RoboClaw 2x5A Motor Controller
**Data Sheet**
Firmware Version 3.1.3+

Feature Overview:

- 2 Channels at 5Amp each, Peak 10Amp
- 3.3V Compliant Outputs
- 5V Tolerant Inputs
- Battery Elimination Circuit (BEC)
- Hobby RC Radio Compatible
- Serial Modes
- TTL Input
- Analog Mode
- 2 Channel Quadrature Decoding
- Thermal Protection
- Lithium Cut Off
- Packet Serial with Error Detection
- High Speed Direction Switching
- Flip Over Switch
- Over Current Protection
- Regenerative Braking
- USB Capable(Optional)
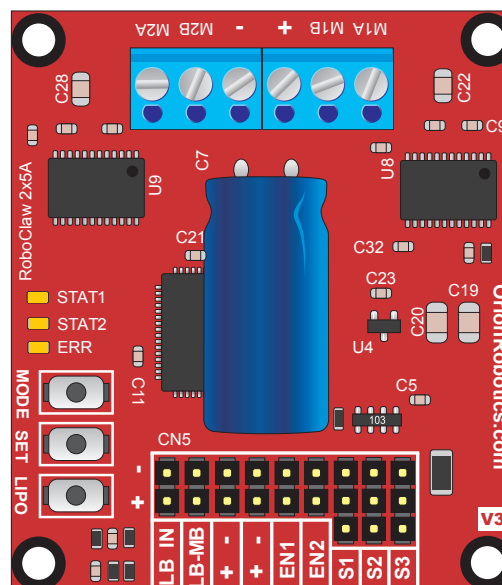
Basic Description

The RoboClaw 2X5 Amp is an extremely efficient, versatile, dual channel synchronous regenerative motor controller. It supports dual quadrature encoders and can supply two brushed DC motors with 5 amps per channel continuous and 10 amp peak.

With support for dual quadrature decoding you get greater control over speed and velocity. Automatically maintain a speed even if load increases. RoboClaw uses PID calculations with feed forward in combination with external quadrature encoders to make an accurate control solution.

RoboClaw is easy to control with several built in modes. It can be controlled from a standard RC receiver/transmitter, serial device, microcontroller or an analog source, such as a potentiometer based joystick. RoboClaw is equipped with screw terminal for fast connect and disconnect. All modes are set by the onboard mode buttons making setup a snap!

Optical Encoders

RoboClaw features dual channel quadrature decoding. RoboClaw gives you the ability to create a closed loop motion system. Now you can know a motors speed and direction giving you greater control over DC motors systems.

Power System

The RoboClaw is equipped with synchronous regenerative motor drivers. This means your battery is recharged when slowing down, braking or reversing. In addition a BEC circuit is included. It can supply a useful current of up to 150mA at 5v. The BEC is meant to provide power to a microcontroller or RC receiver.

Hardware Overview:
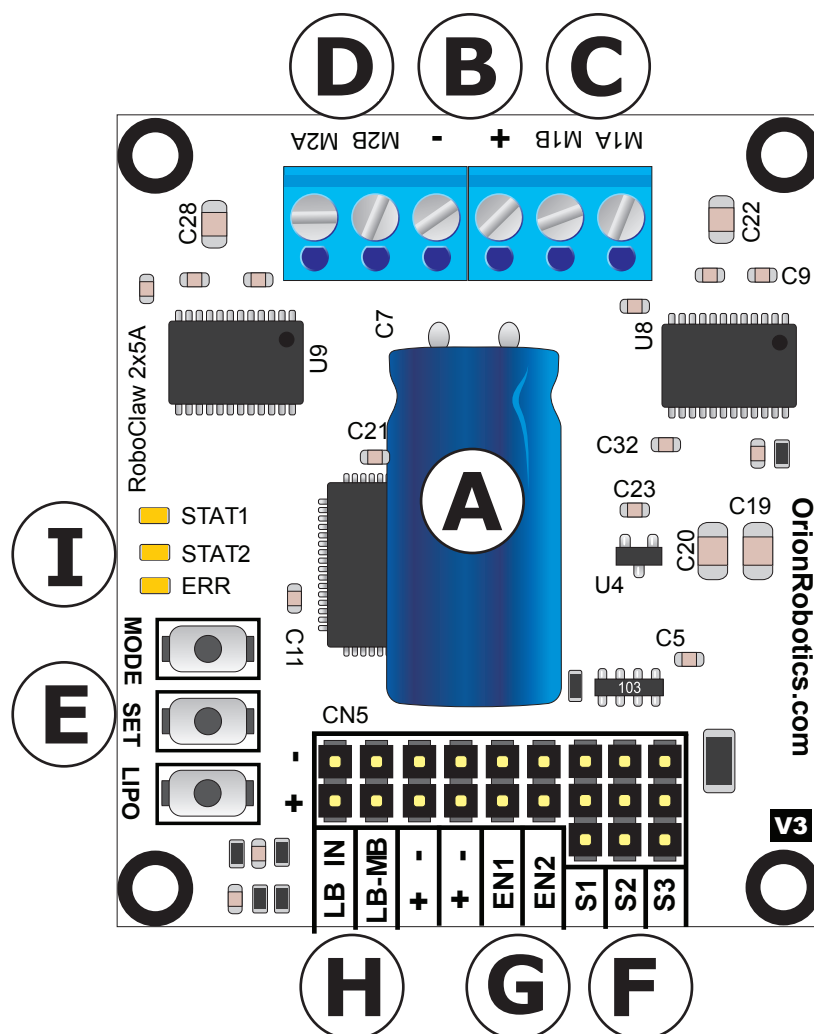
A: Power Stabilizer
B: Main Battery Input
C: Motor Channel 1
D: Motor Channel 2
E: Setup Buttons
F: Control Inputs
G: Encoder Inputs
H: Logic Voltage Source Selection Header
I: Status and Error LED Indicators

Dimensions:
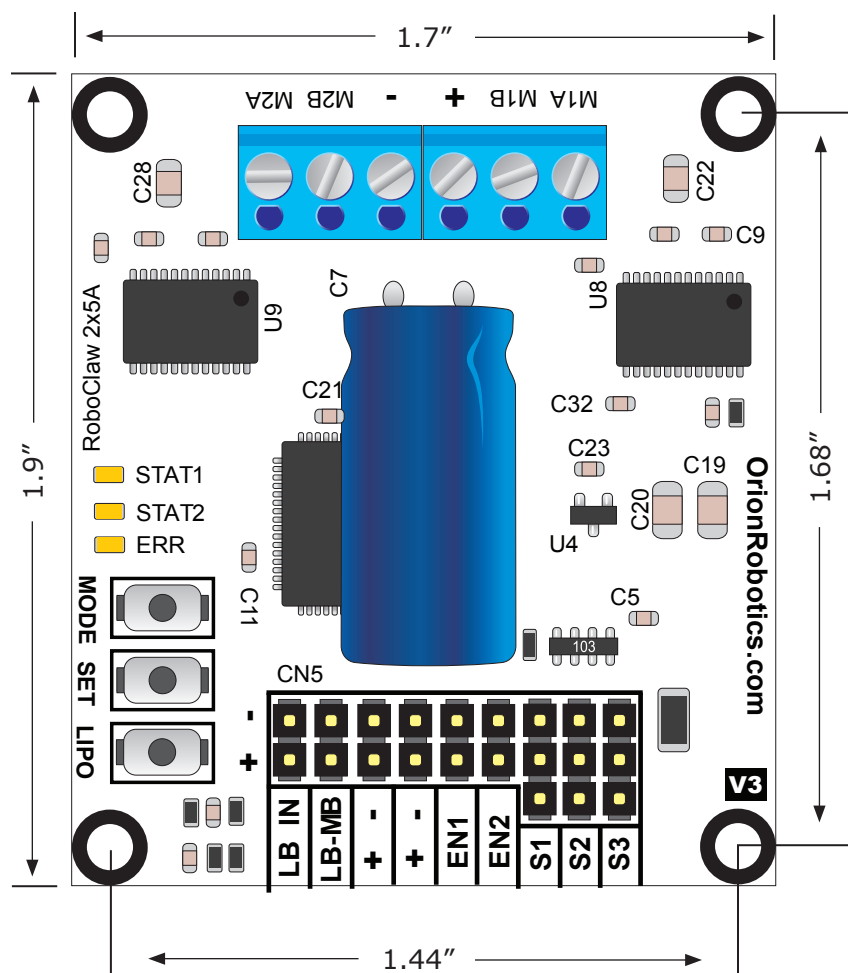
Board Edge: 1.7"W X 1.9"L
Hole Pattern: 0.125D, 1.44"W x 1.68"H

Header Overview

Logic Battery (LB IN)
The logic circuits can be powered from the main battery or a secondary battery wired LB IN. The positive (+) terminal is located at the board edge and ground (-) is the inside pin near the heatsink. Remove LB-MB jumper if power is applied to LB IN.

BEC Source (LB-MB)
RoboClaw logic requires 5VDC which is provided from the on board BEC circuit. The BEC source input is set with the LB-MB jumper. Install a jumper on the 2 pins labeled LB-MB to use the main battery as the BEC power source. Remove this jumper if using a separate logic battery.

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.

Encoder Inputs (EN1 / EN2)
EN1 and EN2 are the inputs from the encoders. Channel A of both EN1 and EN2 are located at the board edge. Channel B pins are located near the heatsink. 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. Which can affect how RoboClaw operates. Refer to the data sheet of the encoder you are using for channel direction.

Control Inputs (S1 / S2 / S3)
S1, S2 and S3 are setup for standard servo style headers I/O, +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 becomes an emergency stop. S3 is active when pulled low. It is internally pull up so it will not accidentally trip when left floating. 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 logic supply. It may be necessary to remove the +5V pin from the RC receivers cable in those cases.

Main Battery Screw Terminals

RoboClaws main power input can be from 6VDC to 34VDC. The connections are marked + and - on the main screw terminal. + is the positive side typically marked with a red wire. The - is the negative side typically marked with a black wire. When connecting the main battery it is a good practice to use a switch to turn the main power on and off. The switch must be rated to handle the maximum current and voltage from the battery. This will vary depending on the type of motors and or power source you are using.

