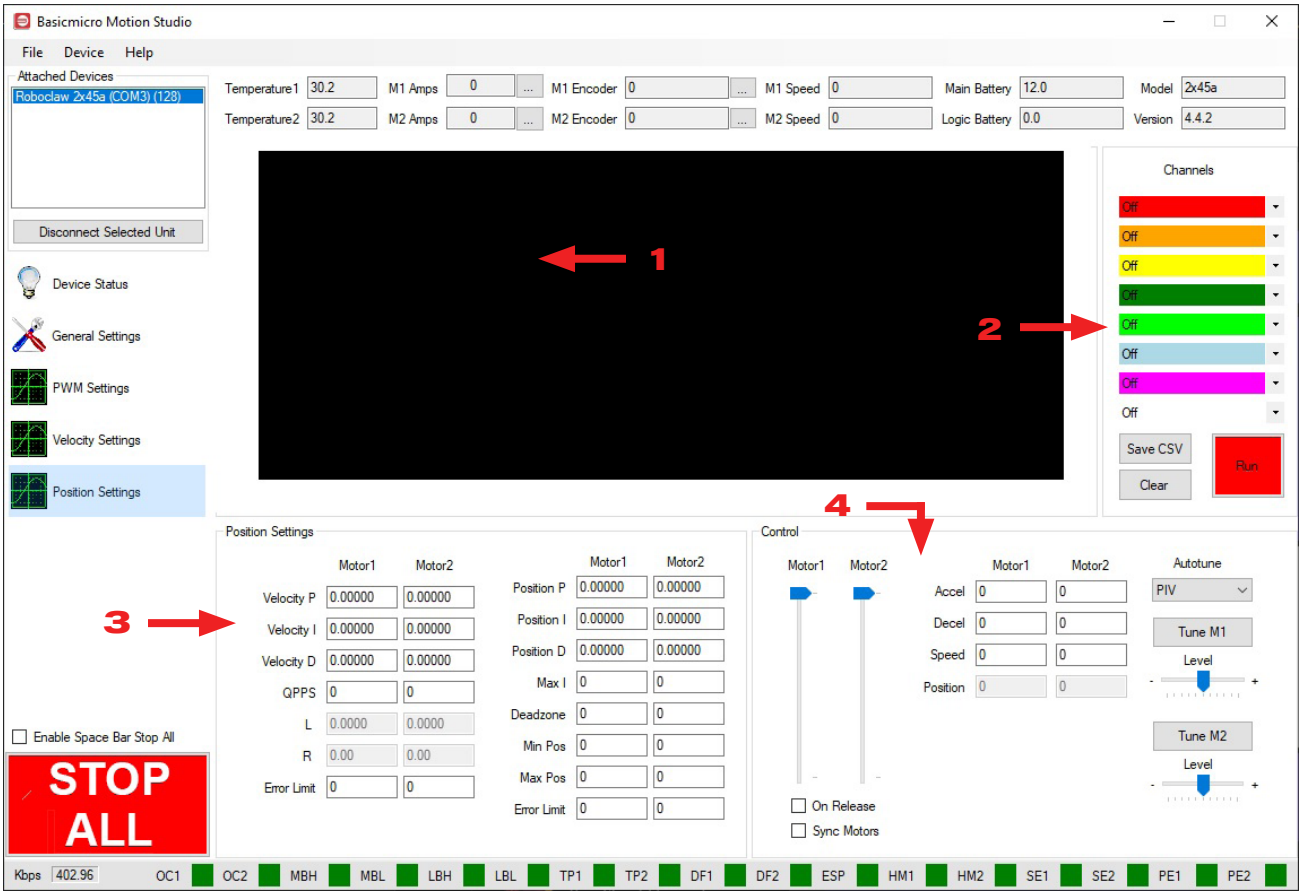
Label	Function	Description
5	Tune M1	Starts the auto-tuning process for channel 1
5	Level	Adjusts PID aggressiveness (left = less, right = more)
5	Tune M2	Starts the auto-tuning process for channel 2
5	Level	Adjusts PID aggressiveness (left = less, right = more)

Graph Channels

Label	Function	Description
4	Scale	Sets vertical scale to fit the range of the specified Channel.
4	Channels	Select the data to plot for a given channel. The selected value will be graphed in the color shown. Available channel options: • Setpoint – The commanded duty, speed, or position (depends on the active screen) • PWM – Actual motor PWM (duty cycle) • Speed – Current encoder speed (1-second average) • |Speed – Current encoder speed (instantaneous) • Speed Error – Current velocity PID error • Position – Current encoder position • Position Error – Current position PID error • Current – Current motor amps • Temperature – Motor/controller temperature (°F) • Main Battery – Main battery voltage • Logic Battery – Logic supply voltageTemperature • Main Battery Voltage • Logic Battery Voltage
4	Save CSV	Save the collected data to a CSV file.
4	Clear	Clears channels graphed line.

Position Settings

The Position settings screen is used to set the encoder and PID settings for position control. The screen is also used for testing and plotting.



Graph

Label	Function	Description
1	Grid	Displays channel data with 100mS update rate and one second horizontal divisions.
2	Scale	Sets vertical scale to fit the range of the specified Channel.

Graph Channels

Label	Function	Description
2	Channels	Select the data to plot for a given channel. The selected value will be graphed in the color shown. Available channel options: • Setpoint – The commanded duty, speed, or position (depends on the active screen) • PWM – Actual motor PWM (duty cycle) • Speed – Current encoder speed (1-second average) • |Speed – Current encoder speed (instantaneous) • Speed Error – Current velocity PID error • Position – Current encoder position • Position Error – Current position PID error • Current – Current motor amps • Temperature – Motor/controller temperature (°F) • Main Battery – Main battery voltage • Logic Battery – Logic supply voltageTemperature • Main Battery Voltage • Logic Battery Voltage
2	Save CSV	Save the collected data to a CSV file.
2	Clear	Clears channels graphed line.

Position Settings

Function	Description
Velocity P	Proportional setting for velocity PID.
Velocity I	Integral setting for velocity PID.
Velocity D	Differential setting for velocity PID.
QPPS	Maximum speed of motor using encoder counts per second.
L	MCP only. Motor Inductance in Henries.
R	MCP only. Motor resistance in Ohms.
Error Limit	The absolute speed PID error value that, when exceeded, triggers a velocity error condition.
Position P	Proportional setting for position PID.
Position I	Integral setting for position PID.
Position D	Differential setting for position PID.
Max I	Maximum integral windup limit.
Deadzone	Zero position deadzone. Increases the "stopped" range.
Min Pos	Minimum encoder position.
Max Pos	Maximum encoder position.

Control

Function	Description
Motor 1	Motor 1 velocity control (0 to +/- maximum motor speed).
Motor 2	Motor 2 velocity control (0 to +/- maximum motor speed).
On Release	Will not update new speed until the slider is released.
Sync Motors	Synchronises Motor 1 and Motor 2 Sliders.
Accel	Acceleration rate used when moving the sliders.
Deccel	Decceleration rate used when moving the sliders.
Speed	Speed to use with slide move.
Position	Numeric value of slider motor position.
Autotune	Method used. PD = Proportional and Differential. PID = Proportional Differential and Integral. PIV = Cascaded Velocity PD + Position P.
Tune M1	Start motor 1 velocity auto tune.
Level	Adjusts motor 1 PID aggressiveness (left = less, right = more)
Tune M2	Start motor 2 velocity auto tune.
Level	Adjusts motor 2 PID aggressiveness (left = less, right = more)

Control Modes

Setup

RoboClaw features 4 control modes, each designed for specific applications. All mode configuration is performed using Motion Studio software, which provides comprehensive setup options for each mode. Below is a brief description of each mode. Refer to the configuration section of this manual for detailed setup instructions.

USB Control

USB can be used in any mode, but is highly susceptible to electrical noise. A high-quality USB cable with ferrite cores at both ends is typically required for reliable communication. When RoboClaw is in packet serial mode and another device (such as an Arduino) is connected, commands from both USB and the device will be executed and may conflict with each other. In non-packet serial modes, motor movement commands via USB will not function, limiting USB communication to status monitoring and configuration settings only.

RC

In RC mode, RoboClaw accepts standard hobby RC radio system inputs or servo pulse signals from microcontrollers like Arduino. The controller interprets these servo pulses to control motor direction and speed, similar to standard RC servo operation. RC mode supports encoder feedback when properly configured (refer to the Encoder section for setup details).

Analog

Analog mode enables motor speed and direction control using voltage inputs from 0V to 2V. Control signals can be supplied by potentiometers or filtered PWM output from a microcontroller. This mode is particularly suitable for joystick positioning systems and non-microcontroller control hardware. Analog mode supports encoder feedback when properly configured (refer to the Encoder section for setup details).

Simple Serial

Simple serial mode accepts TTL level asyncronous serial data to control motor direction and speed. This mode is commonly used with microcontroller or PC interfaces. PC communication requires a MAX232 or equivalent level converter since RoboClaw only accepts TTL level inputs. Simple serial includes a slave select feature, enabling control of multiple RoboClaw units from a single RS-232 port. This mode provides one-way communication only, with RoboClaw receiving but not transmitting data. Encoder feedback is not supported in Simple Serial mode.

Packet Serial

Packet serial mode accepts TTL level RS-232 serial data to control motor direction and speed. This mode is commonly used with microcontroller or PC interfaces. PC communication requires a MAX232 or equivalent level converter since RoboClaw only accepts TTL level inputs. Each RoboClaw in packet serial mode is assigned a unique address (8 addresses available), allowing up to 8 RoboClaw units to operate on the same serial port. This mode fully supports encoder feedback (refer to the Encoder section for setup details).

Voltage Settings

Automatic Battery Detection

RoboClaw's automatic battery detection is enabled by default and samples the main battery voltage at power-up or after reset. This feature sets the maximum voltage protection threshold based on the detected battery voltage, preventing damage from over-voltage conditions. This basic protection scheme is separate from the LiPo-specific detection feature, which must be enabled through Motion Studio for additional battery management capabilities.

LiPo Cell Count Detection

By default, RoboClaw monitors only the maximum voltage of the main battery. The auto-detect LiPo feature must be enabled through Motion Studio to automatically configure battery protection settings based on cell count. When enabled, this feature samples main battery voltage at power-up or after reset to determine LiPo configuration. LiPo batteries have specific minimum and maximum voltage ranges based on cell count. For accurate detection, the battery voltage must be within its normal operating range. Under-charged or over-charged batteries will prevent proper detection and compromise battery protection functions.

When automatic LiPo detection is enabled, verify the detected cell count at startup. The Status2 LED indicates cell count through a series of blinks, with each blink representing one detected LiPo cell.

Undercharged or overcharged batteries can cause an incorrect auto detection voltage.

Manual Voltage Settings

Voltage limits can be configured through Motion Studio or packet serial commands within the board's rated voltage range. When using a power supply, set the minimum voltage threshold 2V below the power supply voltage to accommodate load-induced voltage dips. Set the maximum voltage threshold 2V above the power supply voltage to handle minor regenerative spikes.

Power Supplies

When voltage limits are reached, RoboClaw enters either braking or freewheel mode. While these settings help protect the power supply, they do not fully address regenerative voltage issues. Most power supplies connected to AC mains cannot absorb regenerative energy from decelerating motors. This regenerative voltage can cause the power supply output to rise uncontrollably, potentially damaging both the power supply and RoboClaw.

Additionally, many switching power supplies will shut down or enter protection mode when regenerative voltage is detected, causing unexpected system behavior. A dedicated voltage clamping circuit is required when using a power supply as the main power source. Refer to the Voltage Clamping section of this manual.

Auxiliary Pins

S3, S4 and S5 Setup

The S3, S4, and S5 pins on RoboClaw can be configured for functions such as home switches, limit switches, voltage clamping, and emergency stops.

Configuration is done in Motion Studio by selecting the desired mode for S3, S4, or S5 from the drop-down menu. Each mode is described in the table below. After making changes, settings must be saved before exiting Motion Studio.

CTRL1 and CTRL2 Setup

The CTRL1 and CTRL2 pins, available on select RoboClaw models, can be configured as control signals similar to the S3, S4, and S5 functions. In addition, CTRL1 and CTRL2 are MOSFET-driven outputs and can directly drive loads such as brakes. Refer to the model-specific datasheet for the maximum supported current.

S4 and S5 Motor Channel Assignment

Certain pin functions are specific to individual motor channels. S4 is assigned to Motor 1, and S5 is assigned to Motor 2. When using channel-specific features, the corresponding motor must be connected to the appropriate sensor or input to ensure proper operation.

S Pin	Option	Description
3,4,5	Disable	Disables S4 and S5. Set by default.
3,4,5	Default	Flip switch in RC/Analog mode, or E-Stop (latching) in Serial modes.
3,4,5	E-Stop(Latching)	All stop until RoboClaw is reset and E-Stop clears.
3,4,5	E-Stop	All stop until E-Stop signal clears.
3,4,5	Stop M1	Stops Motor 1 (non-fault condition) while the input signal is active. Motor resumes when signal is released.
3,4,5	Stop M2	Stops Motor 2 (non-fault condition) while the input signal is active. Motor resumes when signal is released.
3,4,5	Stop M1 / M2	Stops Motor 1 and 2 (non-fault condition) while the input signal is active. Motor resumes when signal is released.
3,4,5, CTRL1, CTRL2	Main Battery Warning	Sets pin low (or high if 'Invert' is checked) when the Max Main Battery voltage is exceeded. Can be used to drive an LED or signal line with a TTL-compatible input.
3,4,5, CTRL1, CTRL2	Logic Battery Warning	Sets pin low (or high if 'Invert' is checked) when the Max Logic Battery is exceeded. Can be used to drive an LED or signal line with a TTL-compatible input.
3,4,5, CTRL1, CTRL2	User Output	User-settable PWM output. Signal is 0 to 100% duty value. Can function as a basic analog output if filtered. Set via serial commands (SETAUXDUTYS, GETAUXDUTYS)
3,4,5, CTRL1, CTRL2	Main Battery Output	Outputs a PWM signal representing the logic battery voltage, scaled from 0–100% duty cycle to correspond with 0–100 V input. The signal must be filtered to produce a usable voltage, as the PWM frequency is undefined. User calibration against a known logic battery voltage is required.
3,4,5, CTRL1, CTRL2	Logic Battery Output	Outputs a PWM signal representing the logic battery voltage, scaled from 0–100% duty cycle to correspond with 0–100 V input. The signal must be filtered to produce a usable voltage, as the PWM frequency is undefined. User calibration against a known logic battery voltage is required.

S Pin	Option	Description
3,4,5, CTRL1, CTRL2	Current M1 Output	Outputs a PWM signal representing the motor current, scaled from 0–100% duty cycle, The signal must be filtered to produce a usable voltage, as the PWM frequency is undefined. User calibration against a packet serial current reading is required.
3,4,5, CTRL1, CTRL2	Current M2 Output	Outputs a PWM signal representing the motor current, scaled from 0–100% duty cycle. The signal must be filtered to produce a usable voltage, as the PWM frequency is undefined. User calibration against a packet serial current reading is required.
3,4,5, CTRL1, CTRL2	Temperature 1 Output	Outputs a PWM signal representing Temperature 1, scaled from 0% to 100% corresponding to 0 °C to 100 °C. The signal must be filtered to produce a usable voltage, as the PWM frequency is undefined. User calibration against a packet serial current reading is required.
3,4,5, CTRL1, CTRL2	Temperature 2 Output	Outputs a PWM signal representing Temperature 1, scaled from 0% to 100% corresponding to 0 °C to 100 °C. The signal must be filtered to produce a usable voltage, as the PWM frequency is undefined. User calibration against a packet serial current reading is required.
3,4,5, CTRL1, CTRL2	Voltage Clamp	Activates an external voltage clamp circuit to dissipate excess regenerative energy, recommended when using power supplies. Triggers 1 V above the set limit and deactivates when voltage drops back to the threshold (1 V hysteresis).
3,4,5, CTRL1, CTRL2	Brake	The brake function sets S3, S4 or S5 pins as a control signal for an external brake circuit. On some RoboClaw models CTRL1 and CTRL2 can be used as a control signals and direct drive brakes up to 5 Amps outputs and can also drive a brake directly unlike S3, S4 and S5.
3	RS485 Direction	Outputs a read/write signal for RS-485 adapters.
3	Encoders Enable/ Disable	Toggles encoder support in RC and Analog modes. When the signal is active, encoder feedback is disabled; when inactive, encoder feedback is enabled
3	Flip/Reverse Switch	Flip/Reverse input. When activated (active-low by default), motor direction toggles (ideal for reversing controls if a robot flips over). Use the 'Invert' setting to change the active state.
4,5	Motor *n* Brake Ctrl	Activates when motor stops. Releases when motor starts.
4,5	Motor *n* Home(Auto)	On startup, the motor will move until the switch is triggered or the timeout period elapses. If the switch is already triggered at startup, the motor will move forward for 3 seconds to clear the switch, then return. During normal operation, if the switch is triggered, the motor will stop and the encoder will be zeroed.
4,5	Motor *n* Home(User)	Sets the reverse limit switch to act as a home switch. The motor is driven by user commands until the switch is triggered, at which point the motor stops and the encoder count is reset to zero.
4,5	Motor *n* Home(Auto)/ Limit(Fwd)	Used when two limit switches share the same input. Behavior on power-up is the same as Motor Home (Auto). Once the home switch is triggered, the motor can only move forward until the switch is released. When moving forward, activation of the other limit switch stops the motor, after which it can only move in reverse until the switch is released.
4,5	Motor *n* Home(User)/ Limit(Fwd)	Used when two limit switches share the same input. If the switch is triggered while moving in reverse, the motor stops and the encoder is zeroed. The motor can then only move forward until the switch is released. When moving forward, activation of the other switch stops the motor, after which it can only move in reverse until the switch is released.

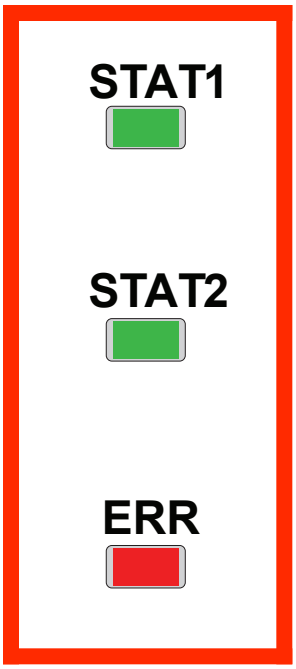
S Pin	Option	Description
4,5	Motor *n* Limit(Fwd)	Stops the motor when triggered while moving forward.
4,5	Motor *n* Limit(Rev)	Stops the motor when triggered while moving in reverse.
4,5	Motor *n* Limit(Both)	Configures the pin as a motor limit input for both directions. Two limit switches can be wired in parallel (OR configuration). The motor's commanded direction when the switch is activated determines whether it functions as a forward or reverse limit.

Status LEDs

Status and Error LEDs

RoboClaw includes 3 LEDs to indicate status. Two green status LEDs labeled STAT1 and STAT2 and one red error LED labeled ERR. When the motor controller is first powered on all 3 LEDs should blink briefly to indicate all LEDs are functional.

The LEDs will behave differently depending on the mode. During normal operation the status 1 LED will remain on continuously or blink when data is received in RC Mode or Serial Modes. The status 2 LED will light when either drive stage is active.



Message Types

RoboClaw indicates three types of operational messages:

1. Faults: When a fault occurs, RoboClaw disables all motor outputs and halts operation. The controller requires either a reset or, in the case of non-latching E-Stops, clearing of the fault condition to resume operation.

2. Warnings: Warning conditions trigger automatic protective responses. For example, when the temperature reaches 85°C, RoboClaw automatically reduces maximum current output until a safe temperature is achieved.

3. Notices: Informational messages about system status. The controller currently implements one notice condition.

LED Blink Sequences

When a warning or fault occurs, the RoboClaw indicates the condition by blinking a sequence using its LEDs. The table below lists each blink pattern and its corresponding cause. For warning conditions, the Error LED will remain steadily lit while the condition is active. For example, if the controller exceeds 85°C, the Error LED will stay on until the temperature drops below 85°C.

LED Status	Condition	Type	Description
Error LED blinks once with short delay. Other LEDs off.	Over 100c Temperature	Fault	Motors freewheel while condition exist.
Error LED blinking 2 times.	Logic Battery High	Fault	Motors freewheel until reset.
Error LED blinking 3 times.	Logic Battery Low	Fault	Motors freewheel until reset.
Error LED blinking 4 times.	Velocity Error Fault	Fault	Triggers if velocity error limit is exceded.
Error LED blinking 5 times.	Position Error Fault	Fault	Triggeres if position error limit is exceded
All three LEDs lit.	E-Stop	Fault	Motors are stopped by braking.
Error LED lit while condition is active.	Over 85c Temperature	Warning	Motor current limit is recalculated based on temperature.
Error LED lit while condition is active.	Over Current	Warning	Motor power is automatically limited.
Error LED lit while condition is active.	Main Battery High	Warning	Motors freewheel while condition exist.
Error LED lit while condition is active.	Main Battery Low	Warning	Motors freewheel while condition exist.
Error LED lit while condition is active.	M1 or M2 Home	Warning	Motor is stopped and encoder is reset to 0
All 3 LED cycle on and off in sequence after power up.	RoboClaw is waiting for new firmware.	Notice	RoboClaw is in boot mode. Use Motion Studio to clear.

