TITLE:

Autonomous car

CANDIDATE(S):

Sivert Løken (10024)
Ole-Martin Steinnes (10019)
Eirik G. Gustafsson (10009)
Vebjørn Bjørlo-Larsen (10017)

DATE:	COURSE CODE:	COURSE TITLE:	RESTRICTION:
25.11.18	IE303812	Real Time Programming (2018 HØST)	None

STUDY PROGRAM:	PAGES/APPENDIX:	LIBRARY NO.:
Engineering - Automation	50/4	None

SUPERVISOR(S):

Blindheim, Ivar
Rekdalsbakken, Webjørn

SUMMARY:

The objective of this project is to examine the possibility of creating an autonomous vehicle capable of mapping and localization using a 250$ Lidar. The result is vehicle which is remotely operated and provides a video stream for visual feedback. While driving it also is capable of drawing a map of the surrounding objects relative to its current location. A bit array representing the map is sent to a graphical user interface, where it gets assembled to an image and displayed. Making the vehicle operate autonomously has not been completed. Some research has been done and a navigational algorithm is available in code.

NTNU

Real-time computing

Autonomous car

Sivert Løken

Ole-Martin Steinnes

Eirik G. Gustafsson

Vebjørn Bjørlo-Larsen

Abstract

The objective of this project is to examine the possibility of creating an autonomous vehicle capable of mapping and localization using a 250$ Lidar. The result is vehicle which is remotely operated and provides a video stream for visual feedback. While driving it also is capable of drawing a map of the surrounding objects relative to its current location. A bit array representing the map is sent to a graphical user interface, where it gets assembled to an image and displayed. Making the vehicle operate autonomously has not been completed. Some research has been done and a navigational algorithm is available in code.

Norwegian University of Science and Technology
Supervisors: Ivar Blindheim and Webjørn Rekdalsbakken

Contents

List of Figures

List of Terms

A* Pronounced a star, is a search algorithm to find a path from a point a to a point b, while avoiding obstacles. 1

Active sensor A sensor that emit energy into the environment, then measure the environmental reaction. 1

API (Application programming interface) is a interface which lets other software use functions from some software the API communicates with. 1

D*/ D*Lite Pronounced d star, is a search algorithm to find a path from a point a to a point b, while avoiding obstacles. Saves the path and if it is applied to the same area, only updates the parts that have changed. 1

Exteroceptive A sensor that acquire information from the robots's environment, for example, distance measurments, light intensity, and sound amplitude. 1

GUI Graphical user interface(GUI) is a interface for a user to interact with a program or robot. Information from the robot can be displayed and input from the user can be sent to the program or robot. 1

IMU Inertial measurement unit measures the acceleration in three directions using an accelerometer. Jaw, pith and tilt is also measured using a gyroscope[2]. 1

LIDAR Light detection and ranging is a surveying method that measures distance to a target by illuminating the target with pulsed laser light and measuring the reflected pulses with a sensor. 1

Passive sensor Sensor that measure ambient environmental energy entering the sensor. 1

Proprioceptive Sensor that measures values internal to the system, for example, whell load, robot arm joint angles, and battery voltage. 1

SLAM Simultaneous localization and mapping aims to recover both the robot path and the environment map using only the data gathered by its proprioceptive and exteroceptive sensors. 1

TCP TCP(Transmission Control Protocol) is a protocol for sending data over a network, the protocol ensures that the data has not been corrupted during transport and if it has it wil be resent. 1

UDP UDP(User Datagram Protocol) is a protocol for sending packets. UDP does not resend data or ensure that the data has arrived. One can compare UDP to a person shouting in a crowded room and a person tries to listen, the person listening may not get all the info but the person shouting will continue to shout. This property makes UDP perfect for example video streaming. 1

A Introduction

Autonomous systems are becoming increasingly popular. If done properly such systems can make both work and life easier for humans. However, the technology is still in development. Several start-ups and established companies race to find a solution to the self driving car. One of the most challenging parts of making a car autonomous is to keep the price low. Good perception comes at a cost, and most autonomous test cars have had a budget of 100 000$ - 200 000$ in sensors. These prices are on the way down, and Bloomberg reports that by 2020 there will be more than 10 000 autonomous cars equipped with lidar sensors[13]. However, do autonomous systems that don't need such precision have to come at a high price? In this project report it will be documented how a small autonomous car would work with a 250$ lidar.

This project is done in the course Real-time Programming at NTNU Ålesund. The topic chosen is relevant to real-world problems. However, the main focus of the project should still be the real-time aspects. Sections where real-time programming is explained, shown, or discussed might disappear under other categories. The main concepts of concurrent programming is explained in sections B.9 and B.10. Furthermore, the real-time aspects can be found in sections C.4, C.5, D, and E.3.

B Theory

B.1 Locomotion

Locomotion is a term that relates to how a robot moves from one location to another. There are several factors to take into account when considering the vehicles locomotion. The primary ones being the environment it will be expected to operate in, how much maneuverability the vehicle needs, and how controllable the vehicle needs to be. For wheeled robots there are generally two designs prevalent. The first one being the conventional Ackerman design which use a linkage between the front wheels to turn and avoid slipping when following a curve. The second is differential drive, which in short turns by varying the relative rate of the wheels on either side of the vehicle.
There is generally an inverse correlation between controllability and maneuverability. The Ackerman approach offers greater maneuverability while differential drive offers greater controllability. For low speed operation differential drive is often preferable.

B.2 Perception

Perception is one of the most important task of any autonomous system. This is done by taking measurements using sensors and then extracting data to acquire knowledge. Since an autonomous car moves around and encounters unforeseen situations often it is reliant on it's perception to perceive these and act accordingly. Sensors are classified as active/passive, and proprioceptive/exteroceptive.

B.3 Localization

Before the system can determine where to go, it needs to know where it's located. There have been several approaches of localization aimed at autonomous systems that have been tested the last years. For a robot to determine it's position over time it needs both exteroceptive and proprioceptive sensor. Otherwise, the position error would get bigger over time.

A GPS-sensor would not be precise enough to use for an indoor robot, given the error often can result in several meters off target. Odometric sensors have several issues when used to determine position. Wheel encoders are

integrated to compute position, over time this will yield a positional error. There are several external sources of error on wheel encoders. For example the wheels slipping would result in inaccurate measurements.

B.4 SLAM

SLAM(simultaneous localization and mapping) is the problem of updating a map with different sensor data while keeping track of the objects position. Object meaning the robot or car who is changing position, and is the object all sensors are located on. Statistical techniques is used to provide an estimate of the objects position and the parameters for the map. SLAM algorithms normally have two main approaches to this goal, deterministic and probabilistic. While the deterministic approach assumes that the sensor data is accurate (not accounting for error), and needs large data-sets of sensor-data with near perfect measurements. The probabilistic approach uses a particle filter to estimate the position, trying to find the best result taking into account that the sensor measurements are inaccurate.

B.5 A*

A*(pronounced A) star is based on the Dijkstra algorithm, which finds the shortest path between a start or source node and all other connected nodes by iteration through the nodes and using path costs to determine the shortest paths. A* uses an heuristic function that estimates the best option in each iteration, best-first search where both path cost and distance to goal is considered in deciding the next node to expand to in each iteration. This can drastically decrease the time it takes to find the shortest path.

B.6 D*/ D* lite

D*(pronounced D star) uses the same principles, but what sets it apart from A* is that it stores known information about the area after the initial shortest path has been found. When new information, such as new obstacles, has been found it only needs to update that specific section of the map, and if this interferes with the current path can more quickly plan a new path around the obstacle. D* lite is not based on the original D* but has the same behavior. This version is easier to understand, use and takes fewer lines of code to implement.

B.7 The observer pattern

Observer pattern works by defining subjects and observers, the subjects has a list of observers and notifies them when a property has changed in the subject[9].

B.8 Lidar

Lidar (Light Radar) is an optical distance measuring technique that is used to quickly measure a physical objects position. By measuring the difference in time or change in wave phase between the emitted laser signal and the reflected light, it can determine the distance to object along with some other properties. It can be used on a wide variety of materials, including metals, rock, chemical compounds, aerosol skies and even single molecules.

B.9 Concurrency

Concurrent programming is something every non-trivial software needs. The ability to run several tasks simultaneously in parallel makes any application able to handle more tasks, making the software more powerful. A word-processor would not be able to format text, while listening for keyboard-input without concurrent programming. The Java API offers several functionality for concurrent programming in it's API.

B.10 Thread-safety

When designing a concurrent software one must always ensure that it's thread-safe. This means no "collisions" between threads using the same variable, or the same method. For this, the Java API provides concurrent concepts such as "synchronized"-methods, "Atomic"-variables, "volatile"-variables, and much more. If an application is not thread-safe, it might end up with corrupt variables which will result in the software misbehaving.

C Methods

C.1 Components

- ODROID-XU4
- Koenig USB-dongle
- ODROID Wi-fi antenna
- Microsoft LifeCam
- Scanse sweep LIDAR
- RoboClaw 2X30A Motor Controller (V5E)
- MAXPOWER LIPOBATTERI 11,1V - 35C - 3S - 6000MAH
- Turnigy 5A (8-26v) SBEC for Lipo
- Metal Gearmotor 37Dx73L

C.2 SLAM

In order to create an autonomous car, it would need to see objects and structures around itself (perception), a way to determine the position relative to it's surroundings (localization), and a way to determine the best route from A to B (navigation). The SLAM-algorithm used in this project is based on CoreSLAM, an algorithm created in less than 200 lines of C code, developed in 2010 [15]. The goal of CoreSLAM was to make an algorithm which was easy to understand and integrate into a particle filter framework. SLAM can be done in several programming languages. However, since languages such as Python and C seems to be preferred in the SLAM-community there were few viable options in Java. BreezySLAM is a C library which has been translated through native bindings to Java, Python, MATLAB, and C++ [12]. BreezySLAM fit all the demands of the project. The algorithm is not to demanding for the Odroid, and it is possible to use in Java.

In order to use the aforementioned library one would need a laser sensor to measure distance to a set of points, over some angle. This project utilizes a Scanse Sweep LiDAR that has up to 1075 measurements per second [11].

The LiDAR's Java library is implemented in the class "Lidar", used in the "LidarThread". Explanations of how the LiDAR thread works and it's integration with SLAM-objects can be found in section C.5.2. The SLAM-library uses several objects to represent the algorithm, the laser sensor, positional change, and the car itself. How these are initiated and used is also written about in section C.5.2

The SLAM-library enables two main methods of SLAM. The first one is DeterministicSLAM which is based on an approach that ignores uncertainties thinking of it's sensor data as exact, and demands a laser sensor that provides large amounts of exact measurements[7]. This approach, while theoretically sound, have problems when applied to real environments since it does not take into account error over time[16]. While the Deterministic method was tested as a possible solution in this project. However, it did not provide satisfying results.

Random Mutation Hill-Climbing is the other method that the SLAM-library provides. This is a probabilistic algorithm which contains a Monte-Carlo particle filter that iterates a search on the cars starting position. This is a mathematical optimization that starts with an arbitrary solution to the answer then tries to find a better one by further increment the result. For each new sensor sample it re-samples it's particles and converges on a new position when the algorithm is done iterating[4]. As can be found in the "RMHCSLAM" class this project use 5000 iterations as opposed to the default 1000. This results in a bigger chance of finding the actual position. In practise this means that the algorithm uses sensor input (sensor fusion) to probabilistic determine it's new position. Inputs in this project are odometry and laser-scans.

It is important to note that the odometry provided to the algorithm is not necessary for the algorithm to detect movement on it's internal coordinate system. Since the Random Mutation Hill-Climbing algorithm predicts it's position based on the LiDAR-scans it will detect movement without any odometry input. However, theoretically to get the best possible result from such a localization algorithm one would need to utilize sensor fusion. Hence, the addition of odometry data. The change of position, angle and distance, is calculated based on encoder-data, wheel radius, and half axle-length as shown below in listing 1, 2, and 3.

```java
/**
     * Sets up objects used in our SLAM application
     *
     * @param sweepDevice     Sweep LiDAR of the car
     * @param slamServer      Server for transmitting SLAM-map
     * @param lidarSpeed      Motor speed of LiDAR
     */
    public void initSlam(SweepDevice sweepDevice,
                         SlamServer slamServer
                         , int lidarSpeed) {

        ...

        // Wheel radius = 30
        // Half axle length = 110
        robot = new Robot(30, 110);
    }
```

Code Listing 1: Creation of a Robot-object

```java
/**
 * Robot-class that represents a WheeledRobot from the Breezyslam-library
 * Specifies wheel radius and half axle length in mm, calculating odometry.
 * This allows the library to calculate the change of position of the robot
 * based on encoder data from motor controller.
 */
public class Robot extends WheeledRobot{

    /**
     * Constructor of the Robot-class. Take in metrics that are
     * needed for Robot-odometry.
     *
     * @param wheel_radius_mm       Radius of wheel in millimeters.
     * @param half_axle_length_mm   Half axle length of in millimeters.
     */
    public Robot(double wheel_radius_mm, double half_axle_length_mm){
        super(wheel_radius_mm,half_axle_length_mm);
    }

    @Override
    protected WheelOdometry extractOdometry(double timestamp, double
        left_wheel_odometry, double right_wheel_odometry) {
        left_wheel_odometry = (-1)*left_wheel_odometry;

        // Return odometry to library.
        return new WheelOdometry(timestamp,left_wheel_odometry,
            right_wheel_odometry);
    }
}
```

Code Listing 2: WheeledRobot-class

```java
    public PoseChange computePoseChange( double timestamp , double
        left_wheel_odometry , double right_wheel_odometry){
```

```
        WheelOdometry odometry = this.extractOdometry(timestamp ,
            left_wheel_odometry , right_wheel_odometry);
        double dxy_mm = 0;
        double dtheta_degrees = 0;
        double dt_seconds = 0;

        if (this.timestamp_seconds_prev > 0){
            double left_diff_degrees, right_diff_degrees;
            left_diff_degrees = (odometry.left_wheel_encoder -
                left_wheel_encoder_prev)*(360.0/8400.0);
            right_diff_degrees = (odometry.right_wheel_encoder -
                right_wheel_encoder_prev)*(360.0/8400.0);

            // Calculating change in time.
            dxy_mm =  this.wheel_radius_mm * (Math.toRadians(
                left_diff_degrees) + Math.toRadians(right_diff_degrees));
            dtheta_degrees = this.wheel_radius_mm / this.half_axle_length_mm
                 * (right_diff_degrees - left_diff_degrees) * 0.6;
            dt_seconds = odometry.timestamp_seconds - this.
                timestamp_seconds_prev;
        }

        // Store current odometry for next time
        this.timestamp_seconds_prev = odometry.timestamp_seconds;
        this.left_wheel_encoder_prev = odometry.left_wheel_encoder;
        this.right_wheel_encoder_prev = odometry.right_wheel_encoder;

        return new PoseChange(dxy_mm, dtheta_degrees, dt_seconds);
    }
```

Code Listing 3: Position change object

While the actual wheel radius is 6 centimeters ($r = 6cm$) and half of the axle length is 18 centimeters ($l = 18cm$), these values had to be altered in the code where position change was calculated. One of the reasons for this is that changing the axle length slightly could possibly compensate for the wheels slipping when turning. The math used for calculate the angle expects there to be no loss of friction (no slipping), but it still needs to account for the actual loss of friction. Decreasing the length of the axle will calculate a smaller angle despite the encoder-values indicate a bigger turn, and thus the error induced by slippage might cancel out. However, it is extremely unlikely that one can completely cancel out this error by testing. Hence, the best way to get rid of this issue would be to change to a wheel-setup that don't have such limitations. Though, for this project that was deemed unnecessary.

C.3 Motor Control

A Roboclaw 2x30A handles the motor control.The only native library [5] available to the motor controller is written in Python. The library itself provides function for controlling individual motors, taking speed and address as inputs. There are separate functions for driving forward and backward. It also contains functions for reading the encoders, making it very suitable to use.

Documentation and readability of the code in the library is poor to non-existent, making it a time consuming task to translate into java code. There are libraries available for embedding python in java and also for calling the JVM from within python, but none suited the requirements to be used with the motor controller.

The option requiring the least amount of work while providing a suitable gateway between python and java is setting up a TCP server written python which implements the native python library, and a TCP client on the java side which connects to the server. The server is single threaded and must be started up a long side the java project.

C.3.1 Python TCP Server

The python server a tcp server which creates a new thread for each connetion. It implements the Roboclaw library and a socket library. It sets up a socket on host = 'localhost' and port = 2004, and it also initializes the connection to the roboclaw motor controller through an Usb-serial port.
There is a subset of functions defined within the server, which goes as follows

```
def getMotorSpeed():
  speed1 = rc.ReadSpeedM1(address)
  speed2 = rc.ReadSpeedM2(address)
  global speed1Payload
  global speed2Payload
  global speed1PayloadPrev
  global speed2PayloadPrev
  if(speed1[0]):
    speed1Payload = speed1[1]
    speed1PayloadPrev = speed1Payload
  else:
```

```
    speed1Payload = speed1PayloadPrev
  if(speed2[0]):
    speed2Payload = speed2[1]
    speed2PayloadPrev = speed2Payload
  else:
    speed2Payload = speed2PayloadPrev

def setM1ForwardSpeed(M1):
  rc.ForwardM1(address,M1)

def setM2ForwardSpeed(M2):
  rc.ForwardM2(address,M2)

def setM1BackwardSpeed(M1):
  rc.BackwardM1(address,M1)

def setM2BackwardSpeed(M2):
  rc.BackwardM2(address,M2)

def Stop():
  rc.BackwardM2(address,0)
  rc.BackwardM1(address,0)
  rc.ForwardM2(address,0)
  rc.ForwardM1(address,0)

def encoderposition():
  enc1 = rc.ReadEncM1(address)
  enc2 = rc.ReadEncM2(address)
  global enc1PayloadPrev
  global enc2PayloadPrev
  global enc1Payload
  global enc2Payload
  if(enc1[0]==1):
    enc1Payload = enc1[1]
    enc1PayloadPrev = enc1Payload
  else:
    enc1Payload = enc1PayloadPrev
  if(enc2[0]==1):
    enc2Payload = enc2[1]
    enc2PayloadPrev = enc2Payload
  else:
    enc2Payload = enc2PayloadPrev
```

There are separate functions for controlling each individual motor which can be seen above. These are all combined to in a single stop function where the

speed is set to 0 to reduce the amount of traffic over the client/server connection needed to stop the motors. The encoder function returns the current set of encoder values, and if it fails to read the encoders, which can happen for various reason, the previous value will get passed as the current value.

When the Python server receives a connection it will begin listening for data. In order to determine which function should be run, a selected set of keywords has been chosen as conditions. Below is an example.

```
data = connection.recv(1024).decode()
if not data:
  break;
if(str(data).lower().strip() == 'getencoderdata'):
    encoderposition()
  payloadUnencoded = "enc1:" + str(enc1Payload[0])+':' + "enc2:"
                                + str(enc2Payload[0])
  print payloadUnencoded
  payload =  str(payloadUnencoded) + '\n'.encode('UTF-8')
  connection.send(payload)
```

If the client wants to retrieve the encoder data, it simply needs to send a string containing only the keyword ”getencoderdata”. Note that it is not case sensitive. The server will then be prompted to retrieve the encoder data from the motor controller. It will then encode the string with a newline sign to make sure the receiving client will parse it correctly, and then send it. This is how all the getter functions are handled on the server side.

When the client needs to set values for the motors, for example forward values or reverse, a string containing keyword followed by an integer will be sent. The server will parse the keyword and enter correct functions where it will use an regular expression pattern to extract the integer from the string. The integer will then be passed as input in corresponding motor function. See example below.

```
    if("setforwardspeedmotorone" in str(data).lower().strip()):
      matches = re.findall('\d+', data)
    m1 = int(matches[0])
    setM1ForwardSpeed(m1)
```

Setting the forward speed of motor one would require a string in the following structure ”setfowardspeedmotorone:32”. This functions in the same way for both reverse, forward in motor two and motor one. Note that the motors the

motors on each side are wired in parallel, thus motor one is the same as both the motors on the left side and motor two is the same as both the motors on the right side.

C.4 GUI

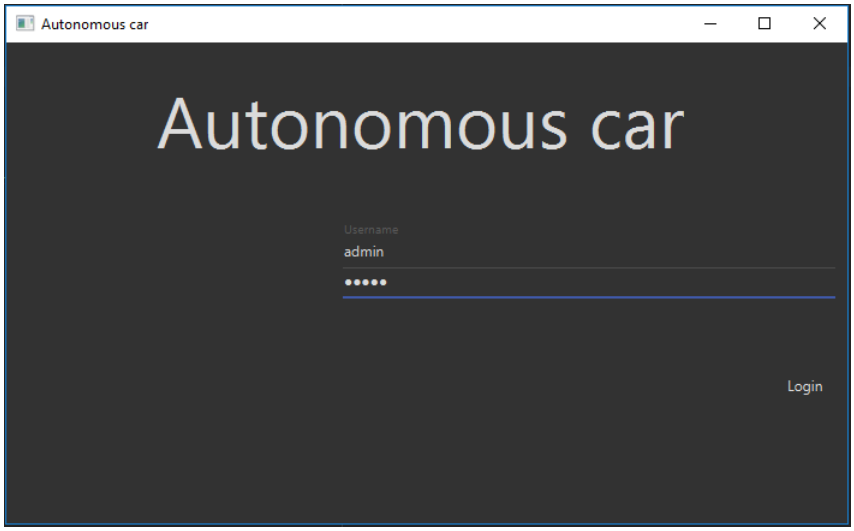


Figure 1: GUI login

Autonomous car GUI starts with a login-screen(figure 1) to protect from unauthorized use of the car's controll system.