Motor Screw Terminals

The motor screw terminals are marked with M1A / M1B for channel 1 and M2A / M2B for channel 2. There is no specific polarities for the motors. However if you want both motors turning in the same direction on a 4 wheeled robot you need to reverse one of the motors polarities.
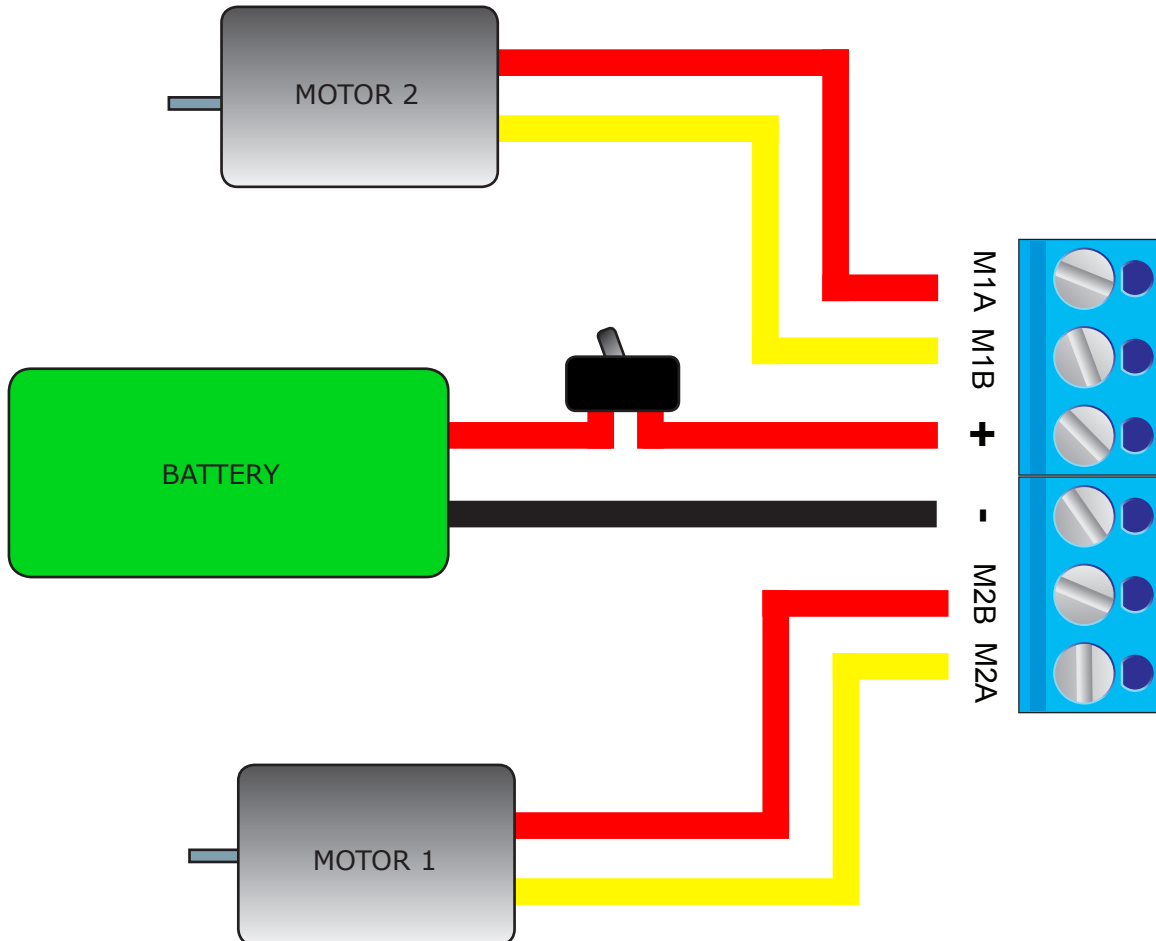
Status and Error LEDs

The RoboClaw has three LEDs. Two Status LEDs and one Error LED. When RoboClaw is first powered up all 3 LEDs should blink briefly to indicate all 3 LEDs are functional. The status LEDs will indicate a status based on what mode RoboClaw is set to.
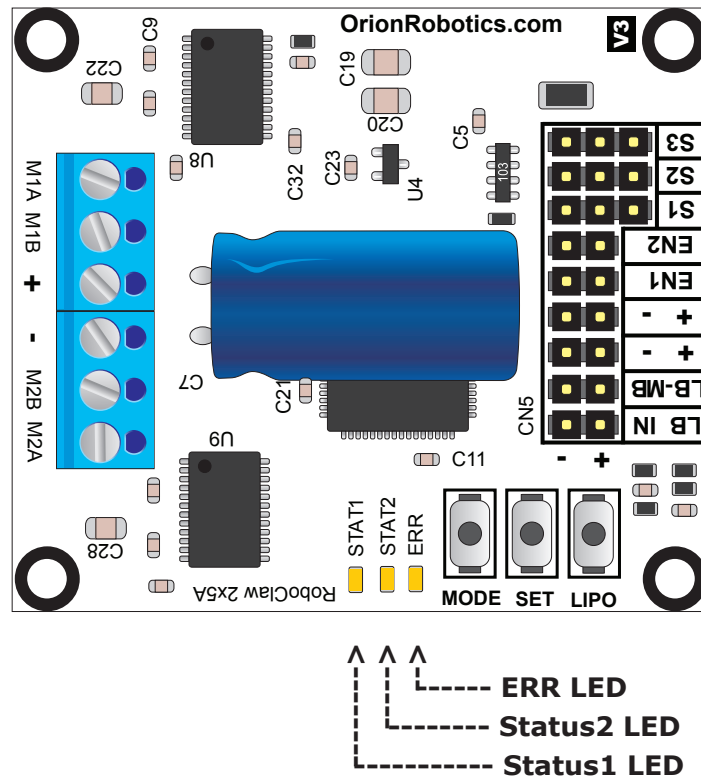
Analog Mode
Status 1 LED = On continuous.
Status 2 LED = On when motor(s) active.

RC Mode
Status 1 LED = On continuous, blink when pulse received.
Status 2 LED = On when motor(s) active.

Serial Modes
Status 1 LED = On continuous, blink on serial receive.
Status 2 LED = On when motor(s) active.

Errors

Over Current	= Error LED on solid. Status 1 or 2 indicates which motor.
Over Heat	= Error LED blinking once with a long pause. Status 1 & 2 off
Driver Error	= Error LED blinking once with a long pause. Status 1 or 2 on
Main Batt Low	= Error LED blinking twice with a long pause.
Main Batt High	= Error LED on/flicker until condition is cleared.
Logic Batt Low	= Error LED blinking three times with a long pause.
Logic Batt High	= Error LED blinking four times with a long pause.

RoboClaw Setup

There are 3 buttons on RoboClaw which are used to set modes and configuration options. The MODE button sets the interface method such as Serial or RC modes. The SET button is used to configure the options for a given mode. The LIPO button doubles as a save button and configuring the low battery voltage cut out function of RoboClaw. See the following tables to navigate RoboClaw setup.
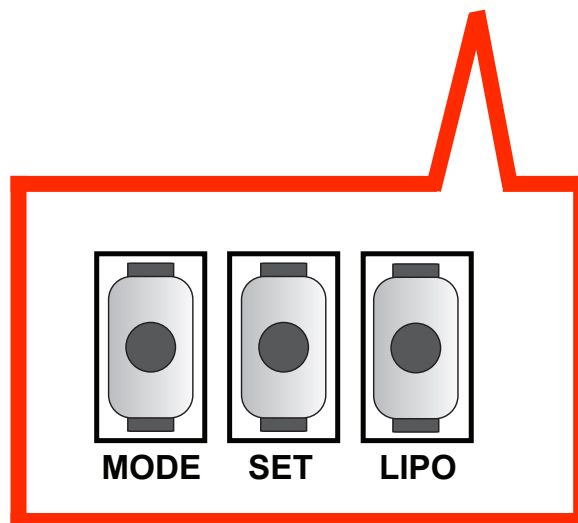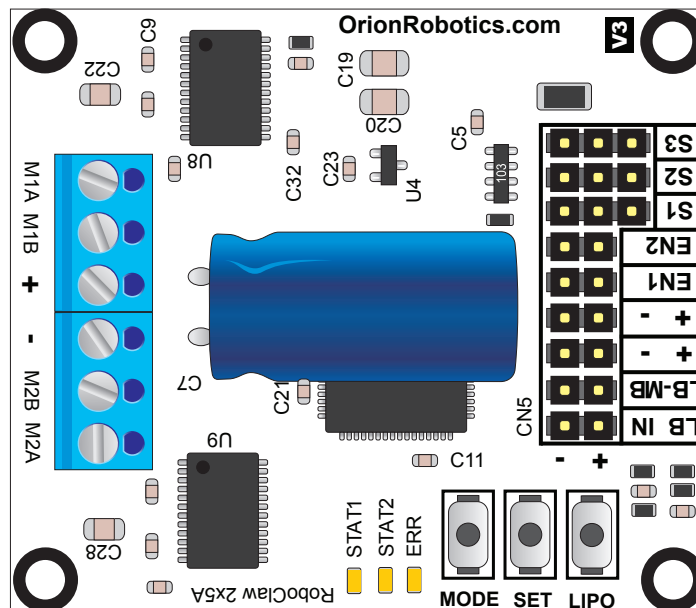
Interface Overview
There are 4 main modes with variations totaling 14 or 15 modes in all. Each mode enables RoboClaw to be controlled in a very specific way. The following list explains each mode and the ideal application.

RC Mode 1 & 2
With RC mode RoboClaw can be controlled from any hobby RC radio system. RC input mode also allows low powered microcontroller such as a Basic Stamp or Nano to control RoboClaw. RoboClaw expects servo pulse inputs to control the direction and speed. Very similar to how a regular servo is controlled. RC mode can not use encoders.

Analog Mode 3 & 4
Analog mode uses an analog signal from 0V to 5V to control the speed and direction of each motor. RoboClaw can be controlled using a potentiometer or filtered PWM from a microcontroller. Analog mode is ideal for interfacing RoboClaw joystick positioning systems or other non microcontroller interfacing hardware. Analog mode can not use encoders.

Simple Serial Mode 5 & 6
In simple serial mode RoboClaw expects TTL level RS-232 serial data to control direction and speed of each motor. Simple serial is typically used to control RoboClaw from a microcontroller or PC. If using a PC a MAX232 type circuit must be used since RoboClaw only works with TTL level input. Simple serial includes a slave select mode which allows multiple RoboClaws to be controlled from a signal RS-232 port (PC or microcontroller). Simple serial is a one way format, RoboClaw only receives data.

Packet Serial Mode 7 through 14
In packet serial mode RoboClaw expects TTL level RS-232 serial data to control direction and speed of each motor. Packet serial is typically used to control RoboClaw from a microcontroller or PC. If using a PC a MAX232 type circuit must be used since RoboClaw only works with TTL level input. In packet serial mode each RoboClaw is assigned an address using the dip switches. There are 8 addresses available. This means up to 8 RoboClaws can be on the same serial port. When using the quadrature decoding feature of RoboClaw packet serial is required since it is a two way communications format. This allows RoboClaw to transmit information about the encoders position and speed.

USB Mode 15(USB Roboclaw only)
In USB mode the RoboClaw's USB port acts as a CDC Virtual Comport in Packet Serial mode with packet address 128. Packet serial mode functionality is available in USB mode as well as baud rates up to 1mbit. There are two ways to activate the USB mode. Power up a USB RoboClaw while it is attached to an active USB cable, or set it to mode 15. If a PC is used to drive RoboClaw mode 15 should be set.

Configuring RoboClaw Modes

The buttons built into RoboClaw are used to set the different configuration options. To set the desired mode follow the steps below:

1. Press and release the MODE button to enter mode setup. The STAT2 LED will begin to blink out the current mode. Each blink is a half second with a long pause at the end of the count. Five blinks with a long pause equals mode 5 and so on.