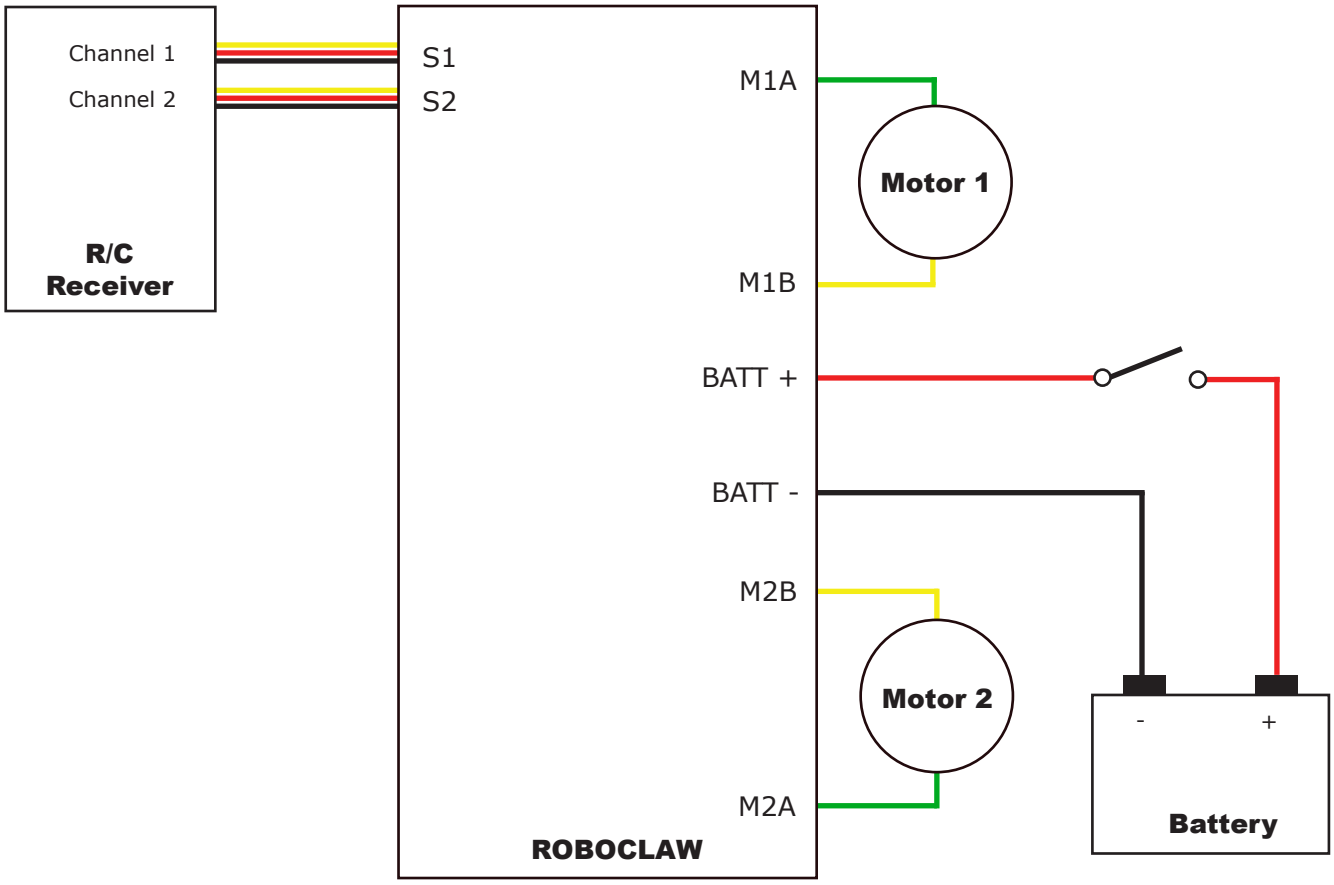
Wiring

Basic Wiring

Each control mode has specific wiring requirements for safe and reliable operation. While the diagram below shows a basic wiring configuration suitable for small motor systems, all RoboClaw installations should include a main battery disconnect switch, regardless of system size. Never underestimate the potential hazards of an uncontrolled motorized system.

RoboClaw is a regenerative motor controller. Motors in motion can generate voltage even when the system is powered off, potentially causing erratic behavior. Always provide a return path to the battery for situations where the system might move when main power is disconnected or a fuse is blown.

Always disconnect the positive terminal first. Disconnecting the negative terminal first can create a ground loop and potentially damage the controller or other connected electronics.



Wiring Safety

Safety is paramount in any system with movement. The wiring diagram below illustrates proper safety practices for RoboClaw installation. A properly rated main power disconnect switch must be installed to provide emergency shutdown capability.

Regenerative Return Path

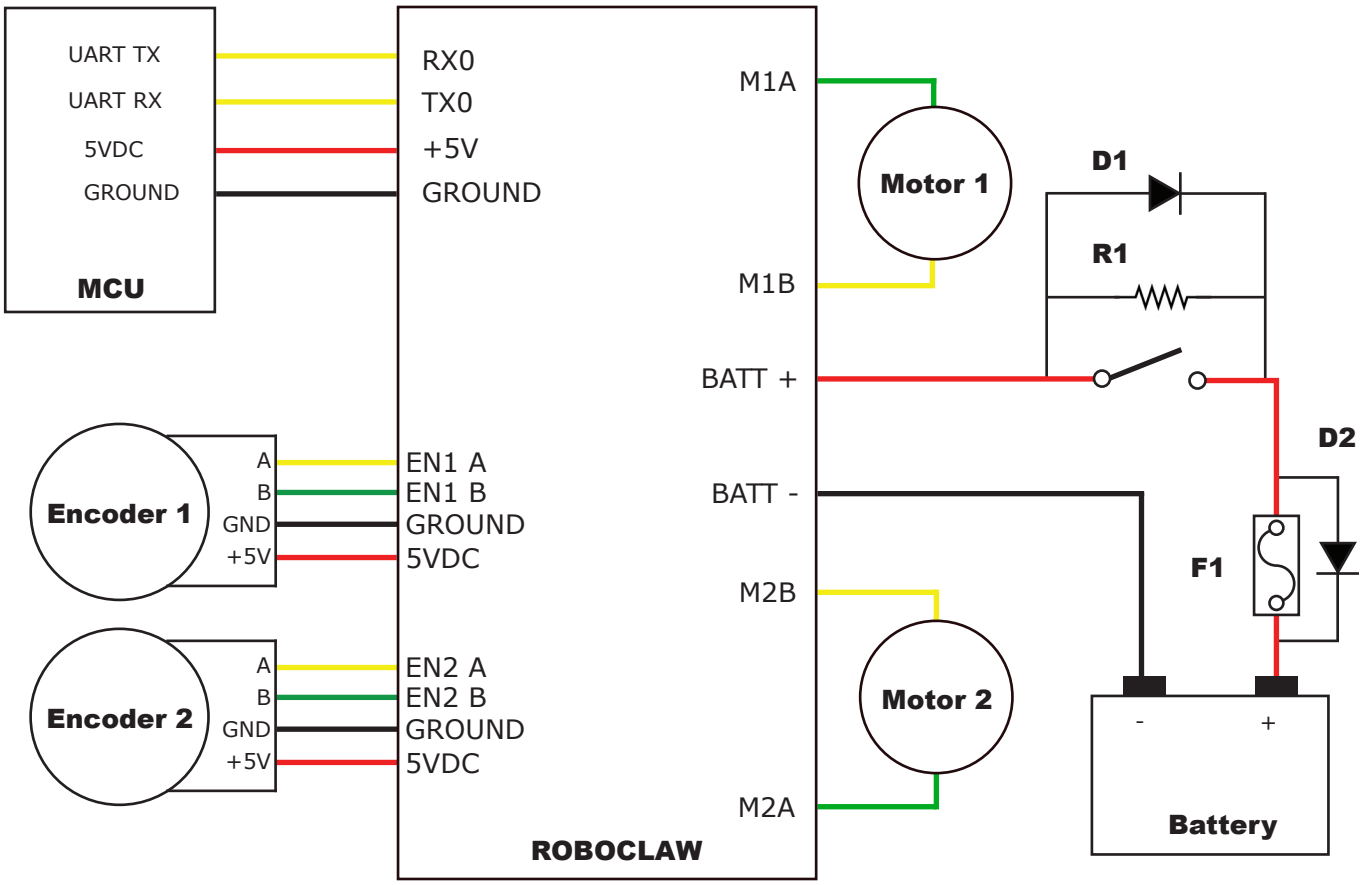
A high-current diode (D1 / D2) may be required to provide a return path to the battery for regenerative voltage when power is disconnected or a fuse opens. When brushed motors are mechanically back-driven, they act as generators and can produce voltage even when the controller is unpowered. Without a return path, this energy can damage the controller. The return path diode requirement depends on the mechanical system and the possibility of back-driving.

Inrush Current

For systems operating at 48 VDC and above, a pre-charge resistor (R1) is recommended to limit inrush current and prevent switch contact arcing. A 1 kΩ, ½ W resistor provides an approximate 15-second pre-charge time.

Encoder Wiring

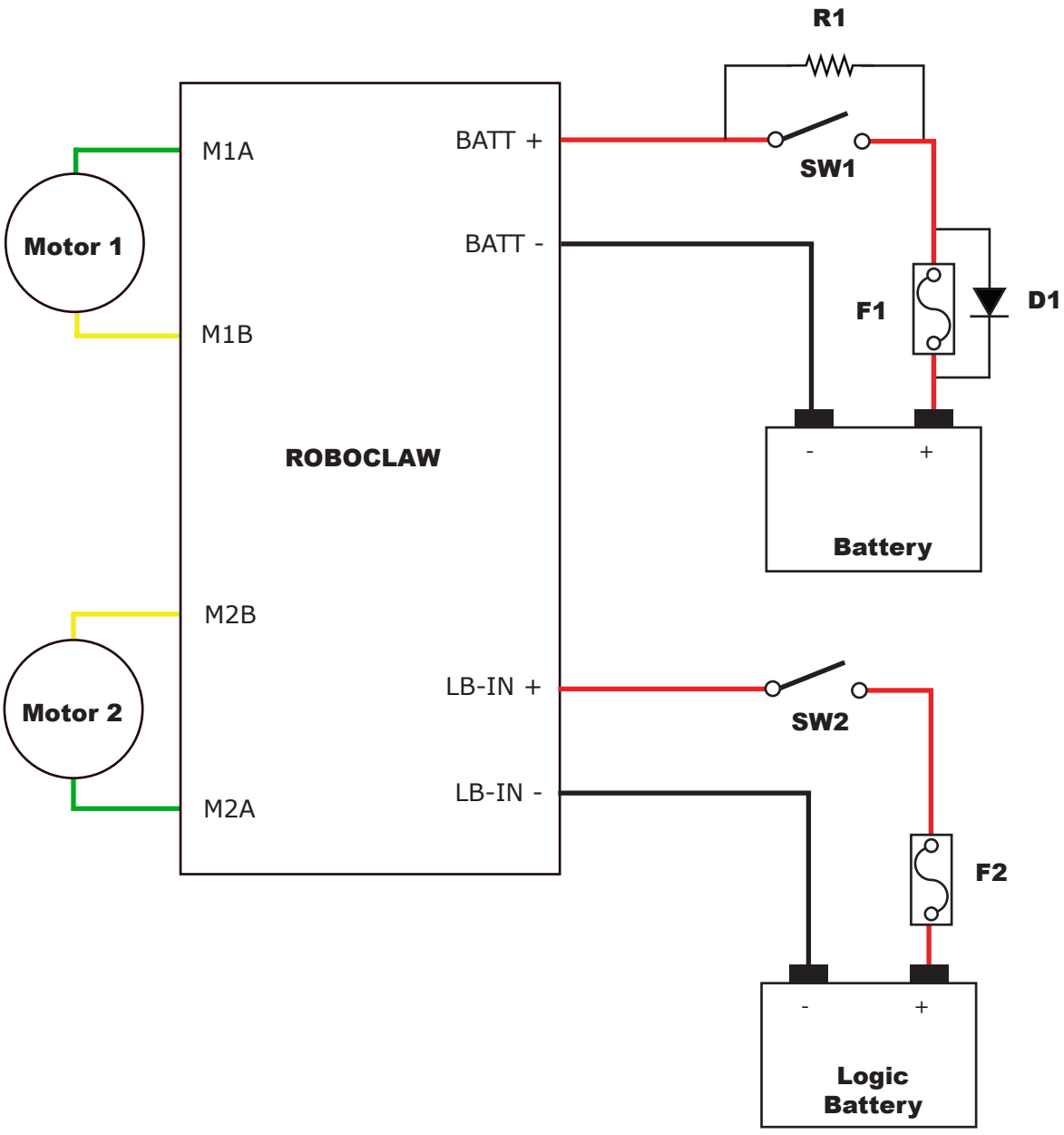
RoboClaw supports multiple feedback devices for closed-loop operation, including quadrature encoders, absolute encoders, potentiometers, and Hall-effect sensors. Some encoders may require pull-up resistors or filter capacitors for proper signal conditioning. Encoder input pins are multifunctional and may be configured for uses beyond encoder feedback. Refer to the Encoder section of this manual for detailed configuration and signal conditioning requirements.



Logic Battery Wiring

RoboClaw supports an optional logic battery connected to the LB-IN input to provide backup power for the logic section. During high-current operation, voltage drops on the main power supply can cause logic brownouts, which may lead to control system instability. Using a separate power source for the logic circuits prevents these brownouts and helps maintain stable operation.

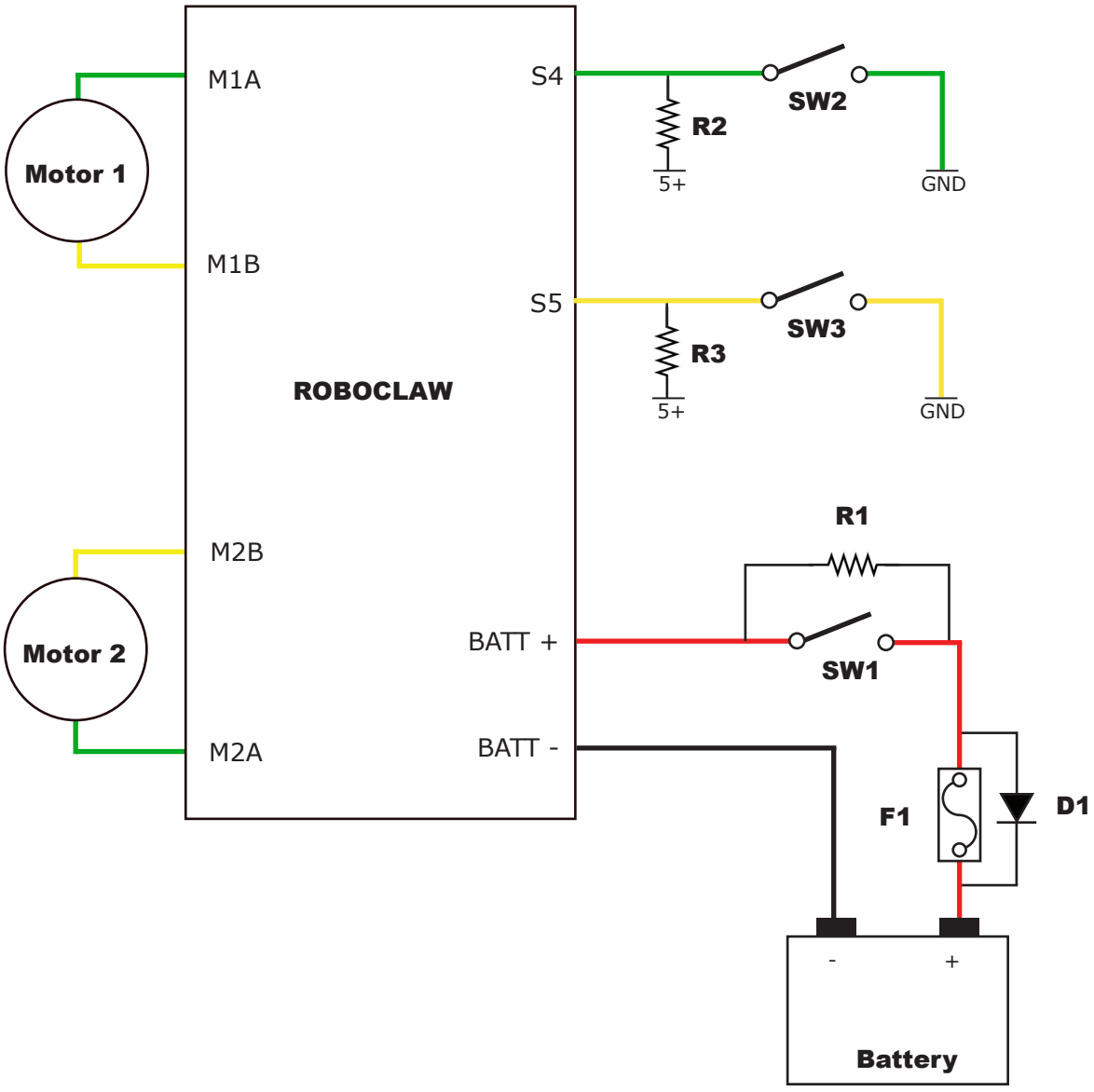
The LB-IN input requires a minimum of 6 VDC, with 7 VDC or higher preferred. Refer to the motor controller datasheet for the maximum allowable input voltage. When a logic battery is installed, it powers the internal 5 V regulator as well as the regulated 5 V user output. The logic battery capacity should be sized based on the current requirements of any devices connected to the 5 V regulated output.



Limit / Home / E-Stop Wiring

S4 controls motor channel 1 and S5 controls motor channel 2. A pull-up resistor to 5VDC or 3.3VDC should be used if wire lengths exceed 6" (150mm). The circuit below shows a NO (normally open) style switch. Connect the NO to S4 or S5 and the COM end to a power ground shared with RoboClaw.

For model specific pinout information please refer to the data sheet for the model being used.



Regenerative Voltage Clamping

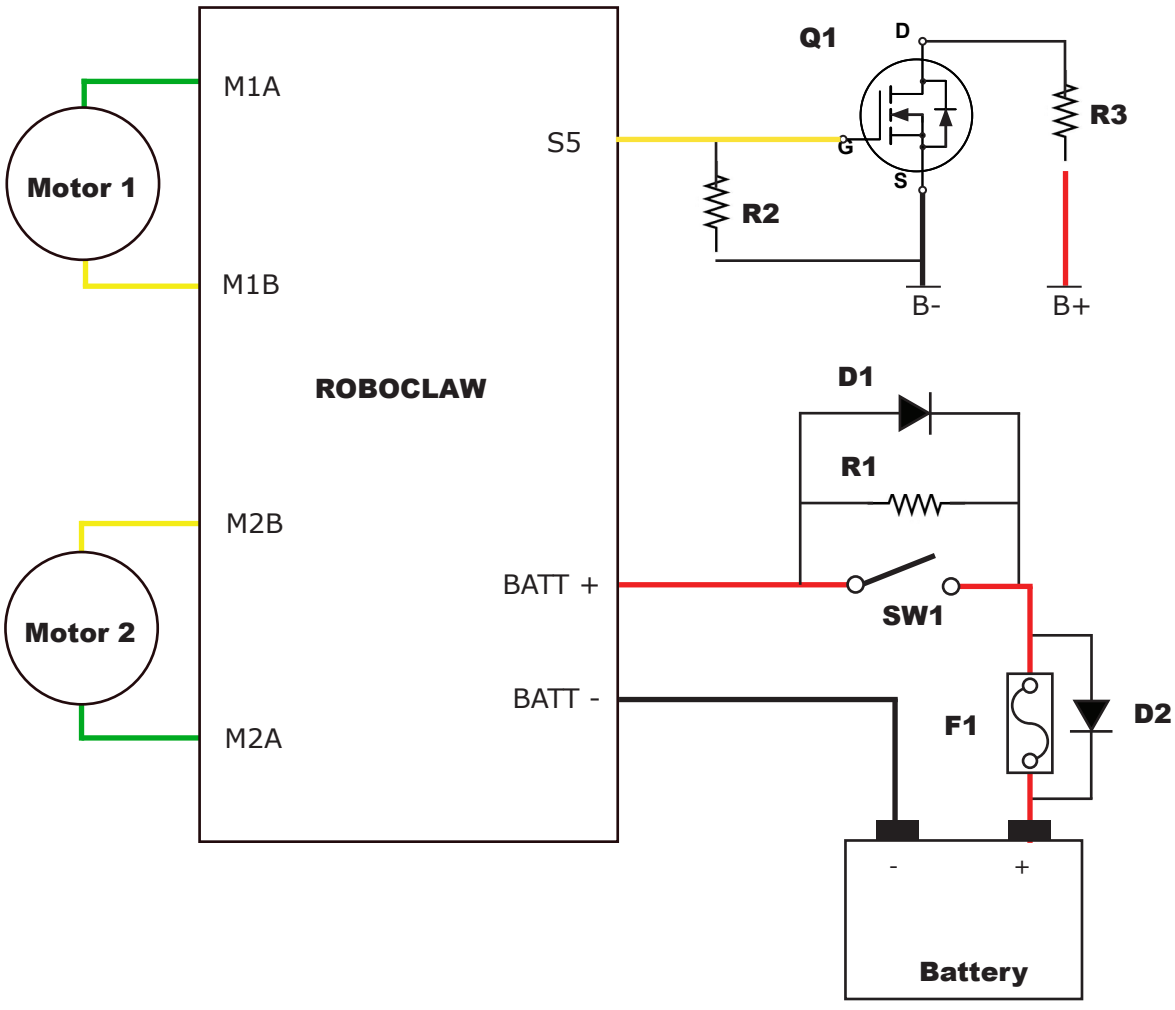
Voltage Clamping

When using power supplies, regenerative voltage must be dissipated. During braking or deceleration, motors can regenerate energy onto the supply bus, causing voltage to rise and potentially damage the controller or power supply.