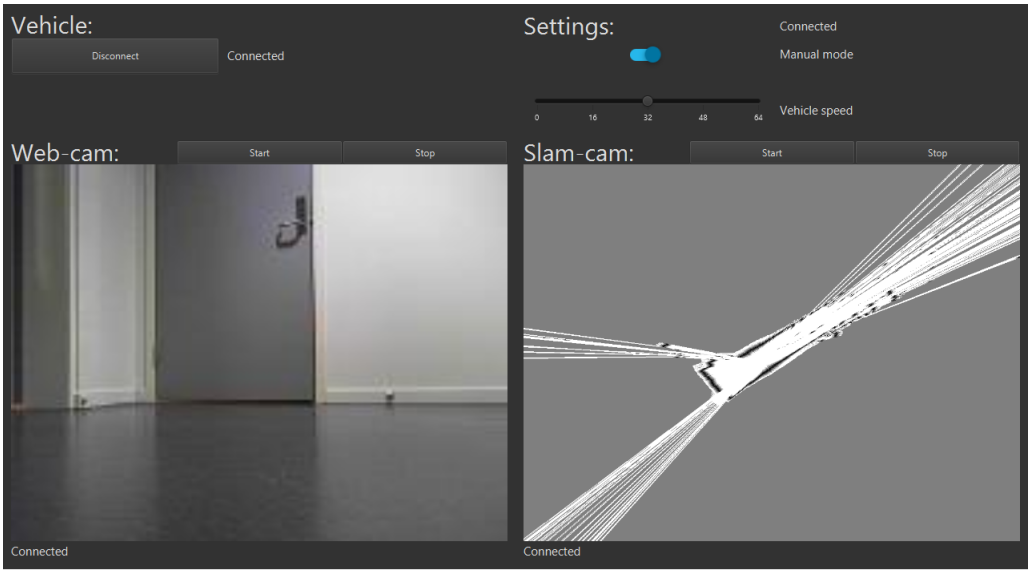


Figure 2: Control system screen

The control system screen contains a connect button which must be pressed before other buttons unlock, this is done so that the GUI doesn't try send commands to a offline server. Next there is a camera feed from the car, for navigating help when manually driving. One can also toggle manual mode on and off, when manual mode is activated the car is controllable with the keys w(forwards), a(left), s(backwards) and d(right). Cars speed can also be set from the GUI this is done by sliding the slider marked "Vehicle speed" and releasing the slider on the desired speed. When driving only one command can be sent at a time, this helps keep the server code simple. Image feed from SLAM library is also displayed, where the white area is places the car can drive and black edges and dots are areas which are visually blocked.

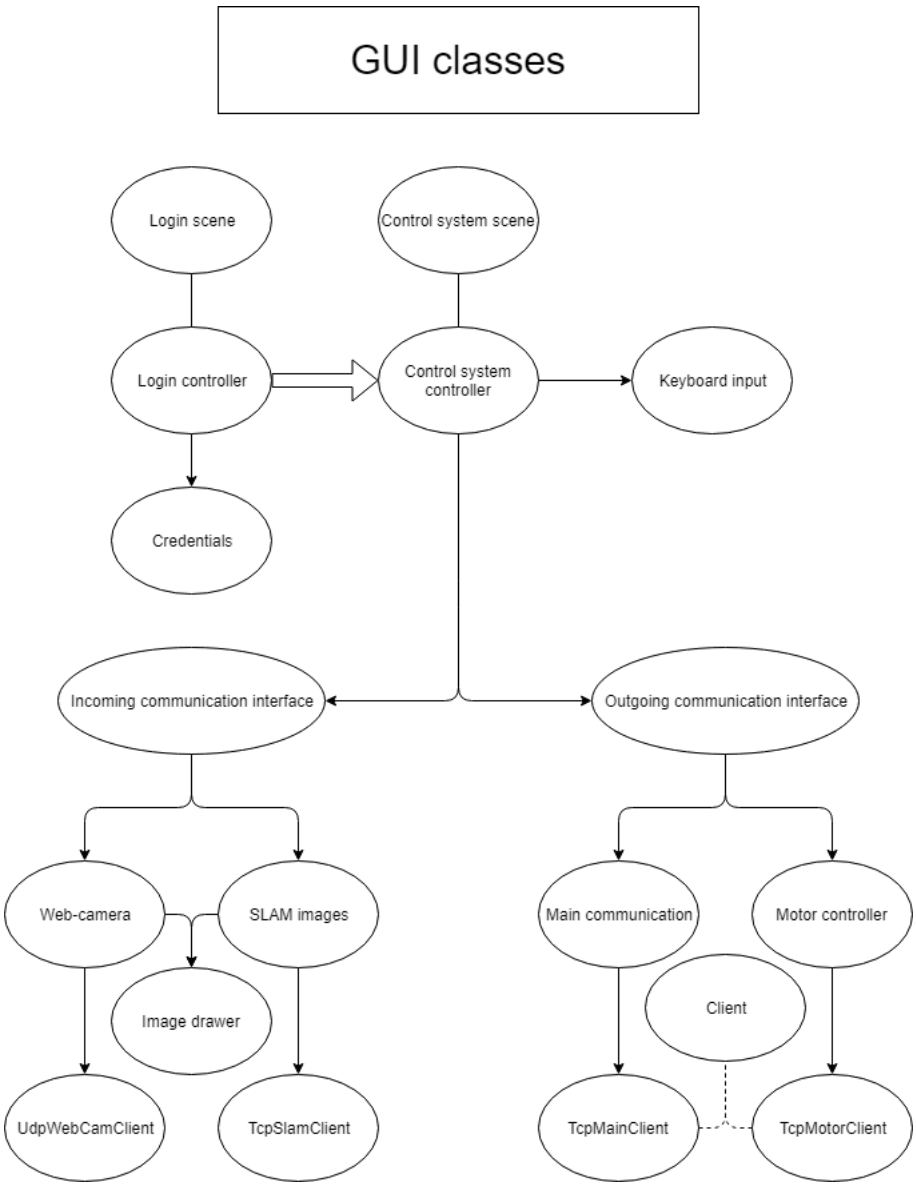


Figure 3: Class diagram over the GUI

In the GUI threads are mainly utilized for sockets to try and connect for a given timeout, and not locking up the GUI. WebCamera and SlamCam classes for the same reason updating GUI happens in a different thread using "Platform.runLater();" which starts a new thread for the code to run

on. In the code snippet showed below, "Platform.runLater();" is used in combination with a lambda expression to update the GUI.

```
1 Platform.runLater(() -> {
2                         // Update GUI buttons and labels.
3                     });
```

Code Listing 4: How GUI elements are updated.

In the GUI there were several threads, WebCam, SlamMap, Main, and MotorControl. Both the WebCam- and SlamMap-thread accepted a stream (former UDP, latter TCP) to receive incoming data to be processed and visualized on the GUI. The MotorControl-thread established a connection with the main software (Odroid) and listened for keyboard-inputs on the computer. Upon input, it could send commands to the motor in order to control it. Furthermore, the main thread established a connection with the main software which let it start some or all modules in the main software (SLAM, WebCam, Motor).

C.5 Main program

The main program referenced other parts in the report is the java-software running on the Odroid. This software contains modules for SLAM, motor-control, and streaming (WebCam and Map). It also talks with two other software-components, the Python-server on the Odroid and the graphical user-interface on a laptop. Components of the software are explained in more detail in the sections below. Furthermore, figure 4 provides a diagram that showcases the most important features of the main program. However, it is important to note that there are several object excluded from this diagram and can not be considered a full representation of the class-structure, but rather one of the threads and their communication.

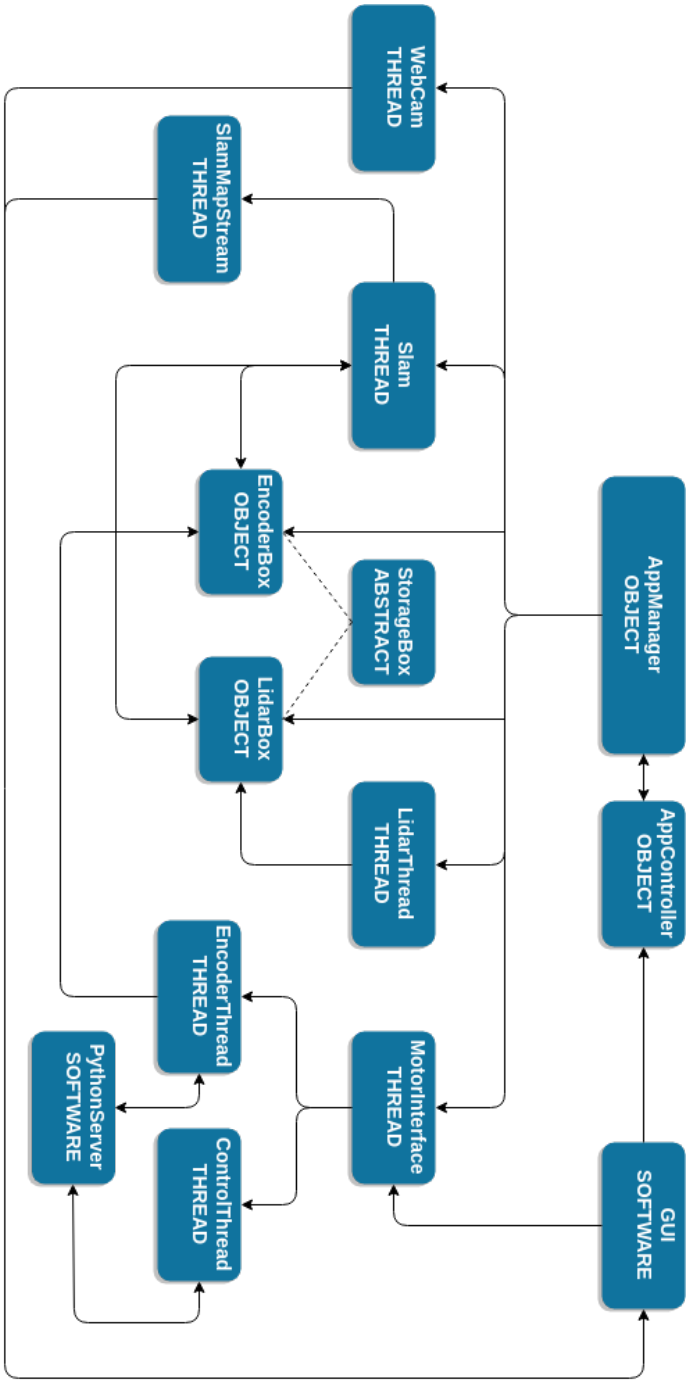


Figure 4: Diagram of most important parts of main program

C.5.1 AppManager

AppManager is the class that makes it possible for the GUI to control each aspect of the software running on the Odroid. It creates a AppController which serves as an interface with the GUI, setting up a socket for the GUI to connect to and methods for the AppManager to recieve messages sent over that socket. The AppManager has a simple communication protocol with the GUI shown in the table below. While it starts and stops tasks in the software it also makes sure that a task only can be started when it is not currently running.

Message	Meaning
MOTORCONTROLLER:START	Start the motor controller task
SLAM:START	Start the slam task
WEBCAM:START	Start the webcam task
MOTORCONTROLLER:STOP	Stop the motor controller task
SLAM:STOP	Stop the slam task
WEBCAM:STOP	Stop the webcam task
STOP	Stop all active tasks

C.5.2 SLAM

Slam is one of the main tasks on the Odroid's software. The task contains a LidarThread and Slam-thread that work together to run the SLAM-algorithm. The LidarThread consists of a Lidar-object and a LidarBox. Specifically the LidarBox extends a StorageBox. Thus, when the LidarThread recieves the scans from the sensor it iterates through each sample and converts them to the correct form for the SLAM-algorithm, as shown in code listing 5.

```
1     @Override
2     public void run() {
3         boolean firstScan = true;
4         while (true) {
5             if (!firstScan) {
6                 // Loops through samples of each scan
7                 List<SweepSample> s = sweepDevice.nextScan();
8                 // Int-array of distances in scan.
9                 int[] distanceA = new int[s.size()];
10                // Scan vector
```

```
                Vector<int[]> scans = new Vector<int[]>();
                // For each sample, get distance.
                for (int i = 0; i <= s.size() - 1; i++) {
                    int dist = s.get(i).getDistance();
                    distanceA[i] = dist * 10;
                }
                // Add distance to scan vector
                scans.addElement(distanceA);
                ns = scans.size();
                // For each scan
                for (int x = 0; x < ns; x++) {
                    int[] scan = scans.elementAt(x);
                    lidarBox.setValue(scan);
                }
            } else {
                List<SweepSample> s = sweepDevice.nextScan();
                firstScan = false;
            }
        }
    }
```

Code Listing 5: Converting LiDAR-data to proper format.

When the sensor data is filtered it gets sent to the storage box which both threads have access to. The storage box is described later in section C.5.4. Furthermore, after being stored, the Slam-object can access the new array of sensor values. This creates a modular structure, which can easily be modified to use other LiDAR models.

The Slam-thread is an integral part of the application. It starts off by building several objects needed for running a SLAM-algorithm, these are mostly provided by the library BreezySLAM [12] with the exception of Robot which is created to calculate the odometry data. In code listing 6 the process of initializing the SLAM-process is shown.

```
/**
 * Sets up objects used in our SLAM application
 *
 * @param lidarBox      StorageBox for lidar values
 * @param encoderBox    StorageBox for encoder values
 */
public void initSlam(EncoderBox encoderBox, LidarBox lidarBox) {

    executorService = Executors.newFixedThreadPool(1);

    // Set fields
    this.slamActive = true;
    this.sampleLimit = 1000;

    // Created StorageBox-instances.
```

```
16      this.encoderBox = encoderBox;
17      this.lidarBox = lidarBox;
18
19      // Start Map-stream thread.
20      this.slamMapStream = new SlamMapStream(8002, MAP_SIZE_PIXELS);
21      this.slamMapStream.start();
22
23      // New objects for running SLAM.
24      poseChange = new PoseChange();
25      myLidar = new Laser(1000, 1000,
26              360, 4000,
27              0, 0.1);
28      slam = new RMHCSLAM(myLidar, 820, 30, 1337);
29      robot = new Robot(30, 110);
30  }
```

Code Listing 6: Creating SLAM objects

It contains one thread which is administered by a ExecutorService [14] with a threadpool of one. For each scan, it generate a new internal map. When this map is generated, the ExecutorService executes a SlamMapStream thread which uploads the latest map to the GUI. Since the threadpool is one, there will never be more than one thread uploading to the GUI at the time. When the thread has uploaded it's map to the GUI, the executor service will properly dispose of the thread. In code listing 6 you can see how the executor service is created, furthermore in code listing 7 it is shown how the executor service is used when uploading the map. The output from the executor service tells a clear story of how the executor service works when used in this instance, shown in figure 5.

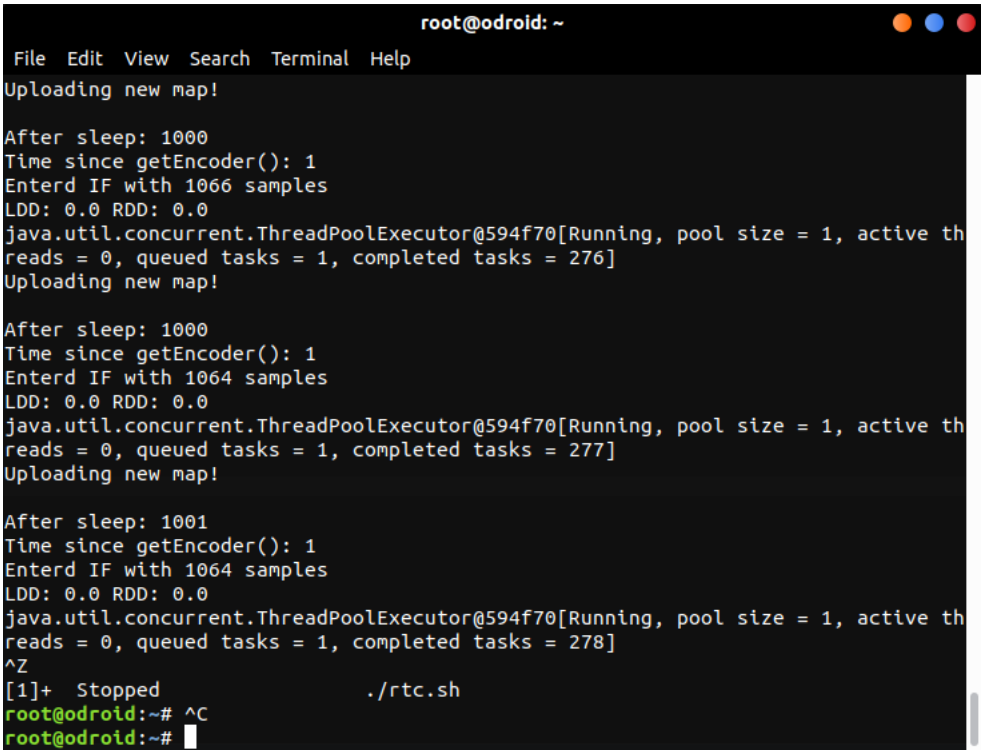


Figure 5: Debug output, showing threadpool.

```
/**
 * Updates the SLAM-algorithm based on lidar scans.
 */
public void run() {

    System.out.println("SLAM activated!");

    do {
        if (lidarBox.isReady()) {

            // Get last lidar-scan (one revolution)
            int[] scan = lidarBox.getValue();


            if (encoderBox.active()) {

                // Encoder one and two from motor controller.
                int enc1, enc2;

                // String array that holds encoder values.
                String[] strings = encoderBox.getValue();

                // If strings do not contain "no response" we know strings
```

```
                    contain
24                  // proper encoder values. Hence, we assign them to
                        PoseChange-object.
25                  if (!strings[1].equalsIgnoreCase("no response")) {
26
27                      // encoder values
28                      String encoder1 = strings[1];
29                      String encoder2 = strings[3];
30
31                      // Parsing encoder values from String to int.
32                      enc1 = Integer.parseInt(encoder1);
33                      enc2 = Integer.parseInt(encoder2);
34
35                      // Computing PoseChange through abstract Robot-class.
36                      poseChange = robot.computePoseChange(((System.
                            currentTimeMillis())) / 1000.0, enc1, enc2);
37                  }
38              }
39              // Update slam with new scan and position change
40              slam.update(scan, poseChange);
41              slam.getmap(mapbytes);
42              slamMapStream.setByteMap(mapbytes);
43              executorService.execute(slamMapStream);
44          }
45      } while (slamActive);
46  }
```

Code Listing 7: Executing the thread.

There are a couple of important points in the "run()"-method shown in listing 7. First of all, for the method to do anything it needs a new set of scans from the LidarBox. When the lidar thread updates the box with new scans it sets a boolean flag "isReady" to true, which indicates to Slam that it can fetch new data. If a motorcontroller is active the EncoderBox will return true when Slam asks whether it is active or not. If it is active, Slam will fetch the encoder data and format it properly for the SLAM-algorithm. When this is done, the new fields will be applied to the algorithm, generating a new position and map. Slam fetches the new map and starts the thread that uploads said map. By default, when Slam is told to stop it will write a map to the local drive.

C.5.3 Motorinterface

Motor Interface is a class which implements two separate threads, one for reading encoder data, and one for sending commands to the motor controller. Since receiving encoder data is paramount to drawing a good map of the surroundings, while also keeping track of the robots position, the encoder thread has higher priority.

The encoder is a runnable thread that sends encoder request to the python server and stores returned values in a storage box. It uses a function from the motor command class to send the following string "getencoderdata" to the python server, and employs a function from the motor client class to listen and parse the response. A sample result would be a string array in the following form "enc1, 1020, enc2, 2020". To avoid network congestion to the python server, a one second delay has been implemented between encoder requests.

The control thread is used to interpret input from the GUI and translate them to motor commands which get sent to the python server. If the user wants to turn the car left, the "a" key on the keyboard running the GUI client must be pressed. That will send a request in the following form "key_pressed:a", which the control thread will interpret as two entries in a string array. It will then string compare and thus find according function, ultimately issue a turnLeft(): function call which will in short send a turn request to the python server.

C.5.4 StorageBox

To share data between threads in manner which is thread safe and efficient, a storage box solution is implemented. This is utilized in two cases, one for sharing Lidar data and one for sharing Encoder data. The box class is made up of an variable that stores values, a flag, and a setter and getter. Below is a sample code from the Lidar Storage Box used in the project. The layout is very similar for the encoder box, and for all purposes and intents functions the same way.

```
1  public class LidarBox extends StorageBox{
2
3      // Holds scans from Lidar.
4      private volatile int[] scans;
5
6      // Holds isReady-flag
7      private volatile boolean isReady;
8
9      // Atomic bool to make box thread-safe
10     private AtomicBoolean atomicBoolean;
11
12     /**
13      * Creates empty scan-array when instance is created.
14      */
15     public LidarBox() {
```

```
16          scans = new int[]{};
17          atomicBoolean = new AtomicBoolean(false);
18      }
19
20      // For thread safety upon expansion
21      private static final AtomicReferenceFieldUpdater<LidarBox,int[]> updater
            =
22              AtomicReferenceFieldUpdater.newUpdater(
23                      LidarBox.class, int[].class, "scans");
24
25      /**
26       * Sets scan values for the last revolution.
27       * @param currentScan current scan values by lidar (last whole
             revolution)
28       */
29      public void setValue(int[] currentScan) {
30          isReady = updater.compareAndSet(this, this.scans, currentScan);
31          atomicBoolean.set(isReady);
32      }
33
34      /**
35       * Returns scan values for last revolution
36       * @return scan values for last revolution
37       */
38      public int[] getValue() {
39          atomicBoolean.set(false);
40          return scans;
41      }
42
43      /**
44       * Returns true if scans is ready to be read. (prev values was changed)
45       * @return true if scans is ready to be read. (prev values was changed)
46       */
47      public boolean isReady() {
48          return atomicBoolean.get();
49      }
50
51  }
```

Code Listing 8: Java Code: Field Setter

Because the storage box is being used by multiple threads, methods for setting values needs to ensure that only thread can write at a time. To achieve this an atomic reference field updater has been utilized.
The Atomic reference field updater is a reflection-based utility that enables atomic updates to designated volatile reference fields of designated classes. While volatile provides memory visibility, so that all threads can access the same value, it will not provide execution control, like for example safeguard against concurrent writing to the same variable. To solve this problem, the compareAndSet method is used, which locks out any competing writes when used. The only limitation is that instead of locking out any competing writes, it only locks out competing writes of the same instance of the Atomic Refer-

ence Updater. Therefore the variable set method "setValue", uses the same instance of atomic reference field updater each time its called. [10]

The compareAndSet function is based on non-blocking compare-and swap algorithm, it returns true if it succeeds writing to a variable, else it will return false.

> *A typical CAS operation works on three operands:*
>
> - *The memory location on which to operate (M)*
> - *The existing expected value (A) of the variable*
> - *The new value (B) which needs to be set*
>
> *The CAS operation updates atomically the value in M to B, but only if the existing value in M matches A, otherwise no action is taken.* ([1] baeldung.com/java-atomic-variables)

If multiple threads tries to set the value simultaneously, only one is able to. This is because when the value get set by one thread, the memory location M will not equal the expected value A for the remaining threads. Thus the they will only receive a false Boolean as an indication that they did not manage to update the value. No threads are locked out, and they can all continue to do work. Since the base variable is volatile, they will be able to reference back to shared memory and update the expected value.