2. Press SET to increment to the next mode. Press MODE to decrement to the previous mode.

3. Press and release the LIPO button to save this mode to memory.

Modes

Mode	Description
1	RC mode
2	RC mode with mixing
3	Analog mode
4	Analog mode with mixing
5	Simple Serial
6	Simple Serial with slave pin
7	Packet Serial Mode - Address 0x80
8	Packet Serial Mode - Address 0x81
9	Packet Serial Mode - Address 0x82
10	Packet Serial Mode - Address 0x83
11	Packet Serial Mode - Address 0x84
12	Packet Serial Mode - Address 0x85
13	Packet Serial Mode - Address 0x86
14	Packet Serial Mode - Address 0x87
15	USB Mode Packet Serial - Address 0x80

Mode Options

After the desired mode is set and saved press and release the SET button for options setup. The STAT2 LED will begin to blink out the current option. Press SET to increment to the next option. Press MODE to decrement to the previous option. Once the desired option is selected press and release the LIPO button to save the option to memory.

RC and Analog Mode Options

Option	Description
1	TTL Flip Switch
2	TTL Flip and Exponential Enabled
3	TTL Flip and MCU Enabled
4	TTL Flip and Exp and MCU Enabled
5	RC Flip Switch
6	RC Flip and Exponential Enabled
7	RC Flip and MCU Enabled
8	RC Flip and Exponential and MCU Enabled

Simple and Packet Serial Mode Options

Option	Description
1	2400bps
2	9600bps
3	19200bps
4	38400bps

Battery Cut Off Settings

The battery settings can be set by pressing and releasing the LIPO button. The STAT2 LED will begin to blink out the current setting. Press SET to increment to the next setting. Press MODE to decrement to the previous setting. Once the desired setting is selected press and release the LIPO button to save this setting to memory.

Battery Options

Option	Description
1	Normal
2	Lead Acid - Auto
3	2 Cell(6v Cutoff)
4	3 Cell(9v Cutoff)
5	4 Cell(12v Cutoff)
6	5 Cell(15v Cutoff)
7	6 Cell(18v Cutoff)
8	7 Cell(21v Cutoff)

RC Mode

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 speed and direction of motor 2. This drive method is similar to how a tank is controlled.

Using RC Mode with feedback for velocity/position control

RC Mode can be used with encoders. Packet Serial commands must be used to enable this option. Velocity and/or Position PID constants must be calibrated for proper operation using Packet Serial commands. Once calibrated values have been set and saved into Roboclaws eeprom, encoder support using velocity or position PID control can be enabled using commands 92 for motor 1 or 93 for motor 2. See the Packet Serial section for more details.

RC Mode With Mixing

This mode is the same as RC mode with the exception of how S1 and S2 control the attached motors. S1 controls speed and direction of both motors 1 and 2. S2 controls steering by slowing one of the motors. This drive method is similar to how a car would be controlled.

RC Mode Options

| Option	Function	Description
1	TTL Flip Switch	Flip switch triggered by low signal.
2	TTL Flip and Exponential Enabled	Softens the center control position. This mode is ideal with tank style robots. Making it easier to control from an RC radio. Flip switch triggered by low signal.
3	TTL Flip and MCU Enabled	Continues to execute last pulse received until new pulse received. Disables Signal loss fail safe and auto calibration. Flip switch triggered by low signal.
4	TTL Flip and Exponential and MCU Enabled	Enables both options. Flip switch triggered by low signal.
5	RC Flip Switch Enabled	Same as mode 1 with flip switch triggered by RC signal.
6	RC Flip and Exponential Enabled	Same as mode 2 with flip switch triggered by RC signal.
7	RC Flip and MCU Enabled	Same as mode 3 with flip switch triggered by RC signal.
8	RC Flip and Exponential and MCU Enabled	Same as mode 4 with flip switch triggered by RC signal.

Servo Pulse Ranges
The RoboClaw expects RC servo 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. 1000us is the default for full reverse and 2000us 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 1000us or larger than 2000us is detected the new pulses will be set as the new ranges.

Pulse	Function
1000us	Full Reverse
2000us	Full Forward

RC Wiring Example

Connect the RoboClaw as shown below. Set mode 1 with option 1. The configuration below uses a separate logic battery so remove the MB-LB jumper. Before powering up anything center the control sticks then turn the radio on first, then the receiver, then RoboClaw. It will take RoboClaw about 1 second to calibrate the neutral position.
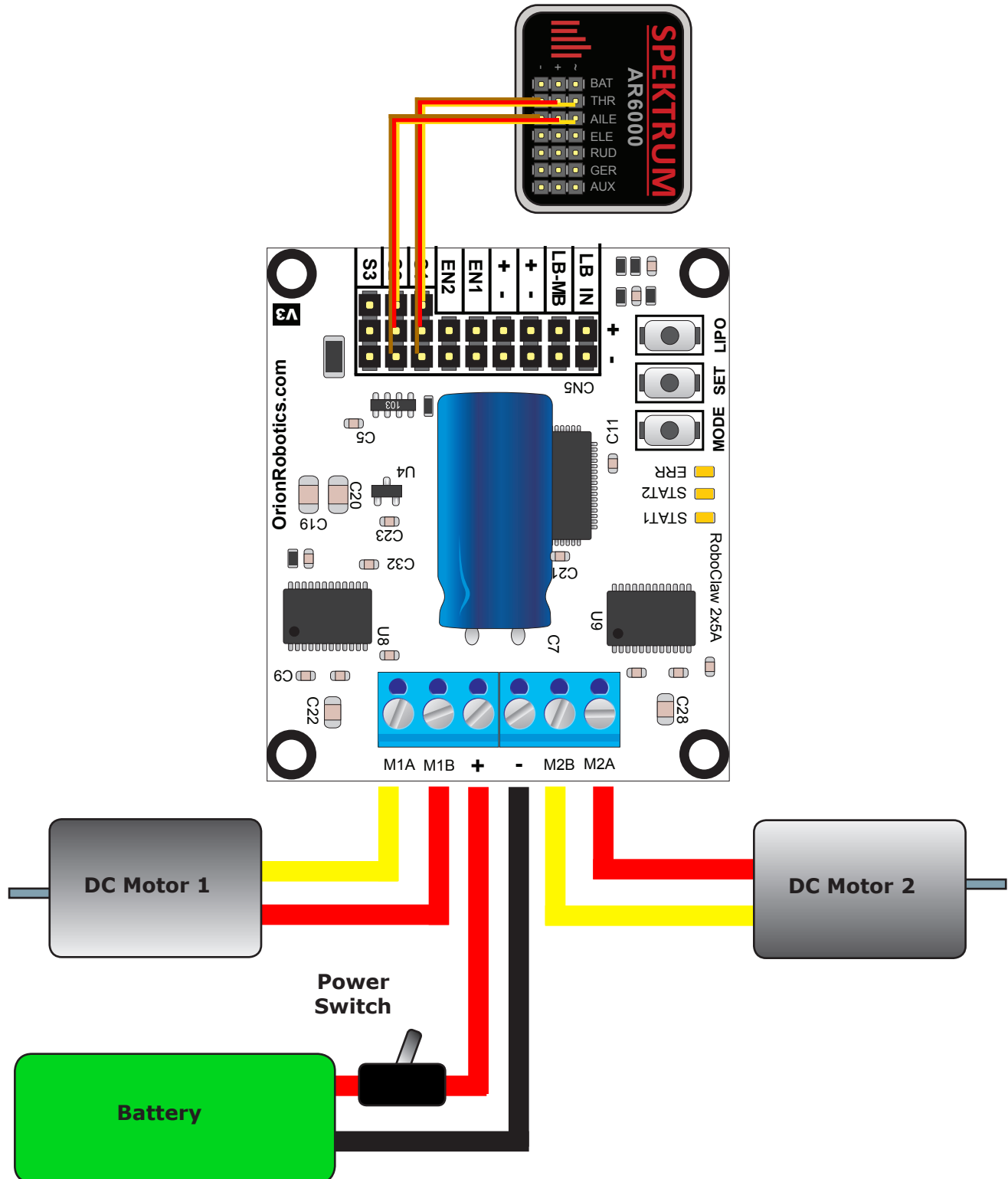
RC Control - Arduino Example

The example will drive a 2 motor 4 wheel robot in reverse, stop, forward, left turn and then right turn. The program was written and tested with a Arduino Uno and P5 connected to S1, P6 connected to S2. Set mode 2 with option 4.

```
//Basic Micro RoboClaw RC Mode. Control RoboClaw
//with servo pulses from a microcontroller.
//Mode settings: Mode 2 with Option 4.


#include <Servo.h>

Servo myservo1;  // create servo object to control a RoboClaw channel
Servo myservo2;  // create servo object to control a RoboClaw channel

int pos = 0;     // variable to store the servo position

void setup()
{
  myservo1.attach(5);  // attaches the RC signal on pin 5 to the servo object
  myservo2.attach(6);  // attaches the RC signal on pin 6 to the servo object
}


void loop()
{
  myservo1.writeMicroseconds(1500);  //Stop
  myservo2.writeMicroseconds(1500);  //Stop
  delay(2000);

  myservo1.writeMicroseconds(1250);  //full forward
  delay(1000);

  myservo1.writeMicroseconds(1500);  //stop
  delay(2000);

  myservo1.writeMicroseconds(1750);  //full reverse
  delay(1000);

  myservo1.writeMicroseconds(1500);  //Stop
  delay(2000);

  myservo2.writeMicroseconds(1250);  //full forward
  delay(1000);

  myservo2.writeMicroseconds(1500);  //Stop
  delay(2000);

  myservo2.writeMicroseconds(1750);  //full reverse
  delay(1000);
}
```

Analog Mode

Analog Mode

Analog mode is used when controlling RoboClaw from a potentiometer or a filtered PWM signal. In this mode S1 and S2 are set as analog inputs. Voltage range is 0V = Full reverse, 1V = Stop and 2V = Full forward.

Using Analog Mode with feedback for velocity/position control

Analog Mode can be used with encoders. Packet Serial commands must be used to enable this option. Velocity and/or Position PID constants must be calibrated for proper operation using Packet Serial commands. Once calibrated values have been set and saved into Roboclaws eeprom, encoder support using velocity or position PID control can be enabled using commands 92 for motor 1 or 93 for motor 2. See the Packet Serial section for more details.

Analog Mode With Mixing

This mode is the same as Analog mode with the exception of how S1 and S2 control the attached motors. S1 controls speed and direction of both motors 1 and 2. S2 controls steering by slowing one of the motors. This drive method is similar to how a car would be controlled.

Analog Mode Options

Option	Function	Description
1	TTL Flip Switch	Flip switch triggered by low signal.
2	TTL Flip and Exponential Enabled	Softens the center control position. This mode is ideal with tank style robots. Making it easier to control from an RC radio. Flip switch triggered by low signal.
3	TTL FLip and MCU Enabled	Continues to execute last pulse received until new pulse received. Disables Signal loss fail safe and auto calibration. Flip switch triggered by low signal.
4	TTL FLip and Exponential and MCU Enabled	Enables both options. Flip switch triggered by low signal.
5	RC Flip Switch Enabled	Same as mode 1 with flip switch triggered by RC signal.
6	RC Flip and Exponential Enabled	Same as mode 2 with flip switch triggered by RC signal.
7	RC Flip and MCU Enabled	Same as mode 3 with flip switch triggered by RC signal.
8	RC Flip and Exponential and MCU Enabled	Same as mode 4 with flip switch triggered by RC signal.