Clamping Circuit

Regenerative energy can be managed using the voltage clamp circuit shown below, activated through an Sx pin. Q1 (BUK965R8) is used as the clamp switch. R2 (10 kΩ) is a gate pull-down resistor that ensures Q1 remains off when the clamp is inactive. R3 is a high-power resistor used to dissipate regenerative energy as heat. Typical R3 values range from 10 Ω at 50 W for smaller motors down to 1 Ω at 100 W or lower for larger systems. Final values should be selected based on testing and observed regenerative voltage.

BasicMicro also offers a ready-to-use voltage clamp module, eliminating the need to design and size discrete components. Refer to the website for details.



Voltage Clamp Setup and Testing

Open Motion Studio and set S5 to the Voltage Clamp option in the drop down and save the setting before exiting the application.

1. Configure the Controller:
   - Open Motion Studio.
   - Set S5 to the Voltage Clamp option in the dropdown menu.
   - Save the setting before exiting the application.
2. Begin Testing:
   - Run the motor at approximately 25% of full speed.
   - Quickly reduce the speed without braking or triggering an emergency stop (E-stop), while monitoring the supply voltage for any spikes.
3. Stepwise Increase:
   - Gradually increase speed and power in 5% increments.
   - At each step, repeat the quick deceleration and monitor the voltage spike.

   Continue this process up to 100% power, or until you observe that the clamp is no longer effectively dissipating voltage spikes.
4. Evaluate Clamp Performance:
   - If overvoltage spikes are not fully clamped, corrective action is required:
   - Use a lower resistance dump resistor, or
   - Add additional bulk capacitance across B+ and B- (typically 5000µF to 10,000µF or more, depending on system requirements).

RC Control

RC Mode

RC mode is typically used when controlling RoboClaw from a hobby RC radio. This mode can also be used to simplify driving RoboClaw from a microcontroller using servo pulses. In this mode S1 controls the direction and speed of motor 1 and S2 controls the direction and speed of motor 2.

RC Mode With Mixing

This mode is the same as RC mode with the exception of how S1 and S2 controls the attached motors. When used with a differentially steered robot, mixing mode allows S1 to control the speed forward and backward and S2 to control steering left and right.

RC Mode with feedback for velocity or position control

RC Mode supports encoder feedback with either velocity or position PID control. Before enabling encoder support, PID constants must be properly calibrated and saved to RoboClaw's EEPROM memory. Enable encoder feedback for RC mode using Motion Studio software or packet serial commands. Refer to the Configuration section of this manual for detailed setup instructions.

RC Control Options

Label	Function	Description
5	Safe Start M1	Enables Safe Start. Input must go to center before Motor will start.
5	Safe Start M2	Enables Safe Start. Input must go to center before Motor will start.
5	RC Timeout	Stops motors if input signal is lost for over 100 ms. Not available in Analog Mode.
5	Analog Error Detect	Stops motors if analog input reads 0 or 2047, indicating a failed potentiometer.
5	Mixing	Enables mixing of S1 and S2 inputs for differential drive control. S1 sets forward/reverse speed; S2 controls turning. Similar to RC car steering. Disable for independent tank-style control.
5	Swap Mixed Channels	Swaps M1 and M2 roles in mixing.
5	Exponential	Adds low-speed control sensitivity.
5	Disable Auto Calibration	Disables auto calibrate. Allows slow MCU to send R/C pulses at lower than normal R/C rates.
5	Flip/Reverse Switch Mode	Enables Flip/Reverse mode via S3. When activated, motor directions are reversed (useful if a robot is flipped). Active-low by default unless the pin is inverted. S3 must be set to Flip/Revese switch in its drop down menu.
5	Enable Encoder 1	Enables Encoder 1 for use in RC or Analog mode. Controls motor speed or position based on selected PID mode. Speed is scaled using the QPPS value; position is limited by set min and max position values.
5	Enable Encoder 2	Enables Encoder 2 for use in RC or Analog mode. Controls motor speed or position based on selected PID mode. Speed is scaled using the QPPS value; position is limited by set min and max position values.
5	Min Position	Sets the minimum input value when Auto Calibration is disabled. (RC: µs, Analog: 0–2V mapped to 0–2047)
5	Center Position	Sets the center input value when Auto Calibration is disabled. (RC: µs, Analog: 0–2V mapped to 0–2047)
5	Max Position	Sets the maximum input value when Auto Calibration is disabled. (RC: µs, Analog: 0–2V mapped to 0–2047)
5	Fwd Deadband (%)	Sets the forward input range near neutral that is ignored (treated as zero throttle).

Label	Function	Description
5	Rev Deadband (%)	Sets the reverse input range near neutral that is ignored (treated as zero throttle).
5	Fwd Limit (%)	Sets the maximum allowed forward output by capping the PWM duty cycle to less than 100%.
5	Rev Limit (%)	Sets the maximum allowed reverse output by capping the PWM duty cycle to less than 100%.

Pulse Ranges

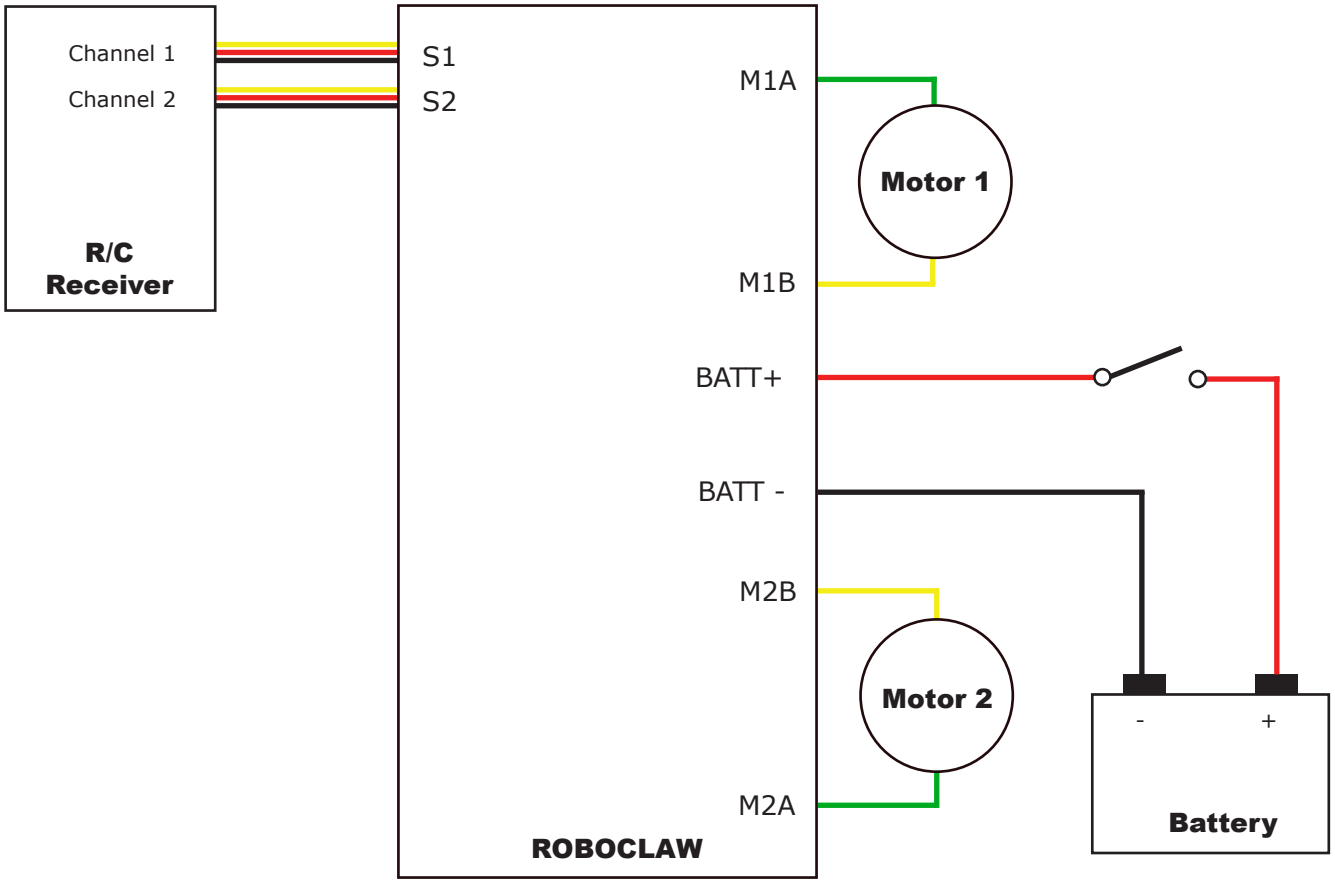
The RoboClaw expects RC pulses on S1 and S2 to drive the motors when the mode is set to RC mode. The center points are calibrated at start up (unless disabled by enabling MCU mode). 1250us is the default for full reverse and 1750us is the default for full forward. The RoboClaw will auto calibrate these ranges on the fly unless auto-calibration is disabled. If a pulse smaller than 1250us or larger than 1750us is detected the new pulse range will be set as the maximum.

Pulse	MCU Mode Enabled	MCU Mode Diabled
Stopped	1520µs	Start Up Auto Calibration
Full Reverse	1120µs	+400µs
Full Forward	1920µs	-400µs

RC Wiring Example

Connect RoboClaw as shown below and set Mode 1 with Option 1. Before powering up, center the control sticks on the transmitter, then power on the radio, followed by the receiver, and finally RoboClaw. RoboClaw requires about one second to calibrate the neutral positions of the RC controller. Once RC pulses are detected and calibration is complete, the Stat1 LED will begin flashing to indicate that signals from the RC receiver are being received.

The 5 VDC line between RoboClaw and the RC receiver is intended to power the receiver. If the receiver has its own power supply, this line must be disconnected to avoid conflicts.



Analog Control

Analog Mode

Analog mode is utilized when controlling the RoboClaw motor controller from a potentiometer or a filtered PWM signal. In this mode, S1 and S2 are configured as analog inputs. The center points are calibrated at startup unless disabled by checking "Disable Auto Calibration" in Motion Studio, which then sets the center at 1V. If a voltage smaller than 0.5V or larger than 1.5V is detected, the new voltage will be set as the minimum or maximum, respectively.

Analog Mode With Mixing

Mixing mode operates similarly to Analog mode, with the exception of how S1 and S2 control the attached motors. When used with a differentially steered robot, Mixing mode allows S1 to control the speed of the robot, both forward and backward, while S2 controls the steering direction, left and right. This configuration simplifies the control scheme for differential drive systems, enabling intuitive operation with utilizing a joystick or two potentiometers.

Analog Mode with Encoders

Analog Mode can be used in conjunction with encoders. To ensure proper operation, the Velocity and/or Position PID constants must be calibrated. Once the calibrated values have been set and saved into the RoboClaw's EEPROM, encoder support using velocity or position PID control can be enabled. Enabling encoders for RC/Analog modes can be done using Motion Studio control software or PacketSerial commands (refer to the section "Configuration Using Motion Studio" for more information).

Analog Control Options

Label	Function	Description
5	Safe Start M1	Enables Safe Start. Input must go to center before Motor will start.
5	Safe Start M2	Enables Safe Start. Input must go to center before Motor will start.
5	RC Timeout	Stops motors if input signal is lost for over 100 ms. Not available in Analog Mode.
5	Analog Error Detect	Stops motors if analog input reads 0 or 2047, indicating a failed potentiometer.
5	Mixing	Enables mixing of S1 and S2 inputs for differential drive control. S1 sets forward/reverse speed; S2 controls turning. Similar to RC car steering. Disable for independent tank-style control.
5	Swap Mixed Channels	Swaps M1 and M2 roles in mixing.
5	Exponential	Adds low-speed control sensitivity.
5	Disable Auto Calibration	Disables auto calibrate. Allows slow MCU to send R/C pulses at lower than normal R/C rates.
5	Flip/Reverse Switch Mode	Enables Flip/Reverse mode via S3. When activated, motor directions are reversed (useful if a robot is flipped). Active-low by default unless the pin is inverted. S3 must be set to Flip/Revese switch in its drop down menu.
5	Enable Encoder 1	Enables Encoder 1 for use in RC or Analog mode. Controls motor speed or position based on selected PID mode. Speed is scaled using the QPPS value; position is limited by set min and max position values.
5	Enable Encoder 2	Enables Encoder 2 for use in RC or Analog mode. Controls motor speed or position based on selected PID mode. Speed is scaled using the QPPS value; position is limited by set min and max position values.
5	Min Position	Sets the minimum input value when Auto Calibration is disabled. (RC: µs, Analog: 0–2V mapped to 0–2047)

Label	Function	Description
5	Center Position	Sets the center input value when Auto Calibration is disabled. (RC: µs, Analog: 0–2V mapped to 0–2047)
5	Max Position	Sets the maximum input value when Auto Calibration is disabled. (RC: µs, Analog: 0–2V mapped to 0–2047)
5	Fwd Deadband (%)	Sets the forward input range near neutral that is ignored (treated as zero throttle).
5	Rev Deadband (%)	Sets the reverse input range near neutral that is ignored (treated as zero throttle).
5	Fwd Limit (%)	Sets the maximum allowed forward output by capping the PWM duty cycle to less than 100%.
5	Rev Limit (%)	Sets the maximum allowed reverse output by capping the PWM duty cycle to less than 100%.

Analog Wiring Example

The RoboClaw motor controller uses a high-speed 12-bit analog-to-digital converter with a 0–2V input range. While the analog pins are protected and 5V tolerant, using a 5V reference limits the usable input range. To bring the potentiometer output within the valid 0–2V range, use a resistor divider circuit as shown below. The potentiometer serves as one half of the divider; values for R1 and R2 depend on the potentiometer's resistance.The values of R1 and R2 depend on the potentiometer resistance:

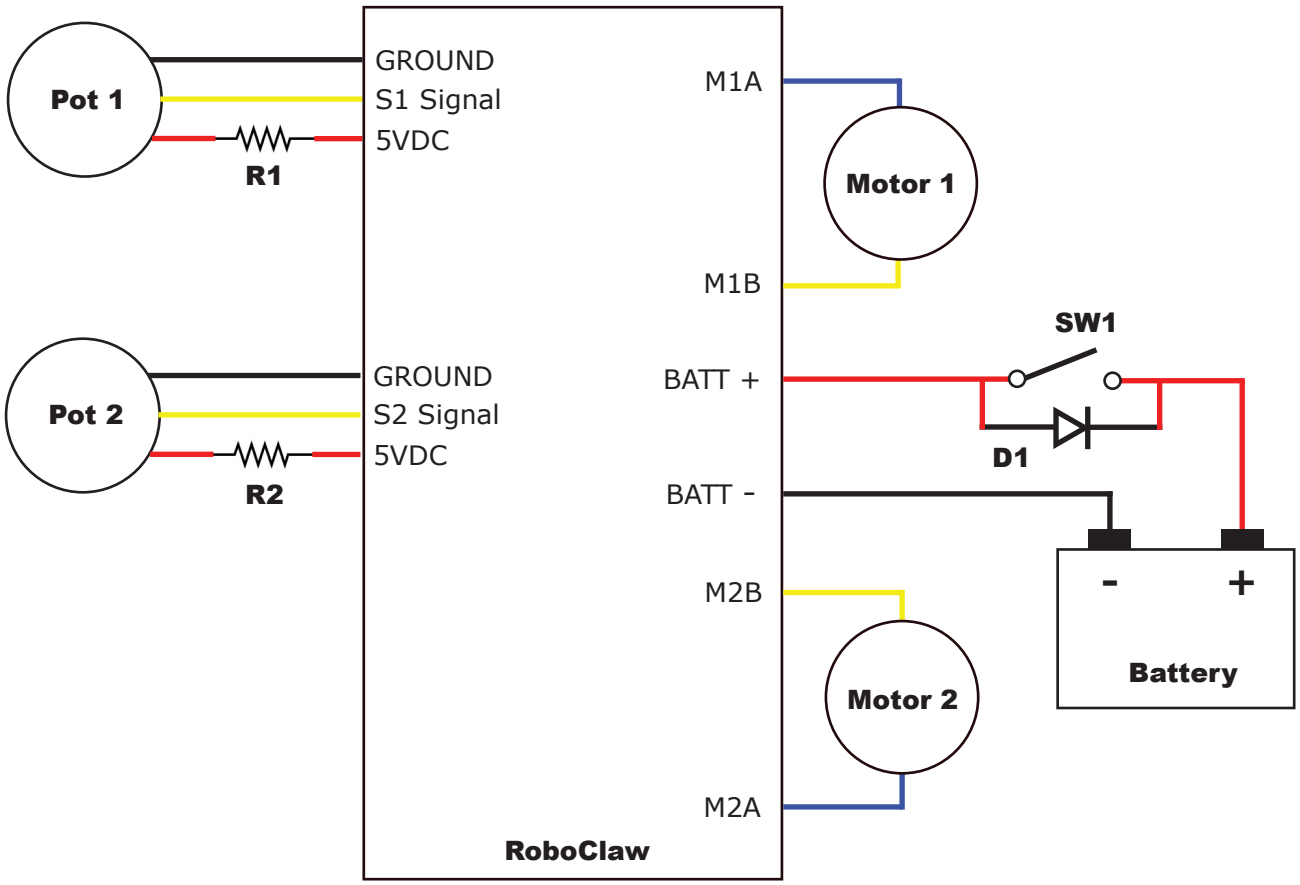
5k potentiometer, R1 / R2 = 7.5k

10k potentiometer, R1 / R2 = 15k

20k potentiometer, R1 / R2 = 30k

Analog inputs are sensitive to electrical noise. If the signal is unstable, add a 0.1 µF capacitor between the analog input and GND to reduce noise.

If using self-centering potentiometers or joysticks, leave Auto Calibration enabled and ensure both are centered before powering on. S1 controls Motor 1; S2 controls Motor 2. For non-centered inputs, enable "Disable Auto Calibration" and set the center manually.



Analog Error Detect

When a potentiometer is used for critical commands like speed or braking, it's important to implement a safe state if the potentiometer or wiring fails. This can be done by adding a 100K pull down resistor (R5/R6) and a 220 Ω resistors (R3/R4) to the GND side of the pot and enabling "Analog Error Detect." These resistors limit the input range to 0.2–1.8 V. If a wire is cut, the input will jump to 0 V or 2 V, triggering the safety feature. R1 and R2 may need to be adjusted:

5k potentiometer, R1 / R2 = 8.2k

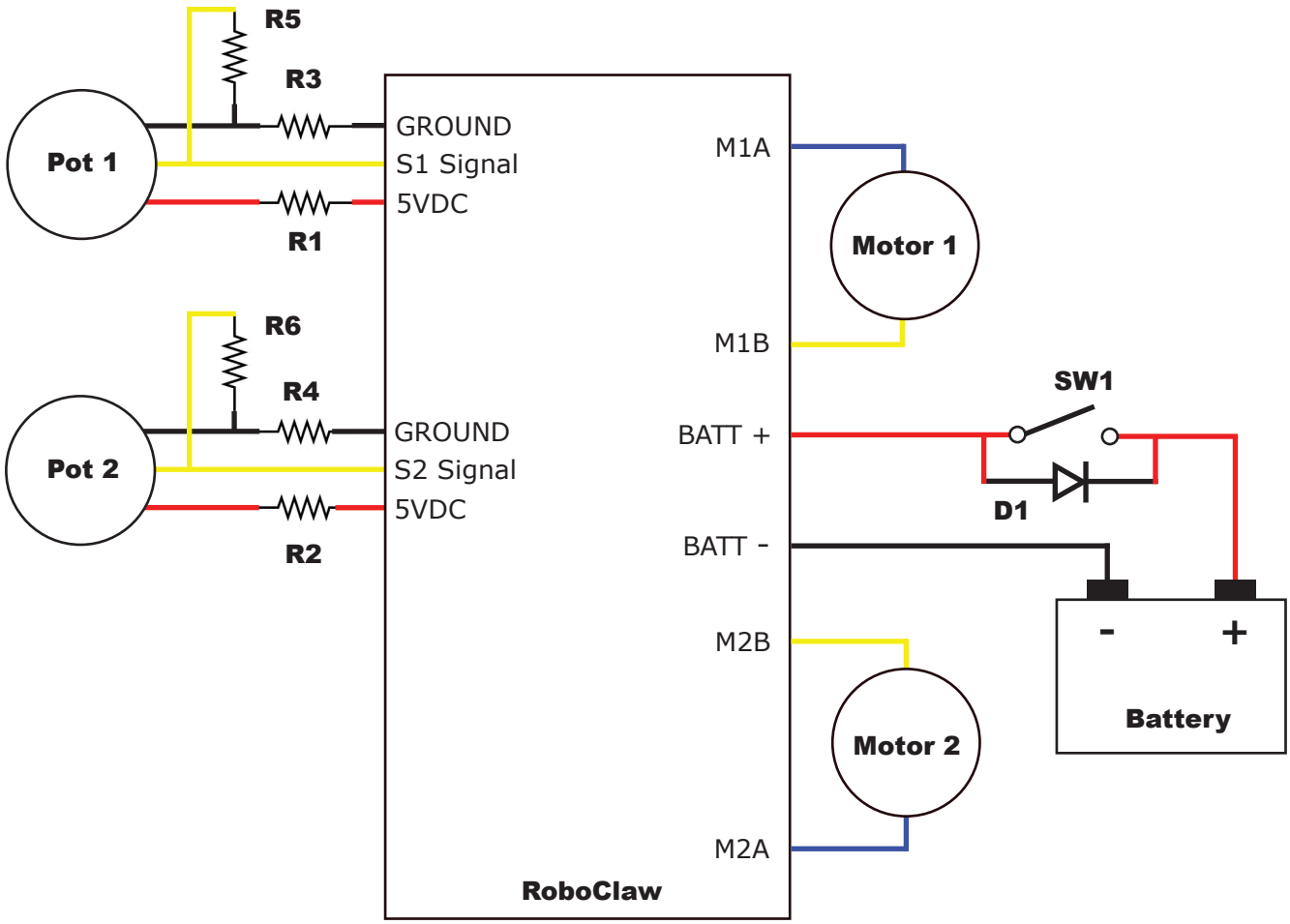
10k potentiometer, R1 / R2 = 16k

20k potentiometer, R1 / R2 = 33k

When Analog Error Detect is triggered, the controller immediately ramps motor power to 0 using the default deceleration rate. Behavior after reaching 0 power depends on the Idle Mode setting from the general settings -> Motors in Motion Studio:

Free Wheeling (default): The controller will freewheel after the idle delay time.