The lidar thread writes values to the Lidarbox while the slam thread reads values from the same box. Since the slam algorithm is only supposed to run once for each set of scan values, the Lidarbox contains a flag which indicates weather or not the int array has been updated. When the atomic reference updater manages to update the value, it will set the local volatile boolean isReady true, which in turn sets the atomic flag isReady() to true. Once another thread, in this case, the slam thread uses the getValue() function to request the int array, the atomic boolean is set false. This ensures that it will not be able to request data until the array has been updated. The use of an atomic boolean flag safeguards against corrupt data, since the atomic operator guarantees the process is not interrupted by the thread scheduler. It will also make sure the other threads sees the operation as instantaneously, and as such it will not have any intermediate state.

C.5.5 Webcam server

The web-camera server (on the main program) uses an API[8] to capture images. It takes a screen-shot from the camera using a webcam capture API, and sends it trough a UDP protocol socket to the GUI. At first, the server streamed a very high quality image with high frame-rate. Since sending one hundred high resolution images per second is a waste of processing power, a 10 millisecond delay was implemented to slow down the frame-rate. Thus, the server stream roughly 10 images per second. In addition the image resolution was reduced to limit the traffic on the network.

C.6 Vehicle

The frame of the vehicle is a rectangle comprised of four aluminum profiles. The width is approximately 60 cm, including the wheels, and length of approximately 30 cm. The motors are mounted in each corner of the frame using bolts and nuts slotted in the aluminum profile. A large thumper wheel is mounted on the axle of each motor using hexagonal brackets and screws. Because the motors are firmly mounted to the frame, the only method for turning the car is to drive the motors in opposite directions(differential drive). In order to improve the turn speeds and controllability the frame is made with a greater width than length, while it is not to wide to drive through door frames.

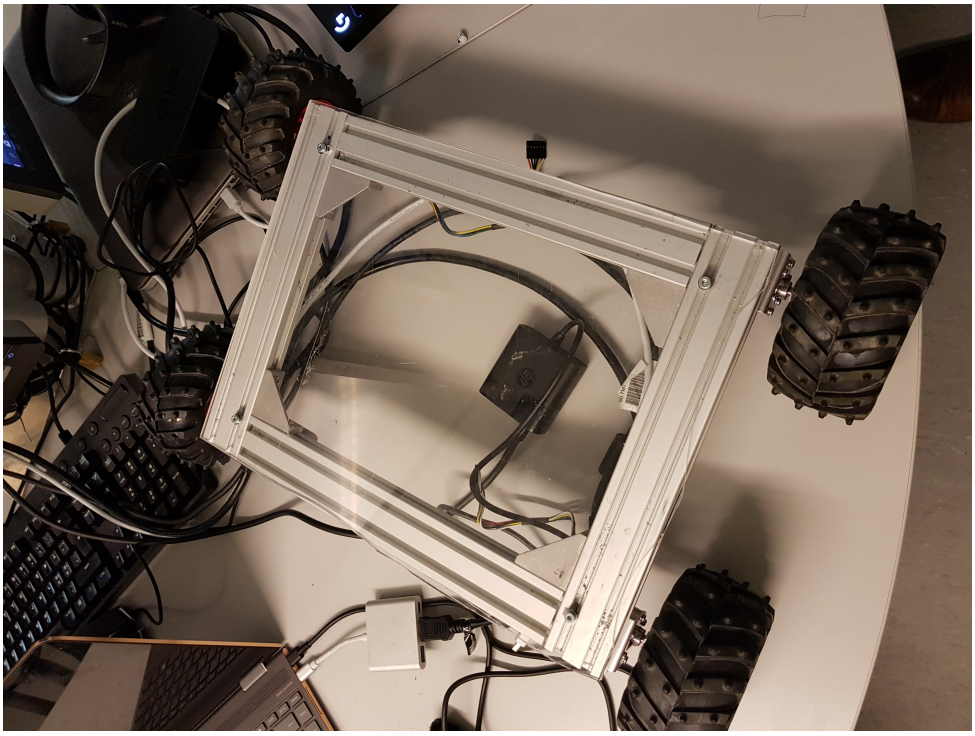


Figure 6: Frame with motors mounted

To create a base for mounting the components to the vehicle, a 0.5 cm thick piece of plexiglass is attached to the top of the aluminum frame. The battery pack is contained in a 3d-printed crib. A Lidar is mounted in the center atop another 3d-printed frame. In front and center the webcamera is mounted using a smaller 3-d printed cradle. The Roboclaw motor control is bolted in place using spacers, since the underside of the circuit board is not flat. Lastly, the Odroid and an usb-hub is also attached to the plexiglass base. All components on top of the car is mounted with double sided tape, except the aforementioned motor controller.

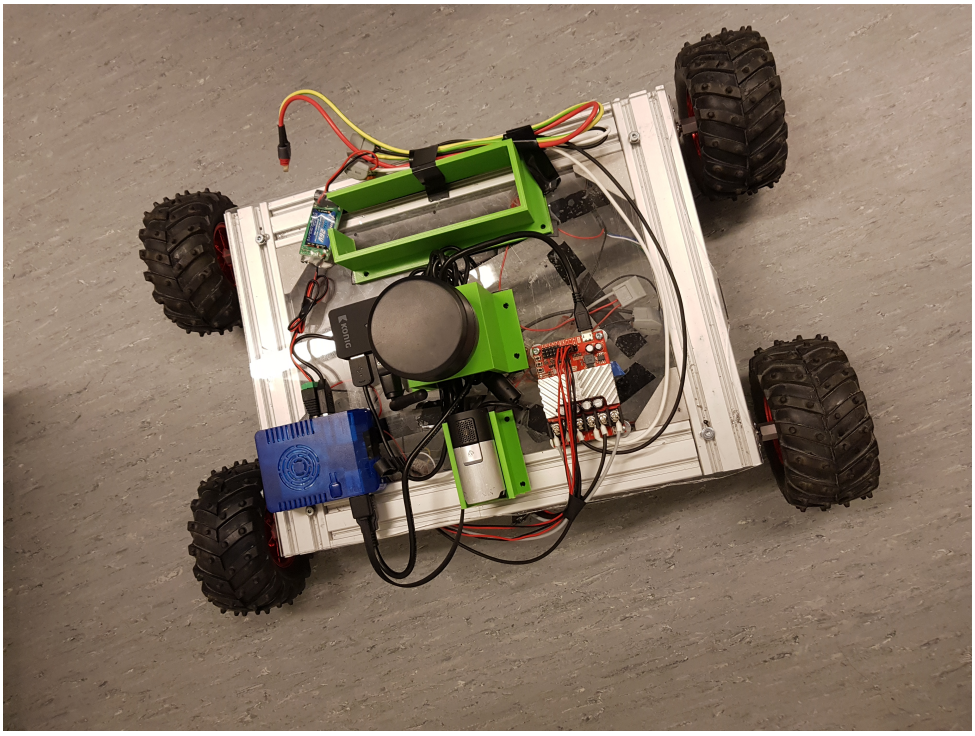


Figure 7: Finished vehicle with components

All wiring for the car is hidden as much as possible with all wires on the bottom car taped to the floor. Wires for lidar, web camera and motor controller is tucked away underneath the lidar for a clean look. Wires that go from the bottom to the top of the car is routed around the front since drilling holes in plexiglass is prone to cracking the glass.

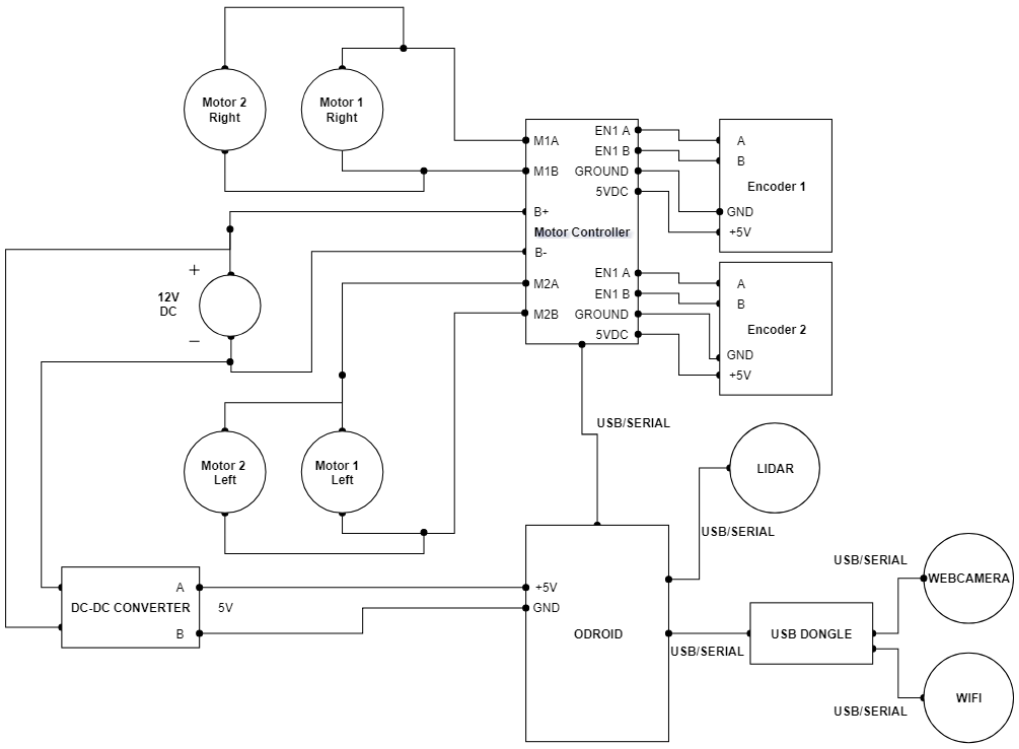


Figure 8: Wiring diagram

The power source of the vehicle is a Lithium-Polymer battery with a voltage of 11.1v and a capacity of 6000 mah. From the battery the wiring runs through a 3-point Wago connector which splits it between the Odroid and the motor controller. The Odroid has a max operating voltage of 5v, therefore a DC-DC converter is mounted between the Wago connector and the barrel plug connecting to the Odroid, ensuring it wont be overloaded. The motor controller has an operating voltage ranging from 6V to 34V, thus there is no risk of overloading. All the motors and encoders are connected to the motor controller, while the rest of the components draws power from the serial ports of the Odroid.

C.7 Navigation

Due to time constraints and difficulties with the map created by the SLAM program, the navigation part of the project ended up being a proof of concept, and not included in the final program.

Before the scan results from the LiDAR can be used it is processed to be compatible with the D*Lite algorithm. The scan image is first binarized to show open areas and walls/obstacles, then basic image morphology is used to limit noise. This was done in Matlab for testing purposes, but a solution using OpenCV was planned to be implemented. A result of the image processing is shown below, with colors inverted to better illustrate the path planning algorithm.

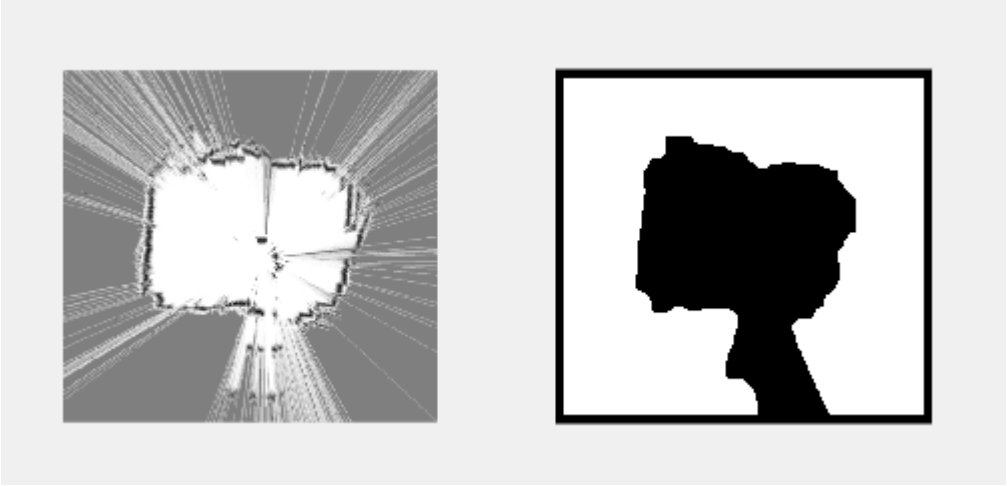


Figure 9: Left image is the original, and the right shows the processed version with inverted colors

The code below shows a basic implementation of the D*Lite algorithm[6]. First the pathfinder, start and goal nodes are initialized, all nodes are set to passable. Then the impassable nodes are found by iterating through the pixels in the processed image, which sets all the black pixels in the processed image to impassable. This is then used when the shortest path between the start and goal node is calculated.

```
1     // Get Scanned Map ...
2
3     // Process Image ...
4
5     // Create pathfinder
6     DStarLite pf = new DStarLite();
7     // Set start and goal nodes
8     pf.init(130, 340, 370, 300);
9
10    // Iterate through the processed image to find impassable nodes (
          obstacles/walls)
```

```
11      for (int y = 0; y < image.getHeight(); y++) {
12          for (int x = 0; x < image.getWidth(); x++) {
13              int gray = image.getRGB(x, y)& 0xFF; // Find grayscale value of
                    the image
14              if (gray != 255) {
15                  //set impassable nodes
16                  pf.updateCell(x,y,-1);
17              }
18          }
19      }
20
21      // Perform the pathfinding
22      pf.replan();
23
24      // Get the path
25      List<State> path = pf.getPath();
26
27      // Use Path ...
28
29      // To visualize the path it is printed in red on the test image.
30      int red=new Color(255,0,0).getRGB();
31      for (State i : path) {
32          image.setRGB(i.x,i.y,red);
33          // Pixels around the path pixels are also coloured for clarity
34          image.setRGB(i.x+1,i.y,red);
35          image.setRGB(i.x-1,i.y,red);
36          image.setRGB(i.x,i.y+1,red);
37          image.setRGB(i.x,i.y-1,red);
38      }
```

Code Listing 9: Basic D*Lite implementation

To visualize the path found a red line was added upon the processed image. An example of this is shown below.

Figure 10: Right image shows path calculated from a D*Lite algorithm

C.8 IMU

The planned implementation was to merge sensor data from the encoders and the IMU to get a more accurate estimation. Implementing an IMU would require an Arduino micro-controller[3]. While this would have been advantageous in our project, it would also have made the whole project more complex and added a couple physical components to our system. Using a Arduino would also have have added battery drainage, and the capacity was already at the limit.

```
1
2  // Includes library to communicate with IMU.
3  #include <SparkFunLSM9DS1.h>
4
5  // Creates an instance of IMU as a object.
6  LSM9DS1 imu;
7
8  // Defines the IMU addresses.
9  #define LSM9DS1_M 0x1E
10 #define LSM9DS1_AG 0x6B
11
12 // Delay between transmitions in milliseconds.
13 const int delayTime = 50;
14
15 // Absolute angle of the IMU.
16 float angle1 = 0;
17 float angle2 = 0;
18
```

```
19 #define DECLINATION -1.20
20
21 void setup() {
22   // Start logging to terminal.
23   Serial.begin(9600);
24
25   // Sets the addresses to the imu unit.
26   imu.settings.device.commInterface = IMU_MODE_I2C;
27   imu.settings.device.mAddress = LSM9DS1_M;
28   imu.settings.device.agAddress = LSM9DS1_AG;
29
30   // Above settings take effect when imu.begin() is called.
31   while (!imu.begin())
32     Serial.println("No connection with IMU.");
33 }
34
35 void loop() {
36
37   // Read values from gyro.
38   if (imu.gyroAvailable())
39     imu.readGyro();
40
41   // Values from magnetometer.
42   if (imu.accelAvailable())
43     imu.readAccel();
44
45   Serial.print("Gyro angle change: ");
46   Serial.print(doGetGyroAngle(imu.calcGyro(imu.gz)), 2);
47
48   //Serial.print("S");
49   //Serial.println(String(angle));
50   //Serial.print("E");
51
52   delay(delayTime);
53 }
54
55 // Calculates angle from gyro.
56 float doGetGyroAngle( float angleVelocity) {
57   return (angleVelocity * delayTime / 1000);
58 }
```

Code Listing 10: Reading gyroscope with Arduino

In code listing 10 an Arduino script which reads and formats IMU-data from a 9DOF-sensor and prints it with a serial writer. Through testing it became evident that the sensor measurements drifted even when the chip remained stationary. It is possible to compensate for some of the drif with a complimentary filter. However, given the quality of this specific sensor it was deemed unlikely that it would improve enough to be beneficial.

D Results

To test the SLAM algorithm, the hallways of the lab building were mapped. Below are some results trough various stages of the implementation.

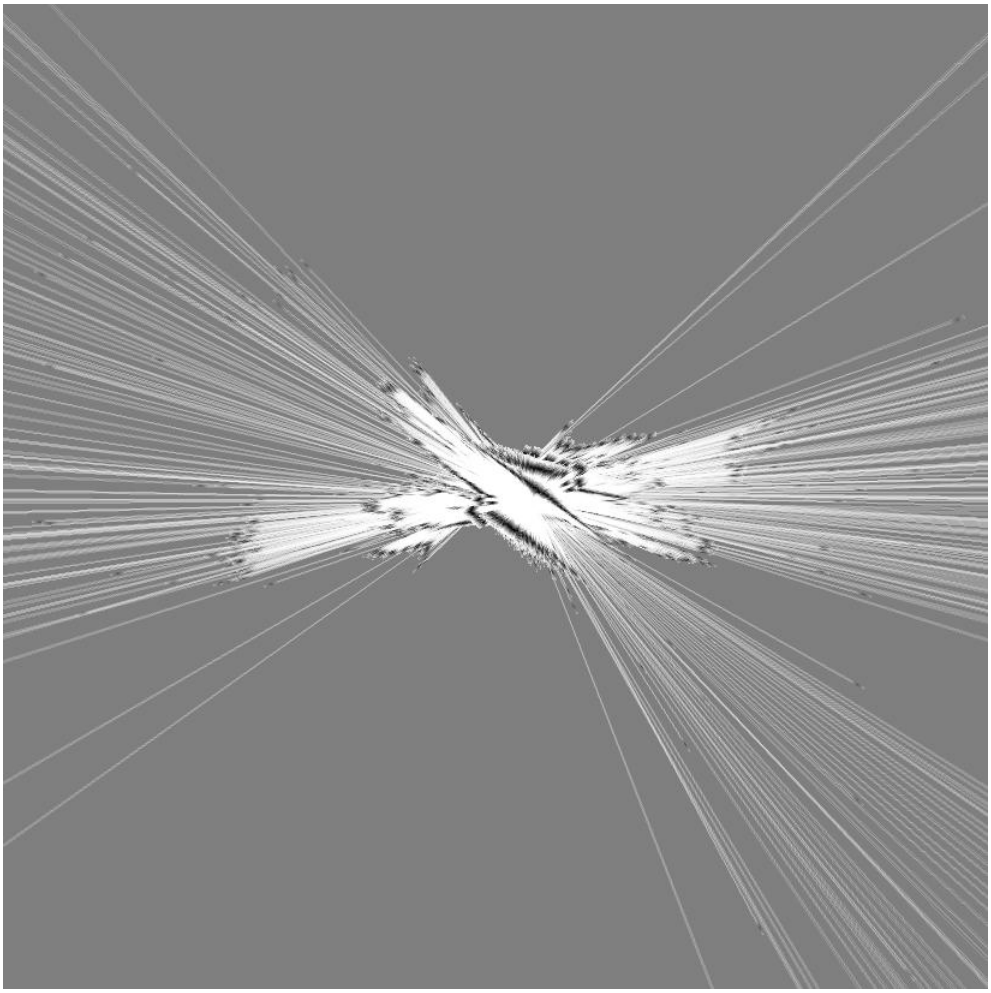


Figure 11: Hallway outside the labs L160 and L163

In figure 11 the SLAM-algorithms positional data were incorrect. It lost track of its current position and tried to re-center itself, but ended up rewriting the hallway several times. Each scan iteration shifted the hallway with a slight angle, yielding a cluttered map.

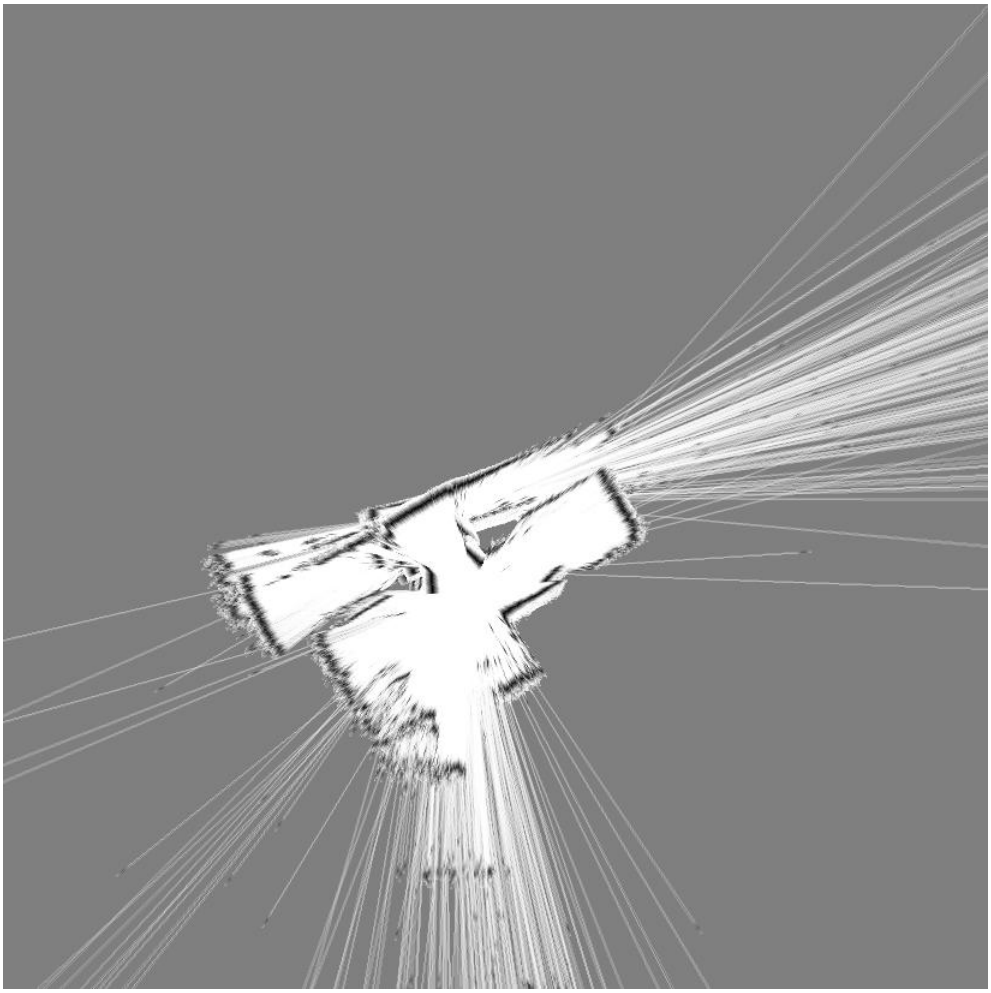


Figure 12: Outside L160, we drove from hallway to the common area

Figure 12 displayed very promising results regarding the mapping. However, this scan was also conducted with incorrect positional data. Encoder values calculated in a manner which didn't limit the values between 0 and 360 degrees. Rework was required and a lot of the code was reverted. Even though this scan was created with incorrect odometric data, the map turned out better than expected. The map has clearly defined areas which the car can travel in (white), and areas that are solid objects or obstacles.