Analog Wiring Example

Connect the RoboClaw as shown below using two potentiometers. Set mode 3 with option 1. Center the potentiometers before applying power or the attached motors will start moving. S1 potentiometer will control motor 1 direction and speed. S2 potentiometer will control motor 2 direction and speed.
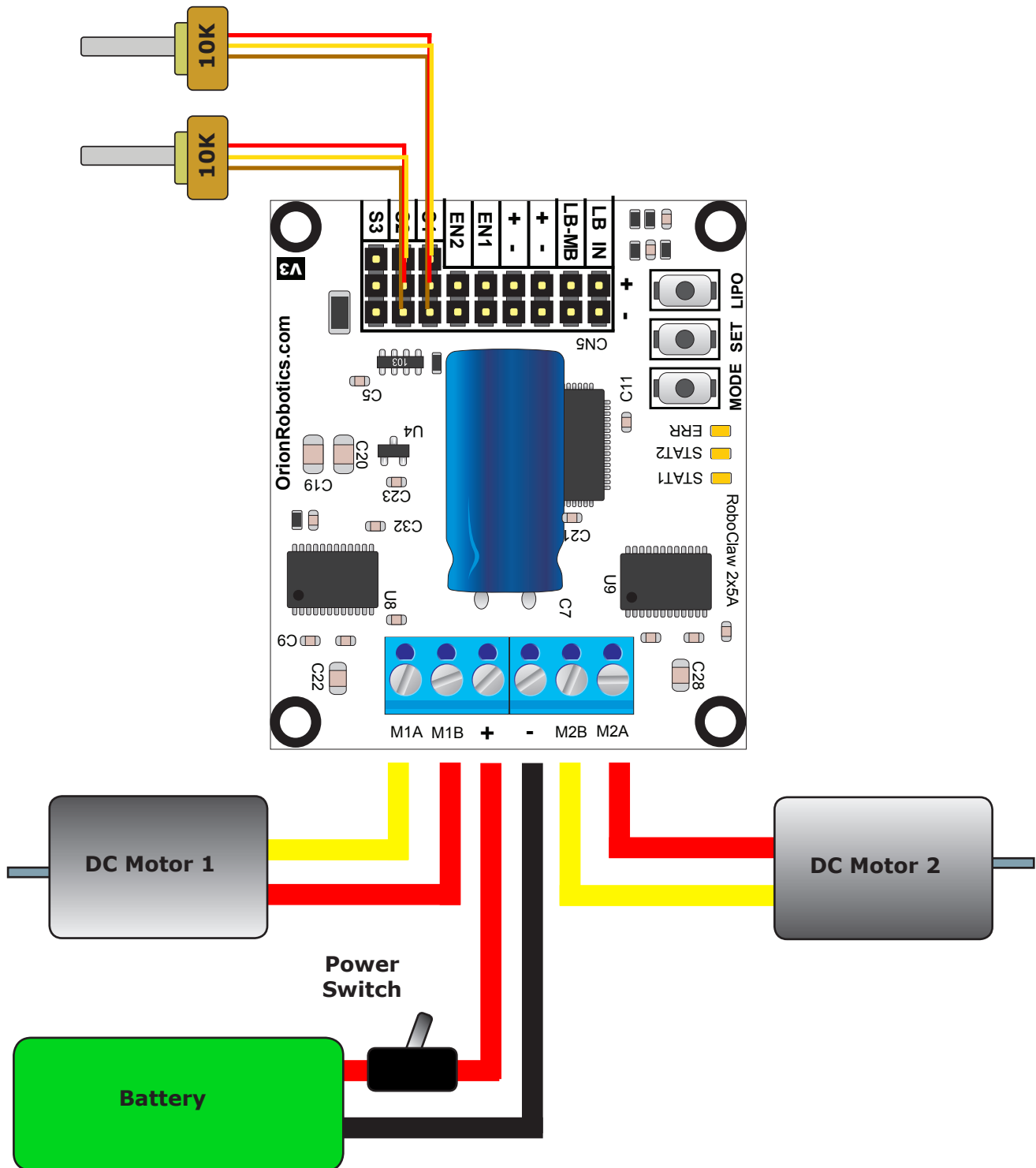
Simple Serial

Simple Serial Mode

In this mode S1 accepts TTL level byte commands. Simple serial mode is one way serial data. RoboClaw can receive only. 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 (See Max232) is required. The baud rate can be changed using the SET button once a serial mode has been selected.

Serial Mode Options

Option	Description
1	2400
2	9600
3	19200
4	38400

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 its S1 pin. Set S2 low (0V) and RoboClaw will ignore all received commands.

To setup up RoboClaw for serial slave make sure all RoboClaws share a common signal ground (GND) shown by the black wire. SOUT (Brown line) is connected to the S1 pin of all three RoboClaws which is the serial in of the RoboClaw. P1, P2 and P3 are connected to individual S2 pins. Only one MCU pin is connected to each RoboClaw S2 pin. To enable a RoboClaw hold its S2 pin high otherwise commands will be ignored.
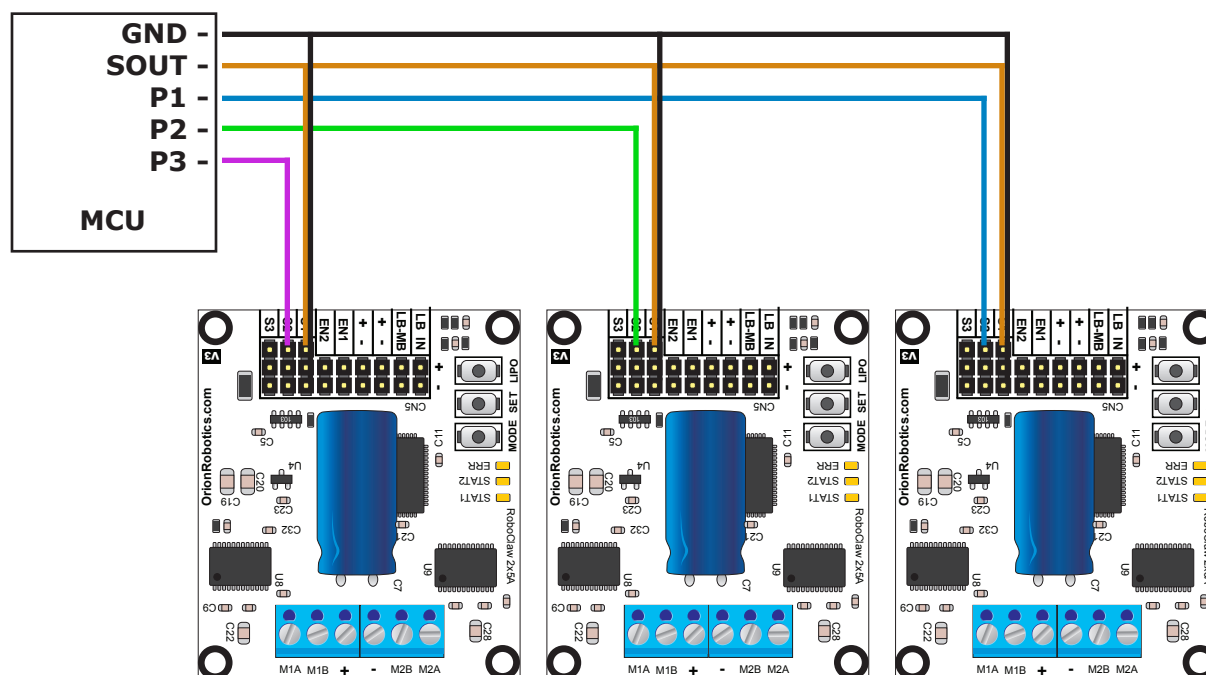
Simple Serial Command Syntax

The RoboClaw simple serial is setup to control both motors with one byte sized command character. Since a byte can be anything from 0 to 255 the control of each motor is split. 1 to 127 controls channel 1 and 128 to 255 controls channel 2. Command character 0 will shut down both channels. Any other values will control speed and direction of the specific channel.

Character	Function
0	Shuts Down Channel 1 and 2
1	Channel 1 - Full Reverse
64	Channel 1 - Stop
127	Channel 1 - Full Forward
128	Channel 2 - Full Reverse
192	Channel 2 - Stop
255	Channel 2 - Full Forward

Simple Serial Wiring Example

In simple serial mode the RoboClaw can only receive serial data. Use the below wiring diagram with the following code examples. Make sure the LB-MB jumper is installed.
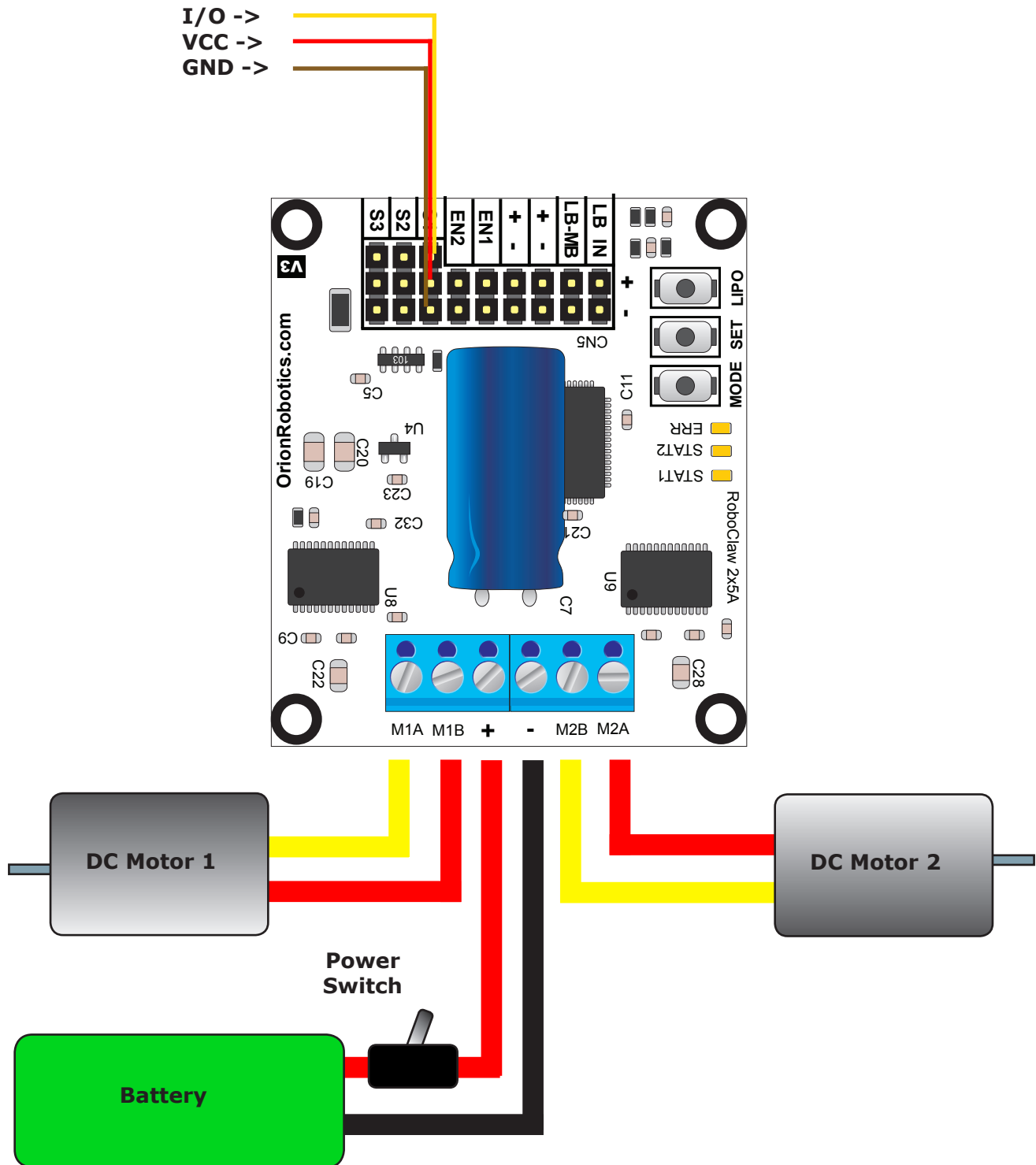
Simple Serial - Arduino Example