Eletric Braking: The controller will remain in braking mode once it reaches 0 duty.



Encoders

Closed Loop Modes

RoboClaw supports a wide range of encoders for closed loop operation. Encoders can be used for velocity control, position control, or a cascaded mode combining velocity and position control. This manual focuses on Quadrature and Absolute encoders, although additional encoder types are also supported.

Encoder Tuning

All encoders require tuning for proper operation. Motion Studio includes an Auto Tune function that can automatically configure the PID values, along with editable fields for manual adjustment. Tuning can also be performed using packet serial commands sent from a user device.

Encoder Signal Conditioning

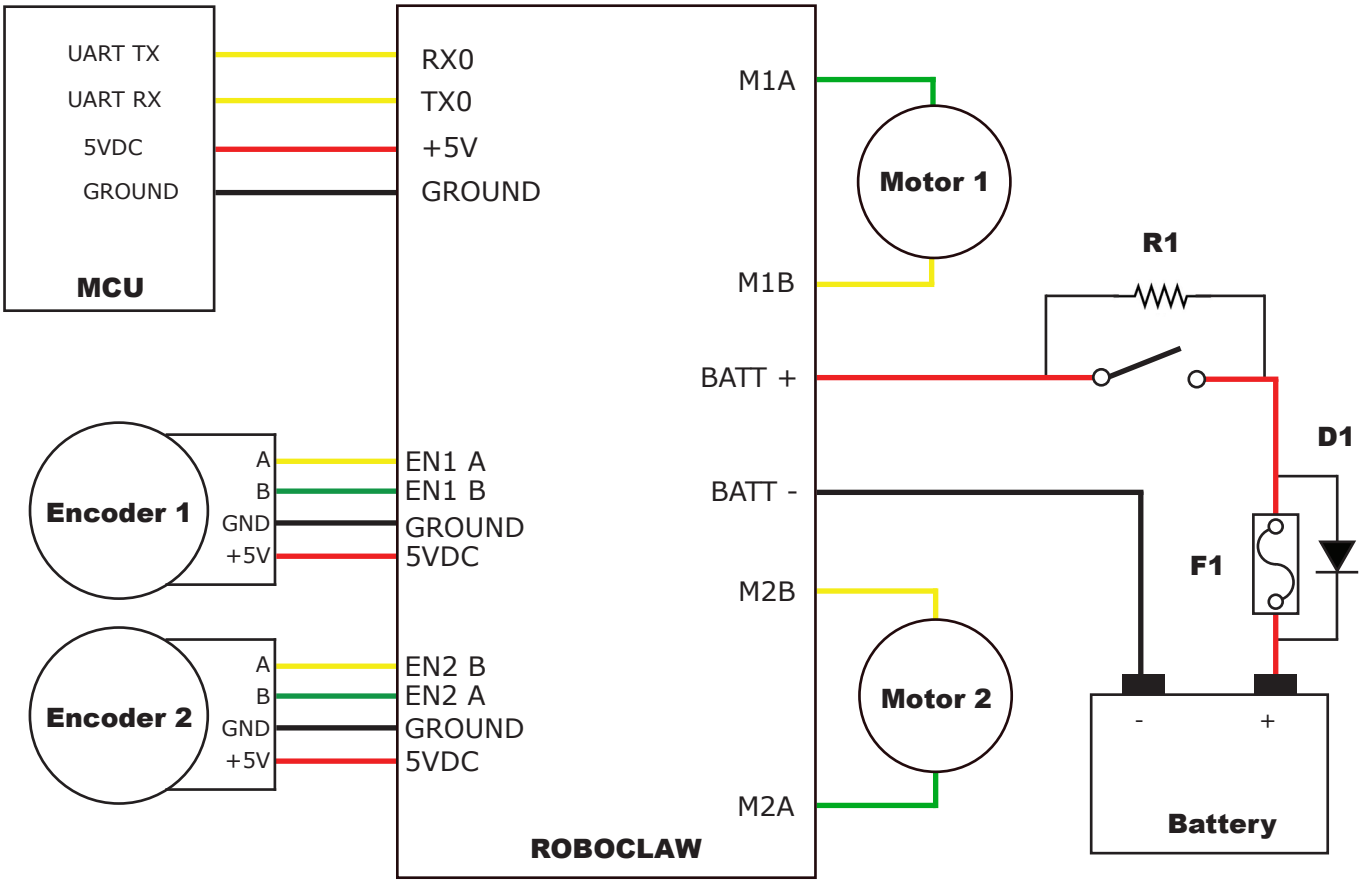
RoboClaw accepts direct quadrature encoder inputs but does not include built-in signal conditioning, as this would limit maximum count speed. Some encoders provide weak or noisy signals that may require external conditioning to ensure reliable operation. This can include adding pull-up resistors to open-collector or open-drain outputs, and using bypass capacitors for power filtering.

Users should verify encoder operation with an oscilloscope, checking both the signal lines and the encoder power supply. Some encoders may inject noise onto the power lines, which can interfere with RoboClaw and other connected devices. For applications in electrically noisy environments, differential encoders are strongly recommended to improve noise immunity and signal integrity.

Quadrature Encoder Wiring

The RoboClaw main header provides +5V and ground (GND) along with A and B input signals for each encoder.

Quadrature encoders are directional. In a two-motor robot, one motor typically rotates clockwise (CW) and the other counterclockwise (CCW). To ensure both encoders count up when the robot moves forward, the A and B inputs on one encoder must be reversed. If both encoders are connected with the leading edge on channel A, one will count up while the other counts down, causing functions such as Mixed Drive Forward to operate incorrectly.



Absolute Encoder Wiring

RoboClaw can read absolute encoders that output an analog voltage. As with the analog input modes for motor control, the encoder voltage must be between 0 V and 2 V. This range matches the input limits of RoboClaw's internal ADC and ensures accurate position measurement.

When using standard potentiometers as absolute encoders, the 5 V supply from RoboClaw must be divided down to 2 V at the potentiometer. This is done with a resistor divider on the 5 V line:

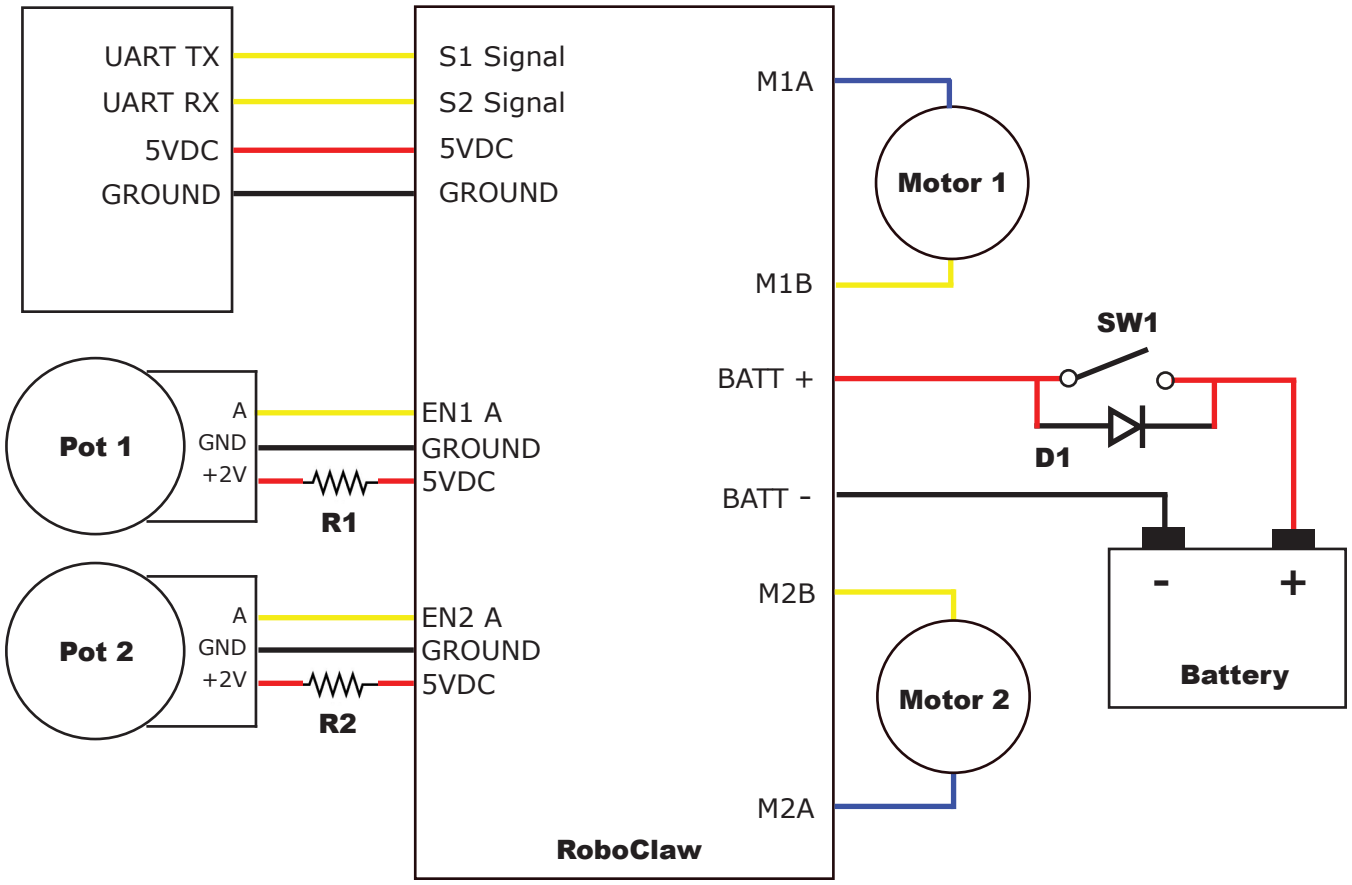
Formula:

R1 ≈ 1.5 × Potentiometer Value

Examples:

- 5 kΩ pot - R1 ≈ 7.5 kΩ
- 10 kΩ pot - R1 ≈ 15 kΩ
- 20 kΩ pot - R1 ≈ 30 kΩ

In the wiring diagram below the 5VDC output shown is only required if the MCU needs a power source. This is provided by RoboClaw's BEC feature. If the MCU has its own power supply, do not connect the 5 V output.



Encoder Tuning

To control motor speed or position with an encoder, the PID must be calibrated for the specific motor and encoder in use. Using Motion Studio, the PID can be tuned manually or with the Auto Tune function. Once tuned, the settings can be saved to onboard EEPROM and will be automatically reloaded each time the unit powers up.

The Velocity Settings window provides Auto Tune for velocity control. The Position Settings window supports tuning of a PD position controller, a PID position controller, or a cascaded Position-with-Velocity (PIV) controller. In cascaded mode, both velocity and position values are determined automatically, but the QPPS value must be set manually before starting. Auto Tune typically produces reasonable results, although manual adjustments may be required for optimal performance.

Auto Tuning

Motion Studio provides an Auto Tune option for velocity and position control. Before running Auto Tune, verify that the motor and encoder operate correctly in the desired direction and that basic PWM control functions as expected. It is recommended to confirm the motor and encoder combination is working properly before using the Auto Tune feature.

Incorrect motor or encoder wiring can cause Auto Tune to lock up and make the controller unresponsive. Fix the wiring and reset the controller to continue.

1. Go to the PWM Settings screen in Motion Studio.

2. Move the motor slider upward to drive the motor forward. Confirm that the encoder count increases. If the value decreases, reverse either the motor wires or the encoder wires and test again.

3. Before Auto Tune can run, the maximum speed of the motor and encoder must be set. For quadrature encoders, the maximum Quadrature Pulses Per Second (QPPS) is the highest speed achievable by the motor and encoder. For absolute encoders, QPPS corresponds to the encoder's maximum rotational speed. Refer to the encoder datasheet to ensure its speed rating is not exceeded. Auto Tune for position control cannot automatically measure QPPS, since position systems usually have limited movement range.

4. To determine QPPS, use the PWM Settings screen to run the motor at 100% duty by moving the slider to full forward or reverse. Record the value shown in the M1 Speed or M2 Speed fields at the top of the window. This value is the maximum QPPS. If the motor cannot be run at full speed due to physical constraints, an estimated maximum encoder count per second must be used.

5. Enter the QPPS value into the QPPS field under Settings, making sure to assign the correct value to the corresponding motor channel. Even identical motors and encoders may have slightly different maximum QPPS values, so measure each channel individually.

6. Start Auto Tune by selecting the Auto Tune button for the motor channel to be tuned. The Auto Tune process will calculate and apply the PID values for that channel.

Manual Velocity Calibration Procedure

1. **Determine QPPS**
Find the quadrature pulses per second (QPPS) value for the motor. The simplest method is to run the motor at 100% duty in Motion Studio and read the encoder speed value. If physical constraints prevent running at full speed, estimate the maximum encoder counts per second the motor can produce.

2. **Set Initial PID Values**
In the Velocity Control window, set P = 1, I = 0, D = 0. Move the motor using the slider controls in Motion Studio.

- If the motor does not move, it may be wired incorrectly or the P value may need to be increased.
- If the motor immediately runs at maximum speed, the motor or encoder wires are reversed. In this case, the controller attempts to reach the target speed but receives encoder feedback in the opposite direction, causing it to drive at full power. Reverse either the motor wires or the encoder wires (not both) and retest.

3. **Adjust P Value**
With basic control established, set a moderate speed and gradually increase P until the actual speed approaches the setpoint. If the motor begins vibrating at higher P values, reduce P to approximately two-thirds of that value before proceeding.

4. **Adjust I Value**
Increase I in increments of 0.1. The I term helps the motor reach and hold the exact target speed. An I value set too high will cause oscillation: the motor overshoots, then undershoots, producing a back-and-forth vibration. Reduce I if this occurs.

5. **Adjust D Value (if required)**
In most cases, D = 0 is sufficient. The D term is only needed if stable control cannot be achieved with P and I alone. D can help dampen overshoot, allowing higher P and I values, but it also introduces noise that may cause speed oscillations.

Manual Position Calibration Procedure

1. **Verify QPPS Value**
Position mode requires the QPPS value from Velocity mode to be set as described previously. For simple position control, set the Velocity PID values (P = 0, I = 0, D = 0).

2. **Initial Position PID Setup**
Set Position I = 0 and D = 0. Set Position P = 2000 as a starting point. To test the motor, a Speed value must also be entered. A good starting value is the same as the QPPS (maximum motor speed). Configure the minimum and maximum position values to safe limits:

- Motors without mechanical stops: use ±2 billion.
- Motors with physical dead stops (e.g., linear actuators): manually move the motor to each stop to determine the limits, and leave a safety margin before each stop.

When using quadrature encoders, the motor must be homed on every power-up since quadrature counts are relative to the starting position unless explicitly set or reset.

3. **Tune P and D**
At this stage the motor should move and stop, though not necessarily close to the target position. Gradually increase P until overshoot occurs when changing position with the slider. Then increase D (with I still set to 0). D provides damping and reduces overshoot as the motor approaches the target. D is often 5–20 times larger than P, but this ratio varies. Adjust P and D iteratively until the motor moves smoothly and stops reasonably close to the target. This forms a simple PD control.

4. **Consider I Adjustment**
Once position control is stable, I can be adjusted to help reach the exact setpoint. However, most systems with mechanical play or backlash will exhibit oscillation ("hunting") when I is introduced. To mitigate this, configure a deadzone, which defines a range around the target position that the controller treats as acceptable.

5. **Adjust Imax**
The Imax value limits the maximum wind-up of the I calculation. Increasing Imax allows I to influence a larger range of movement but also raises the risk of oscillation if the system is not tuned correctly or if Imax is set too high.

Serial Communications

Serial Communications Introduction

RoboClaw supports multiple serial communication modes for control, monitoring, and configuration. These modes provide flexible options to interface with a wide range of host devices, including microcontrollers, single-board computers, and PCs. Depending on the application, serial communication can be used for simple one-way commands, full bidirectional packet-based control, or advanced feedback and status reporting.

RoboClaw's serial I/O accepts 5 V input signals and provides 3.3 V outputs, making it compatible with a wide range of interfaces without requiring additional level shifting.

PC Serial Connections

When connecting RoboClaw directly to a PC through a standard RS-232 serial port, level shifting is required. RoboClaw's inputs accept 5 V logic signals, while RS-232 ports on a PC operate at higher voltage levels that are not directly compatible. A device such as a MAX232 or equivalent level shifter must be used to safely convert between RS-232 signal levels and RoboClaw's logic levels. Connecting an RS-232 port directly to RoboClaw without level shifting will result in damage to the controller.

Example Libraries

Official RoboClaw libraries are available to streamline development and provide reliable communication using the Packet Serial protocol. These libraries handle CRC16 generation, byte formatting, and packet structure, allowing development with simple, high-level functions instead of low-level serial code.

Libraries are provided for the following platforms:

Arduino (C++) – Includes functions like ForwardM1() and ReadEncM1() for direct motor control and encoder access

C# (.NET for Windows and Mono for Linux)

Python (Raspberry Pi, Linux, macOS, etc.)

Libraries can be downloaded from the BasicMicro Downloads page:
https://www.basicmicro.com/downloads_ep_45-1.html

Source code is also available on GitHub:
https://github.com/basicmicro

Additional tutorials, wiring guides, and usage examples can be found at:
https://resources.basicmicro.com

Custom Code and Library Support

BasicMicro offers support for custom code and library development to assist with specialized applications or unique integration requirements. This service helps streamline development by providing tailored examples, modified libraries, or complete code solutions based on project needs. Contact support for more information on custom development options.

Simple Serial

Simple Serial communication does not include error correction. Packet Serial is recommended for reliable communication.

Simple Serial Mode

In Simple Serial mode, RoboClaw operates as a receive only device. S1 accepts TTL level byte commands. A standard 8N1 format is used. Which is 8 bits, no parity bits and 1 stop bit. If you are using a microcontroller you can interface directly to RoboClaw. If you are using a PC a level shifting circuit (eg: Max232) is required.

In Simple Serial Mode, Sending updates every 50 ms or faster is advised to minimize the impact of communication errors.

Simple Serial Buad Rates

When the motor controller is set to Simple Serial mode in Motion Studio, the Baudrate option becomes available. The RoboClaw supports reliable communication at baud rates from 2400 up to 460800.

Baud Rate Options
2400
9600
19200
38400
57600
115200
230400
460800

Timeout

Sets the communication timeout period (0–25.5 seconds). If no valid command is received within the selected time, RoboClaw will automatically stop both motors.

This feature provides a safety mechanism: if communication is lost, interrupted, or the controlling device crashes, the motors will not continue running indefinitely. A value of 0 disables the timeout function.

Slave Select

Enables the slave select feature in Simple Serial mode. When enabled, the S2 pin acts as an enable/disable control for the RoboClaw:

- S2 High - RoboClaw is enabled and will respond to commands.
- S2 Low - RoboClaw ignores all commands.

This option is useful when multiple RoboClaw controllers share the same serial line. By toggling the S2 pin, you can select which controller is active while others remain idle, preventing command conflicts.

Standard Serial Syntax

In Simple Serial mode, RoboClaw is controlled using a single-byte command. The command value determines both direction and power or stop for each channel. This byte value (0–255, or –128 to 127) is divided between the two motor channels:

- Channel 1: responds to values 1–127
- Channel 2: responds to values 128–255 (or –1 to –127)
- Value 0: stops both channels

The table below lists the available command values and their associated functions. These commands provide basic one-way control without error checking or feedback.