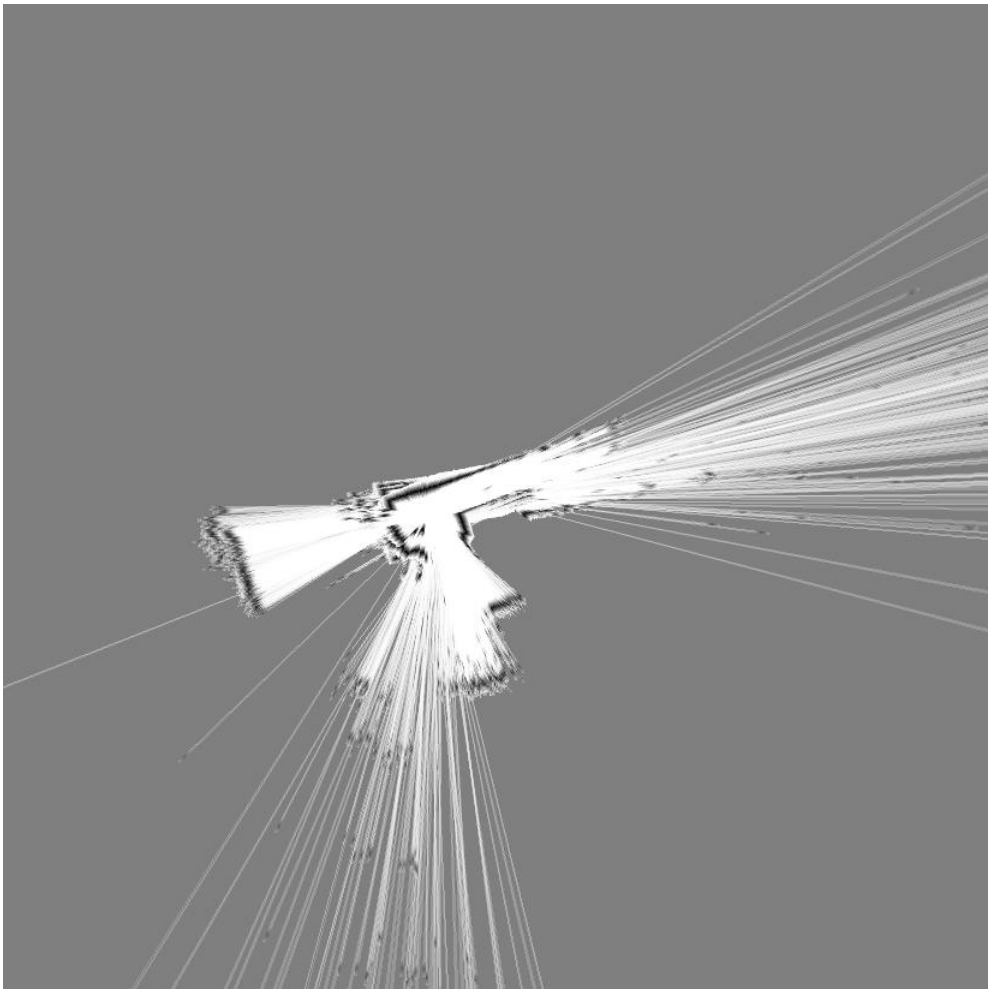


Figure 13: Outside L160, drove 2-3 meters down the hallway

In figure 13 a short scan was conducted, which detected corners and walls correctly, however this was also with incorrect encoder values as in figure 12

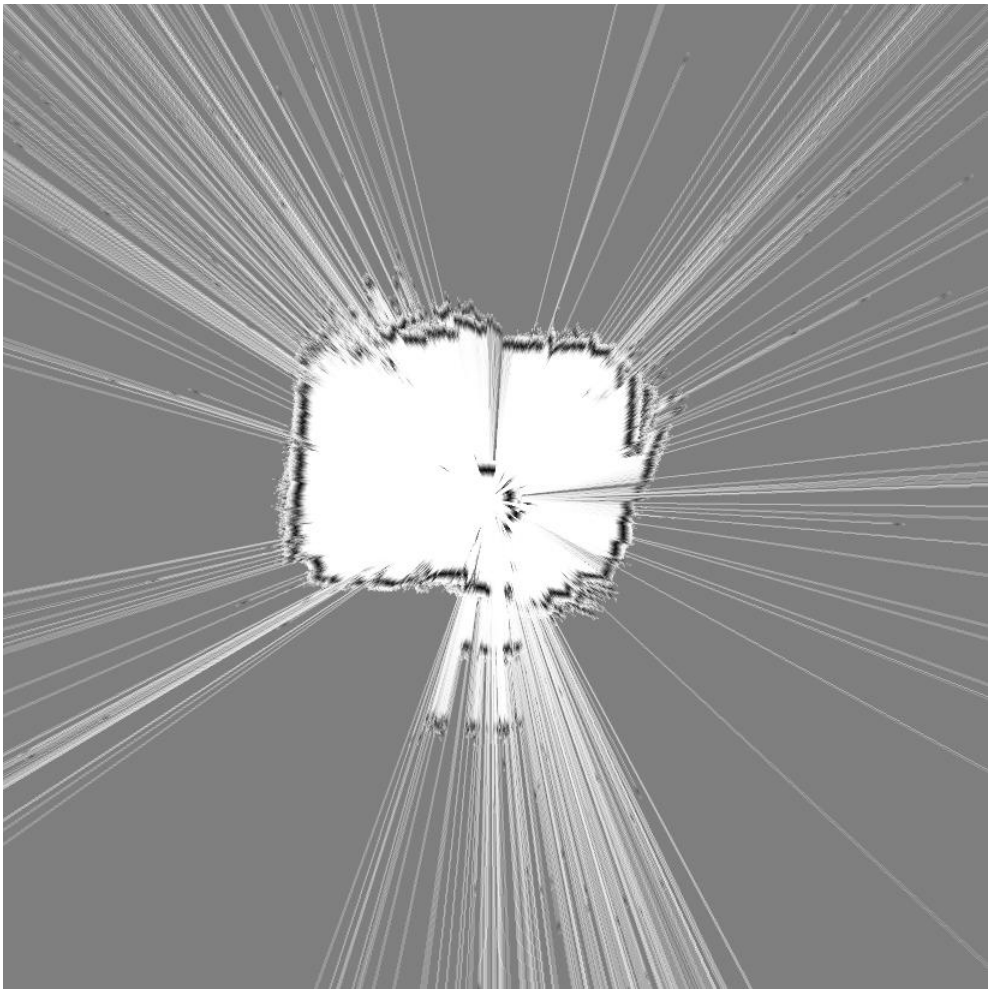


Figure 14: On the table in L160

In figure 14 one can observe that the LIDAR recognizes walls and cabinets. However, this was taken on the same SLAM version as figure 13 and 12.

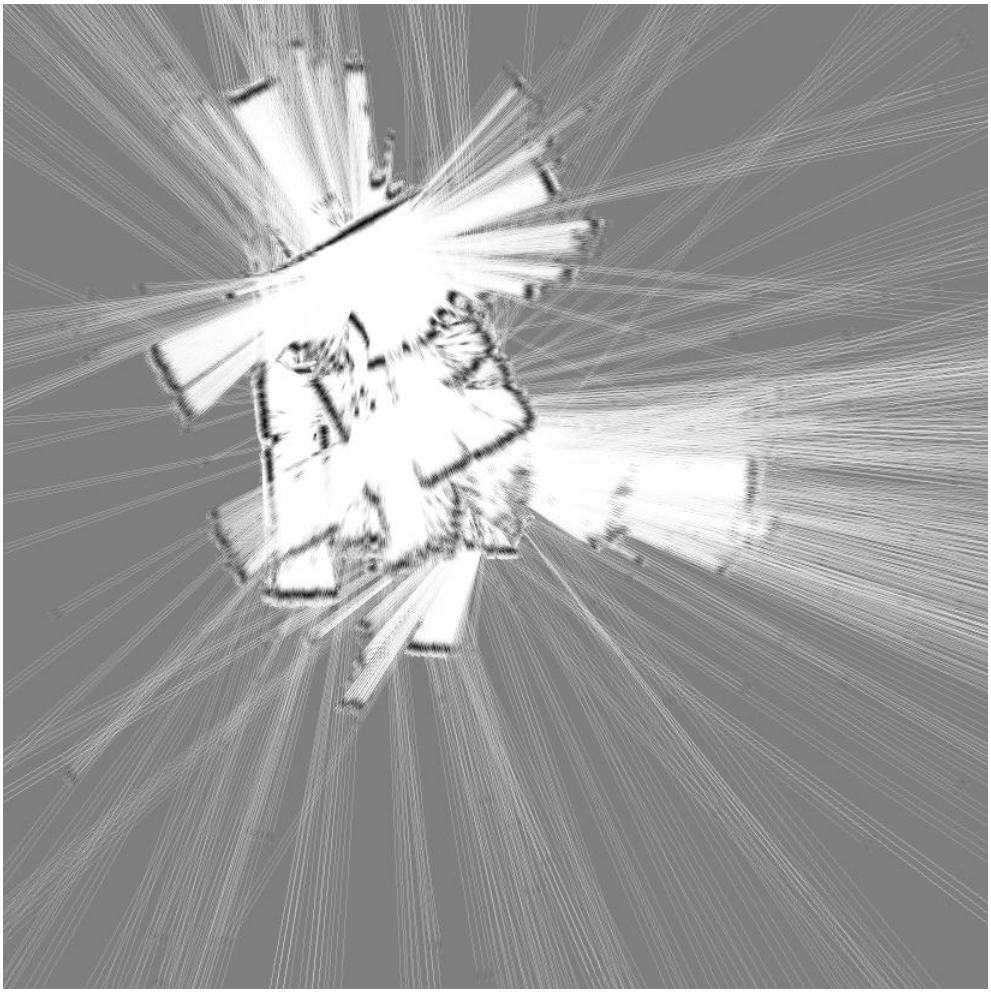


Figure 15: Common area outside L160

In figure 15 the odometric data in the program had several issues. The SLAM-algorithm parameters is close to where they should be. Communication with the motor-controller was a bottle-neck, and had to be rewritten properly. If the odometric data is not fast enough it would result in the mapping-algorithm to lose positional data, and thus get disoriented.

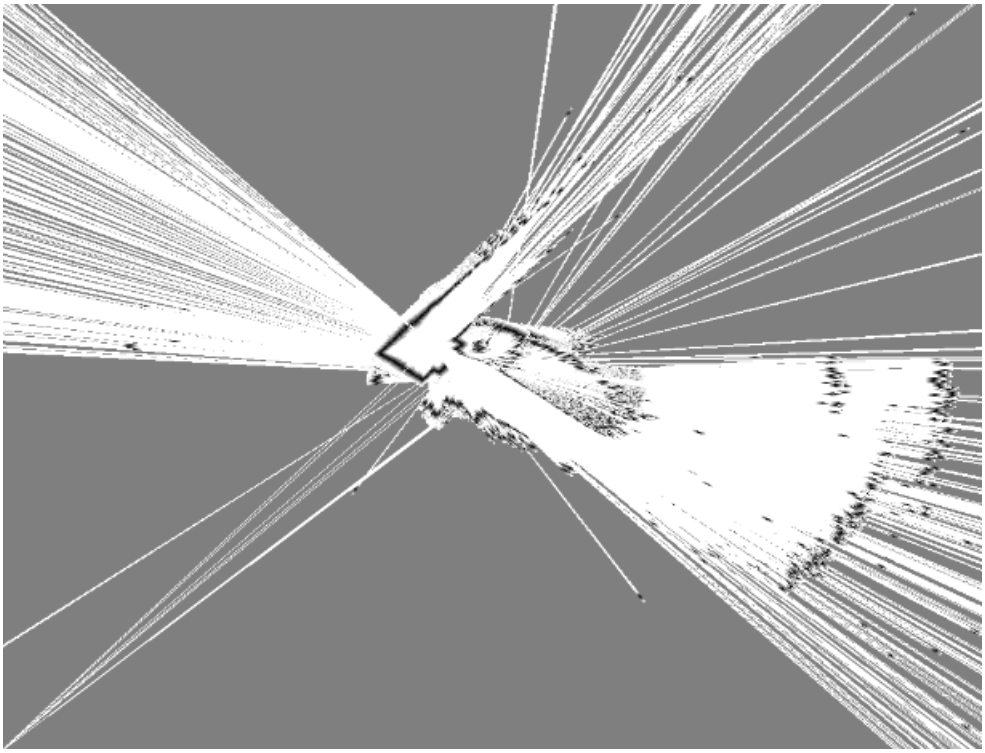


Figure 16: Mapped from common area to hallway outside L167

The final mapping was done outside the computer lab (L167), the edges in the hallway is clearly visible but the image is distorted. One can clearly see point-cloud rays that extend far beyond the boundaries of the environment.

E Discussion

E.1 SLAM

After compensating for physical input errors the SLAM scans was acceptable(most of the time). However, there are still issues with the random mutation algorithm. For the most time it is pretty good, but one can clearly see that, over time, the positional error increases. Certainly the SLAM-algorithm alone is not the issue but a certain amount of error is to be expected with the random mutation approach. Throughout the testing it was observed that odometry had little impact on the results since the algorithm weights LiDAR measurements heavier. There are certain areas in the building where LiDAR data should be weighted heavier than odometry. For example, when the car is travelling in an area which has several distinctive features such as a stairwell, benches, and corners. However, when the car is travelling through an area without those features(such as the hallway) odometry should be used to compensate when the algorithm is not able to determine change in position from the LiDAR-scans provided to the algorithm.

Some time were spent trying to optimize the settings of the particle filter, such as increasing the number of iterations it makes trying to find the best results. Increasing the number of iterations did improve the results, though it also had an impact on the Odroid running the software. Making the number of iterations bigger do demand more computational power, thus it becomes something each project has to consider based on their hardware. To be able to up the computational time to some extent the LiDAR-scanning and SLAM-processing was moved to two different threads, which freed up some additional space for the algorithm within the timeline. The restrictions on time are dependent on how much time each scan of the LiDAR takes. In order to process every scan the LiDAR does, the SLAM-algorithm cannot use more time than the LiDAR-scan takes. However, that is the only limit the SLAM-algorithm has when it comes to computational time. This limit is an result of how the algorithm takes scans as input. Being designed for a general purpose for cheap LiDARs it is limited. Instead of taking in each measurement

E.2 Hardware

The car is able to drive forward without slipping, but when turning using differential drive, it causes the wheels slip on the ground and the absolute angle of the car is hard to extract precisely. A possible solution is to change the way the car turns. Replacing the front engines with casters, and only using two motors with differential drive in the back, would not introduce the same problems with tire slipping, ulitimatly reducing error in calculated turn angle.

The plexiglass which components are mounted on, might also be reconsidered, since expansion and redesign is a problem. Components where going to be mounted by drilling holes and use spacers, bolts and nuts to hold them down. This technique was used to hold the motor controller but drilling holes easily introduced cracks in the glass. For the rest of the components double sided tape was utilized, which is surprisingly strong and components have not yet fallen off. Having expand-ability and a modular design in mind, plexiglass is not the best solution and should be replaced with something with the same properties as the car's frame see figure 6.

E.3 Software

E.3.1 Concurrency

Throughout the project, when adding new modules or functionality, the concurrency aspects were discussed to a great degree. Trying to find the most efficient, thread-safe and viable solutions that would ensure a stable software while also being expandable. As of now the main program is thread-safe. Since navigation was not implemented as planned, the core of the thread-communication is the Slam-class. Using storage boxes with atomic- and volatile-fields ensure that it is, in fact, thread-safe. Since the demand for using such objects are that more than one thread interact with the field, it was necessary to implement. With the abstraction of the storage box it is now possible to further build upon the software with similar communication between threads. The executor service implemented for the Map-stream ensured that only one map was uploaded at the time. However, this might seem a bit excessive since there was only one task to be executed. In accordance with thread-safety design, it is appropriate to ensure that one Map-stream was finished before another could start. As such, it seems reasonable to im-

plement the executor service.

Since the navigation-modules were not implemented in the main software, much of the planned thread-communication did not appear in the source-code. The planned implementation was to create a cluster of navigation threads. One thread would format the map for the navigation-algorithm, another thread would run the map through the algorithm while the third thread would use the output from the algorithm to send motor commands to the control thread. The communication between these threads would mostly be done by storage boxes. A executor service could have been implemented by the thread that handles the algorithm's output to execute motor-command tasks when the need occurs. There would also be created an interface that both the GUI and the navigation-control thread could use to control the motors. However, such an interface would need to be thread-safe and thus be synchronized. This would lock the object to whichever thread using it, as was intended.

To improve the threading in the GUI, it might have been feasible to let the Main- and MotorControl-threads utilize the same socket, especially since the Main-thread rarely sends commands over the socket. This would not have been hard to implement, and it would cause in fewer points of failure(sockets). Discussion on improvements of the GUI-thread design can be found in section E.3.3.

E.3.2 Main

The main program is built with sturdiness and thread safety in mind, but because of the program's modules there is not much to consider regarding threads. Each of the modules has their own thread on which they do their calculations, and since the motor-controller has software which reports encoder changes since last report, we felt a shared object containing encoder data is not required.

For the java classes communicating with the python server have thread priority is used, this is implemented so the encoder readings get priority over sending motor commands. This is implemented so that java is not in the middle of a encoder request, when a motor command is sent.

E.3.3 GUI

When the GUI was designed, "the observer pattern" was not known, so both video streams(web-cam and slam images) where implemented by a thread getting a reference to an ImageViewer object. This is not the best style and "the observer pattern" ideology should have been utilized, to allow the GUI controller full control over GUI components. When status labels where implemented this was done using "the observer pattern", in this case the labels are updated in the GUI controller. This approach yields better code structure and the streams should have been implemented using the same method. "The observer pattern" is described in the theory section B.7. In the rest of the GUI code the focus has been on high cohesion and low coupling.

E.4 Navigation

Another integral part of an autonomous system is navigation. To plan the path the car should take to a goal position a path planning algorithm is needed. The algorithm should plan the path based on known information (map of the area), and should be able to re-plan the path based on new information gained while driving, such as new obstacles in the original path. To achieve this, multiple algorithms have been developed the last 50 years, but the main algorithms considered for this project were A* and D* (D* Lite).

Both algorithms are based on the Dijkstra algorithm, which finds the shortest path between a start or source node and all other connected nodes by iteration through the nodes and using path costs to determine the shortest paths. This can be modified to stop when a goal node has been reached, but for the purposes of this project this is still a very inefficient algorithm, as nodes in all directions are considered equally, regardless of if they bring us closer to the goal or not. A* improves upon this by using a heuristic function that estimates the best option in each iteration. A* uses a best-first search where both path cost and distance to goal is considered in deciding the next node to expand to in each iteration. This can drastically decrease the time it takes to find the shortest path. D* uses the same principles, but what sets it apart from A* is that it stores known information about the area after the initial shortest path has been found. When new information, such as new obstacles, has been found it only needs to update that section of the map, and if this interferes with the current path can more quickly plan a new path

around the obstacle. This seems to be the most optimal algorithm for our project, thus a lighter variation of D*, D* Lite, was considered best for this project.

F Conclusion

The car is able to recognize its position and draw a two-dimensional map of it's surroundings with various degrees of accuracy. One of the main problems is the low sampling rate of the Lidar, making the positional aspect of slam susceptible to error. This problem can be seen in the form of the map getting re-drawn upon the previous map with a shift in angle. This ends creating a picture consisting of a stack of maps, where each entry is tilted slightly compared to the previous one. This problem is more prevalent when the car is turning or when increasing the velocity. In some cases the car does not have enough distinct objects in the surroundings to recognize its position, and it preforms few scan iterations to re-gain its bearings, sometimes causing it to re-drew the map perpendicular on the previous map. To help mitigate the problem, odometry based on encoder data was introduced to the equation. This should in theory help remove edge cases and improve the positional accuracy, while in practice it was lowly weighted by the algorithm and no immediate improvement could be seen.

The software generally preforms well. It has been constructed to maintain both object oriented and concurrency practices. The structure is modular with abstracted classes and and a main application manager that has a superior control over the program. Problems can be encountered when restarting the services. A few can be attributed to physical properties, such as the inrush current causing components to receive less than ideal amount of power. Others are caused by sockets and some objects not being discarded properly, thus causing problems when re-instantiated. In the GUI the structure and principles applied are similar and the visuals are structured in a simple setup. If there is one thing to change about the GUI it would be the re-sizing behaviour. When enlarging it keeps all components the same size and pushes them to the top left corner, when shrinking it places the components on top of each other.

References

[1] An introduction to Atomic Variables in java. URL: https://www.baeldung.com/java-atomic-variables.

[2] Norhafizan Ahmad et al. "Reviews on various inertial measurement unit (IMU) sensor applications". In: *International Journal of Signal Processing Systems* 1.2 (2013), pp. 256–262.

[3] Yusuf Abdullahi Badamasi. "The working principle of an Arduino". In: *Electronics, computer and computation (icecco), 2014 11th international conference on*. IEEE. 2014, pp. 1–4.

[4] Suraj Bajracharya. "BreezySLAM: A Simple, efficient, cross-platform Python package for Simultaneous Localization and Mapping (thesis)". In: (2014).

[5] BASICMICRO. Library for motorcontroller server. URL: http://www.basicmicro.com/downloads.