The following example will start both channels in reverse, then full speed forward. The program was written and tested with a Arduino Uno and Pin 5 connected to S1. Set mode 5 and option 3.

```
//Basic Micro RoboClaw Simple Serial Test
//Switch settings: SW2=ON and SW5=ON
//Make sure Arduino and Robo Claw share common GND!

#include "BMSerial.h"

BMSerial mySerial(5,6);

void setup() {
  mySerial.begin(19200);
}

void loop() {
  mySerial.write(1);
  mySerial.write(-1);
  delay(2000);
  mySerial.write(127);
  mySerial.write(-127);
  delay(2000);
}
```

Packet Serial

Packet Serial Mode
Packet serial is a buffered bidirectional serial mode. More sophisticated instructions can be sent to RoboClaw. The basic command structures consists of an address byte, command byte, data bytes and a checksum. The amount of data each command will send or receive can vary.

Address
Packet serial requires a unique address. With up to 8 addresses available you can have up to 8 RoboClaws bussed on the same RS232 port. There are 8 packet modes 7 to 14. Each mode has a unique address. The address is selected by setting the desired packet mode using the MODE button.

Packet Modes

Mode	Description
7	Packet Serial Mode - Address 0x80 (128)
8	Packet Serial Mode - Address 0x81 (129)
9	Packet Serial Mode - Address 0x82 (130)
10	Packet Serial Mode - Address 0x83 (131)
11	Packet Serial Mode - Address 0x84 (132)
12	Packet Serial Mode - Address 0x85 (133)
13	Packet Serial Mode - Address 0x86 (134)
14	Packet Serial Mode - Address 0x87 (135)

Packet Serial Baud Rate
When in serial mode or packet serial mode the baud rate can be changed to one of four different settings in the table below. These are set using the SET button as covered in Mode Options.

Serial Mode Options

Option	Description
1	2400
2	9600
3	19200
4	38400

Checksum Calculation

All packet serial commands use a 7 bit checksum to prevent corrupt commands from being executed. Since the RoboClaw expects a 7bit value the 8th bit is masked. The checksum is calculated as follows:

Checksum = (Address + Command + Data bytes) & 0x7F

When calculating the checksum all data bytes sent or received must be added together. The hexadecimal value 0X7F is used to mask the 8th bit.

Commands 0 - 7 Standard Commands
The following commands are the standard set of commands used with packet mode. The command syntax is the same for commands 0 to 7:

Address, Command, ByteValue, Checksum

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. Example with RoboClaw address set to 128:

```
Send: 128, 0, 127, ((128+0+127) & 0X7F)
```

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. Example with RoboClaw address set to 128:

```
Send: 128, 1, 127, ((128+0+127) & 0X7F)
```

2 - Set Minimum Main Voltage
Sets main battery (B- / B+) minimum voltage level. If the battery voltages drops below the set voltage level RoboClaw will shut down. The value is cleared at start up and must set after each power up. The voltage is set in .2 volt increments. A value of 0 sets the minimum value allowed which is 6V. The valid data range is 0 - 120 (6V - 30V). The formula for calculating the voltage is: (Desired Volts - 6) x 5 = Value. Examples of valid values are 6V = 0, 8V = 10 and 11V = 25. Example with RoboClaw address set to 128:

```
Send: 128, 2, 25, ((128+2+25) & 0X7F)
```

3 - Set Maximum Main Voltage
Sets main battery (B- / B+) maximum voltage level. The valid data range is 0 - 154 (0V - 30V). If you are using a battery of any type you can ignore this setting. During regenerative breaking a back voltage is applied to charge the battery. When using an ATX type power supply if it senses anything over 16V it will shut down. By setting the maximum voltage level, RoboClaw before exceeding it will go into hard breaking mode until the voltage drops below the maximum value set. The formula for calculating the voltage is: Desired Volts x 5.12 = Value. Examples of valid values are 12V = 62, 16V = 82 and 24V = 123. Example with RoboClaw address set to 128:

```
Send: 128, 3, 82, ((128+3+82) & 0X7F)
```

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. Example with RoboClaw address set to 128:

```
Send: 128, 4, 127, ((128+4+127) & 0X7F)]
```

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. Example with RoboClaw address set to 128:

```
Send: 128, 5, 127, ((128+5+127) & 0X7F)
```

6 - Drive M1 (7 Bit)
Drive motor 1 forward and reverse. Valid data range is 0 - 127. A value of 0 = full speed reverse, 64 = stop and 127 = full speed forward. Example with RoboClaw address set to 128:

```
Send: 128, 6, 96, ((128+6+96) & 0X7F)
```

7 - Drive M2 (7 Bit)
Drive motor 2 forward and reverse. Valid data range is 0 - 127. A value of 0 = full speed reverse, 64 = stop and 127 = full speed forward. Example with RoboClaw address set to 128:

```
Send: 128, 7, 32, ((128+7+32) & 0X7F)
```

Commands 8 - 13 Mix Mode Commands

The following commands are mix mode commands and used to control speed and turn. Before a command is executed valid drive and turn data is required. You only need to send both data packets once. After receiving both valid drive and turn data RoboClaw will begin to operate. At this point you only need to update turn or drive data.

8 - Drive Forward

Drive forward in mix mode. Valid data range is 0 - 127. A value of 0 = full stop and 127 = full forward. Example with RoboClaw address set to 128:

```
Send: 128, 8, 127, ((128+8+127) & 0x7F)
```

9 - Drive Backwards

Drive backwards in mix mode. Valid data range is 0 - 127. A value of 0 = full stop and 127 = full reverse. Example with RoboClaw address set to 128:

```
Send: 128, 9, 127, ((128+9+127) & 0x7F)
```

10 - Turn right

Turn right in mix mode. Valid data range is 0 - 127. A value of 0 = stop turn and 127 = full speed turn. Example with RoboClaw address set to 128:

```
Send: 128, 10, 127, ((128+10+127) & 0x7F1)
```

11 - Turn left

Turn left in mix mode. Valid data range is 0 - 127. A value of 0 = stop turn and 127 = full speed turn. Example with RoboClaw address set to 128:

```
Send: 128, 11, 127, ((128+11+127) & 0x7F)
```

12 - Drive Forward or Backward (7 Bit)

Drive forward or backwards. Valid data range is 0 - 127. A value of 0 = full backward, 64 = stop and 127 = full forward. Example with RoboClaw address set to 128:

```
Send: 128, 12, 96, ((128+12=96) & 0x7F)
```

13 - Turn Left or Right (7 Bit)

Turn left or right. Valid data range is 0 - 127. A value of 0 = full left, 0 = stop turn and 127 = full right. Example with RoboClaw address set to 128:

```
Send: 128, 13, 0, ((128+13=0) & 0x7F)
```