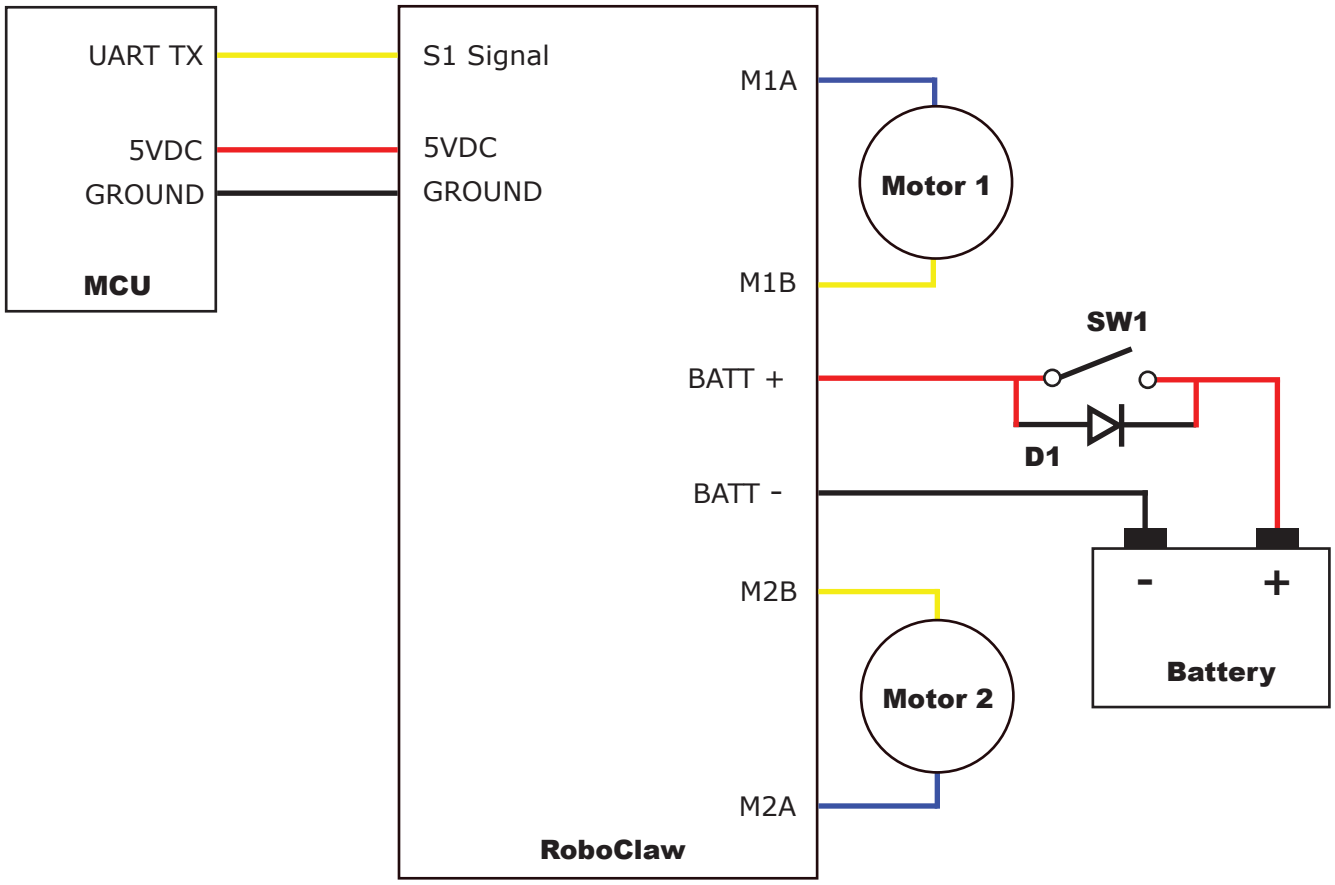
Binary	Unsigned	Signed	Action
0000000	0	0	Stop Channel 1 & 2
00000001	1	+1	Channel 1 – Full Reverse
01000000	64	+64	Channel 1 – Stop
01111111	127	+127	Channel 1 – Full Forward
10000000	128	–128	Channel 2 – Full Reverse
11000000	192	–64	Channel 2 – Stop
11111111	255	–1	Channel 2 – Full Forward

Simple Serial Wiring Example

The follwoing wiring diagram is a basic RoboClaw setup for Simple Serial operation using the main battery as the sole power source.

The 5 VDC connection, provided by RoboClaw's Battery Elimination Circuit (BEC), should only be used if the microcontroller (MCU) requires power from RoboClaw. If the MCU has its own independent power supply, the 5 VDC line must not be connected.

The 5 VDC output from RoboClaw's BEC should only be used if the MCU requires power. If the MCU has its own supply, do not connect the 5 VDC line.



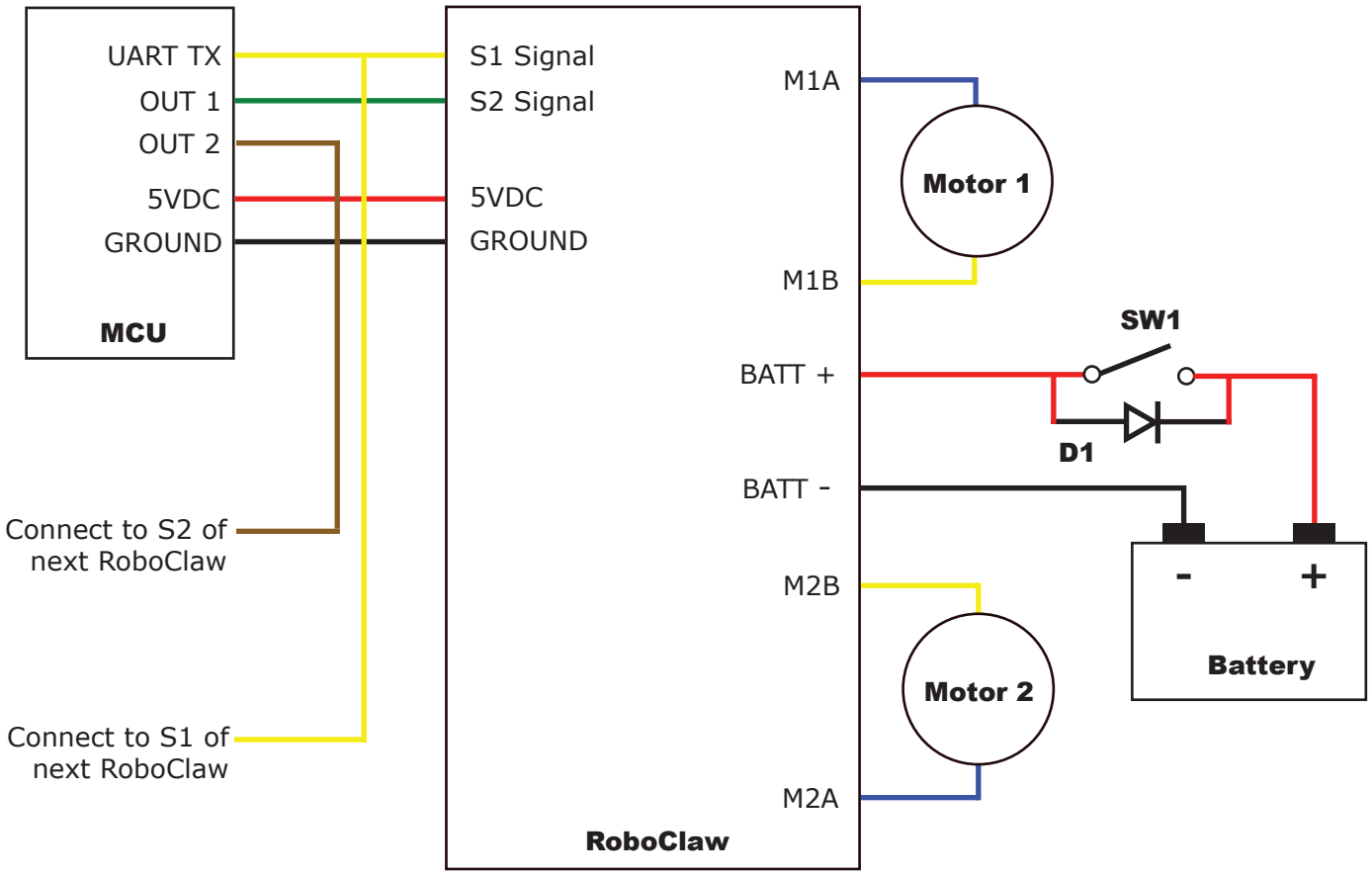
Simple Serial Mode With Slave Select

Slave select is used when more than one RoboClaw is on the same serial bus. When slave select is set to ON the S2 pin becomes the select pin. Set S2 high (5V) and RoboClaw will execute the next set of commands sent to S1 pin. Set S2 low (0V) and RoboClaw will ignore all received commands.

Any RoboClaw connected to a bus must share a common signal ground (GND) shown by the black wire. The S1 pin of RoboClaw is the serial receive pin and should be connected to the transmit pin of the MCU. All RoboClaw's S1 pins will be connected to the same MCU transmit pin. Each RoboClaw S2 pin should be connected to a unique I/O pin on the MCU. S2 is used as the control pin to activate the attached RoboClaw. To enable a RoboClaw hold its S2 pin high otherwise any commands sent are ignored.

The 5VDC connection, which utilizes RoboClaw's BEC (Battery Elimination Circuit) feature, is only necessary if your microcontroller (MCU) requires a power source. If your MCU has an independent power supply, do not connect the 5VDC line.

Consult the specific RoboClaw model's datasheet for detailed pinout information.



Packet Serial

Packet Serial Mode

Packet serial is a buffered bidirectional serial mode. The basic command structures consist of an address byte, command byte, data bytes and a CRC16 16bit checksum. The amount of data each command will send or receive can vary.

Libraries and Resources

Official RoboClaw libraries are available to streamline development and provide reliable communication using the Packet Serial protocol. These libraries handle CRC generation, byte formatting, and packet structure, allowing development with simple, high-level functions instead of low-level serial code. Libraries are available at www.basicmicro.com and www.github.com/basicmicro. Tutorials and examples can be found at resources.basicmicro.com.

Packet Serial Address

When using packet serial over the TTL serial pins (S1/S2), each RoboClaw on the bus must have a unique address. Up to 8 devices can share the same RS232 port when properly wired. The address is set in Motion Studio using the "Packet Serial Address" dropdown. The default address is 128 and does not need to be changed if you're only using one RoboClaw.

When sending packet serial commands over USB, the address byte can be any value from 0x80 to 0x87, since each USB connection is already treated as a unique device.

Packet Serial Settings

Label	Function	Description
2	Packet Serial Address	Sets the RoboClaw's address for packet serial mode, enabling control of multiple units from a single serial port using unique addresses. Default address is 128.
2	Baudrate	Sets the serial communication speed. Default is 2400
2	Timeout	Sets the communication timeout (0–25.5 seconds). If no serial data (at least one byte) is received within the set time, the motors will stop. A setting of 0 disables this feature.
2	Simple Serial	Enables slave select for Simple Serial mode. Set pin S2 high to enable the attached RoboClaw. Pull S2 low and all commands will be ignored. Used when configuring a serial network.

Packet Timeout

RoboClaw enforces a 10 ms inter-byte timeout during packet reception. If any byte in a packet arrives more than 10 ms after the previous one, any data received up to that point is discarded. This prevents incomplete or corrupted packets from being processed.

To recover from a communication error, the device communicating with RoboClaw should introduce a 10 ms delay before sending a new packet. This allows RoboClaw’s buffer to reset automatically.

Packet Acknowledgement

RoboClaw sends an acknowledgment byte (0xFF) after successfully receiving and validating write-only packet commands, such as motor control or settings commands. If the packet is invalid or corrupted, no acknowledgment is sent.

CRC16 Checksum Calculation

RoboClaw uses a CRC16 (Cyclic Redundancy Check) to validate all packet communications. This method is more robust than a basic checksum and helps prevent corrupted data from causing unintended behavior.

CRC16 is used both when receiving commands and validating responses. For incoming packets, RoboClaw checks the CRC16 to ensure the data is valid before processing it. For received data, the CRC16 can be used to verify integrity on the host side. To validate a response, calculate the CRC16 using the address byte, command byte, and all data bytes—excluding the final two CRC16 bytes. If the calculated value matches the received CRC16, the data was transmitted without error.

CRC16 Checksum Example

The following Arduino function calculates a 16-bit Cyclic Redundancy Check (CRC) for RoboClaw serial communication packets. This checksum ensures data integrity by detecting transmission errors. The function uses the CCITT CRC16 polynomial (0x1021) and should be called with your command packet data and byte count. The resulting CRC value must be appended to your serial commands when communicating with the RoboClaw controller from your Arduino.

```
//Calculates CRC16 of nBytes of data in byte array message
unsigned int crc16(unsigned char *packet, int nBytes) {
    unsigned int crc = 0;  // Initialize CRC to 0

    for (int byte = 0; byte < nBytes; byte++) {
        crc = crc ^ ((unsigned int)packet[byte] << 8);
        for (unsigned char bit = 0; bit < 8; bit++) {
            if (crc & 0x8000) {
                crc = (crc << 1) ^ 0x1021;
            } else {
                crc = crc << 1;
            }
        }
    }
    return crc;
}

// Example usage function
void setup() {
    Serial.begin(9600);

    // Test the CRC function
    unsigned char testData[] = {0x80, 0x00};  // Example RoboClaw command
    unsigned int result = crc16(testData, 2);

    Serial.print("CRC16 result: 0x");
    Serial.println(result, HEX);
}

void loop() {
    // Main loop code here
}
```

Multi-Byte Packet Serial Commands

Some RoboClaw Packet Serial commands use values larger than a single byte. These are usually 2-byte (word) or 4-byte (long) values. Since most microcontrollers including the Arduino send one byte at a time over serial, larger values will need to be split into individual bytes before sending and recombined when receiving. RoboClaw expects big-endian order, most significant byte (MSB) comes first.

Splitting 32-bit long into 4 bytes

```
unsigned long myValue = 123456789;
byte byte3 = myValue >> 24;        // Highest byte (MSB)
byte byte2 = myValue >> 16;
byte byte1 = myValue >> 8;
byte byte0 = myValue;              // Lowest byte (LSB)
```

Splitting 16-bit word into 2 bytes

```
unsigned int myValue = 30000;
byte byte1 = myValue >> 8;         // High byte
byte byte0 = myValue;              // Low byte
```

Rebuilding 16-bit long from 2 bytes

```
unsigned int myValue = (byte1 << 8) | byte0;
```

Rebuilding 32-bit long from 4 bytes

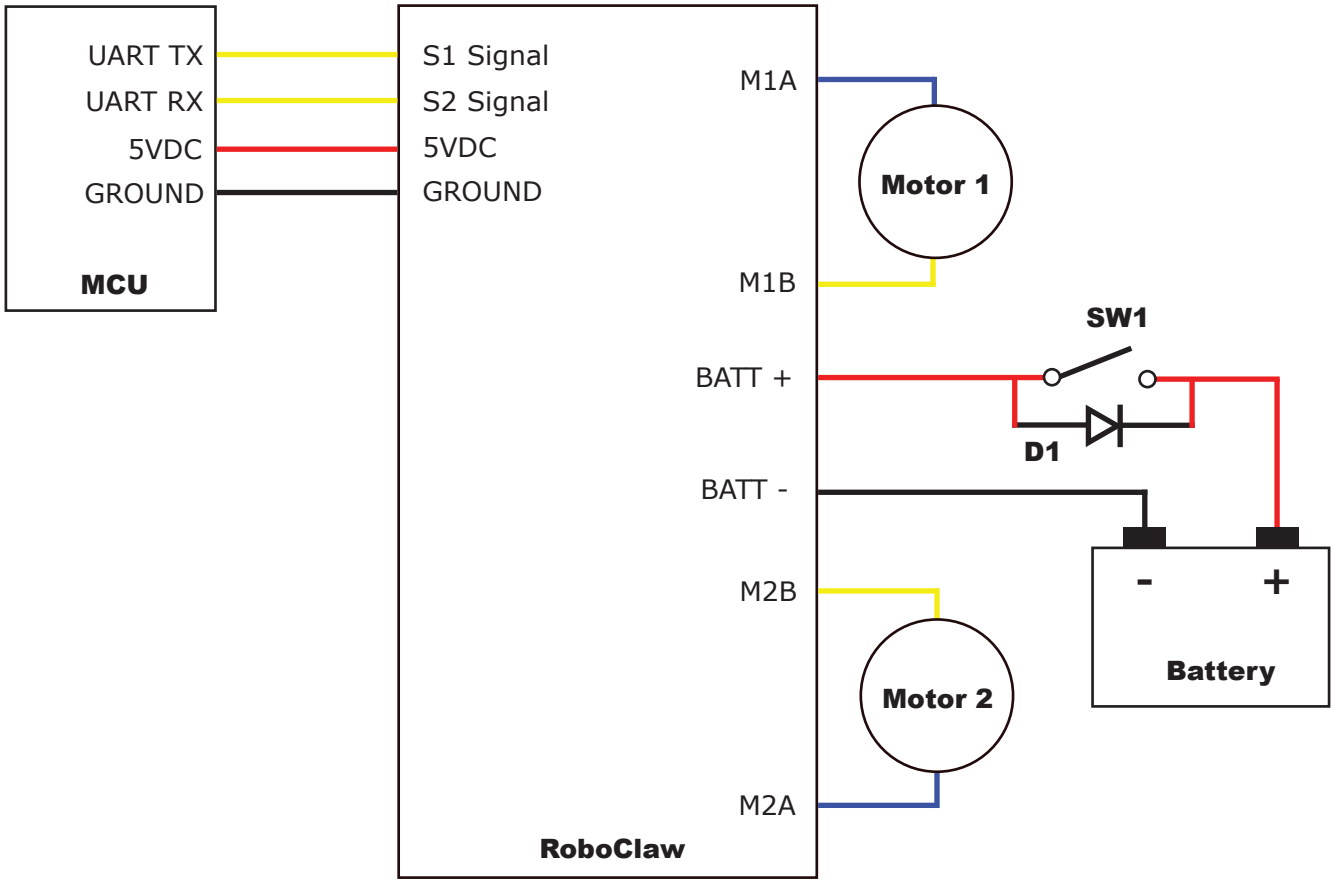
```
unsigned long myValue = ((unsigned long)byte3 << 24) |
                        ((unsigned long)byte2 << 16) |
                        ((unsigned long)byte1 << 8)  |
                        byte0;
```

Packet Serial Wiring

In packet serial mode the RoboClaw can transmit and receive serial data. A microcontroller with a UART is recommended. The UART will buff er the data received from RoboClaw. When a request for data is made to RoboClaw the return data will have at least a 1ms delay after the command is received if the baud rate is set at or below 38400. This will allow slower processors and processors without UARTs to communicate with RoboClaw.

The diagram below illustrates the main battery as the sole power source. The 5VDC connection is only required if your MCU needs a power supply. This voltage is provided by RoboClaw's built-in BEC (Battery Eliminator Circuit).

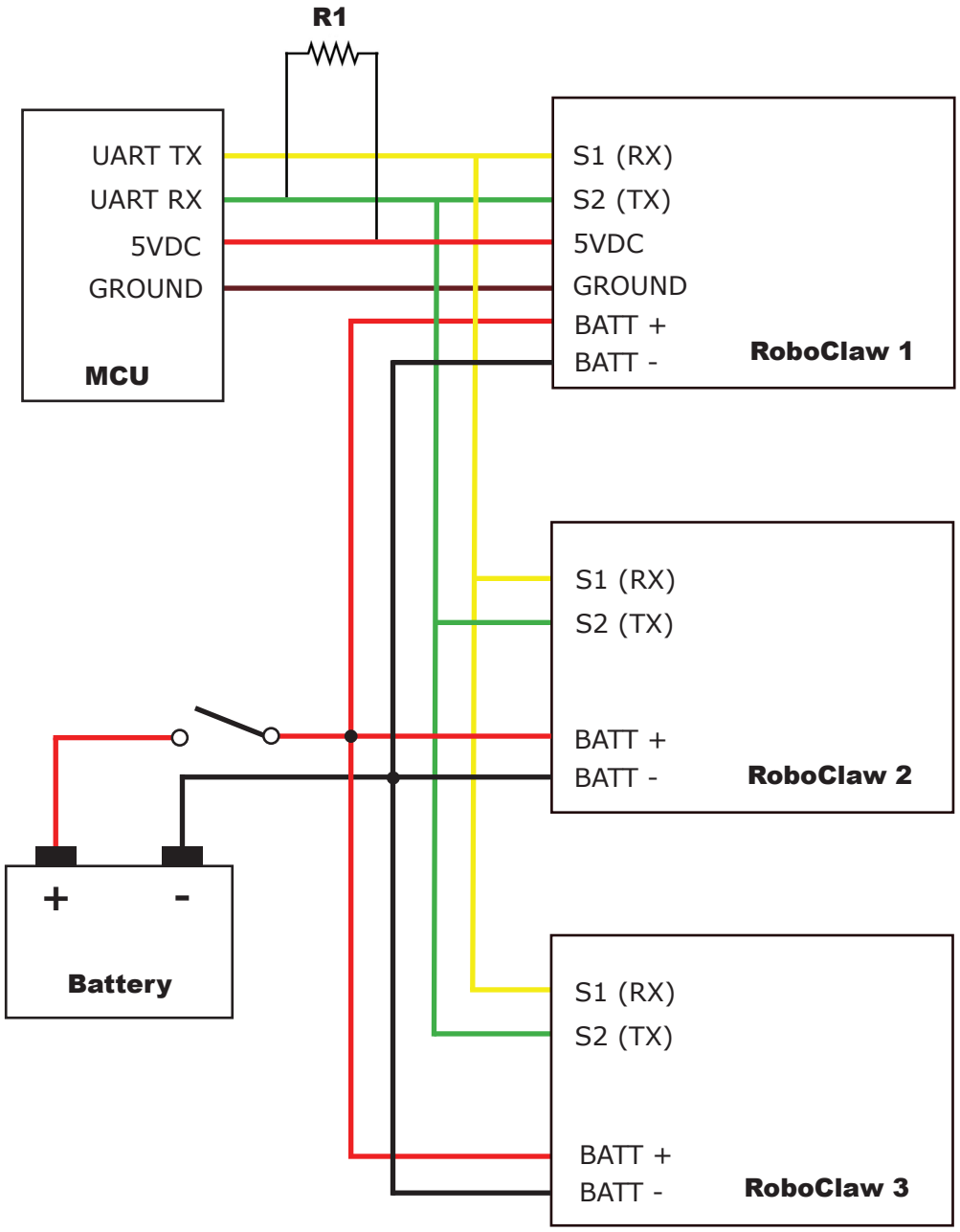
The 5 VDC output from RoboClaw's BEC should only be used if the MCU requires power. If the MCU has its own supply, do not connect the 5 VDC line.