[6] Daniel Beard. *DStarLiteJava*. Library for D* Lite navigation algorithm. 2017. URL: https://github.com/daniel-beard/DStarLiteJava.

[7] A. I. Eliazar and R. Parr. "DP-SLAM 2.0". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. Vol. 2. Apr. 2004, 1314–1320 Vol.2. DOI: 10.1109/ROBOT.2004.1308006.

[8] Bartosz Firyn. *Webcam Capture API*. Driver for accesing webcam from java application. 2018. URL: https://github.com/sarxos/webcam-capture.

[9] Adam Freeman. "The Observer Pattern". In: *Pro Design Patterns in Swift*. Springer, 2015, pp. 447–472.

[10] Tom Hawtin. Explenation and use of Atomic Field Reference Updater. URL: https://stackoverflow.com/questions/8262982/atomicreferencefieldupdater-methods-set-get-compareandset-semantics.

[11] Daniel J.H. Hyun Hwang. *sweep-sdk*. Library to get data from lidar sensor. 2018. URL: https://github.com/scanse/sweep-sdk.

[12] Simon D. Levy. *BreezySLAM*. Simple, efficient, open-source package for Simultaneous Localization and Mapping. 2018. URL: https://github.com/simondlevy/BreezySLAM.

[13] Jie Ma. "Waymo, Uber Driverless Projects Make Scanning Sensors Cheaper". In: (Sept. 2018). URL: https://www.bloomberg.com/news/articles/2018-09-05/waymo-uber-driverless-projects-make-scanning-sensors-cheaper.

[14] Oracle. JavaDoc of the ExecutorService-class. URL: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html.

[15] Bruno Steux and Oussama El Hamzaoui. "CoreSLAM: a SLAM Algorithm in less than 200 lines of C code". In: (Nov. 2018).

[16] Hui Wang, Michael Jenkin, and Patrick Dymond. "Deterministic topological visual SLAM". In: *Proceedings of the Fifth Symposium on Information and Communication Technology*. ACM. 2014, pp. 126–135.

G Appendices

1. Motor Controller Roboclaw User Manual / Data Sheet
2. Sweep Lidar Manual / Data Sheet
3. ODROID-XU4 Manual

BASICMICRO

RoboClaw Series Brushed DC Motor Controllers

RoboClaw Solo
RoboClaw 2x5A
RoboClaw 2x7A
RoboClaw 2x15A
RoboClaw 2x30A
RoboClaw 2x45A
RoboClaw 2x45A ST
RoboClaw 2x60A
Roboclaw 2x60HV
Roboclaw 2x120A
Roboclaw 2x160A
Roboclaw 2x200A

User Manual

Firmware 4.1.20 and Newer
Hardware V3, V4, V5 and V6
User Manual Revision 5.6

Contents

Firmware History

RoboClaw is an actively maintained product. New firmware features will be available from time to time. The table below outlines key revisions that could affect the version of RoboClaw you currently own.

Revision	Description
4.1.20	• Added Default Deceleration setting • Added Forward/Reverse Limit support • Added Forward Limit/Reverse Home support • Fixed power up Home switch state(now reads the initial state if HOME/Manual Home is enabeld • Home/Limit switch now supports on the fly changes • Fixed short PWM glitch on power up • Resets internal position counter when encoder is reset(fixes position movement glitches after Encoder Reset) • Fixed sign magnitude on HV units
4.1.19	• Adjusted deadtime on MC5,7,15,30,60
4.1.18	• Fixed invalid conditional in MC120-160 A/D sample timer • Fixed Set/Zero Encoders • Added RC Encoder Mode selection Option using RC/TTL signal on S3
4.1.17	• Adjusted RC/Analog deadband filtering • Changed A/D sampling to prevent PWM noise • Added GetDefaultAccel commands • Changed RC/Analog w/ Encoder modes to use DefaultAccel for Accel/Deccel • Added MC60A V5 • Added MC160A • Removed Sign Magnitude mode on HV models(cant do noise free A/D sampling in Sign Magnitude mode on HV boards) • Fixed power on zero position movement in RC w/ Encoder mode.
4.1.16	• Adjusted RC/Analog controlled Velocity and Position Control functions. • Fixed Sync Motor option in Position Settings window.

Warnings

There are several warnings that should be noted before getting started. Damage can easily result by not properly wiring RoboClaw. Harm can also result by not properly planning emergency situations. Any time mechanical movement is involved the potential for harm is present. The following information can help avoid damage to RoboClaw, connected devices and help reduce the potential for harm or bodily injury.

Disconnecting the negative power terminal is not the proper way to shut down a motor controller. Any connected I/O to RoboClaw will create a ground loop and cause damage to RoboClaw and attached devices.

Brushed DC motors are generators when spun. A robot being pushed or coasting can create enough voltage to power RoboClaws logic intermittenly creating an unsafe state. Always stop the motors before powering down RoboClaw.

RoboClaw has a minimum power requirement. Under heavy loads, without a logic battery and, brownouts can happen. This will cause erratic behavior. A logic battery should be used in these situations.

Never reverse the main battery wires Roboclaw will be permenantly damaged.

Never disconnect the motors from RoboClaw when under power. Damage will result.

Introduction

Motor Selection

When selecting a motor controller several factors should be considered. All DC brushed motors will have two current ratings, maximum stall current and continuous current. The most important rating is the stall current. Choose a model that can support the stall current of the motor selected to insure the motor can be driven properly without damage to the motor controller.

Stall Current

A motor at rest is in a stall condition. This means during start up the motors stall current will be reached. The loading of the motor will determine how long maximum stall current is required. A motor that is required to start and stop or change directions rapidly but with light load will still require maximum stall current often.

Running Current

The continuous current rating of a motor is the maximum current the motor can run without overheating and eventually failing. The average running current of the motor should not excede the continuous current rating of the motor.

Shut Down

To shut down a motor controller the positive power connections should be removed first after the motors have stopped moving. Powering off in an emergency, a properly sized switch or contactor can be used. A path to ground for regeneration energy to return to the battery should always be provided. This can be accomplish by using a power diode with proper ratings to provide a path across the switch or contactor when in an open circuit state.

Run Away

During development of your project caution should be taken to avoid run away conditions. The wheels of a robot should not be in contact with any surface until all development is complete. If the motor is embedded, ensure you have a safe and easy method to remove power from RoboClaw as a fail safe.

Wire Lengths

Wire lengths to the motors and from the battery should be kept as short as possible. Longer wires will create increased inductance which will produce undesirable effects such as electrical noise or increased current and voltage ripple. The power supply/battery wires must be as short as possible. They should also be sized appropriately for the amout of current being drawn. Increased inductance in the power source wires will increase the ripple current/voltage at the RoboClaw which can damage the filter caps on the board or even causing voltage spikes over the rated voltage of the Roboclaw, leading to board failure.

Power Sources

A battery is recommended as the main power source for the motor controller. Some power supplies can also be used without additional hardware if they have built in voltage clamps or if used with very low current motors. Most Linear and Switching power supplies are not capable of handling the regeneration energy generated by DC motors. Switching power supplies will momentarily reduce voltage and/or shut down, causing brown outs which will leave the controller in an unsafe state. The MCPs minimum and maximum voltage levels can be set to prevent some of these voltage spikes, however this will cause the motors to brake when slowing down in an attempt to reduce the over voltage spikes. This will also limit power output when accelerating motors or when the load changes to prevent undervoltage conditions. Voltage clamp solutions may be required for higher power motors when using power supplies.

Logic Power

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 low quality 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. When wiring encoders make sure the direction of spin is correct to the motor direction. Incorrect encoder connections can cause a run away state. Refer to the encoder section of this user manual for proper setup.

Getting Started

Initial Setup

RoboClaw offers several methods of control. Each control scheme has several configuration options. The following is quick start guide which will cover the basic initialization of RoboClaw. Most control schemes require very little configuration. The control options are covered in detail in this manual. The following is a basic setup procedure.

1. Read the Introduction and Hardware Overview sections of this manual. It is important to ensure the RoboClaw model chosen is rated to drive the selected motors. RoboClaw must be paired by the motor stall current ratings. Not running current.

2. 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.

3. The RoboClaw main modes can be configured using Ion Studio or on-board buttons. Ion Studio is the preferred method of configuration with additional options not available using the on-board buttons. However these additional options are not critical to RoboClaw's operation. This manual covers both configuration methods.

4. Once the configuration is complete see Wiring section of this manual. The basic wiring diagram should only be used for basic for testing purposes. The Safety Wiring diagram is recommended for safe and reliable operation.

Encoder Setup

RoboClaw supports several encoder types. All encoders require tunning to properly pair with the selected motors. The auto tune function can automatically tune for most all combinations. However some manual adjustment maybe required. The final auto tune settings can be adjusted for optimal performance.

1. Once Initial Setup is complete. Attached an encoder to your motor and wire as shown in the encoder section of this manual. Make sure the encoder can be powered from a 5VDC power source.

2. After the encoder is wired double check the wiring. Then proceed to the auto tune function in the Encoder section of this manual.

3. Auto tune will work in most all cases. Some manual tweaks may be necessary. If additional assistance is required contact support at support@ionmc.com

Hardware Overview

I/O

RoboClaw's I/O is setup to interface to both 5V and 3.3V logic. This is accomplished by internally current limiting and clipping any voltages over 3.3V. RoboClaw outputs 3.3V which will work with any 5V or 3.3V logic. This is also done to protect the I/O 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. Channel A of both EN1 and EN2 are located at the board edge on the pin header. Channel B pins are located near the heatsink on the pin header. The A and B channels are labeled appropriately on screw terminal versions.

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 logic side of RoboClaw can be powered from a secondary battery wired to LB IN. The positive (**+**) terminal is located at the board edge and ground (**-**) is the inside pin closest to the heatsink. Remove the LB-MB jumper if a secondary battery for logic will be used.

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. On models without this jumper the power source is selected automatically.

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

Main Battery Screw Terminals

The main power input can be from 6VDC to 34VDC on a standard RoboClaw and 10.5VDC to 60VDC on an HV (High Voltage) RoboClaw. The connections are marked **+** and **-** on the main screw terminal. The plus (**+**) symbol marks the positive terminal and the negative (**-**) marks the negative terminal. The main battery wires should be as short as possible.

Do not reverse main battery wires. Roboclaw will be permenantly damaged.

Main Battery Disconnect

The main battery should have a disconnect in case of a run away situation and power needs to be cut. 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. A typically solution would be an inexpensive contactor which can be sourced from sites like Ebay. A power diode rated for the maximum current the battery will deliver should be placed across the switch/contactor to provide a path back to the battery when disconnected while the motors are spinning. The diode will provice a path back to the battery for regenerative power even if the switch is opened.

Motor Screw Terminals

The motor screw terminals are marked with M1A / M1B for channel 1 and M2A / M2B for channel 2. For both motors to turn in the same direction the wiring of one motor should be reversed from the other in a typical differential drive robot. The motor and battery wires should be as short as possible. Long wires can increase the inductance and therefore increase potentially harmful voltage spikes.

Easy to use Libraries

Source code and Libraries are available on the Ion Motion Control website. Libraries are available for Arduino(C++), C# on Windows(.NET) or Linux(Mono) and Python(Raspberry Pi, Linux, OSX, etc).

Ion Studio Overview

Ion Studio

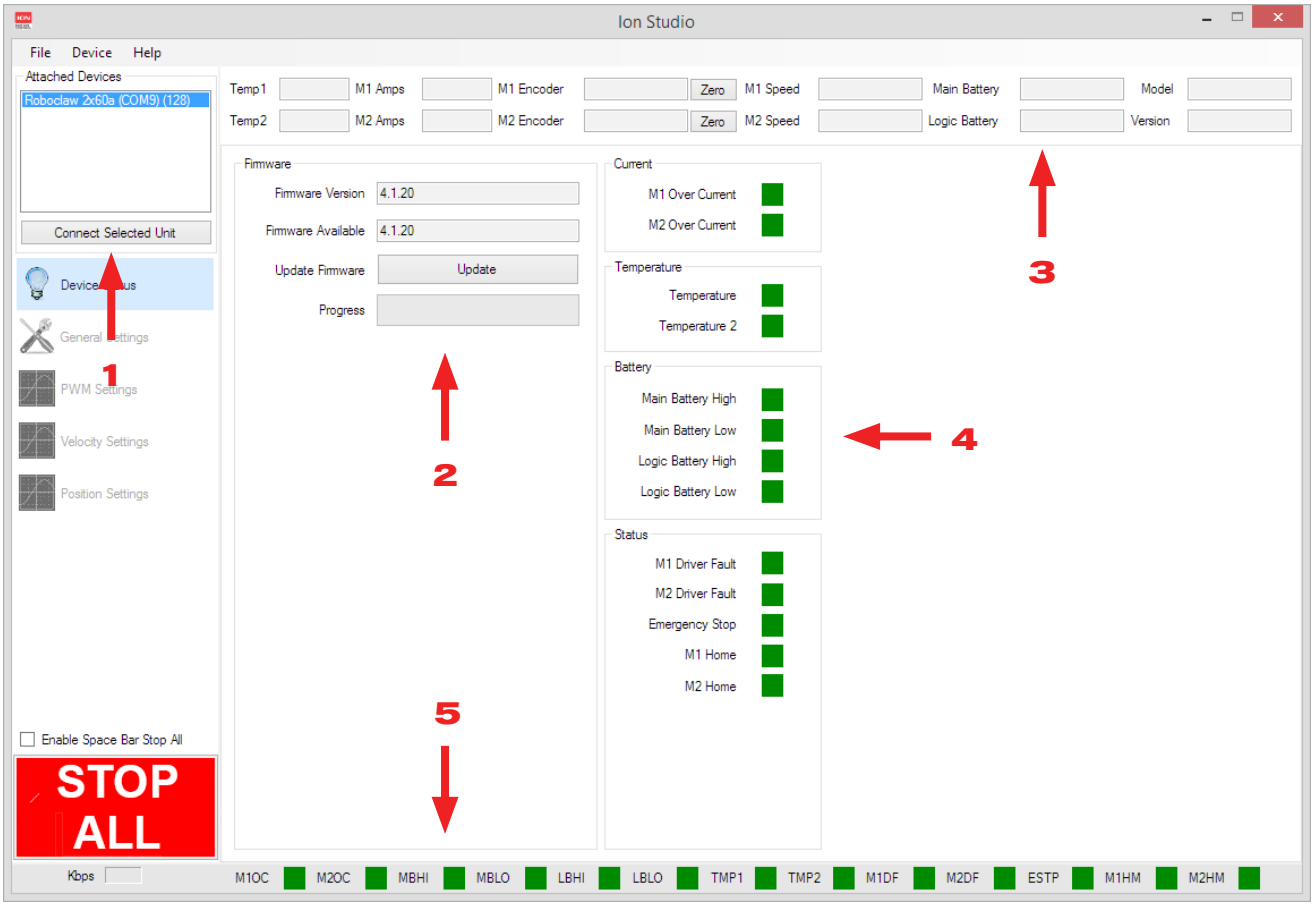
The Ion Studio software suite is design to configure, monitor and maintain RoboClaw. It's used to configure all the available RoboClaw modes and options. Ion Studio can be used to monitor and control RoboClaw. It can be download from http://www.ionmc.com. Once installed, each time Ion Studio is ran it will check for the latest version online.

Connection

This is the first screen shown when first running Ion Studio. From this screen you can select a detected RoboClaw and connect (1). More than one RoboClaw can be connected at a time. Box (1) is where the desired RoboClaw is selected.

After the RoboClaw is detected and it's firmware version is checked (2). If a newer firmware version is available it can be updated by clicking the Update Firmware button (2).

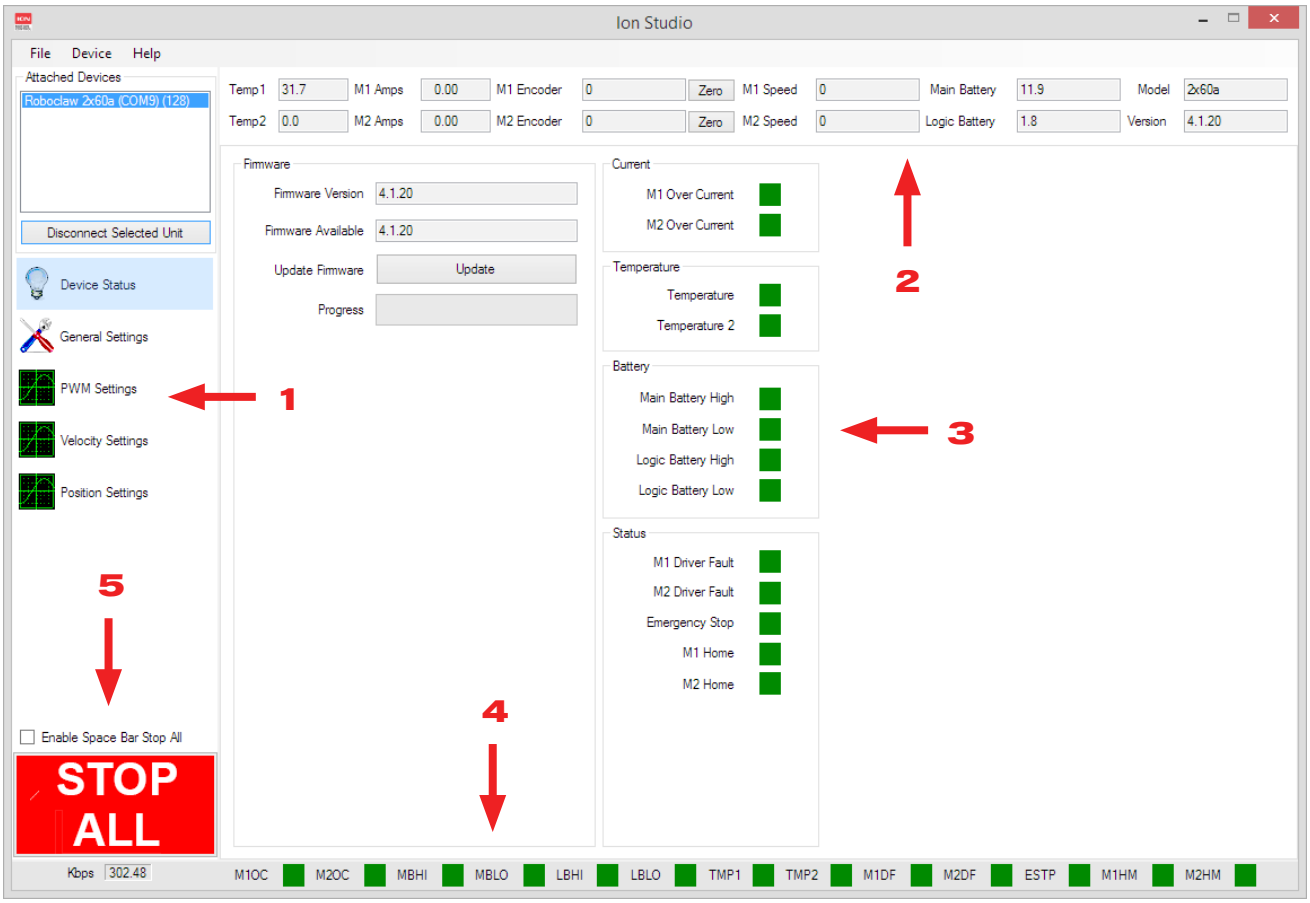
Fields (3,4,5) display current values and status. The feilds at the top of the screen (3) show the current value in each monitored parameter and are updated live once a RoboClaw is connected. Status indicators (4,5) indicate the current condition of the named monitor parameter. Green indicates operationing within the defined parameter. Yellow indicates a warning. Red indicates a fault.



Device Status

Once a RoboClaw is connected, the connection screen becomes active (1) and is now the Device Status screen. All status indicators (3,4) and monitored parameter feilds (2) will update to reflect the current status and values of the connected RoboClaw.

When an RoboClaw is connected the Stop All (5) button becomes active. There is a small check box to activate the Stop All function by using the space bar on the keyboard. This is safety feature and is the quickest method to stop all motor movements when using Ion Studio.



Device Status Screen Layout

Label	Function	Description
1	Window 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.

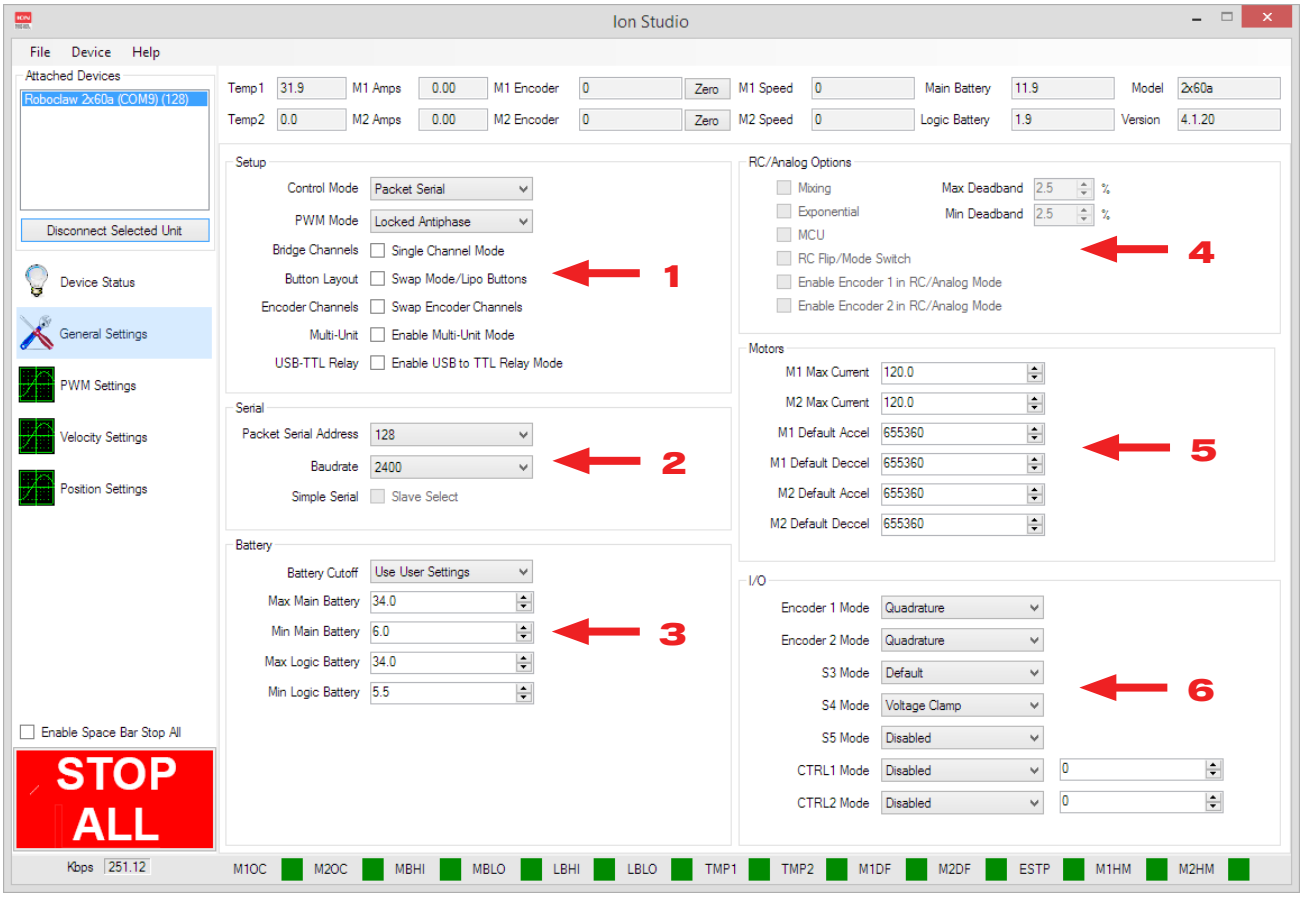
Status Indicator (4)

The status indicators shown at the bottom of the screen are an abbreviated duplication of the main status indicators shown on the device status screen.