Packet Serial Wiring

In packet mode the RoboClaw can transmit and receive serial data. A microcontroller with a UART is recommended. The UART will buffer the data received from RoboClaw. When a request for data is made to RoboClaw the return data will always have at least a 1ms delay after the command is received. This will allow slower processors and processors without UARTs to communicate with RoboClaw.
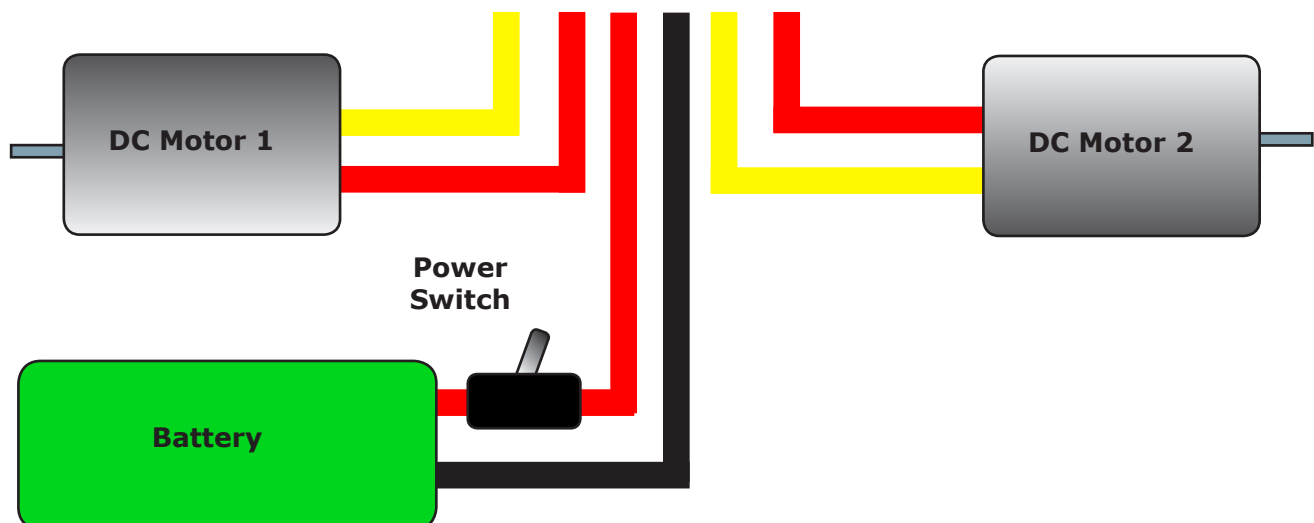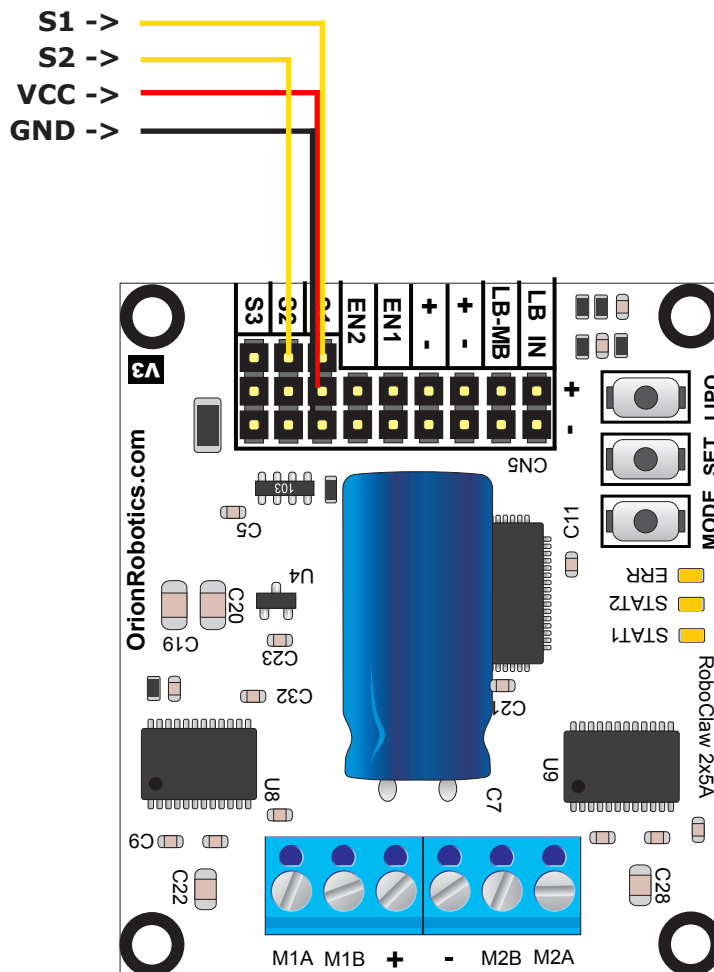
Packet Serial - Arduino Example
The example will start the motor channels independently. Then start turns with mix mode commands. The program was written and tested with a Arduno Uno and P5 connected to S1. Set mode 7 and option 3.

```
//Basic Micro RoboClaw Packet Serial Test Commands 0 to 13.
//Switch settings: SW3=ON and SW5=ON.

#include "BMSerial.h"
#include "RoboClaw.h"

#define address 0x80

RoboClaw roboclaw(5,6);

void setup() {
  roboclaw.begin(19200);
}

void loop() {
  roboclaw.ForwardM1(address,64);  //Cmd 0
  roboclaw.BackwardM2(address,64); //Cmd 5
  delay(2000);
  roboclaw.BackwardM1(address,64); //Cmd 1
  roboclaw.ForwardM2(address,64);  //Cmd 6
  delay(2000);
  roboclaw.ForwardBackwardM1(address,96);      //Cmd 6
  roboclaw.ForwardBackwardM2(address,32);      //Cmd 7
  delay(2000);
  roboclaw.ForwardBackwardM1(address,32);      //Cmd 6
  roboclaw.ForwardBackwardM2(address,96);      //Cmd 7
  delay(2000);

  //stop motors
  roboclaw.ForwardBackwardM1(address,0);
  roboclaw.ForwardBackwardM2(address,0);

  delay(10000);

  roboclaw.ForwardMixed(address, 64);     //Cmd 8
  delay(2000);
  roboclaw.BackwardMixed(address, 64);    //Cmd 9
  delay(2000);
  roboclaw.TurnRightMixed(address, 64);  //Cmd 10
  delay(2000);
  roboclaw.TurnLeftMixed(address, 64);        //Cmd 11
  delay(2000);
  roboclaw.ForwardBackwardMixed(address, 32);  //Cmd 12
  delay(2000);
  roboclaw.ForwardBackwardMixed(address, 96);  //Cmd 12
  delay(2000);
  roboclaw.LeftRightMixed(address, 32);  //Cmd 13
  delay(2000);
  roboclaw.LeftRightMixed(address, 96);  //Cmd 13
  delay(2000);

  //stop motors
  roboclaw.ForwardMixed(address, 0);

  delay(10000);
}
```

Advanced Packet Serial

Version, Status, and Settings Commands
The following commands are used to read board status, version information and set configuration values.

Command	Description
21	Read Firmware Version
24	Read Main Battery Voltage
25	Read Logic Battery Voltage
26	Set Minimum Logic Voltage Level
27	Set Maximum Logic Voltage Level
49	Read Motor Currents
55	Read Motor 1 Velocity PID Constants
56	Read Motor 2 Velocity PID Constants
57	Set Main Battery Voltages
58	Set Logic Battery Voltages
59	Read Main Battery Voltage Settings
60	Read Logic Battery Voltage Settings
63	Read Motor 1 Position PID Constants
64	Read Motor 2 Position PID Constants
82	Read Temperature
90	Read Error Status
91	Read Encoder Mode
92	Set Motor 1 Encoder Mode
93	Set Motor 1 Encoder Mode
94	Write Settings to EEPROM

21 - Read Firmware Version
Read RoboClaw firmware version. Returns up to 32 bytes and is terminated by a null character. Command syntax:

```
Send: [Address, 21]
Receive: ["RoboClaw 10.2A v1.3.9, Checksum]
```

The command will return up to 32 bytes. The return string includes the product name and firmware version. The return string is terminated with a null (0) character.

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. Command syntax:

```
Send: [Address, 24]
Receive: [Value.Byte1, Value.Byte0, Checksum]
```

The command will return 3 bytes. Byte 1 and 2 make up a word variable which is received MSB first and is 10th of a volt. A returned value of 300 would equal 30V. Byte 3 is the checksum. It is calculated the same way as sending a command and can be used to validate the data.

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. Command syntax:

```
Send: [Address, 25]
Receive: [Value.Byte1, Value.Byte0, Checksum]
```

The command will return 3 bytes. Byte 1 and 2 make up a word variable which is received MSB first and is 10th of a volt. A returned value of 50 would equal 5V. Byte 3 is the checksum. It is calculated the same way as sending a command and can be used to validate the data.

26 - Set Minimum Logic Voltage Level

Sets logic input (LB- / LB+) minimum voltage level. If the battery voltages drops below the set voltage level RoboClaw will shut down. The value is cleared at start up and must set after each power up. The voltage is set in .2 volt increments. A value of 0 sets the minimum value allowed which is 3V. The valid data range is 0 - 120 (6V - 28V). The formula for calculating the voltage is: (Desired Volts - 6) x 5 = Value. Examples of valid values are 3V = 0, 8V = 10 and 11V = 25.

```
Send: [128, 26, 0, (154 & 0X7F)]
```

27 - Set Maximum Logic Voltage Level

Sets logic input (LB- / LB+) maximum voltage level. The valid data range is 0 - 144 (0V - 28V). By setting the maximum voltage level RoboClaw will go into shut down and requires a hard reset to recovers. The formula for calculating the voltage is: Desired Volts x 5.12 = Value. Examples of valid values are 12V = 62, 16V = 82 and 24V = 123.

```
Send: [128, 27, 82, (213 & 0X7F)]
```

49 - Read Motor Currents

Read the current draw from each motor in 10ma increments. Command syntax:

```
Send: [Address, 49]
Receive: [M1Cur.Byte1, M1Cur.Byte0, M2Cur.Byte1, M2Cur.Byte0, Checksum]
```

The command will return 5 bytes. Bytes 1 and 2 combine to represent the current in 10ma increments of motor1. Bytes 3 and 4 combine to represent the current in 10ma increments of motor2 . Byte 5 is the checksum.

55 - Read Motor 1 P, I, D and QPPS Settings

Read the PID and QPPS Settings. Command syntax:

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

56 - Read Motor 2 P, I, D and QPPS Settings

Read the PID and QPPS Settings. Command syntax:

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

57 - Set Main Battery Voltages
Set the Main Battery Voltages cutoffs, Min and Max. Command syntax:

```
Send: [Address, 57, Min(2 bytes), Max(2bytes, Checksum]
```

58 - Set Logic Battery Voltages
Set the Logic Battery Voltages cutoffs, Min and Max. Command syntax:

```
Send: [Address, 58, Min(2 bytes), Max(2bytes, Checksum]
```

59 - Read Main Battery Voltage Settings
Read the Main Battery Voltage Settings. Command syntax:

```
Send: [Address, 59]
Receive: [Min(2 bytes), Max(2 bytes), Checksum]
```

60 - Read Logic Battery Voltage Settings
Read the Main Battery Voltage Settings. Command syntax:

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

63 - Read Motor 1 Position P, I, D Constants
Read the Position PID Settings. Command syntax:

```
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), Checksum]
```

64 - Read Motor 2 Position P, I, D Constants
Read the Position PID Settings. Command syntax:

```
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), Checksum]
```

82 - Read Temperature
Read the board temperature. Value returned is in 0.1 degree increments. Command syntax:

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

90 - Read Error Status
Read the current error status. Command syntax:

```
Send: [Address, 90]
Receive: [Error, Checksum]
```

Error Mask
Normal	0x00
M1 OverCurrent	0x01
M2 OverCurrent	0x02
E-Stop	0x04
Temperature	0x08
Main Battery High	0x10
Main Battery Low	0x20
Logic Battery High	0x40
Logic Battery Low	0x80

91 - Read Encoder Mode
Read the encoder mode for both motors. Command syntax:

```
Send: [Address, 91]
Receive: [Mode1, Mode2, Checksum]
```

92 - Set Motor 1 Encoder Mode
Set the Encoder Mode for motor 1. Command syntax:

```
Send: [Address, 92, Mode, Checksum]
```

93 - Set Motor 2 Encoder Mode
Set the Encoder Mode for motor 1. Command syntax:

```
Send: [Address, 93, Mode, Checksum]
```

Encoder Mode bits
Bit 7	Enable RC/Analog Encoder support
Bit 6-1	N/A
Bit 0	Quadrature(0)/Absolute(1)

94 - Write Settings to EEPROM
Writes all settings to non-volatile memory. Command syntax:

```
Send: [Address, 94]
Receive: [Checksum]
```

Quadrature Decoding

Quadrature Encoder Wiring

RoboClaw is capable of reading two quadrature encoders one for each motor channel. The main RoboClaw header provides two +5VDC connections with dual A and B input signals.