Multi-Unit Packet Serial Wiring

In Packet Serial mode, up to eight RoboClaw units can be controlled by one serial connection. Each RoboClaw must have Multi-Unit Mode enabled and be assigned a unique packet serial address, which can be configured in Motion Studio.

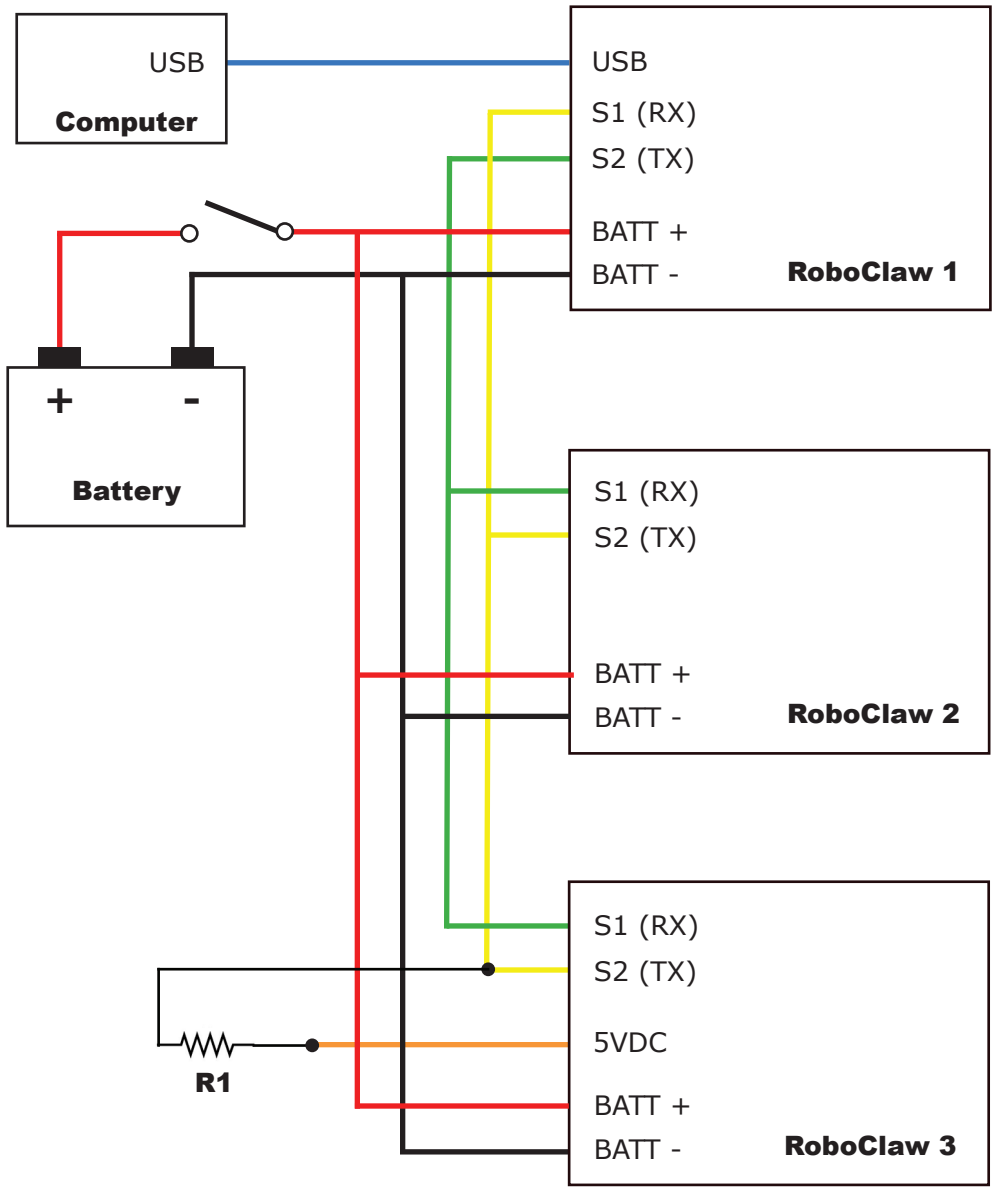
Connect the S1 and S2 pins of each RoboClaw directly to the MCU's TX and RX pins. A pull-up resistor (R1) is required on the MCU RX pin, with a recommended value between 1 kΩ and 4.7 kΩ.



Multi-Unit Wiring With USB

In Packet Serial mode, up to eight RoboClaw units can be controlled through a single USB connection. Each RoboClaw must have Multi-Unit Mode enabled and be assigned a unique packet serial address, which can be configured in Motion Studio.

When multiple RoboClaws are used in Packet Serial mode, one unit is connected to the USB host (PC or SBC) and acts as the primary interface. Connect S1 on the primary RoboClaw to S2 on each additional RoboClaw, and connect S2 on the primary RoboClaw to S1 on each additional RoboClaw. A pull-up resistor (R1) is required on the S2 pin of the RoboClaw, with a recommended value between 1 kΩ and 4.7 kΩ.



Packet Serial Commands

Command Groups

The Packet Serial command set is divided into functional groups. Status commands provide feedback such as encoder counts, motor speed, currents, voltages, and error states. Settings commands configure controller parameters including PID values, voltage and current limits, and operating modes. Open loop commands control motor channels directly without encoder feedback. Closed loop commands use encoder feedback for velocity and position control, enabling accurate and repeatable operation.

Status Commands

Status commands provide feedback from RoboClaw, allowing the host system to monitor performance and operating conditions in real time. These commands can be used to read encoder counts, motor speeds, PWM values, currents, voltages, temperatures, and error states, as well as reset or set encoder values when needed. By using these commands, applications can track motor position, verify system health, detect faults, and adjust control logic dynamically

Command	Description
16	Get Motor 1 Encoder
17	Get Motor 2 Encoder
18	Get Motor 1 Speed
19	Get Motor 2 Speed
20	Reset Encoders
21	Get Fimrware Version
22	Set Motor 1 Encoder Count
23	Set Motor 2 Encoder Count
24	Get Main Battery Voltage
25	Get Logic Battery Voltage
30	Get Motor 1 Instantaneous Speed
31	Get Motor 2 Instantaneous Speed
47	Get Buffers
48	Get PWM Values
49	Get Currents
73	Get Status (all status information)
78	Get Encoder Values
79	Get Instantaneous Speeds
82	Get Temperature 1
83	Get Temperature 2
90	Get Error
100	Get Main and Logic Voltage
101	Get Temperatures
103	Get Encoder Status Bits
108	Get Speeds (encoder counts per second)
111	Get Speed Errors (encoder counts per second)
114	Get Position Errors (encoder counts)

16 - Read Encoder Count/Value M1

Read M1 encoder count/position.

```
Send: [Address, 16]
Receive: [Enc1(4 bytes), Status, CRC16(2 bytes)]
```

Quadrature encoders have a range of 0 to 4,294,967,295. Absolute encoder values are converted from an analog voltage into a value from 0 to 2047 for the full 2v range.

The status byte tracks counter underflow, direction and overflow. The byte value represents:

Bit0 - Counter Underflow (1= Underflow Occurred, Clear After Reading)
Bit1 - Direction (0 = Forward, 1 = Backwards)
Bit2 - Counter Overflow (1= Underflow Occurred, Clear After Reading)
Bit3 - Reserved
Bit4 - Reserved
Bit5 - Reserved
Bit6 - Reserved
Bit7 - Reserved

17 - Read Quadrature Encoder Count/Value M2

Read M2 encoder count/position.

```
Send: [Address, 17]
Receive: [EncCnt(4 bytes), Status, CRC16(2 bytes)]
```

Quadrature encoders have a range of 0 to 4,294,967,295. Absolute encoder values are converted from an analog voltage into a value from 0 to 2047 for the full 2v range.

The Status byte tracks counter underflow, direction and overflow. The byte value represents:

Bit0 - Counter Underflow (1= Underflow Occurred, Cleared After Reading)
Bit1 - Direction (0 = Forward, 1 = Backwards)
Bit2 - Counter Overflow (1= Underflow Occurred, Cleared After Reading)
Bit3 - Reserved
Bit4 - Reserved
Bit5 - Reserved
Bit6 - Reserved
Bit7 - Reserved

18 - Read Encoder Speed M1

Read M1 counter speed. Returned value is in pulses per second. RoboClaw keeps track of how many pulses received per second for both encoder channels.

```
Send: [Address, 18]
Receive: [Speed(4 bytes), Status, CRC16(2 bytes)]
```

Status indicates the direction (0 – forward, 1 - backward).

19 - Read Encoder Speed M2

Read M2 counter speed. Returned value is in pulses per second. RoboClaw keeps track of how many pulses received per second for both encoder channels.

```
Send: [Address, 19]
Receive: [Speed(4 bytes), Status, CRC16(2 bytes)]
```

Status indicates the direction (0 – forward, 1 - backward).

20 - Reset Quadrature Encoder Counters

Will reset both quadrature decoder counters to zero. This command applies to quadrature encoders only.

```
Send: [Address, 20, CRC16(2 bytes)]
Receive: [0xFF]
```

21 - Read Firmware Version

Read RoboClaw firmware version. Returns up to 48 bytes(depending on the Roboclaw model) and is terminated by a line feed character and a null character.

```
Send: [Address, 21]
Receive: [“RoboClaw 10.2A v4.1.11”,10,0, CRC16(2 bytes)]
```

The command will return up to 48 bytes. The return string includes the product name and firmware version. The return string is terminated with a line feed (10) and null (0) character.

22 - Set Quadrature Encoder 1 Value

Set the value of the Encoder 1 register. Useful when homing motor 1. This command applies to quadrature encoders only.

```
Send: [Address, 22, Value(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

23 - Set Quadrature Encoder 2 Value

Set the value of the Encoder 2 register. Useful when homing motor 2. This command applies to quadrature encoders only.

```
Send: [Address, 23, Value(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

24 - Read Main Battery Voltage Level

Read the main battery voltage level connected to B+ and B- terminals. The voltage is returned in 10ths of a volt(eg 300 = 30v).

```
Send: [Address, 24]
Receive: [Value(2 bytes), CRC16(2 bytes)]
```

25 - Read Logic Battery Voltage Level

Read a logic battery voltage level connected to LB+ and LB- terminals. The voltage is returned in 10ths of a volt(eg 50 = 5v).

```
Send: [Address, 25]
Receive: [Value.Byte1, Value.Byte0, CRC16(2 bytes)]
```

30 - Read Raw Speed M1

Read the pulses counted in that last 300th of a second. This is an unfiltered version of command 18. Command 30 can be used to make a independent PID routine. Value returned is in encoder counts per second.

```
Send: [Address, 30]
Receive: [Speed(4 bytes), Status, CRC16(2 bytes)]
```

The Status byte is direction (0 – forward, 1 - backward).

31 - Read Raw Speed M2

Read the pulses counted in that last 300th of a second. This is an unfiltered version of command 19. Command 31 can be used to make a independent PID routine. Value returned is in encoder counts per second.

```
Send: [Address, 31]
Receive: [Speed(4 bytes), Status, CRC16(2 bytes)]
```

The Status byte is direction (0 – forward, 1 - backward).

47 - Read Buffer Length

Read both motor M1 and M2 buffer lengths. This command can be used to determine how many commands are waiting to execute.

```
Send: [Address, 47]
Receive: [BufferM1, BufferM2, CRC16(2 bytes)]
```

The return values represent how many commands per buffer are waiting to be executed. The maximum buffer size per motor is 64 commands(0x3F). A return value of 0x80(128) indicates the buffer is empty. A return value of 0 indiciates the last command sent is executing. A value of 0x80 indicates the last command buffered has finished.

48 - Read Motor PWM values

Read the current PWM output values for the motor channels. The values returned are +/-32767. The duty cycle percent is calculated by dividing the Value by 327.67.

```
Send: [Address, 48]
Receive: [M1 PWM(2 bytes), M2 PWM(2 bytes), CRC16(2 bytes)]
```

49 - Read Motor Currents

Read the current from each motor in 10ma increments. The amps value is calculated by dividing the value by 100.

```
Send: [Address, 49]
Receive: [M1 Current(2 bytes), M2 Currrent(2 bytes), CRC16(2 bytes)]
```

Current readings are minimally filtered due to the high speed control loop and represent short sampling intervals, reported values may different from instruments such as a DVM which reports a long-term averaged measurement. (See Operational Requirements)

Motor current will not be equal to battery current. When operating below 100% duty, the motor only receives a fraction of the battery voltage during each PWM cycle. To produce the same torque at a lower effective voltage, the motor draws a higher current. The battery supplies only the average current, while the motor sees short, higher current pulses. For example, using a 24 VDC battery:

- 25% duty: Motor ~6V, about 4× battery current
- 50% duty: Motor ~12V, about 2× battery current
- 75% duty: Motor ~18V, about 1.3× battery current

73 - Read All Status information

Returns a complete set of RoboClaw status data, including system tick, error flags, temperatures, voltages, currents, encoders, and speed/position information. The timertick counter runs at 20 kHz and starts at power-up.

```
Send:[Address, 73]
Receive:[timertick(4 bytes),
                errors/warnings(4 bytes),
                temp1(2 bytes),
                temp2(2 bytes),
                Main Battery Voltage(2 bytes),
                Logic Battery Voltage(2 bytes),
                M1 current Duty(2 bytes),
                M2 current Duty(2 byte),
                M1 Current(2 bytes),
                M2 Current(2 bytes),
                M1 Encoder(4 bytes),
                M2 Encoder(4 bytes),
                M1 Speed(4 bytes),
                M2 Speed(4 bytes),
                M1 ISpeed(4 bytes),
                M2 ISpeed(4 bytes),
                M1 Speed Errors(2 bytes),
                M2 Speed Errors(2 bytes),
                M1 Position Errors(2 bytes),
                M2 Position Errors(2 bytes),
                CRC16(2 bytes)]
```

This command consolidates data otherwise available through the following individual status commands:

- Temp1 – See GETTEMP (82)
- Temp2 – See GETTEMP2 (83)
- Main Battery Voltage – See GETMBATT (24)
- Logic Battery Voltage – See GETLBATT (25)
- M1/M2 Duty – See GETPWMS (48)
- M1/M2 Current – See GETCURRENTS (49)
- M1/M2 Encoder – See GETENCODERS (78)
- M1/M2 Speed – See GETSPEEDS (108)
- M1/M2 ISpeed – See GETISPEEDS (79)
- M1/M2 Speed Errors – See GETSPEEDERRORS (111)
- M1/M2 Position Errors – See GETPOSERRORS (114)

78 - Read Encoder Counters

Read M1 and M2 encoder counters. Quadrature encoders have a range of 0 to 4,294,967,295. Absolute encoder values are converted from an analog voltage into a value from 0 to 2047 for the full 2V analog range.

```
Send: [Address, 78]
Receive: [Enc1(4 bytes), Enc2(4 bytes), CRC16(2 bytes)]
```

79 - Read ISpeeds Counters

Read M1 and M2 instantaneous speeds. Returns the speed in encoder counts per second for the last 300th of a second for both encoder channels.

```
Send: [Address, 79]
Receive: [ISpeed1(4 bytes), ISpeed2(4 bytes), CRC16(2 bytes)]
```

82 - Read Temperature

Read the board temperature. Value returned is in 10ths of degrees.

```
Send: [Address, 82]
Receive: [Temperature(2 bytes), CRC16(2 bytes)]
```

83 - Read Temperature 2

Read the second board temperature(only on supported units). Value returned is in 10ths of degrees.

```
Send: [Address, 83]
Receive: [Temperature(2 bytes), CRC16(2 bytes)]
```

90 - Read Status

Read the current unit status.

```
Send: [Address, 90]
Receive: [Status, CRC16(2 bytes)]
```

Function	Status Bit Mask
Normal	0x0000
M1 OverCurrent Warning	0x0001
M2 OverCurrent Warning	0x0002
E-Stop	0x0004
Temperature Error	0x0008
Temperature2 Error	0x0010
Main Battery High Error	0x0020
Logic Battery High Error	0x0040
Logic Battery Low Error	0x0080
M1 Driver Fault	0x0100
M2 Driver Fault	0x0200
Main Battery High Warning	0x0400
Main Battery Low Warning	0x0800
Termperature Warning	0x1000
Temperature2 Warning	0x2000
M1 Home	0x4000
M2 Home	0x8000

100 - Get Volts

Returns the measured voltages of the main and logic batteries. A value of 125 corresponds to 12.5 VDC.

```
Send: [Address, 100]
Receive:[Main Battery Voltage(2 bytes), Logic Battery Voltage(2 bytes), CRC16(2 bytes)]
```

101 - Get Temperatures

Returns the readings from onboard temperature sensors 1 and 2. A value of 250 corresponds to 25 °C.

```
Send: [Address, 101]
Receive:[Temp1(2 bytes), Temp2(2 bytes), CRC16(2 bytes)]
```

103 - Get Encoder Status Bits

Returns status information for the Motor 1 and Motor 2 encoders, including direction and counter underflow/overflow conditions.

```
Send: [Address,103]
Receive:[M1 Encoder Status(1 byte), M2 Encoder Status(1 byte),CRC16(2 bytes)]
```

Encoder Status Bits:

Bit 0 – Counter Underflow (1 = underflow occurred, cleared after read)
Bit 1 – Direction (0 = forward, 1 = reverse)
Bit 2 – Counter Overflow (1 = overflow occurred, cleared after read)
Bit 3 – Reserved
Bit 4 – Reserved
Bit 5 – Reserved
Bit 6 – Reserved
Bit 7 – Reserved

108 - Read Motor Average Speeds

Read M1 and M2 average speeds. Return the speed in encoder counts per second for the last second for both encoder channels.

```
Send: [Address, 108]
Receive: [Speed1(4 bytes), Speed2(4 bytes), CRC16(2 bytes)]
```

111 - Read Speed Errors

Read current calculated speed error in encoder counts per second.

```
Send: [Address, 111]
Receive; [M1Error(4 bytes), M2Error(4 bytes), CRC16(2 bytes)]
```

114 - Read Position Errors

Read current calculated position error in encoder counts.

```
Send: [Address, 114]
Receive: [M1Error(4 bytes), M2Error(4 bytes], CRC16(2 bytes)]
```

Settings Commands

Settings commands configure both persistent settings and access runtime parameters on the RoboClaw. Persistent commands allow configuration of values such as PID gains, encoder modes, current limits, and voltage thresholds, which are stored in non-volatile memory and automatically reloaded at startup. Runtime commands provide access to live data such as motor currents, battery voltage, encoder counts, and other system metrics. These commands are essential for system setup, tuning, and real-time monitoring.