Label	Description
M1OC	Motor 1 over current.
M2OC	Motor 2 over current.
MBHI	Main battery over voltage.
MBLO	Main battery under voltage.
LBHI	Logic battery over voltage.
LBLO	Logic battery under voltage.
TMP1	Temperature 1
TMP2	Optional temperature 2 on some RoboClaw models.
M1DF	Motor driver 1 fault.
M2DF	Motor driver 2 fault.
ESTP	Emergency stop. When active.
M1HM	Motor 1 homed or limit switch active. When option in use.
M2HM	Motor 2 homed or limit switch active. When option 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.



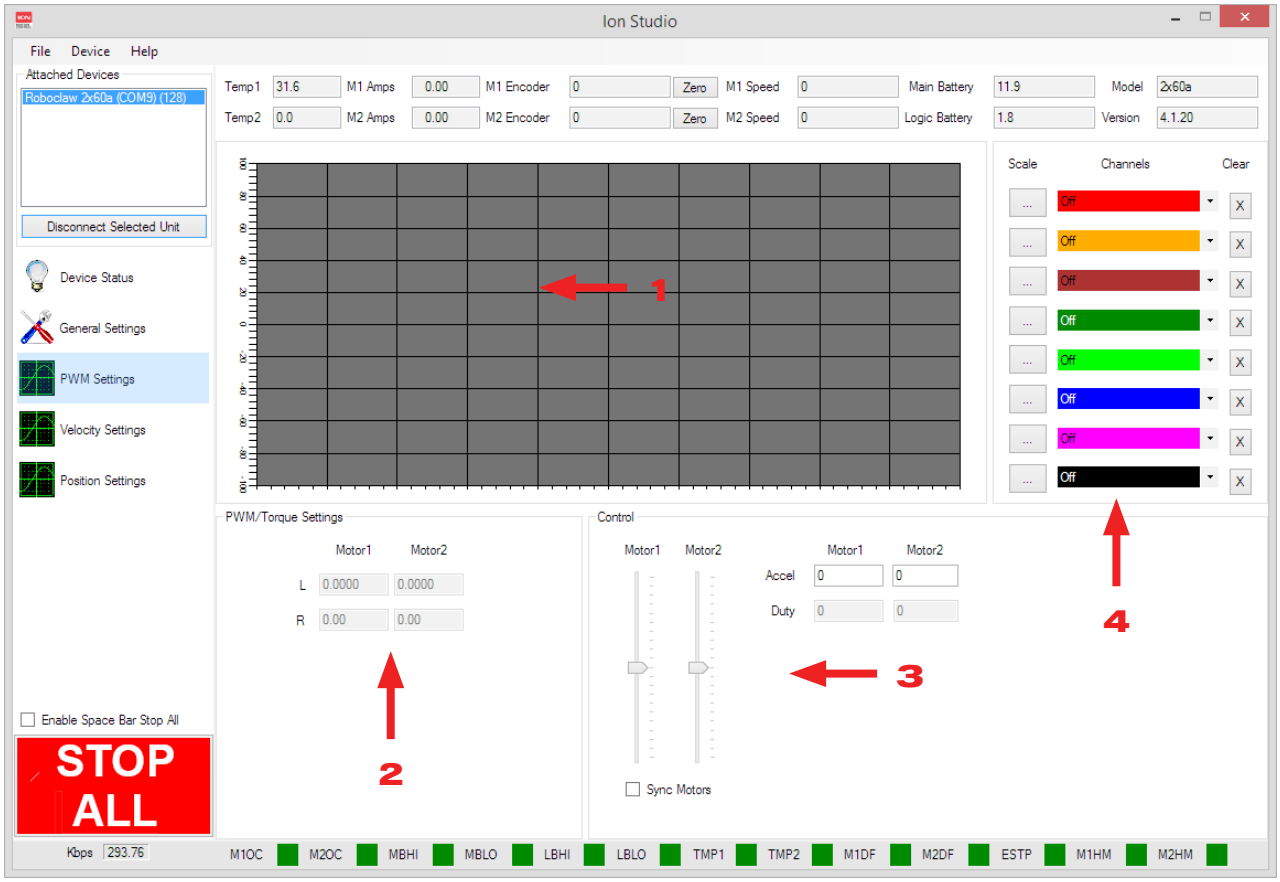
Configuration Options

Each control mode will have several configuration options. Some options will appear grayed out to indicate the option is not available for the selected mode. All setting changes will need to be saved and the RoboClaw reset in order to take. Select Save Settings under File in the menu bar.

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	RC/Analog Options	Configure RC and Ananlog control options.
5	Motors	Motor current, accel and deccel settings.
6	I/O	Set encoder input type. Set S3, S4 and S5 configuration options. Enabling output pins on certain models of RoboClaw.

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.



(1) Graph

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

(2) PWM/Torque Settings

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

(3) Control

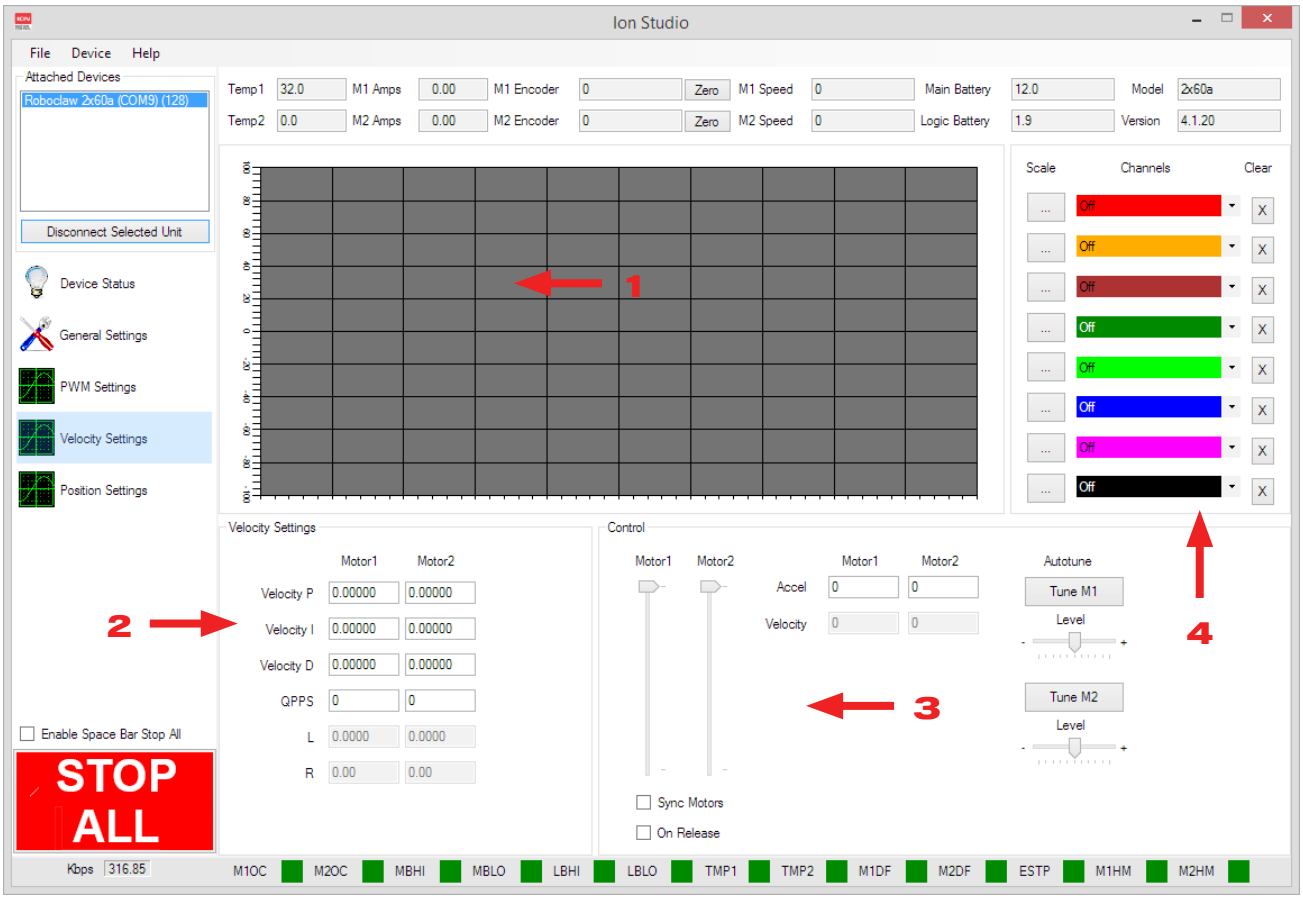
Function	Description
Motor 1	Controls motor 1 duty percentage forward and reverse.
Motor 2	Controls motor 2 duty percentage forward and reverse.
Sync Motors	Synchronises Motor 1 and Motor 2 Sliders.
Accel	Acceleration rate used when moving the sliders.
Duty	Displays the numberic value of the motor slider in 10ths of a Percent (0 to +/-1000).

(4) Graph Channels

Function	Description
Scale	Sets vertical scale to fit the range of the specified Channel.
Channels	Select data to display on the channel. The channel is graphed in the color shown. Channel options: • M1 or M2 Setpoint - User input for channel • M1 or M2 PWM - Motor PWM output • M1 or M2 Velocity - Motors Encoder Velocity • M1 or M2 Position - Motors Encoder Position • M1 or M2 Current - Motor running current • Temperature • Main Battery Voltage • Logic Battery Voltage
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.



(1) Graph

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

(2) Velocity Settings

Function	Description
Velocity P	Proportional setting for PID.
Velocity I	Integral setting for PID.
Velocity D	Differential setting for 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.

(3) Control

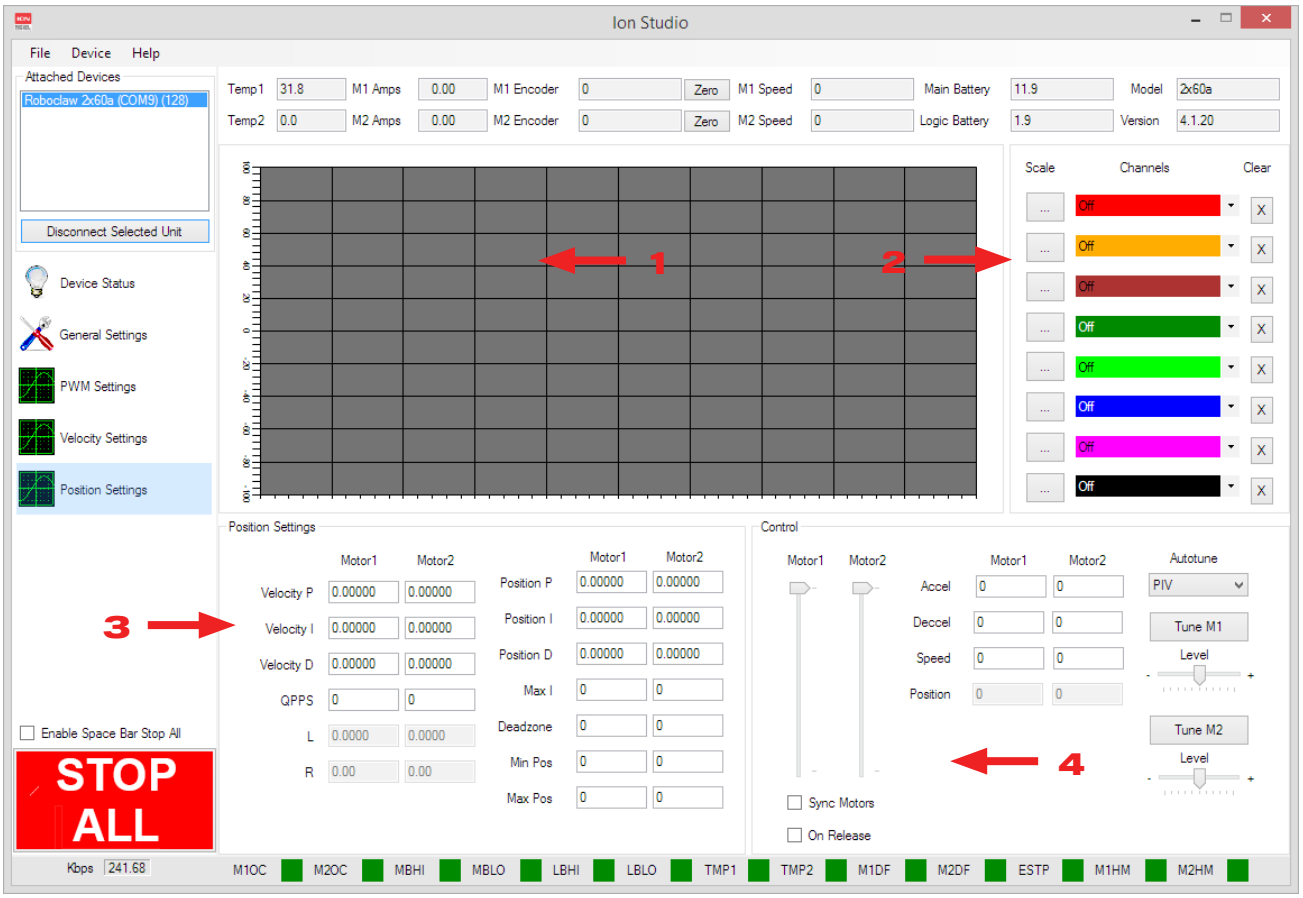
Function	Description
Motor 1	Motor 1 velocity control (0 to +/- maximum motor speed).
Motor 2	Motor 2 velocity control (0 to +/- maximum motor speed).
Sync Motors	Synchronises Motor 1 and Motor 2 Sliders.
On Release	Will not update new speed until the slider is released.
Accel	Acceleration rate used when moving the sliders.
Velocity	Shows the numeric value for the sliders current position.
Tune M1	Start motor 1 velocity auto tune.
Level	Adjust auto tune 1 values agressiveness. Sllide left for softer control.
Tune M2	Start motor 2 velocity auto tune.
Level	Adjust auto tune 2 values agressiveness. Sllide left for softer control.

(4) Graph Channels

Function	Description
Scale	Sets vertical scale to fit the range of the specified Channel.
Channels	Select data to display on the channel. The channel is graphed in the color shown. Channel options: • M1 or M2 Setpoint - User input for channel • M1 or M2 PWM - Motor PWM output • M1 or M2 Velocity - Motors Encoder Velocity • M1 or M2 Position - Motors Encoder Position • M1 or M2 Current - Motor running current • Temperature • Main Battery Voltage • Logic Battery Voltage
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.



(1) Graph

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

(2) Graph Channels

Function	Description
Scale	Sets vertical scale to fit the range of the specified Channel.
Channels	Select data to display on the channel. The channel is graphed in the color shown. Channel options: • M1 or M2 Setpoint - User input for channel • M1 or M2 PWM - Motor PWM output • M1 or M2 Velocity - Motors Encoder Velocity • M1 or M2 Position - Motors Encoder Position • M1 or M2 Current - Motor running current • Temperature • Main Battery Voltage • Logic Battery Voltage
Clear	Clears channels graphed line.

(3) 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.
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.

(4) 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).
Sync Motors	Synchronises Motor 1 and Motor 2 Sliders.
On Release	Will not update new speed until the slider is released.
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	Adjust auto tune 1 values agressiveness. Sllide left for softer control.
Tune M2	Start motor 2 velocity auto tune.
Level	Adjust auto tune 2 values agressiveness. Sllide left for softer control.

Firmware Updates

Ion Studio Setup

Download and install the Ion Studio application. Win7 or newer is required. When opening Ion Studio, it will check for updates and search for a USB Windows Driver to verify installation. If the USB driver is not found, Ion Studio will install it.

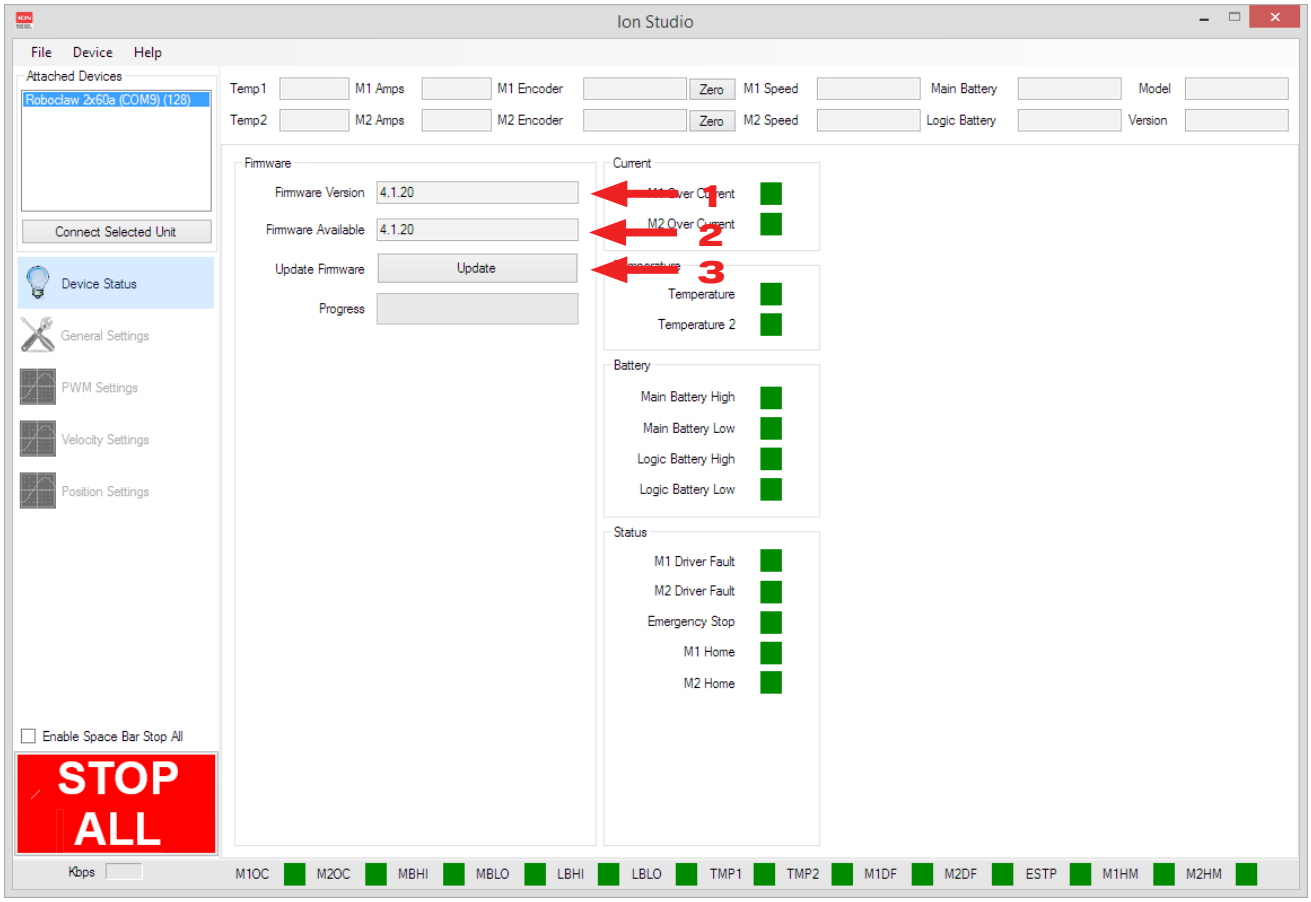
1. Open the Ion Studio application.

2. Apply a reliable power source such as a fully charge battery to power the motor controller.

3. Connect the powered motor controller to a USB port on your computer with Ion Studio already open.

Firmware Update

Once Ion Studio detects the motor controller it will display the current firmware version in the Firmware Version field (1). Each time Ion Studio is started it will check for a new version of its self which will always include new firmware. If an update is required Ion Studio will download the latest version and display it in the firmware available field (2).



1. When a new version of firmware is shown click the update button (3) to start the process.

2. Ion Studio will begin to update the firmware. While the firmware update is in progress the onboard LEDs will begin to flash. The onboard flash memory will first be erased. It is important power is not lost during this process or the motor controller will no longer function. There is no recovery if power fails during the erase process.

3. Once the firmware update is complete the motor controller will reset. Click the "Connect Selected Unit" button to re-connect.

Control Modes

Setup

RoboClaw has several fucntional control modes. There are two methods to configure these modes. Using the built-in buttons or Ion Studio. This manaul covers both methods of configuration. Ion Studio offers greater options for each mode and can be easier to configure the RoboClaw in several situations. However the built-in buttons are more than adequate in most all modes. Refer to the configuration section of this manual for mode setup instructions using Ion Studio or the built-in buttons.

There are 4 main modes with several variations. Each mode enables RoboClaw to be controlled in a very specific way. The following list explains each mode and the ideal application.

USB Control

USB can be used in any mode. When RoboClaw is in packet serial mode and another device, such as an Arduino, is connected commands from the USB and Arduino will be executed and can potential over ride one another. However if Roboclaw is not in packet serial mode, motor movement commands will not function. USB packet serial commands can then only be used to read status information and set configuration settings.