In a two motor robot configuration one motor will spin clock wise (CW) while the other motor will spin counter clock wise (CCW). The A and B inputs for one of the two encoders must be reversed as shown. If either encoder is connected wrong one will count up and the other down this will cause commands like mix drive forward to not work properly.
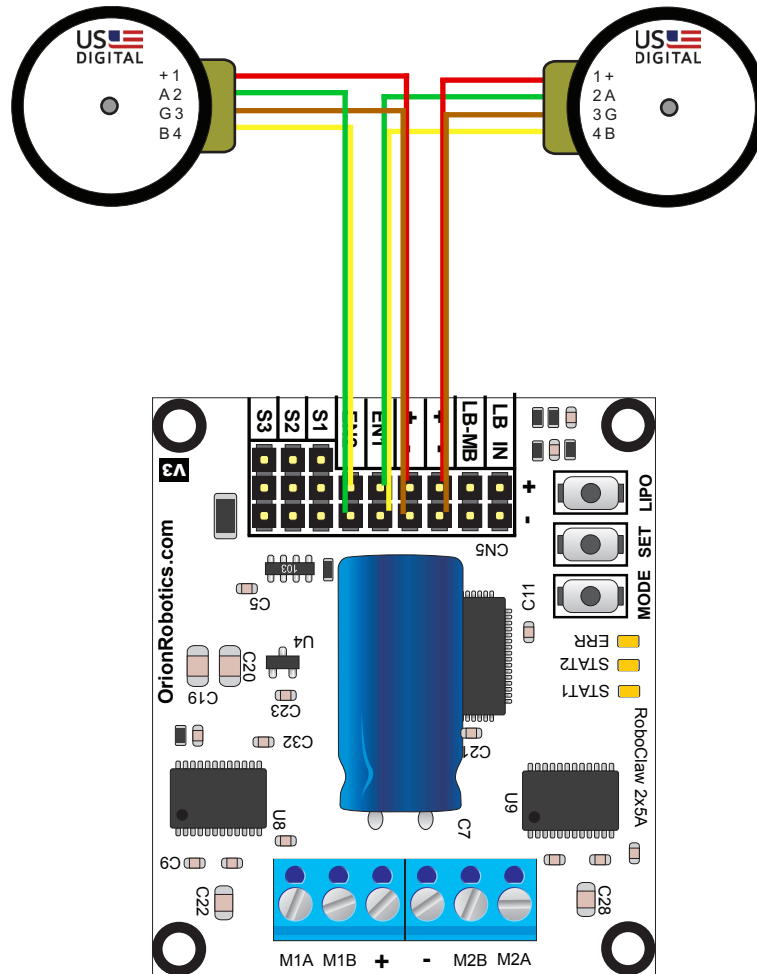
Quadrature Encoder Commands

The following commands are used in dealing with the quadrature decoding counter registers. The quadrature decoder is a simple counter that counts the incoming pulses, tracks the direction and speed of each pulse. There are two registers one each for M1 and M2. (Note: A microcontroller with a hardware UART is recommended for use with packet serial modes).

Command	Description
16	Read Quadrature Encoder Register for M1.
17	Read Quadrature Encoder Register for M2.
18	Read M1 Speed in Pulses Per Second.
19	Read M2 Speed in Pulses Per Second.
20	Resets Quadrature Encoder Registers for M1 and M2.

16 - Read Quadrature Encoder Register M1

Read decoder M1 counter. Since CMD 16 is a read command it does not require a checksum. However a checksum value will be returned from RoboClaw and can be used to validate the data. Command syntax:

```
Send: [Address, CMD]
Receive: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2,
Checksum]
```

The command will return 6 bytes. Byte 1,2,3 and 4 make up a long variable which is received MSB first and represents the current count which can be any value from 0 - 4,294,967,295. Each pulse from the quadrature encoder will increment or decrement the counter depending on the direction of rotation.

Byte 5 is the status byte for M1 decoder. It tracks counter underflow, direction, overflow and if the encoder is operational. 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

Byte 6 is the checksum. It is calculated the same way as sending a command, Sum all the values sent and received except the checksum and mask the 8th bit.

17 - Read Quadrature Encoder Register M2

Read decoder M2 counter. Since CMD 16 is a read command it does not require a checksum. However a checksum value will be returned from RoboClaw and can be used to validate the data. Command syntax:

```
Send: [Address, CMD]
Receive: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]
```

The command will return 6 bytes. Byte 1,2,3 and 4 make up a long variable which is received MSB first and represents the current count which can be any value from 0 - 4,294,967,295. Each pulse from the quadrature encoder will increment or decrement the counter depending on the direction of rotation.

Byte 5 is the status byte for M1 decoder. It tracks counter underflow, direction, overflow and if the encoder is operational. 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

Byte 6 is the checksum.

18 - Read 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 decoder channels. Since CMD 18 is a read command it does not require a checksum to be sent. However a checksum value will be returned from RoboClaw and can be used to validate the data. Command syntax:

```
Send: [Address, CMD]
Receive: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]
```

The command will return 6 bytes. Byte 1,2,3 and 4 make up a long variable which is received MSB first and is the current ticks per second which can be any value from 0 - 4,294,967,295. Byte 5 is the direction (0 – forward, 1 - backward). Byte 6 is the checksum.

19 - Read 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 decoder channels. Since CMD 19 is a read command it does not require a checksum to be sent. However a checksum value will be returned from RoboClaw and can be used to validate the data. Command syntax:

```
Send: [Address, CMD]
Receive: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]
```

The command will return 6 bytes. Byte 1,2,3 and 4 make up a long variable which is received MSB first and is the current ticks per second which can be any value from 0 - 4,294,967,295. Byte 5 is the direction (0 – forward, 1 - backward). Byte 6 is the checksum.

20 - Reset Quadrature Encoder Counters

Will reset both quadrature decoder counters to zero.

```
Send: [128, 20, ((128+20) & 0x7F)]
```

Advanced Motor Control
The following commands are used to control motor speeds, acceleration and distance using the quadrature encoders. All speeds are given in quad pulses per second (QPPS) unless otherwise stated. Quadrature encoders of different types and manufactures can be used. However many have different resolutions and maximum speeds at which they operate. So each quadrature encoder will produce a different range of pulses per second.

Command	Description
28	Set PID Constants for M1.
29	Set PID Constants for M2.
30	Read Current M1 Speed Resolution 125th of a Second.
31	Read Current M2 Speed Resolution 125th of a Second.
32	Drive M1 With Signed Duty Cycle. (Encoders not required)
33	Drive M2 With Signed Duty Cycle. (Encoders not required)
34	Mix Mode Drive M1 / M2 With Signed Duty Cycle. (Encoders not required)
35	Drive M1 With Signed Speed.
36	Drive M2 With Signed Speed.
37	Mix Mode Drive M1 / M2 With Signed Speed.
38	Drive M1 With Signed Speed And Acceleration.
39	Drive M2 With Signed Speed And Acceleration.
40	Mix Mode Drive M1 / M2 With Speed And Acceleration.
41	Drive M1 With Signed Speed And Distance. Buffered.
42	Drive M2 With Signed Speed And Distance. Buffered.
43	Mix Mode Drive M1 / M2 With Speed And Distance. Buffered.
44	Drive M1 With Signed Speed, Acceleration and Distance. Buffered.
45	Drive M2 With Signed Speed, Acceleration and Distance. Buffered.
46	Mix Mode Drive M1 / M2 With Speed, Acceleration And Distance. Buffered.
47	Read Buffer Length.
50	Mix Drive M1 / M2 With Individual Speed and Acceleration
51	Mix Drive M1 / M2 With Individual Speed, Accel and Distance
52	Drive M1 With Duty and Accel. (Encoders not required)
53	Drive M2 With Duty and Accel. (Encoders not required)
54	Mix Drive M1 / M2 With Duty and Accel. (Encoders not required)
61	Set Position PID Constants for M1.
62	Set Position PID Constants for M2
65	Drive M1 with signed Speed, Accel, Deccel and Position
66	Drive M2 with signed Speed, Accel, Deccel and Position
67	Drive M1 & M2 with signed Speed, Accel, Deccel and Position

28 - Set 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), Checksum]
```

Each value is made up of 4 bytes for a long. To write the registers a checksum value is used. This prevents an accidental write.

29 - Set 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), Checksum]
```

Each value is made up of 4 bytes for a long. To write the registers a checksum value is used. This prevents an accidental write.

30 - Read Current Speed M1

Read the current pulse per 125th of a second. This is a high resolution version of command 18 and 19. Command 30 can be used to make a independent PID routine. The resolution of the command is required to create a PID routine using any microcontroller or PC used to drive RoboClaw. The command syntax:

```
Send: [Address, CMD]
Receive: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]
```

The command will return 5 bytes, MSB sent first for a long. The first 4 bytes are a 32 byte value (long) that repersent the speed. The 5th byte (Value2) is direction (0 – forward, 1 - backward). is A checksum is returned in order to validate the data returned.

31 - Read Current Speed M2

Read the current pulse per 125th of a second. This is a high resolution version of command 18 and 19. Command 31 can be used to make a independent PID routine. The resolution of the command is required to create a PID routine using any microcontroller or PC used to drive RoboClaw. The command syntax:

```
Send: [Address, CMD]
Receive: [Value1.Byte3, Value1.Byte2, Value1.Byte1, Value1.Byte0, Value2, Checksum]
```

The command will return 5 bytes, MSB sent first for a long. The first 4 bytes are a 32 byte value (long) that repersent the speed. The 5th byte (Value2) is direction (0 – forward, 1 - backward). is A checksum is returned in order to validate the data returned.

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. The command syntax:

```
Send: [Address, CMD, Duty(2 Bytes), Checksum]
```

The duty value is signed and the range is +-1500.

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, CMD, Duty(2 Bytes), Checksum]
```

The duty value is signed and the range is +-1500.

34 - Mix Mode 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, CMD, DutyM1(2 Bytes), DutyM2(2 Bytes), Checksum]
```

The duty value is signed and the range is +-1500.

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. The command syntax:

```
Send: [Address, CMD, Qspeed(4 Bytes), Checksum]
```

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses.

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. The command syntax:

```
Send: [Address, CMD, Qspeed(4 Bytes), Checksum]
```

4 Bytes (long) are used to expressed the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses.

37 - Mix Mode 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. The command syntax:

```
Send: [Address, CMD, QspeedM1(4 Bytes), QspeedM2(4 Bytes), Checksum]
```

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses.