Command	Description
14	Set Serial Timeout
15	Read Serial Timeout
28	Set M1 Velocity PID settings
29	Set M2 Velocity PID settings
55	Read M1 Velocity PID settings
56	Read M2 Velocity PID settings
57	Set Main Voltage Limits
58	Set Logic Voltage Limits
59	Read Min/Max Main Voltage Limits
60	Read Min/Max Logic Voltage Limits
61	Set M1 Position PID settings
62	Set M2 Position PID settings
63	Read M1 Position PID settings
64	Read M2 Position PID settings
68	Set M1 Default Accel
69	Set M2 Default Accel
70	Set M1 Default Speed
71	Set M2 Default Speed
72	Read Default Speeds
74	Set Pin Functions
75	Read Pin Functions
76	Set RC/Analog Control Settings
77	Read RC/Analog Control Settings
80	Restore Defaults
81	Read Default Accelerations
91	Get Encoder Mode
92	Set M1 Encoder Mode
93	Set M2 Encoder Mode
96	Set USB Serial Number
97	Read USB Serial Number
98	Set Config
99	Get Config
102	Set Auxiliary Pin Duty Setting
104	Read Auxiliary Pin Duty Settings
105	Set M1 Homing Timeout
106	Set M2 Homing Timeout
107	Read Homing Timeout

Command	Description
109	Set Speed Error Limits (in encoder counts per second)
110	Read Speed Error Limits
112	Set Position Error Limits (in encoder counts)
113	Read Position Error Limits
133	Set M1 Max Current Limit
134	Set M2 Max Current Limit
135	Read M1 Max Current Limit
136	Read M2 Max Current Limit
148	Set PWM Modes (Inductive or Resistive)
149	Read PWM Modes
160	Set PWM Idle Delays and Idle Modes
161	Read PWM Idle Delays and Idle Modes
160	Set PWM Idle Delays and Idle Modes
161	Read PWM Idle Delays and Idle Modes

14 - Set Serial Timeout

Sets the serial communication timeout in 100ms increments. When serial bytes are received in the time specified both motors will stop automatically. Range is 0 to 25.5 seconds (0 to 255 in 100ms increments).

```
Send: [Address, 14, Value, CRC16(2 byes)]
Receive: [0xFF]
```

15 - Read Serial Timeout

Read the current serial timeout setting. Range is 0 to 255.

```
Send: [Address, 15]
Receive: [Value, CRC16(2 bytes)]
```

28 - Set Velocity PID Constants M1

Several motor and quadrature combinations can be used with RoboClaw. In some cases the default PID values will need to be tuned for the systems being driven. This gives greater flexibility in what motor and encoder combinations can be used. The RoboClaw PID system consist of four constants starting with QPPS, P = Proportional, I= Integral and D= Derivative. The defaults values are:

```
QPPS = 44000
P = 0x00010000
I = 0x00008000
D = 0x00004000
```

QPPS is the speed of the encoder when the motor is at 100% power. P, I, D are the default values used after a reset. Command syntax:

```
Send: [Address, 28, D(4 bytes), P(4 bytes), I(4 bytes), QPPS(4 byte), CRC16(2 bytes)]
Receive: [0xFF]
```

29 - Set Velocity PID Constants M2

Several motor and quadrature combinations can be used with RoboClaw. In some cases the default PID values will need to be tuned for the systems being driven. This gives greater flexibility in what motor and encoder combinations can be used. The RoboClaw PID system consist of four constants starting with QPPS, P = Proportional, I= Integral and D= Derivative. The defaults values are:

```
QPPS = 44000
P = 0x00010000
I = 0x00008000
D = 0x00004000
```

QPPS is the speed of the encoder when the motor is at 100% power. P, I, D are the default values used after a reset. Command syntax:

```
Send: [Address, 29, D(4 bytes), P(4 bytes), I(4 bytes), QPPS(4 byte), CRC16(2 bytes)]
Receive: [0xFF]
```

55 - Read Motor 1 Velocity PID and QPPS Settings

Read the PID and QPPS Settings.

```
Send: [Address, 55]
Receive: [P(4 bytes), I(4 bytes), D(4 bytes), QPPS(4 byte), CRC16(2 bytes)]
```

56 - Read Motor 2 Velocity PID and QPPS Settings

Read the PID and QPPS Settings.

```
Send: [Address, 56]
Receive: [P(4 bytes), I(4 bytes), D(4 bytes), QPPS(4 byte), CRC16(2 bytes)]
```

57 - Set Main Battery Voltages

Sets the main battery voltage cutoffs (minimum and maximum). Values are specified in tenths of a volt; multiply the voltage by 10 (e.g., 12.5 VDC = 125). An automatic maximum battery offset is also applied, which defaults to 2.0 V (value = 20).

```
Send: [Address, 57, Min(2 bytes), Max(2bytes), AutoMaxMainBatOffset(1 byte),
CRC16(2bytes)]
Receive: [0xFF]
```

58 - Set Logic Battery Voltages

Set the Logic Battery Voltages cutoffs, Min and Max. Min and Max voltages are in 10th of a volt increments. Multiply the voltage by 10 (12.5VDC = 125).

```
Send: [Address, 58, Min(2 bytes), Max(2bytes, CRC16(2 bytes)]
Receive: [0xFF]
```

59 - Read Main Battery Voltage Settings

Read the Main Battery Voltage Settings. The voltage is calculated by dividing the value by 10

```
Send: [Address, 59]
Receive: [Min(2 bytes), Max(2 bytes), AutoMaxMainBatOffset(1byte), CRC16(2 bytes)]
```

60 - Read Logic Battery Voltage Settings

Read the Logic Battery Voltage Settings. The voltage is calculated by dividing the value by 10

```
Send: [Address, 60]
Receive: [Min(2 bytes), Max(2 bytes), CRC16(2 bytes)]
```

61 - Set Motor 1 Position PID Constants

The RoboClaw Position PID system consist of seven constants starting with P = Proportional, I= Integral and D= Derivative, MaxI = Maximum Integral windup, Deadzone in encoder counts, MinPos = Minimum Position and MaxPos = Maximum Position. The defaults values are all zero.

```
Send: [Address, 61, D(4 bytes), P(4 bytes), I(4 bytes), MaxI(4 bytes),
      Deadzone(4 bytes), MinPos(4 bytes), MaxPos(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

Position constants are used only with the Position commands, 65,66 and 67 or when encoders are enabled in RC/Analog modes.

62 - Set Motor 2 Position PID Constants

The RoboClaw Position PID system consist of seven constants starting with P = Proportional, I= Integral and D= Derivative, MaxI = Maximum Integral windup, Deadzone in encoder counts, MinPos = Minimum Position and MaxPos = Maximum Position. The defaults values are all zero.

```
Send: [Address, 62, D(4 bytes), P(4 bytes), I(4 bytes), MaxI(4 bytes),
Deadzone(4 bytes), MinPos(4 bytes), MaxPos(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

Position constants are used only with the Position commands, 65,66 and 67 or when encoders are enabled in RC/Analog modes.

63 - Read Motor 1 Position PID Constants

Read the Position PID Settings.

```
Send: [Address, 63]
Receive: [P(4 bytes), I(4 bytes), D(4 bytes), MaxI(4 byte), Deadzone(4 byte),
MinPos(4 byte), MaxPos(4 byte), CRC16(2 bytes)]
```

64 - Read Motor 2 Position PID Constants

Read the Position PID Settings.

```
Send: [Address, 64]
Receive: [P(4 bytes), I(4 bytes), D(4 bytes), MaxI(4 byte), Deadzone(4 byte),
MinPos(4 byte), MaxPos(4 byte), CRC16(2 bytes)]
```

68 - Set M1 Default Duty Acceleration

Sets the default acceleration and deceleration values for Motor 1 when using duty cycle commands (32, 33, 34) or when operating in Standard Serial, RC, or Analog PWM modes. Both acceleration and deceleration are specified as 4-byte values.

```
Send: [Address, 68, Accel(4 bytes), Decel(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

69 - Set M2 Default Duty Acceleration

Sets the default acceleration and deceleration values for Motor 2 when using duty cycle commands (32, 33, 34) or when operating in Standard Serial, RC, or Analog PWM modes. Both acceleration and deceleration are specified as 4-byte values.

```
Send: [Address, 69, Accel(4 bytes), Decel(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

70 - Set Motor1 Default Speed

Set M1 default speed for use with M1 position command and RC or analog modes when position control is enabled. This sets the percentage of the maximum speed set by QPSS as the default speed. The range is 0 to 32767.

```
Send: [Address, 70, Value(2 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

71 - Set Motor 2 Default Speed

Set M2 default speed for use with M2 postion command and RC or analog modes when position control is enabled. This sets the percentage of the maximum speed set by QPSS as the default speed. The range is 0 to 32767.

```
Send: [Address, 71, Value(2 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

72 - Read Default Speed Settings

Read current default speeds for M1 and M2.

```
Send: [Address, 72]
Receive: [M1Speed(2 bytes), M2Speed(2 bytes), CRC16(2 bytes)]
```

74 - Set S3, S4, S5 and CTRL1 / CTRL2 Modes

Configures the function of the S3, S4, and S5 I/O pins. Each pin can be independently assigned as an emergency stop, voltage clamp output, homing input, or other supported function. Refer to the Auxiliary Pin section of this manual for detailed descriptions of each mode.

```
Send: [Address, 74, S3mode, S4mode, S5mode, CTRL1mode, CTRL2mode, CRC16(2 bytes)]
Receive: [0xFF]
```

Available Modes

Mode	S3	S4	S5	CTRL1	CTRL2
0x00	Default	Disabled	Disabled		
0xC1	E-Stop (Latching)	E-Stop (Latching)	E-Stop (Latching)		
0x14	Voltage Clamp	Voltage Clamp	Voltage Clamp	Voltage Clamp	Voltage Clamp
0x24	RS485 Direction				
0x80	Encoder Toggle	Encoder Toggle	Encoder Toggle		
0x04	Brake	Brake	Brake	Brake	Brake
0xE2		Home (Auto)	Home (Auto)		
0x62		Home (User)	Home (User)		
0xF2		Home (Auto)/Limit (Fwd)	Home (Auto)/Limit (Fwd)		
0x72		Home (User)/Limit (Fwd)	Home (User)/Limit (Fwd)		
0x12		Limit (Fwd)	Limit (Fwd)		
0x22		Limit (Rev)	Limit (Rev)		
0x32		Limit (Both)	Limit (Both)		
0x11	Stop M1	Stop M1	Stop M1		
0x21	Stop M2	Stop M2	Stop M2		
0x41	E-Stop	E-Stop	E-Stop		
0x84	User Output	User Output	User Output	User Output	User Output
0x94	Main Batt Out	Main Batt Out	Main Batt Out	Main Batt Out	Main Batt Out
0xA4	Logic Batt Out	Logic Batt Out	Logic Batt Out	Logic Batt Out	Logic Batt Out
0xB4	M1 Current Out	M1 Current Out	M1 Current Out	M1 Current Out	M1 Current Out
0xC4	M2 Current Out	M2 Current Out	M2 Current Out	M2 Current Out	M2 Current Out
0xD4	Temperature 1 Out	Temperature 1 Out	Temperature 1 Out	Temperature 1 Out	Temperature 1 Out
0xE4	Temperature 2 Out	Temperature 2 Out	Temperature 2 Out	Temperature 2 Out	Temperature 2 Out
0x40	Flip Switch	Flip Switch	Flip Switch		
0x34	Main Batt Warning	Main Batt Warning	Main Batt Warning	Main Batt Warning	Main Batt Warning
0x44	Logic Batt Warning	Logic Batt Warning	Logic Batt Warning	Logic Batt Warning	Logic Batt Warning

Mode Notes

0x84–0xE4 Outputs: Provides PWM signals that can be filtered into an analog voltage to approximate measurement values. User calibration of the filtering circuit is required.

75 - Get S3, S4 amd S5 Modes

Returns the current mode settings for the S3, S4, and S5 pins, as well as CTRL1 and CTRL2. See Command 74 for detailed mode descriptions.

```
Send: [Address, 75]
Receive: [S3mode, S4mode, S5mode, CTRL1mode, CTRL2mode, CRC16(2bytes)]
```

76 - Set DeadBand for RC/Analog controls

Sets the deadband percentage for RC and Analog control inputs, specified in tenths of a percent. The default value is 25 (2.5%). The valid range is 0 (no deadband) to 250 (25%). Separate values are provided for M1 Reverse, M1 Forward, M2 Reverse, and M2 Forward.

```
Send: [Address, 76, M1 Reverse, M1 Forward, M2 Reverse, M2 Forward, CRC16(2byes)]
Receive: [0xFF]
```

77 - Read DeadBand for RC/Analog controls

Returns the deadband settings for RC and Analog control inputs, reported in tenths of a percent. Values are provided for M1 Reverse, M1 Forward, M2 Reverse, and M2 Forward.

```
Send: [Address, 77]
Receive: [M1 Reverse, M1 Forward, M2 Reverse, M2 Forward, CRC16(2bytes)]
```

80 - Restore Defaults

Reset Settings to factory defaults.

```
Send: [Address, 80, 0xE22EAB7A(4 bytes), CRC16(2bytes)]
Receive: [0xFF]
```

81 - Read Default Duty Acceleration Settings

Returns the default acceleration and deceleration values for Motor 1 and Motor 2 when using duty cycle commands or operating in Standard Serial, RC, or Analog PWM modes. Each value is returned as a 4-byte number.

```
Send: [Address, 81]
Receive: [M1Accel(4 bytes),M1Decel(4 bytes),M2Accel(4 bytes),M2Decel(4 bytes),CRC16(2
bytes)]
```

91 - Read Encoder Mode

Returns the encoder mode settings for Motor 1 and Motor 2.

```
Send: [Address, 91]
Receive: [Enc1Mode, Enc2Mode, CRC16(2 bytes)]
```

Encoder Mode bits

Bit 7	Enable RC/Analog Encoder support
Bit 6-1	Reserved (not used)
Bit 0	Encoder type: 0 = Quadrature, 1 = Absolute

92 - Set Motor 1 Encoder Mode

Sets the encoder mode for Motor 1. See Command 91 for bit definitions.

```
Send: [Address, 92, Mode, CRC16(2 bytes)]
Receive: [0xFF]
```

93 - Set Motor 2 Encoder Mode

Sets the encoder mode for Motor 2. See Command 91 for bit definitions.

```
Send: [Address, 93, Mode, CRC16(2 bytes)]
Receive: [0xFF]
```

96 - Set USB Serial Number

Sets the USB serial number string. The command requires exactly 36 bytes. If the chosen serial number is shorter, pad the remaining bytes with a value of 0 (NULL), not the ASCII character "0". The default is all NULL bytes, which indicates no USB serial number is set.

```
Send: [Address, 96, SerialNumber (36 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

97 - Read USB Serial Number

Reads the current USB serial number. The response always returns 36 bytes, including any NULL padding.

```
Send: [Address, 97, CRC (2 bytes)]
Receive: [SerialNumber (36 bytes), CRC16(2 bytes)]
```

98 - Set Standard Config Settings

Set config bits for standard settings.

```
Send: [Address, 98, Config(2 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

Function	Config Bit Mask
RC Mode	0x0000
Analog Mode	0x0001
Simple Serial Mode	0x0002
Packet Serial Mode	0x0003
Battery Mode Off	0x0000
Battery Mode Auto	0x0004
Battery Mode 2 Cell	0x0008
Battery Mode 3 Cell	0x000C
Battery Mode 4 Cell	0x0010
Battery Mode 5 Cell	0x0014
Battery Mode 6 Cell	0x0018
Battery Mode 7 Cell	0x001C
Mixing	0x0020

Function	Config Bit Mask
Exponential	0x0040
MCU	0x0080
BaudRate 2400	0x0000
BaudRate 9600	0x0020
BaudRate 19200	0x0040
BaudRate 38400	0x0060
BaudRate 57600	0x0080
BaudRate 115200	0x00A0
BaudRate 230400	0x00C0
BaudRate 460800	0x00E0
FlipSwitch	0x0100
Packet Address 0x80	0x0000
Packet Address 0x81	0x0100
Packet Address 0x82	0x0200
Packet Address 0x83	0x0300
Packet Address 0x84	0x0400
Packet Address 0x85	0x0500
Packet Address 0x86	0x0600
Packet Address 0x87	0x0700
Slave Mode	0x0800
Relay Mode	0X1000
Swap Encoders	0x2000
Swap Buttons	0x4000
Multi-Unit Mode	0x8000

99 - Read Standard Config Settings

Read config bits for standard settings See Command 98.

```
Send: [Address, 99]
Receive: [Config(2 bytes), CRC16(2 bytes)]
```

102 - Set Auxillary Pin Duty Setting

Sets the PWM duty for auxiliary pins configured as User Output or Auto Homing. Duty values range from 0 (0%) to 32767 (100%). In Auto Homing mode, this duty sets the homing speed as a percentage of the configured default speed. This command does not set homing timeout, use command 105 and 106 to set timeouts and 107 to read timeouts.

```
Send: [Address, 102, S3duty(2 bytes), S4duty(2 bytes), S5duty(2 bytes), CTRL1duty(2 bytes),
CTRL2duty(2 bytes), CRC16(2 bytes)]
Receive: [0xff]
```

104 - Read Auxiliary Pin Duty settings

Returns the PWM duty values for S3, S4, S5, CTRL1, CTRL2 (each 0–32767 = 0–100% duty). For pins in Auto Homing mode, this duty corresponds to the homing speed percentage. Homing timeouts are not returned, use command 107 to read timeouts.

```
Send: [Address, 104]
Receive: [S3duty(2 bytes),S4duty(2 bytes),S5duty(2 bytes),CTRL1duty(2 bytes),CTRL2duty(2
bytes),CRC16(2 bytes)]
```

105 - Set Auto Homing timeout for M1

Sets the percentage of duty or max speed and the timeout value for automatic homing of motor 1. If the motor is set up for velocity or position control the percentage of maximum speed is used, otherwise percent duty is used. The range for duty/speed is 0 to 32767. The timeout value is 32 bits and is set in increments of 1/300 of a second. As an example, a timeout value of 10 seconds would be set as 3000.

```
Send: [Address, 105, Timeout(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

106 - Set Auto Homing timeout for M2

Sets the percentage of duty or max speed and the timeout value for automatic homing of motor 2. If the motor is set up for velocity or position control the percentage of maximum speed is used, otherwise percent duty is used. The range for duty/speed is 0 to 32767. The timeout value is 32 bits and is set in increments of 1/300 of a second. As an example, a timeout value of 10 seconds would be set as 3000.

```
Send: [Address, 106, Timeout(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

107 - Get Auto Homing Timeout Settings

Reads the configured auto homing timeout values for both motors. Each value represents the maximum homing time in milliseconds.

```
Send: [Address, 107]
Receive: [M1 timeout(4 bytes), M2 timeout(4 bytes), CRC16(2 bytes)]
```

109 - Set Speed Error Limits

Set motor speed error limits in encoder counts per second.

```
Send: [Address, 109, M1Limit(4 bytes), M2Limit(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

110 - Read Speed Error Limits

Read the current speed error limit values.

```
Send: [Address, 110]
Receive: [M1Limit(4 bytes), M2Limit(4 bytes), CRC16(2 bytes)]
```

112 - Set Position Error Limits
Set motor position error limits in encoder counts.

```
Send: [Address, 112, M1Limit(4 bytes), M2Limit(4 byes), CRC16(2 bytes)]
Receive: [0xFF]
```

113 - Read Position Error Limits
Read the current motor position error limits.

```
Send: [Address, 113]
Receive: [M1Limit(4 bytes), M2Limit(4 bytes), CRC16(2 bytes)]
```

133 - Set M1 Max Current Limit
Set Motor 1 Maximum Current Limit. Current value is in 10ma units. To calculate multiply current limit by 100.

```
Send: [Address, 134, MaxCurrent(4 bytes), 0, 0, 0, 0, CRC16(2 bytes)]
Receive: [0xFF]
```

134 - Set M2 Max Current Limit
Set Motor 2 Maximum Current Limit. Current value is in 10ma units. To calculate multiply current limit by 100.

```
Send: [Address, 134, MaxCurrent(4 bytes), 0, 0, 0, 0, CRC16(2 bytes)]
Receive: [0xFF]
```

135 - Read M1 Max Current Limit
Read Motor 1 Maximum Current Limit. Current value is in 10ma units. To calculate divide value by 100. MinCurrent is always 0.

```
Send: [Address, 135]
Receive: [MaxCurrent(4 bytes), MinCurrent(4 bytes), CRC16(2 bytes)]
```

136 - Read M2 Max Current Limit
Read Motor 2 Maximum Current Limit. Current value is in 10ma units. To calculate divide value by 100. MinCurrent is always 0.

```
Send: [Address, 136]
Receive: [MaxCurrent(4 bytes), MinCurrent(4 bytes), CRC16(2 bytes)]
```

148 - Set PWM Mode

Configures the PWM mode for motor 1 and motor 2.

```
Send: [Address, 148, M1 PWM Mode(1 byte), M2 PWM Mode(1 byte), CRC16(2 bytes)]
Receive: [0xFF]
```

0 = Inductive mode
1 = Resisitive with 5% blanking
2 = Resisitive with 10% blanking
3 = Resisitive with 15% blanking
4 = Resisitive with 20% blanking

149 - Read PWM Mode

Read PWM Drive mode for motor 1 and motor 2. See Command 148.

```
Send: [Address, 149]
Receive: [M1 PWM Mode(1 byte), M2 PWM Mode(1 byte), CRC16(2 bytes)]
```

160 - Set PWM Idle Delays and Idle Modes

Configures the idle behavior for the controller when PWM input reaches zero. Two modes are available:

- Freewheel (default): Motors coast after the idle delay expires. Motors are held in a braking state during the idle delay. The delay sets how long the controller waits after PWM reaches zero before switching to freewheel (in tenths of a second).
- Braking: Motors immediately brake when PWM reaches zero. In this mode, the idle delay value is ignored.

The command uses a single byte, where:

- Bit 7 (0x80): Idle mode (0 = Freewheel, 1 = Braking)
- Bits 0–6: Idle delay time in tenths of a second (0–127, up to 12.7 seconds)

```
Send: [Address, 160, ModeDelay (1 byte), CRC16(2 bytes)]
Receive: [0xFF]
```

161 - Read PWM Idle Delays and Idle Modes

Reads back the current PWM idle mode and idle delay settings for both motor channels. See command 160 – Set PWM Idle Delays and Idle Modes for details on how the returned byte values are structured (bit 7 = mode, bits 0–6 = idle delay in tenths of a second).

```
Send: [Address, 161, CRC (2 bytes)]
Receive: [M1ModeDelay (1 byte), M2ModeDelay (1 byte), CRC16(2 bytes)]
```

Non-Motor Commands

These commands manage controller configuration and user data storage. They provide access to non-volatile memory for saving and restoring system settings, as well as read/write functions for user-defined EEPROM locations.

Command	Description
94	Write Settings to NVM
95	Read Settings from NVM
252	Read Non-Volatile data previously written
253	Write Non-Volatile data previously written

94 - Write Settings to NVM

Writes all configuration settings to non-volatile flash memory. Saved values are automatically loaded at power-up. Settings are only written to flash when this command is issued. No arguments are required other than the 4-byte key (0xE22EAB7A) to prevent accidental writes.

```
Send: [Address, 94, 0xE22EAB7A(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

95 - Read Settings from NVM

Reloads all configuration settings from non-volatile flash memory into active memory, overwriting any unsaved changes. This command can be used at any time to restore saved settings without cycling power.

```
Send: [Address, 95, CRC16(2bytes)]
Receive: [0xFF]
```

252 - Write User EEPROM Memory Location

Write a 16 bit value to user EEPROM. The memory address range is 0 to 255.

```
Send: [Address, 252, Memory Address, Value(2 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

253 - Read User EEPROM Memory Location

Read a 16 bit value from user EEPROM address. The memory address range is 0 to 255.