RC

Using RC mode RoboClaw can be controlled from any hobby RC radio system. RC input mode also allows low powered microcontrollers such as a Basic Stamp 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 use encoders if properly setup(See Encoder section).

Analog

Analog mode uses an analog signal from 0V to 2V 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 with joystick positioning systems or other non microcontroller interfacing hardware. Analog mode can use encoders if properly setup(See Encoder section).

Simple Serial

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 or an equivilent level converter circuit must be used since RoboClaw only works with TTL level inputs. 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 can only receive data. Encoders are not supported in Simple Serial mode.

Packet Serial

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 or an equivilent level converter circuit must be used since RoboClaw only works with TTL level input. In packet serial mode each RoboClaw is assigned a unique address. There are 8 addresses available. This means up to 8 RoboClaws can be on the same serial port. Encoders are support in Packet Serial mode(See Encoder section).

Configuration Using Ion Studio

Mode Setup

Download and install the Ion Studio application from http://www.ionmc.com. A PC with Windows 7 or newer is required. Ion Studio will check for a newer version each time it is ran. It will then search for the USB RoboClaw Windows Driver to verify installation. If the USB driver is not found Ion Studio will install it.

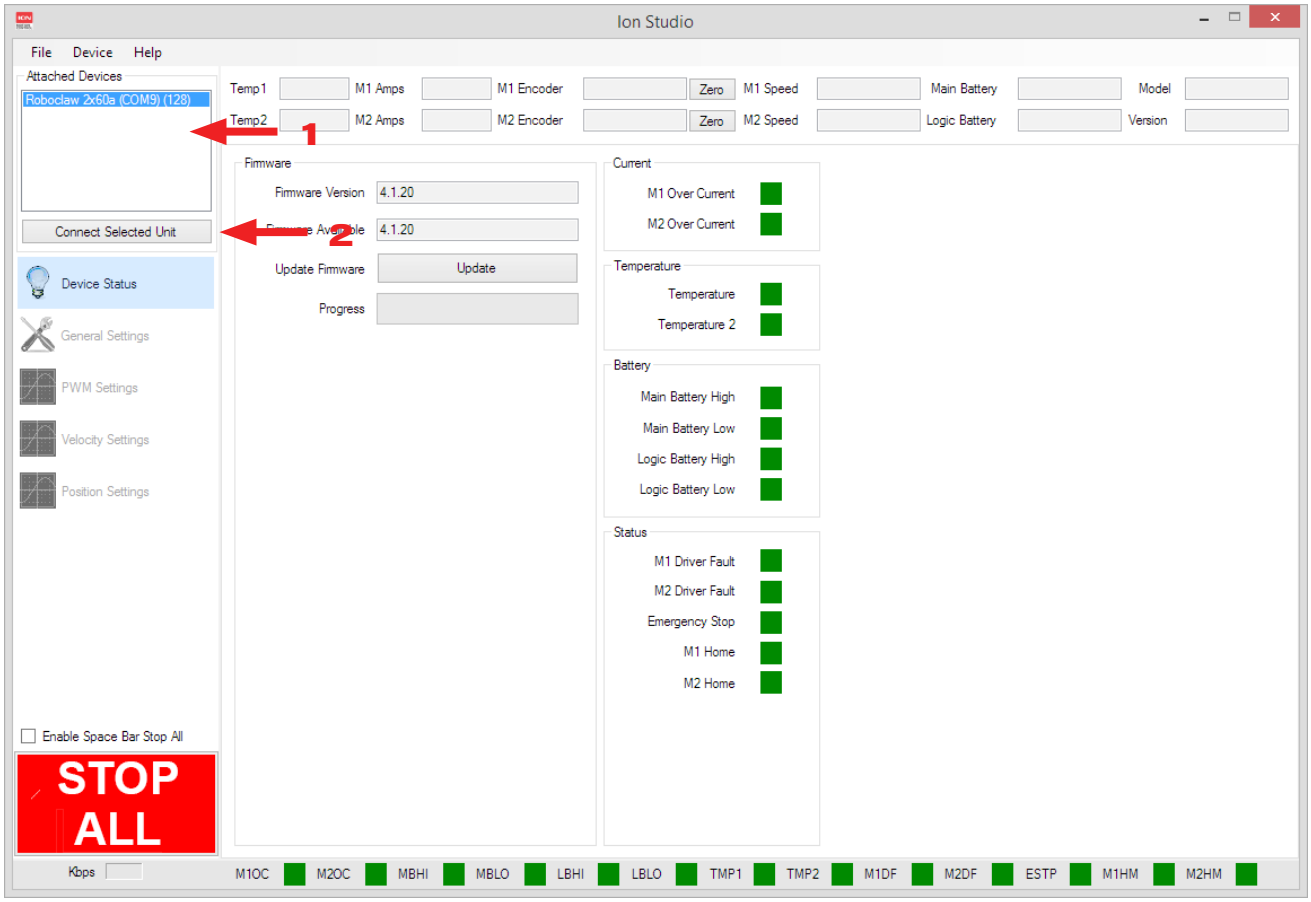
1. Open the Ion Studio application.

2. Apply a reliable power source such as a fully charge battery to power up RoboClaw.

3. Connect the powered RoboClaw to a USB port on your computer with Ion Studio already open. The RoboClaw USB driver may need to be installed Ion Studio will automatically handling installing the required driver.

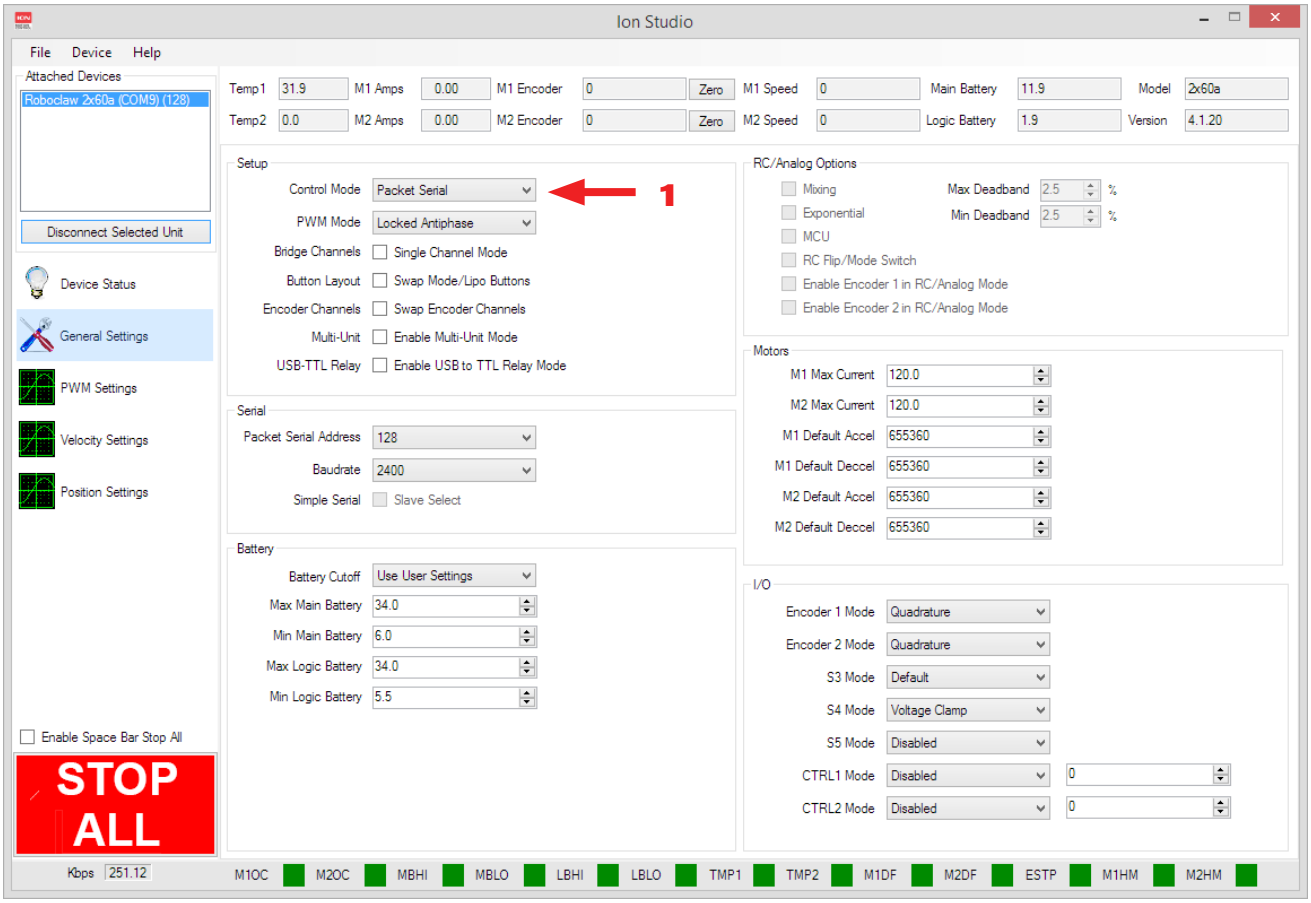
4. When RoboClaw is detected, it will appear in the Attached Device window (1).

5. Once RoboClaw appears in the Attached Device window (1), click the connect button (2).



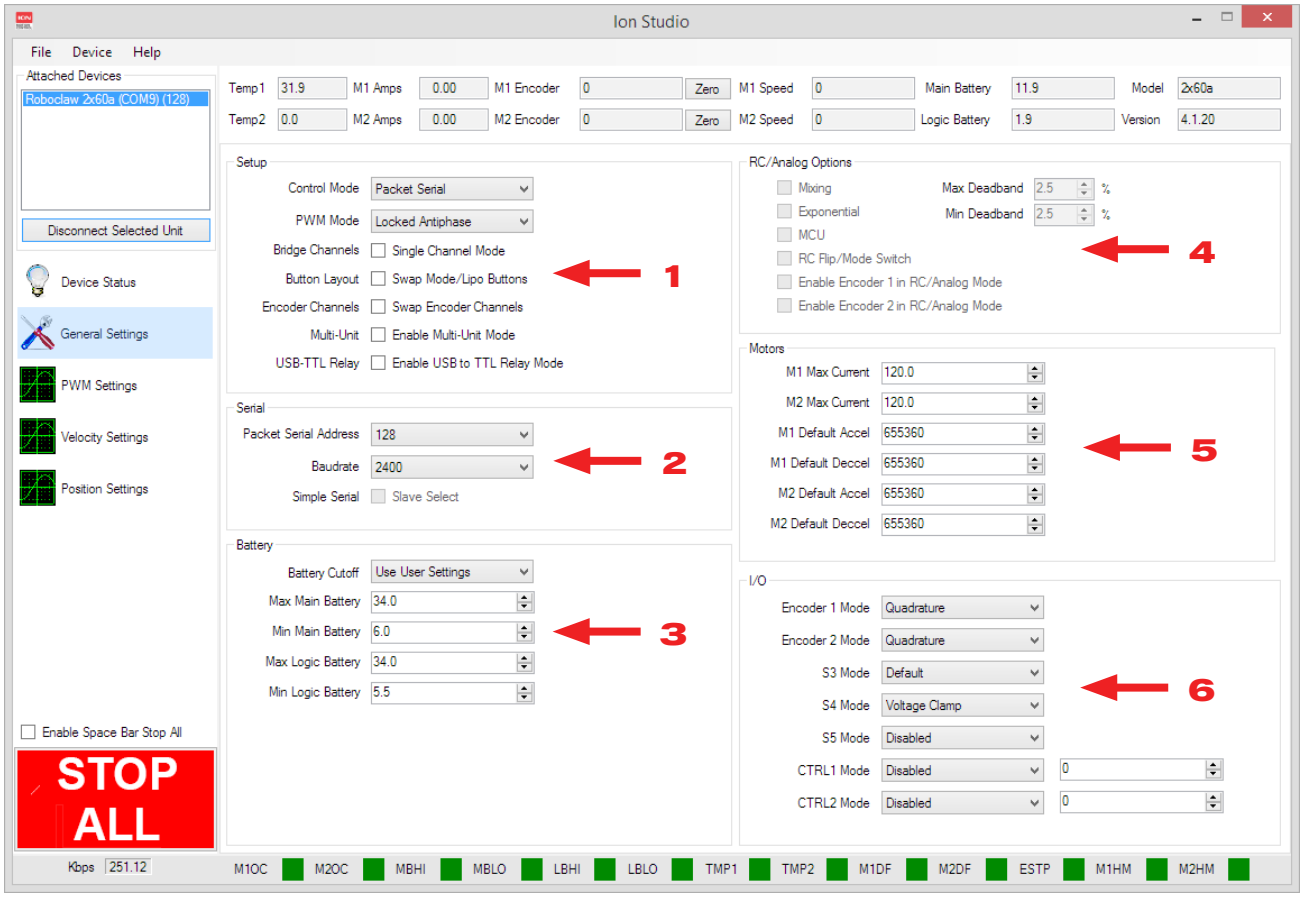
Control Mode Setup

Select the Control Mode drop down (1). There are 4 main modes. See the Control Modes section of this manual for a detailed explanation of each available mode.



Control Mode Options

The general settings screen is used to configure RoboClaw. Each control mode will have several configuration options. Grayed out options are not available for the selected mode. Once all settings are configured they must be saved to RoboClaw. This is done by selecting Save Settings from the File menu in the menu bar.



(1) Setup

Main drop down for setting the control modes and confirguration options.

Function	Description
Control Mode	Drop down to set main control mode. Some options may grey out if not available in the selected mode.
PWM Mode	Drop down to set the main MOSFET driving scheme. This option should never be change but in rare circumstances.
Bridge Channels	Used to bridge motor channe 1 and 2. This option must be set before physically bridging the channels. Or damage will result.
Button Layout	Swaps Mode and LIPO button interface. Only affects hardware V5 and RoboClaw 2x15, 2x30 and 2x45.
Encoder Channels	This option will swap encoder channels. Pair encoder 1 to motor channel 2 and encoder 2 to motor channel 1.
Mulit-Unit	Sets S2 pin to open drain. Allows multiple Roboclaws to be controlled from a single serial port.
USB-TTL Relay	Enables RoboClaw to pass data from USB through S1 (RX) and S2 (TX). Allows several RoboClaws to be networked from one USB connection. All connected RoboClaw's baud rates must be set to the same.

(2) Serial

Settings for serial modes. Set packet address, baudrate and slave select.

Function	Description
Packet Serial Address	Sets RoboClaw address for packet serial mode. Allows multiple Roboclaws to be controlled from a single Serial port.
Baudrate	Sets the baudrate in all serial modes.
Simple Serial	Sets simple serial mode with slave select. Set pin S2 high to enable the attached RoboClaw. Pull S2 low and all commands will be ignored.

(3) Battery

Main and logic battery voltage settings. Sets cut off and protection limits.

Function	Description
Battery Cut Off	Sets main battery cut off based on LiPo cell count. Can also be set to auto detect or User Settings for manual configuration. Auto detect requires a properly charged battery. User Settings allows editing of the voltage values manually. See Battery Settings.
Max Main Battery	Sets main battery maximum voltage. If the main battery voltage goes above the set maximum value running motors will go into brake mode.
Min Main Battery	Sets main battery minimum voltage. If the main battery voltage falls below the set minimum value running motors will go into freewheel.
Max Logic Battery	Sets logic battery maximum voltage. If logic battery voltage goes above the maximum set value RoboClaw will shut down until the voltage is corrected and a reset.
Min Logic Battery	Sets logic battery minimum voltage. If logic battery voltage goes below the minimum set value RoboClaw will shut down until the voltage is corrected and a reset.

(4) RC/Analog Options

Configure RC and Ananlog control options. Set control type in RC and Analog modes.

Function	Description
Mixing	Mixes S1 and S2 inputs for control of a differentially steered robot. S1 controls direction (forward / reverse) and speed. S2 controls turning left or right with speed. Simliar to how a RC car would be controlled. Turn this mode off for tank style control.
Exponential	Enable increased control range at slow speed.
MCU	Disables auto calibrate. Allows slow MCU to send R/C pulses at lower than normal R/C rates.
RC Flip/Mode Switch	R/C pulse switched. Use radio channel to toggle and change all motor direction. Used when a robot is flipped upside down to reverse steering control.
Enable Encoder 1 in RC/ AnalogMode	Enables encoder 1 to be used in RC or Analog mode. Will control motor by speed or position depending on which PID control is set. The range of speed is mapped to the RC control using the QPPS value as the maximum speed. The position range is controlled by maximum and minimum position settings.
Enable Encoder 2 in RC/ AnalogMode	Enables encoder 1 to be used in RC or Analog mode. Will control motor by speed or position depending on which PID control is set. The range of speed is mapped to the RC control using the QPPS value as the maximum speed. The position range is controlled by maximum and minimum position settings.
Max Deadband	Sets maximum range of control signal seen as 0 (Stopped).
Min Deadband	Sets minimum range of control signal seen as 0 (Stopped).

(5) Motors

Motor current limit settings. The accel and deccel settings apply to RC, Analog and commands with no Accel and Deccel arugements.

Function	Description
M1 Max Current	Sets maximum motor current for channel 1. Can not exceed RoboClaw rated peak current.
M2 Max Current	Sets maximum motor current for channel 2. Can not exceed RoboClaw rated peak current.
M1 Default Accel	Sets the ramp rate of acceleration for motor channel 1. A value of 1 to 655,360 can be used. Value of 0 sets the internal default for Accel and Deccel. Value of 655,360 equals 100mS full forward to reverse ramping rate.
M1 Default Deccel	Sets the ramp rate of decceleration for motor channel 1. A value of 1 to 655,360 can be used. Value of 0 sets the internal default for Accel and Deccel. Value of 655,360 equals 100mS full forward to reverse ramping rate.
M2 Default Accel	Sets the ramp rate of acceleration for motor channel 2. A value of 1 to 655,360 can be used. Value of 0 sets the internal default for Accel and Deccel. Value of 655,360 equals 100mS full forward to reverse ramping rate.
M2 Default Deccel	Sets the ramp rate of decceleration for motor channel 2. A value of 1 to 655,360 can be used. Value of 0 sets the internal default for Accel and Deccel. Value of 655,360 equals 100mS full forward to reverse ramping rate.

(6) I/O

Set encoder input type. Set S3, S4 and S5 configuration options. Enabling output pins on certain models of RoboClaw. Set limit, homing, voltage clamp, E-stop options.

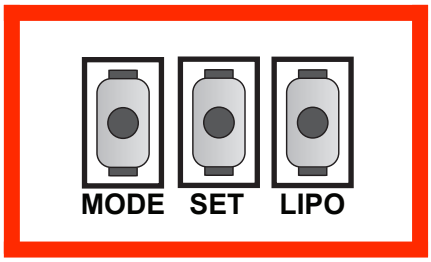
Function	Description
Encoder 1 Mode	Sets encoder type for encoder 1.
Encoder 2 Mode	Sets encoder type for encoder 2.
S3 Mode	Sets the default function for S3.
S4 Mode	Sets the default function for S4.
S5 Mode	Sets the default function for S5.
CTRL1 Mode	Enables output pins on certain models of RoboClaw. A value of 0 to 65535 can be used to set the pin's default PWM output. Value can be changed by commands during run time.
CTRL2 Mode	Enables output pins on certain models of RoboClaw. A value of 0 to 65535 can be used to set the pin's default PWM output. Value can be changed by commands during run time.

Configuration with Buttons

Mode Setup

The 3 buttons on RoboClaw are used to set the different 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 the mode. The LIPO button doubles as a save button and configuring the low battery voltage cut out function of RoboClaw. 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	Function	Description
1	R/C mode	Control with standard R/C pulses from a R/C radio or MCU. Controls a robot like a tank. S1 controls motor 1 forward or reverse and S2 controls motor 2 forward or reverse.
2	R/C mode with mixing	Same as Mode 1 with mixing enabled. Channels are mixed for differentially steered robots (R/C Car). S1 controls forward or reverse and S2 controls left or right.
3	Analog mode	Control using analog voltage from 0V to 2V. S1 controls motor 1 and S2 controls motor 2.
4	Analog mode with mixing	Same as Mode 3 with mixing enabled. Channels are mixed for differentially steered robots (R/C Car). S1 controls forward or reverse and S2 controls left or right.
5	Standard Serial	Use standard serial communications for control.
6	Standard Serial with slave pin	Same as Mode 5 with a select pin. Used for networking. RoboClaw will ignore commands until pin goes high.
7	Packet Serial Mode - Address 0x80	Control using packet serial mode with a specific address for networking several motor controllers together.
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	

Mode Options

Each mode will have several possible configuration settings. The settings need to be setup after the initial mode is selected. Follow the steps below.

1. 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 setting.
2. Press SET to increment to the next option. Press MODE to decrement to the previous option.
3. Once the desired option is selected press and release the LIPO button to save the option to memory.

RC and Analog Mode Options

Option	Function	Description
1	TTL Flip Switch	Logic level switch. Toggle to change all motor direction. Used when a robot is flipped upside down to reverse steering control.
2	TTL Flip and Exponential Enabled	Option 1 combined with increased control range at slow speed.
3	TTL Flip and MCU Enabled	Disables auto calibrate. Allows slow MCU to send R/C pulses at lower than normal R/C rates.
4	TTL Flip and Exp and MCU Enabled	Option 2 and 3 combined.
5	RC Flip Switch	R/C pulse switched. Use radio channel to toggle and change all motor direction. Used when a robot is flipped upside down to reverse steering control.
6	RC Flip and Exponential Enabled	Option 5 combined with increased control range at slow speed.
7	RC Flip and MCU Enabled	Disables auto calibrate and auto stop due to R/C signal loss. Allows slow MCU to send R/C pulses at lower than normal R/C rates.
8	RC Flip and Exponential and MCU Enabled	Option 6 and 7 combined.

Standard Serial and Packet Serial Mode Options

Option	Baud Rate	Description
1	2400bps	Standard RS-232 serial data rate.
2	9600bps	Standard RS-232 serial data rate.
3	19200bps	Standard RS-232 serial data rate.
4	38400bps	Standard RS-232 serial data rate.
5	57600bps	Standard RS-232 serial data rate.
6	115200bps	Standard RS-232 serial data rate.
7	230400bps	Standard RS-232 serial data rate.
8	460800bps	Standard RS-232 serial data rate.

Battery Cut Off Settings

The RoboClaw is able to protect the main battery by utilizing a battery voltage cut off. The cut off voltage will vary depending on the size of battery used. The table below shows the battery option setting with the type of battery it will protect and at what voltage the cutoff will kick in. The battery settings can be set by following the steps below.