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. The command syntax:

```
Send: [Address, CMD, Accel(4 Bytes), Qspeed(4 Bytes), Checksum]
```

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses. The acceleration is measured in speed 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. The command syntax:

```
Send: [Address, CMD, Accel(4 Bytes), Qspeed(4 Bytes), Checksum]
```

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses. The acceleration is measured in speed 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 - Mix Mode 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. The command syntax:

        Send: [Address, CMD, Accel(4 Bytes), QspeedM1(4 Bytes), QspeedM2(4 Bytes), Checksum]

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses. The acceleration is measured in speed 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 - Buffered 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. The command syntax:

        Send: [Address, CMD, QSpeed(4 Bytes), Distance(4 Bytes), Buffer(1 Byte), Checksum]

4 Bytes(long) are used to express the pulses per second. 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 - Buffered 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. The command syntax:

        Send: [Address, CMD, QSpeed(4 Bytes), Distance(4 Bytes), Buffer(1 Byte), Checksum]

4 Bytes(long) are used to express the pulses per second. 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 - Buffered Mix Mode 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. The command syntax:

```
Send: [Address, CMD, QSpeedM1(4 Bytes), DistanceM1(4 Bytes),
       QSpeedM2(4 Bytes), DistanceM2(4 Bytes), Buffer(1 Byte), Checksum]
```

4 Bytes(long) are used to express the pulses per second. 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 - Buffered 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. The command syntax:

```
Send: [Address, CMD, Accel(4 bytes), QSpeed(4 Bytes), Distance(4 Bytes),
       Buffer(1 Byte), Checksum]
```

4 Bytes(long) are used to express the pulses per second. 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 - Buffered 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. The command syntax:

```
Send: [Address, CMD, Accel(4 bytes), QSpeed(4 Bytes), Distance(4 Bytes),
       Buffer(1 Byte), Checksum]
```

4 Bytes(long) are used to express the pulses per second. 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 - Buffered Mix Mode 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. The command syntax:

```
Send: [Address, CMD, Accel(4 Bytes), QSpeedM1(4 Bytes), DistanceM1(4 Bytes),
QSpeedM2(4 bytes), DistanceM2(4 Bytes), Buffer(1 Byte), Checksum]
```

4 Bytes(long) are used to express the pulses per second. 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.

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, CMD]
Receive: [BufferM1(1 Bytes), BufferM2(1 Bytes), Checksum]
```

The return values represent how many commands per buffer are waiting to be executed. The maximum buffer size per motor is 31 commands. 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.

50 - Mix Mode Drive M1 / M2 With Signed Speed And Individual Accelerations

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. The command syntax:

```
Send: [Address, CMD, AccelM1(4 Bytes), QspeedM1(4 Bytes), AccelM2(4 Bytes),
QspeedM2(4 Bytes), Checksum]
```

4 Bytes (long) are used to express the pulses per second. Quadrature encoders send 4 pulses per tick. So 1000 ticks would be counted as 4000 pulses. The acceleration is measured in speed 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 Mix Mode 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. The command syntax:

```
Send: [Address, CMD, AccelM1(4 Bytes), QSpeedM1(4 Bytes), DistanceM1(4 Bytes),
AccelM2(4 Bytes), QSpeedM2(4 bytes), DistanceM2(4 Bytes), Buffer(1 Byte), Check-
sum]
```

4 Bytes(long) are used to express the pulses per second. 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.

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. The command syntax:

```
Send: [Address, CMD, Duty(2 bytes), Accel(2 Bytes), Checksum]
```

The duty value is signed and the range is +-1500.
The accel value range is 0 to 65535

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. The command syntax:

```
Send: [Address, CMD, Duty(2 bytes), Accel(2 Bytes), Checksum]
```

The duty value is signed and the range is +-1500.
The accel value range is 0 to 65535

54 - Mix Mode 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), Checksum]
```

The duty value is signed and the range is +-1500.
The accel value range is 0 to 65535

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, CMD, P(4 bytes), I(4 bytes), D(4 bytes), MaxI(4 bytes), Deadzone(4
bytes), MinPos(4 bytes), MaxPos(4 bytes)
```

Position constants are used only with the Position commands, 65,66 and 67 and RC or Analog mode when in absolute mode with encoders or potentiometers.

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, CMD, P(4 bytes), I(4 bytes), D(4 bytes), MaxI(4 bytes), Deadzone(4
bytes), MinPos(4 bytes), MaxPos(4 bytes)
```

Position constants are used only with the Position commands, 65,66 and 67 and RC or Analog mode when in absolute mode with encoders or potentiometers.

65 - 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. The command syntax:

```
Send: [Address, CMD, Accel(4 bytes), QSpeed(4 Bytes), Deccel(4 bytes), Position(4
Bytes), Buffer(1 Byte), Checksum]
```

66 - 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. The command syntax:

```
Send: [Address, CMD, Accel(4 bytes), QSpeed(4 Bytes), Deccel(4 bytes), Position(4
Bytes), Buffer(1 Byte), Checksum]
```

67 - 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. The command syntax:

```
Send: [Address, CMD, Accel(4 bytes), QSpeed(4 Bytes), Deccel(4 bytes), Position(4
Bytes),Buffer(1 Byte), Checksum]
```

USB Information

USB RoboClaw Power
The USB RoboClaw is self powered. Which means it is not powered from the USB cable. The USB RoboClaw must be externally powered to function correctly.

USB RoboClaw Connection
The USB RoboClaw should have its USB cable connected before powering it up unless USB mode is specifically set (mode 15). If the master controller (the PC) is powered up the USB RoboClaw will automatically detect it is connected to a powered USB master and will enter USB mode. In some cases it may be necessairy to set USB mode manually by setting RoboClaw to mode 15.

USB Comport and baudrate
The USB RoboClaw will be detected as a CDC Virtual Comport. When connected to a Windows PC a driver must be installed. The driver is available for download. On Linux or OSX the RoboClaw will be automatically detected as a virtual comport and an appropriate driver will automatically be loaded.

Unlike a real Comport the USB CDC Virtual Comport does not need a baud rate to be set. It will always communicate at the fastest speed the master and slave device can reach. This will typically be 1mbit/s.

Reading Quadrature Encoder - Arduino Example

The example was tested with an Arduino Uno. RoboClaw was connected as shown in both packet serial wiring and quadrature encoder wiring diagrams.

The example will read the speed, total ticks and direction of each encoder. Connect to the program using a terminal window set to 38400 baud. The program will display the values of each encoders current count along with each encoder status bit in binary and the direction bit. As the encoder is turned it will update the screen.

```
//Basic Micro RoboClaw Packet Serial Mode.
//Switch settings: SW3=ON, SW4=ON, SW5=ON

#include "BMSerial.h"
#include "RoboClaw.h"

#define address 0x80

#define Kp 0x00010000
#define Ki 0x00008000
#define Kd 0x00004000
#define qpps 44000

BMSerial terminal(0,1);
RoboClaw roboclaw(5,6);

void setup() {
  terminal.begin(38400);
  roboclaw.begin(38400);

  roboclaw.SetM1Constants(address,Kd,Kp,Ki,qpps);
  roboclaw.SetM2Constants(address,Kd,Kp,Ki,qpps);
}

void loop() {
  uint8_t status;
  bool valid;

  uint32_t enc1= roboclaw.ReadEncM1(address, &status, &valid);
  if(valid){
    terminal.print("Encoder1:");
    terminal.print(enc1,HEX);
    terminal.print(" ");
    terminal.print(status,HEX);
    terminal.print(" ");
  }
  uint32_t enc2 = roboclaw.ReadEncM2(address, &status, &valid);
  if(valid){
    terminal.print("Encoder2:");
    terminal.print(enc2,HEX);
    terminal.print(" ");
    terminal.print(status,HEX);
    terminal.print(" ");
  }
  uint32_t speed1 = roboclaw.ReadSpeedM1(address, &status, &valid);
  if(valid){
    terminal.print("Speed1:");
    terminal.print(speed1,HEX);
    terminal.print(" ");
  }
  uint32_t speed2 = roboclaw.ReadSpeedM2(address, &status, &valid);
  if(valid){
    terminal.print("Speed2:");
    terminal.print(speed2,HEX);
    terminal.print(" ");
  }
  terminal.println();

  delay(100);
}
```

Speed Controlled by Quadrature Encoders - Arduino Example

The following example was written using an Arduino UNO. RoboClaw was connected as shown in both packet serial wiring and quadrature encoder wiring diagrams.

The example will command a 4wheel robot to move forward, backward, right turn and left turn slowly. You can change the speed by adjusting the value of Speed and Speed2 variables.

```
//Basic Micro RoboClaw Packet Serial Mode.
//Switch settings: SW3=ON, SW4=ON, SW5=ON

#include "BMSerial.h"
#include "RoboClaw.h"

#define address 0x80


#define Kp 0x00010000
#define Ki 0x00008000
#define Kd 0x00004000
#define qpps 44000

BMSerial terminal(0,1);
RoboClaw roboclaw(5,6);

void setup() {
  terminal.begin(38400);
  roboclaw.begin(38400);

  roboclaw.SetM1Constants(address,Kd,Kp,Ki,qpps);
  roboclaw.SetM2Constants(address,Kd,Kp,Ki,qpps);
}

void displayspeed(void)
{
  uint8_t status;
  bool valid;

  uint32_t enc1= roboclaw.ReadEncM1(address, &status, &valid);
  if(valid){
    terminal.print("Encoder1:");
    terminal.print(enc1,DEC);
    terminal.print(" ");
    terminal.print(status,HEX);
    terminal.print(" ");
  }
  uint32_t enc2 = roboclaw.ReadEncM2(address, &status, &valid);
  if(valid){
    terminal.print("Encoder2:");
    terminal.print(enc2,DEC);
    terminal.print(" ");
    terminal.print(status,HEX);
    terminal.print(" ");
  }
```

```
    uint32_t speed1 = roboclaw.ReadSpeedM1(address, &status, &valid);
    if(valid){
      terminal.print("Speed1:");
      terminal.print(speed1,DEC);
      terminal.print(" ");
    }
    uint32_t speed2 = roboclaw.ReadSpeedM2(address, &status, &valid);
    if(valid){
      terminal.print("Speed2:");
      terminal.print(speed2,DEC);
      terminal.print(" ");
    }
    terminal.println();
}

void loop() {
  roboclaw.SpeedAccelDistanceM1(address,12000,12000,48000);
  uint8_t depth1,depth2;
  do{
    displayspeed();
    roboclaw.ReadBuffers(address,depth1,depth2);
  }while(depth1);
  roboclaw.SpeedAccelDistanceM1(address,12000,-12000,48000);
  do{
    displayspeed();
    roboclaw.ReadBuffers(address,depth1,depth2);
  }while(depth1);

}
```

Electrical Characteristics

Characteristic	Rating	Min	Typ	Max
Pulse Per Second	PPS	0		8,000,000
Main Battery (B+ / B-)	VDC	6		34
Logic Battery (LB+ / LB-)	VDC	6	12	34
External Current Draw (BEC)	mA			150
Logic Circuit	mA		30	
Motor Current Per Channel	A		5	10
I/O Input	VDC	0		5
I/O Output	VDC	0		3.3
Analog Voltage Range	VDC	0		2
Tempature Range	C	-40		+125

Warranty
Orion Robotics warranties its products against defects in material and workmanship for a period of 90 days. If a defect is discovered, Orion Robotics will, at our discretion, repair, replace, or refund the purchase price of the product in question. Contact us at support@orionrobotics.com. No returns will be accepted without the proper authorization.

Copyrights and Trademarks
Copyright© 2013 by Orion Robotics, Inc. All rights reserved. RoboClaw and USB RoboClaw are registered trademarks of Basic Micro Inc. Other trademarks mentioned are registered trademarks of their respective holders.

Disclaimer
Orion Robotics cannot be held responsible for any incidental, or consequential damages resulting from use of products manufactured or sold by Orion Robotics or its distributors. No products from Orion Robotics should be used in any medical devices and/or medical situations. No product should be used in a life support situation.

Contacts

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

Discussion List
A web based discussion board is maintained at http://forums.orionrobotics.com**.**

Technical Support
Technical support is made available by sending an email to support@orionrobotics.com. All email will be answered within 48 hours. All general syntax and programming questions, unless deemed to be a software issue, will be referred to the on-line discussion forums.