```
Send: [Address, 253, Memory Address]
Receive: [Value(2 bytes), CRC16(2 bytes)]
```

Open Loop Commands

Open loop commands control motor channels directly using duty cycle values. These commands are typically used with motors that do not have encoders, where precise position or speed feedback is not required. In this mode, motor operation is based on applied power only; variations in load or voltage may cause differences between commanded and actual speed. Acceleration commands provide control by ramping the duty cycle instead of applying it instantly.

Command	Description
0	Set Motor 1 Forward
1	Set Motor 1 Backward
4	Set Motor 2 Forward
5	Set Motor 2 Backward
6	Set Motor 1 7-Bit Value
7	Set Motor 2 7-Bit Value
8	Set Mixed Forward
9	Set Mixed Backward
10	Set Mixed Right
11	Set Mixed Left
12	Set Mixed Forward/Backward
13	Set Mixed Left/Right
32	Set Motor 1 Duty
33	Set Motor 2 Duty
34	Set Mixed Duty (M1 and M2 together)
52	Set Motor 1 Duty with Acceleration
53	Set Motor 2 Duty with Acceleration
54	Set Mixed Duty with Acceleration

0 - Drive Forward M1

Drive motor 1 forward. Valid data range is 0 - 127. A value of 127 = full speed forward, 64 = about half speed forward and 0 = full stop.

```
Send: [Address, 0, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

1 - Drive Backwards M1

Drive motor 1 backwards. Valid data range is 0 - 127. A value of 127 full speed backwards, 64 = about half speed backward and 0 = full stop.

```
Send: [Address, 1, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

4 - Drive Forward M2

Drive motor 2 forward. Valid data range is 0 - 127. A value of 127 full speed forward, 64 = about half speed forward and 0 = full stop.

```
Send: [Address, 4, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

5 - Drive Backwards M2

Drive motor 2 backwards. Valid data range is 0 - 127. A value of 127 full speed backwards, 64 = about half speed backward and 0 = full stop.

```
Send: [Address, 5, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

6 - Drive M1 (7 Bit)

Drive motor 1 forward or reverse. Valid data range is 0 - 127. A value of 0 = full speed reverse, 64 = stop and 127 = full speed forward.

```
Send: [Address, 6, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

7 - Drive M2 (7 Bit)

Drive motor 2 forward or reverse. Valid data range is 0 - 127. A value of 0 = full speed reverse, 64 = stop and 127 = full speed forward.

```
Send: [Address, 7, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

8 - Drive Forward

Sets the forward drive variable in mixed mode. Valid range: 0–127, where 0 = stop and 127 = full forward. Motion will not occur until both a drive command (8, 9 or 12) and a turn command (10, 11 or 13) have been sent. Once motion begins, drive and turn can be updated independently.

```
Send: [Address, 8, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

9 - Drive Backwards

Sets the reverse drive variable in mixed mode. Valid range: 0–127, where 0 = stop and 127 = full reverse. Motion will not occur until both a drive command (8, 9 or 12) and a turn command (10, 11 or 13) have been sent. Once motion begins, drive and turn can be updated independently.

```
Send: [Address, 9, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

10 - Turn right

Sets the turn variable for right rotation in mixed mode. Valid range: 0–127, where 0 = stop turn and 127 = full right turn. Motion will not occur until a drive command (8, 9, or 12) is also active.

```
Send: [Address, 10, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

11 - Turn left

Sets the turn variable for left rotation in mixed mode. Valid range: 0–127, where 0 = stop turn and 127 = full left turn. Motion will not occur until a drive command (8, 9, or 12) is also active.

```
Send: [Address, 11, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

12 - Drive Forward or Backward (7 Bit)

Combined forward/reverse drive command in mixed mode. Valid range: 0–127, where 0 = full reverse, 64 = stop, and 127 = full forward. Motion will not occur until a turn command (10, 11, or 13) is also active.

```
Send: [Address, 12, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

13 - Turn Left or Right (7 Bit)

Combined left/right turn command in mixed mode. Valid range: 0–127, where 0 = full left, 64 = stop, and 127 = full right. Motion will not occur until a drive command (8, 9, or 12) is also active.

```
Send: [Address, 13, Value, CRC16(2 bytes)]
Receive: [0xFF]
```

32 - Drive M1 With Signed Duty Cycle

Drive M1 using a duty cycle value. The duty cycle is used to control the speed of the motor without a quadrature encoder.

```
Send: [Address, 32, Duty(2 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The duty value is signed and the range is -32767 to +32767 (eg. +-100% duty).

33 - Drive M2 With Signed Duty Cycle

Drive M2 using a duty cycle value. The duty cycle is used to control the speed of the motor without a quadrature encoder. The command syntax:

```
Send: [Address, 33, Duty(2 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The duty value is signed and the range is -32768 to +32767 (eg. +-100% duty).

34 - Drive M1 / M2 With Signed Duty Cycle

Drive both M1 and M2 using a duty cycle value. The duty cycle is used to control the speed of the motor without a quadrature encoder. The command syntax:

```
Send: [Address, 34, DutyM1(2 Bytes), DutyM2(2 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The duty value is signed and the range is -32768 to +32767 (eg. +-100% duty).

52 - Drive M1 With Signed Duty And Acceleration

Drive M1 with a signed duty and acceleration value. The sign indicates which direction the motor will run. The acceleration values are not signed. This command is used to drive the motor by PWM and using an acceleration value for ramping. Accel is the rate per second at which the duty changes from the current duty to the specified duty.

```
Send: [Address, 52, Duty(2 bytes), Accel(2 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The duty value is signed and the range is -32768 to +32767(eg. +-100% duty). The accel value range is 0 to 655359(eg maximum acceleration rate is -100% to 100% in 100ms).

53 - Drive M2 With Signed Duty And Acceleration

Drive M1 with a signed duty and acceleration value. The sign indicates which direction the motor will run. The acceleration values are not signed. This command is used to drive the motor by PWM and using an acceleration value for ramping. Accel is the rate at which the duty changes from the current duty to the specified dury.

```
Send: [Address, 53, Duty(2 bytes), Accel(2 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The duty value is signed and the range is -32768 to +32767 (eg. +-100% duty). The accel value range is 0 to 655359 (eg maximum acceleration rate is -100% to 100% in 100ms).

54 - Drive M1 / M2 With Signed Duty And Acceleration

Drive M1 and M2 in the same command using acceleration and duty values for each motor. The sign indicates which direction the motor will run. The acceleration value is not signed. This command is used to drive the motor by PWM using an acceleration value for ramping. The command syntax:

```
Send: [Address, CMD, DutyM1(2 bytes), AccelM1(4 Bytes), DutyM2(2 bytes),
      AccelM1(4 bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The duty value is signed and the range is -32768 to +32767 (eg. +-100% duty). The accel value range is 0 to 655359 (eg maximum acceleration rate is -100% to 100% in 100ms).

Closed Loop Commands

Closed loop commands use encoder feedback for precise speed and position control. Unlike open loop operation, closed loop mode continuously adjusts motor output based on encoder readings to maintain the commanded speed or position, even under changing loads or conditions. These commands include options for acceleration, distance targets, and position-based moves, enabling accurate and repeatable motor control.

Command	Description
35	Set Motor 1 Speed
36	Set Motor 2 Speed
37	Set Mixed Speed (M1 and M2 together)
38	Set Motor 1 Speed with Acceleration
39	Set Motor 2 Speed with Acceleration
40	Set Mixed Speed with Acceleration
41	Set Motor 1 Speed with Distance
42	Set Motor 2 Speed with Distance
43	Set Mixed Speed with Distance
44	Set Motor 1 Speed with Accel and Distance
45	Set Motor 2 Speed with Accel and Distance
46	Set Mixed Speed with Accel and Distance
50	Set Mixed Speed with Two Accelerations
51	Set Mixed Speed with Two Accelerations and Distance
65	Set Motor 1 Speed, Accel, Deccel, and Position
66	Set Motor 2 Speed, Accel, Deccel, and Position
67	Set Mixed Speed, Accel, Deccel, and Position
119	Set Motor 1 Position
120	Set Motor 2 Position
121	Set Mixed Position (M1 and M2 together)
122	Set Motor 1 Position with Speed
123	Set Motor 2 Position with Speed
124	Set Mixed Position with Speed
125	Set Motor 1 Percentage Position
126	Set Motor 2 Percentage Position
127	Set Mixed Percentage Position

35 - Drive M1 With Signed Speed

Drive M1 using a speed value. The sign indicates which direction the motor will turn. This command is used to drive the motor by quad pulses per second. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate as fast as possible until the defined rate is reached.

```
Send: [Address, 35, Speed(4 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

36 - Drive M2 With Signed Speed

Drive M2 with a speed value. The sign indicates which direction the motor will turn. This command is used to drive the motor by quad pulses per second. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent, the motor will begin to accelerate as fast as possible until the rate defined is reached.

```
Send: [Address, 36, Speed(4 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

37 - Drive M1 / M2 With Signed Speed

Drive M1 and M2 in the same command using a signed speed value. The sign indicates which direction the motor will turn. This command is used to drive both motors by quad pulses per second. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate as fast as possible until the rate defined is reached.

```
Send: [Address, 37, SpeedM1(4 Bytes), SpeedM2(4 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

38 - Drive M1 With Signed Speed And Acceleration

Drive M1 with a signed speed and acceleration value. The sign indicates which direction the motor will run. The acceleration values are not signed. This command is used to drive the motor by quad pulses per second and using an acceleration value for ramping. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate incrementally until the rate defined is reached.

```
Send: [Address, 38, Accel(4 Bytes), Speed(4 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The acceleration is measured in speed increase per second. An acceleration value of 12,000 QPPS with a speed of 12,000 QPPS would accelerate a motor from 0 to 12,000 QPPS in 1 second. Another example would be an acceleration value of 24,000 QPPS and a speed value of 12,000 QPPS would accelerate the motor to 12,000 QPPS in 0.5 seconds.

39 - Drive M2 With Signed Speed And Acceleration

Drive M2 with a signed speed and acceleration value. The sign indicates which direction the motor will run. The acceleration value is not signed. This command is used to drive the motor by quad pulses per second and using an acceleration value for ramping. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate incrementally until the rate defined is reached.

```
Send: [Address, 39, Accel(4 Bytes), Speed(4 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The acceleration is measured in speed increase per second. An acceleration value of 12,000 QPPS with a speed of 12,000 QPPS would accelerate a motor from 0 to 12,000 QPPS in 1 second. Another example would be an acceleration value of 24,000 QPPS and a speed value of 12,000 QPPS would accelerate the motor to 12,000 QPPS in 0.5 seconds.

40 - Drive M1 / M2 With Signed Speed And Acceleration

Drive M1 and M2 in the same command using one value for acceleration and two signed speed values for each motor. The sign indicates which direction the motor will run. The acceleration value is not signed. The motors are sync during acceleration. This command is used to drive the motor by quad pulses per second and using an acceleration value for ramping. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate incrementally until the rate defined is reached.

```
Send: [Address, 40, Accel(4 Bytes), SpeedM1(4 Bytes), SpeedM2(4 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The acceleration is measured in speed increase per second. An acceleration value of 12,000 QPPS with a speed of 12,000 QPPS would accelerate a motor from 0 to 12,000 QPPS in 1 second. Another example would be an acceleration value of 24,000 QPPS and a speed value of 12,000 QPPS would accelerate the motor to 12,000 QPPS in 0.5 seconds.

41 - M1 Drive With Signed Speed And Distance

Drive M1 with a signed speed and distance value. The sign indicates which direction the motor will run. The distance value is not signed. This command is buffered. This command is used to control the top speed and total distance traveled by the motor. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second.

```
Send: [Address, 41, Speed(4 Bytes), Distance(4 Bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

42 - M2 Drive With Signed Speed And Distance

Drive M2 with a speed and distance value. The sign indicates which direction the motor will run. The distance value is not signed. This command is buffered. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second.

```
Send: [Address, 42, Speed(4 Bytes), Distance(4 Bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

43 - Drive M1 / M2 With Signed Speed And Distance

Drive M1 and M2 with a speed and distance value. The sign indicates which direction the motor will run. The distance value is not signed. This command is buffered. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second.

```
Send: [Address, 43, SpeedM1(4 Bytes), DistanceM1(4 Bytes),
            SpeedM2(4 Bytes), DistanceM2(4 Bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

44 - M1 Drive With Signed Speed, Accel And Distance

Drive M1 with a speed, acceleration and distance value. The sign indicates which direction the motor will run. The acceleration and distance values are not signed. This command is used to control the motors top speed, total distanced traveled and at what incremental acceleration value to use until the top speed is reached. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second.

```
Send: [Address, 44, Accel(4 bytes), Speed(4 Bytes), Distance(4 Bytes),
            Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

45 - M2 Drive With Signed Speed, Accel And Distance

Drive M2 with a speed, acceleration and distance value. The sign indicates which direction the motor will run. The acceleration and distance values are not signed. This command is used to control the motors top speed, total distanced traveled and at what incremental acceleration value to use until the top speed is reached. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second.

```
Send: [Address, 45, Accel(4 bytes), Speed(4 Bytes), Distance(4 Bytes),
            Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

46 - Drive M1 / M2 With Signed Speed, Accel And Distance

Drive M1 and M2 with a speed, acceleration and distance value. The sign indicates which direction the motor will run. The acceleration and distance values are not signed. This command is used to control both motors top speed, total distanced traveled and at what incremental acceleration value to use until the top speed is reached. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second.

```
Send: [Address, 46, Accel(4 Bytes), SpeedM1(4 Bytes), DistanceM1(4 Bytes),
SpeedM2(4 bytes), DistanceM2(4 Bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

50 - Drive M1 / M2 With Signed Speed And Individual Acceleration

Drive M1 and M2 in the same command using one value for acceleration and two signed speed values for each motor. The sign indicates which direction the motor will run. The acceleration value is not signed. The motors are sync during acceleration. This command is used to drive the motor by quad pulses per second and using an acceleration value for ramping. Different quadrature encoders will have different rates at which they generate the incoming pulses. The values used will differ from one encoder to another. Once a value is sent the motor will begin to accelerate incrementally until the rate defined is reached.

```
Send: [Address, 50, AccelM1(4 Bytes), SpeedM1(4 Bytes), AccelM2(4 Bytes),
      SpeedM2(4 Bytes), CRC16(2 bytes)]
Receive: [0xFF]
```

The acceleration is measured in speed increase per second. An acceleration value of 12,000 QPPS with a speed of 12,000 QPPS would accelerate a motor from 0 to 12,000 QPPS in 1 second. Another example would be an acceleration value of 24,000 QPPS and a speed value of 12,000 QPPS would accelerate the motor to 12,000 QPPS in 0.5 seconds.

51 - Buffered Drive M1 / M2 With Signed Speed, Individual Accel And Distance

Drive M1 and M2 with a speed, acceleration and distance value. The sign indicates which direction the motor will run. The acceleration and distance values are not signed. This command is used to control both motors top speed, total distanced traveled and at what incremental acceleration value to use until the top speed is reached. Each motor channel M1 and M2 have separate buffers. This command will execute immediately if no other command for that channel is executing, otherwise the command will be buffered in the order it was sent. Any buffered or executing command can be stopped when a new command is issued by setting the Buffer argument. All values used are in quad pulses per second.

```
Send: [Address, 51, AccelM1(4 Bytes), SpeedM1(4 Bytes), DistanceM1(4 Bytes),
AccelM2(4 Bytes), SpeedM2(4 bytes), DistanceM2(4 Bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

The Buffer argument can be set to a 1 or 0. If a value of 0 is used the command will be buffered and executed in the order sent. If a value of 1 is used the current running command is stopped, any other commands in the buffer are deleted and the new command is executed.

65 - Buffered Drive M1 with signed Speed, Accel, Deccel and Position

Move M1 position from the current position to the specified new position and hold the new position. Accel sets the acceleration value and deccel the decceleration value. QSpeed sets the speed in quadrature pulses the motor will run at after acceleration and before decceleration.

```
Send: [Address, 65, Accel(4 bytes), Speed(4 Bytes), Deccel(4 bytes),
      Position(4 Bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

66 - Buffered Drive M2 with signed Speed, Accel, Deccel and Position

Move M2 position from the current position to the specified new position and hold the new position. Accel sets the acceleration value and deccel the decceleration value. QSpeed sets the speed in quadrature pulses the motor will run at after acceleration and before decceleration.

```
Send: [Address, 66, Accel(4 bytes), Speed(4 Bytes), Deccel(4 bytes),
      Position(4 Bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

67 - Buffered Drive M1 & M2 with signed Speed, Accel, Deccel and Position

Move M1 & M2 positions from their current positions to the specified new positions and hold the new positions. Accel sets the acceleration value and deccel the decceleration value. QSpeed sets the speed in quadrature pulses the motor will run at after acceleration and before decceleration.

```
Send: [Address, 67, AccelM1(4 bytes), SpeedM1(4 Bytes), DeccelM1(4 bytes),
            PositionM1(4 Bytes), AccelM2(4 bytes), SpeedM2(4 Bytes), DeccelM2(4 bytes),
                  PositionM2(4 Bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

119 - Drive M1 with Position

Moves Motor 1 from its current position to the specified target position and holds that position. The default acceleration, deceleration, and speed settings are applied. The command supports buffered execution, meaning multiple position commands can be queued for sequential execution.

```
Send: [Address, 119, Position (4 bytes), Buffer, CRC16(2 bytes)]
Receivr: [0xFF]
```

120 - Drive M2 with Position

Moves Motor 2 from its current position to the specified target position and holds that position. The default acceleration, deceleration, and speed settings are applied. The command supports buffered execution, meaning multiple position commands can be queued for sequential execution.

```
Send: [Address, 120, Position (4 bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

121 - Set Mixed Position M1 and M2

Moves both Motor 1 and Motor 2 from their current positions to the specified target positions and holds them. The default acceleration, deceleration, and speed settings are applied. This command supports buffered execution, allowing multiple position commands to be queued for sequential operation.

```
Send: [Address, 121, M1Position (4 bytes), M2Position, (4 bytes), Buffer, CRC16(2 bytes)]
Receive; [0xFF]
```

122 - Set Motor 1 Position with Speed

Moves Motor 1 from its current position to the specified target position at the defined speed, then holds that position. Default acceleration and deceleration values are applied. This command supports buffered execution, allowing multiple position/speed moves to be queued and executed in sequence.

```
Send; [Address, 122, Speed (4 bytes), Position (4 bytes), Buffer, CRC16(2 bytes)]
Receiver: [0xFF]
```

123 - Set Motor 2 Position with Speed

Moves Motor 2 from its current position to the specified target position at the defined speed, then holds that position. Default acceleration and deceleration values are applied. This command supports buffered execution, allowing multiple position/speed moves to be queued and executed in sequence.

```
Send: [Address, 123, Speed (4 bytes), Position (4 bytes), Buffer, CRC16(2 bytes)]
Receiver: [0xFF]
```

124 - Set Mixed Position with Speed M1 and M2

Moves Motor 1 and Motor 2 from their current positions to the specified target positions at the defined speeds, then holds those positions. Default acceleration and deceleration values are applied. This command supports buffered execution, allowing multiple motion commands to be queued and executed in sequence.

```
Send: [Address, 124, SpeedM1 (4 bytes), PositionM1 (4 bytes), SpeedM2 (4 bytes),
PositionM2 (4 bytes), Buffer, CRC16(2 bytes)]
Receive: [0xFF]
```

125 - Set Motor 1 Percentage Position

Moves Motor 1 from its current position to the specified target, expressed as a percentage of the defined position range, and holds that position. Default acceleration, deceleration, and speed values are used. This command supports buffered execution, allowing multiple motion commands to be queued and executed in sequence.

```
Send:[Address, 125, PercentPosM1 (2 bytes), Buffer, CRC16(2 bytes)]
Receive:[0xFF]
```

The percentage value is mapped to the configured minimum and maximum position limits (0% = minimum position, 100% = maximum position).

126 - Set Motor 2 Percentage Position

Moves Motor 2 from its current position to the specified target, expressed as a percentage of the defined position range, and holds that position. Default acceleration, deceleration, and speed values are used. This command supports buffered execution, allowing multiple motion commands to be queued and executed in sequence.

```
Send:[Address, 126, PercentPosM2 (2 bytes), Buffer, CRC16(2 bytes)]
Receive:[0xFF]
```

The percentage value is mapped to the configured minimum and maximum position limits (0% = minimum position, 100% = maximum position).

127 - Set Mixed Percentage Position

Moves Motor 1 and Motor 2 from their current positions to the specified targets, each expressed as a percentage of the defined position range, and holds those positions. Default acceleration, deceleration, and speed values are used. This command supports buffered execution, allowing multiple motion commands to be queued and executed in sequence.

```
Send:[Address, 127, PercentPosM1 (2 bytes), PercentPosM2 (2 bytes), Buffer, CRC16(2 bytes)]
Receive:[0xFF]
```

The percentage value is mapped to the configured minimum and maximum position limits (0% = minimum position, 100% = maximum position).

Warranty

BasicMicro warrants its products against defects in material and workmanship for a period of one (1) year from the date of purchase. If a defect is discovered, BasicMicro will, at its sole discretion, repair the product, replace it, or refund the original purchase price. To initiate a warranty claim, contact us at sales@basicmicro.com

Copyrights and Trademarks

Disclaimer

BasicMicro is not responsible for any incidental, indirect, or consequential damages arising from the use or misuse of its products. Products must not be used in medical or life-support applications without prior written authorization issued by a duly authorized corporate officer of BasicMicro, Inc.

By using BasicMicro products, the purchaser agrees to assume all risks associated with their use and to indemnify and hold harmless BasicMicro, Inc., its officers, employees, and distributors from any and all claims, liabilities, damages, costs, or expenses (including reasonable attorneys' fees) arising from such use.

Contacts

Email: sales@basicmicro.com
Tech support: support@basicmicro.com
Web: http://www.basicmicro.com

Technical Support

Technical support is available via email at support@basicmicro.com, by opening a support ticket through the BasicMicro website on the Contact Us page, or by calling 1-800-535-9161 during normal business hours.