1. Press and release the LIPO button. The STAT2 LED will begin to blink out the current setting.
2. Press SET to increment to the next setting. Press MODE to decrement to the previous setting.
3. Once the desired setting is selected press and release the LIPO button to save this setting to memory.

Battery Options

Option	Setting	Description
1	Disabled	6VDC is the default cut off when disabled.
2	Auto Detect	Battery must not be overcharged or undercharge! See Battery Settings.
3	3 Cell	9VDC is the cut off voltage.
4	4 Cell	12VDC is the cut off voltage.
5	5 Cell	15VDC is the cut off voltage.
6	6 Cell	18VDC is the cut off voltage.
7	7 Cell	21VDC is the cut off voltage.
8	8 Cell	24VDC is the cut off voltage.

Battery Settings

Automatic Battery Detection on Startup

Auto detect will sample the main battery voltage on power up or after a reset. All Lipo batteries, depending on cell count will have a minimum and maximum safe voltage range. The attached battery must be within this acceptable voltage range to be correctly detected. Undercharged or overcharged batteries will cause false readings and RoboClaw will not properly protect the battery. If the automatic battery detection mode is enabled using the on-board buttons, the Stat2 LED will blink to indicate the battery cell count that was detected. Each blink indicates the number of LIPO cells detected. When automatic battery detection is used the number of cells detected should be confirmed on power up.

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

Manual Voltage Settings

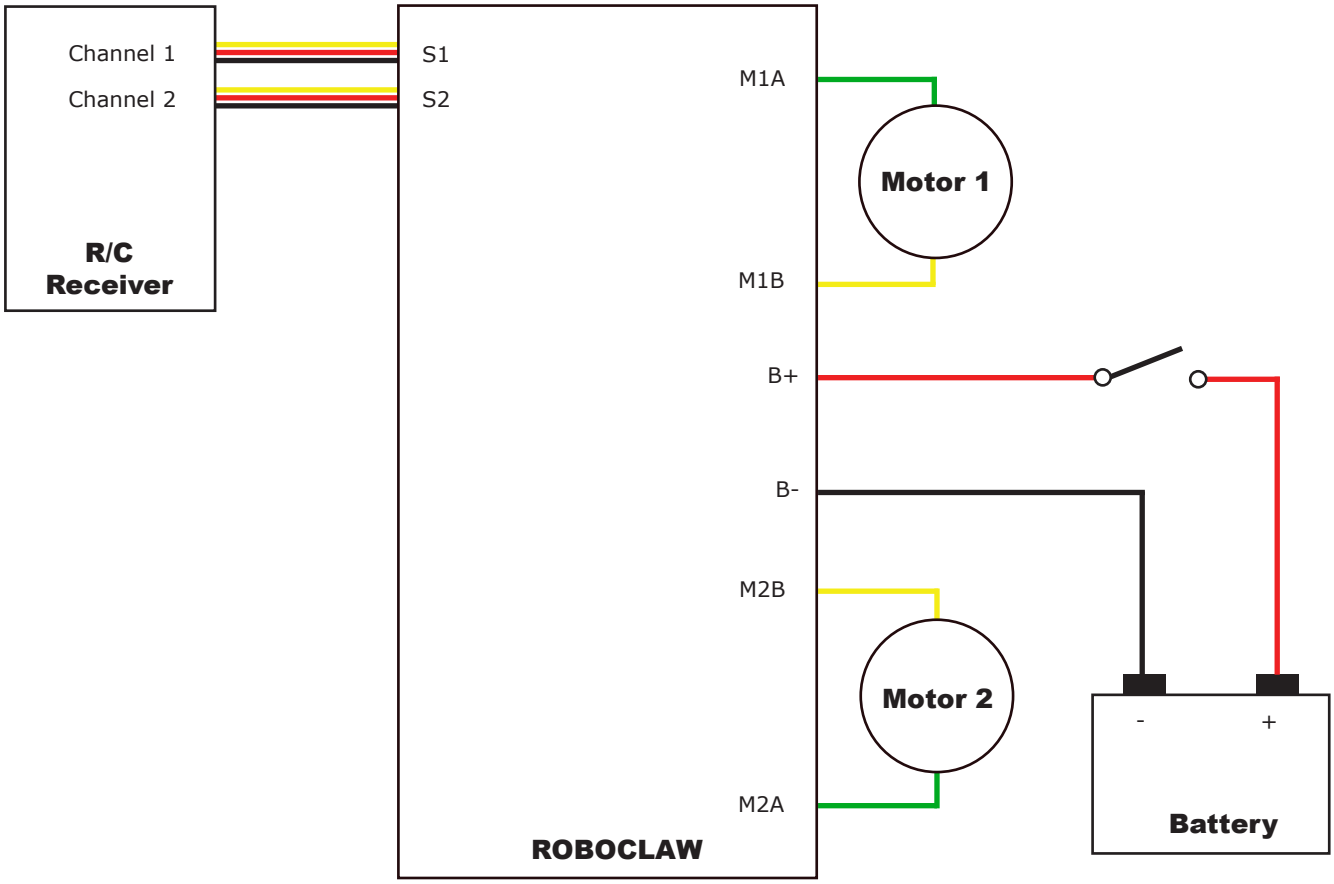
The minimum and maximum voltage can be set using the Ion Studio application or packet serial commands. Values can be set to any value between the boards minimum and maximum voltage limits. This feature can be useful when using a power supply to power RoboClaw. A minimum voltage just below the power supply voltage of 2VDC will prevent the power supply voltage from dipping too low under heavy load. A maximum voltage set to just above the power supply voltage 2VDC will help protect the power supply from regenerative voltage spikes if an external voltage clamp circuit is not being used. However when the minimum or maximum voltages are reached RoboClaw will go into either braking or freewheel mode. This feature will only help to protect a power supply not correct regenerative voltages issues. A voltage clamping circuit is required to correct any regenerative voltage issues when a power supply is used as the main power source. See Voltage Clamping.

Wiring

Basic Wiring

The MCP has many control modes and each mode may have unique wiring requirments to ensure safe and reliable operation. The diagram below illustrates a very basic wiring configuration used in a small motor system where safety concerns are minimal. This is the most basic wiring configuration possible. Any wiring of RoboClaw should include a main battery shut off switch, even when safety concerns are minimal. Never underestimate a motorized system in an uncontrolled condition.

In addition, RoboClaw is a regenerative motor controller. If the motors are moved when the system is off, it could cause potential erratic behavior due to the regenerative voltages powering the system. A return path to the battery should always be supplied if the system can move when main power is disconnected or a fuse is blown.



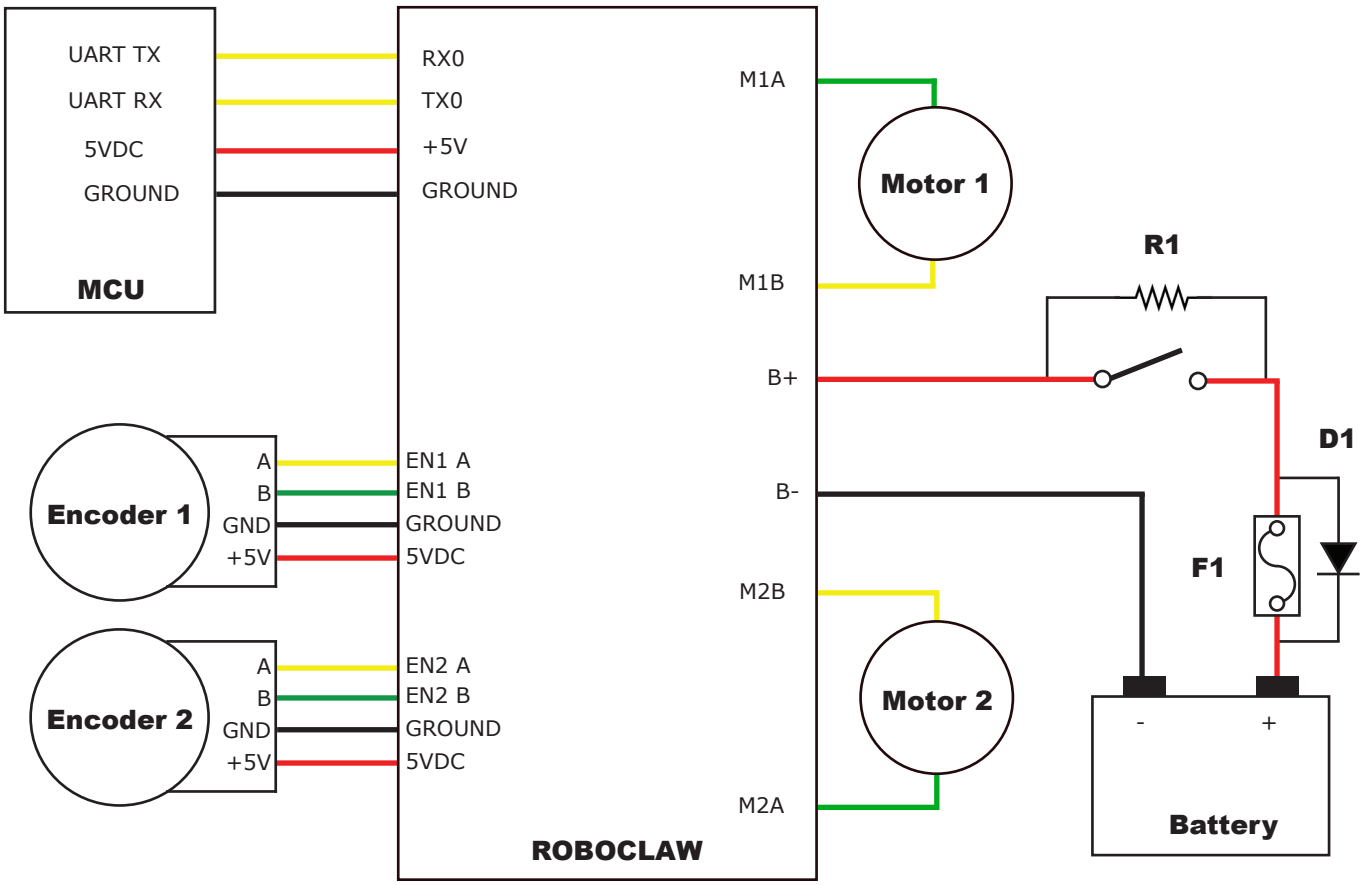
Never disconnect the negative battery lead before disconnecting the positive!

Safety Wiring

In all system with movement, safety is a concern. The wiring diagram below illustrates a properly wired system with several safety features. An external main power cut off is required for safety. When the RoboClaw is switched off or the fuse is blown, a high current diode (D1) is required to create a return path to the battery for any regenerative voltages. The use of a pre-charge resistor (R1) is required to avoid high inrush currents and arcing. A pre-charge resistor (R1) should be 1K, 1/2Watt for a 60VDC motor controller which will give a pre-charge time of about 15 seconds. A lower resistances can be used with lower voltages to decrease the pre-charge time.

Encoder Wiring

A wide range of sensors are supported including quadrature encoders, absolute encoders, potentiometers and hall effect sensors for closed loop operation. The encoder pins are not exclusive to supporting encoders and have several functions available. See Encoder section of this manual for additional information.

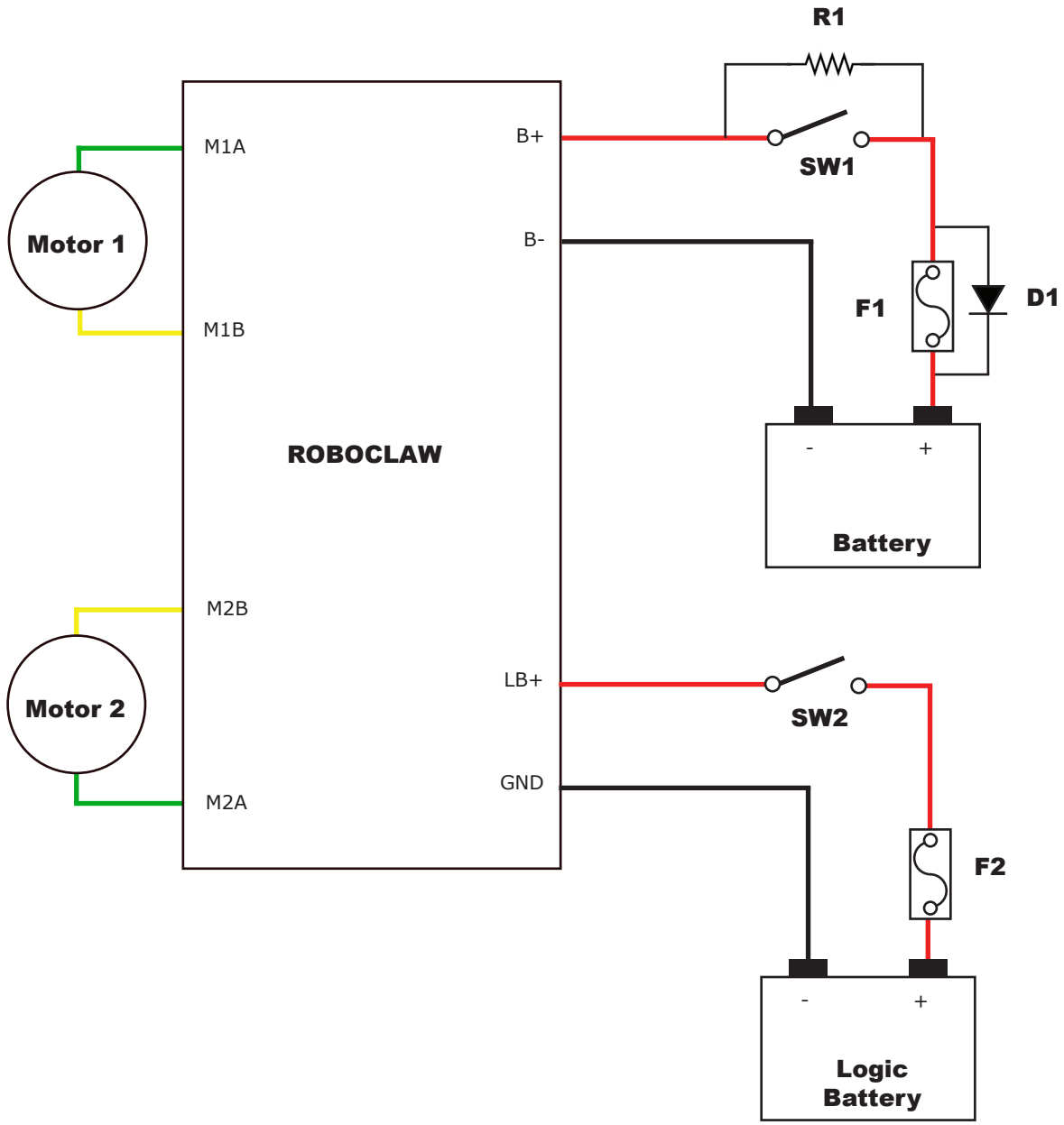


Logic Battery Wiring

An optional logic battery is supported. Under heavy loads the main power can suffer voltage drops, causing potential logic brown outs which may result in uncontrolled behavior. A separate power source for the motor controllers logic circuits, can remedy potential problems from main power voltage drops. The logic battery maximum input voltage is 34VDC with a minimum input voltage of 6VDC. The 5V regulated user output is supplied by the secondary logic battery if supplied. The mAh of the logic battery should be determined based on the load of attached devices powered by the regulated 5V user output.

Logic Battery Jumper

A logic battery is used in the configurtion below. Some models of RoboClaw have a jumper to set the logic battery. On models where the LB-MB header is present the jumper must be removed when a logic battery is used. If the header for LB-MB is not present, then the RoboClaw will automatically set the logic battery power source.

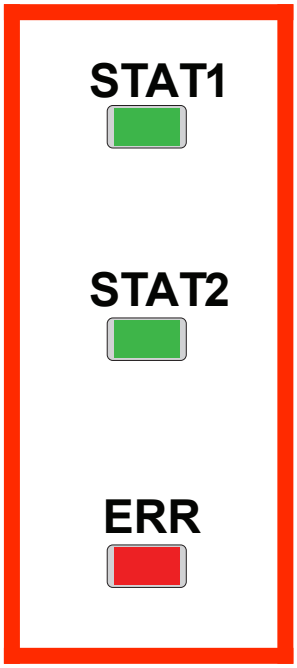


Status LEDs

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

There are 3 types of message RoboClaw can indicate. The first type is a fault. When a fault occurs, both motor channel outputs will be disabled and RoboClaw will stop any further actions until the unit is reset, or in the case of non-latching E-Stops, the fault state is cleared. The second message type is a warning. When a warnings occurs both motor channel outputs will be controlled automatically depending on the warning condition. As an example if an over temperature of 85c is reach RoboClaw will reduce the maximum allowed current until a safe temperature is reached. The final message type is a notice. Currently there is only one notice indicated.

LED Blink Sequences

When a warning or fault occurs RoboClaw will use the LEDs to blink a sequence. The below table details each sequence and the cause.

LED Status	Condition	Type	Description
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 blinks once with short delay. Other LEDs off.	Over 100c Temperature	Fault	Motors freewheel while condition exist.
Error LED lit while condition is active.	Over Current	Warning	Motor power is automatically limited.
Error LED blinking twice. STAT1 or STAT2 indicates channel.	Driver Fault	Fault	Motors freewheel. Damage detected.
Error LED blinking three times.	Logic Battery High	Fault	Motors freewheel until reset.
Error LED blinking four times.	Logic Battery Low	Fault	Motors freewheel until reset.
Error LED blinking five times.	Main Battery High	Fault	Motors are stopped by braking until reset.
Error LED lit while condition is active.	Main Battery High	Warning	Motors are stopped by braking 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 IonMotion PC setup utility to clear.

Inputs

S3, S4 and S5 Setup

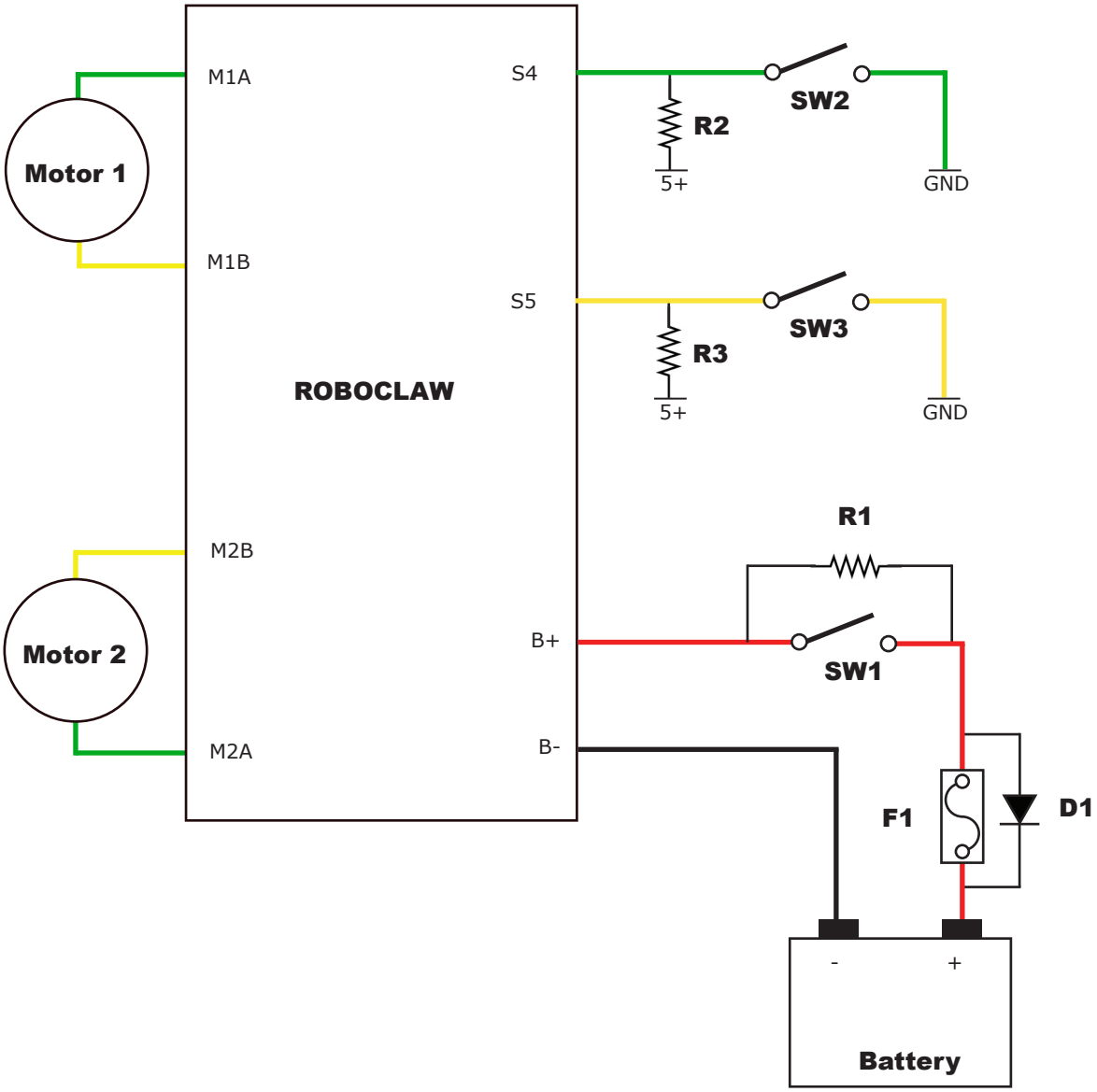
RoboClaw S3, S4 and S5 inputs support the use of home switches, limit switches, Voltage Clamping and E-Stops. A limit switch is used to detect the travel limits. Travel limits are typically used on a linear slide to detect when the assembly has reached the end of travel. A home switch is used to create a known start position. In some situations both may be required.

Open Ion Studio and select the S3, S4 or S5 options drop down. There are several options to choose from. Each option is explained below. After setting S3, S4 and S5 options save the settings before exiting Ion Studio.

Option	Description
Disable	Disables S4 and S5. Set by default.
E-Stop(Latching)	All stop until RoboClaw is reset.
E-Stop	All stop until switch released.
Voltage Clamp	Used to control a voltage clamp circuit. Dumps the regenerative voltages. For use with power supplies.
Motor Home(Auto)	Moves motor in reverse rotation until switch tripped.
Motor Home(User)	User controls direction of motor to reach switch.
Limit(Forward)	Motor moves forward rotation until switch tripped.
Limit(Reverse)	Motor moves in reverse rotation until switch tripped.

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.



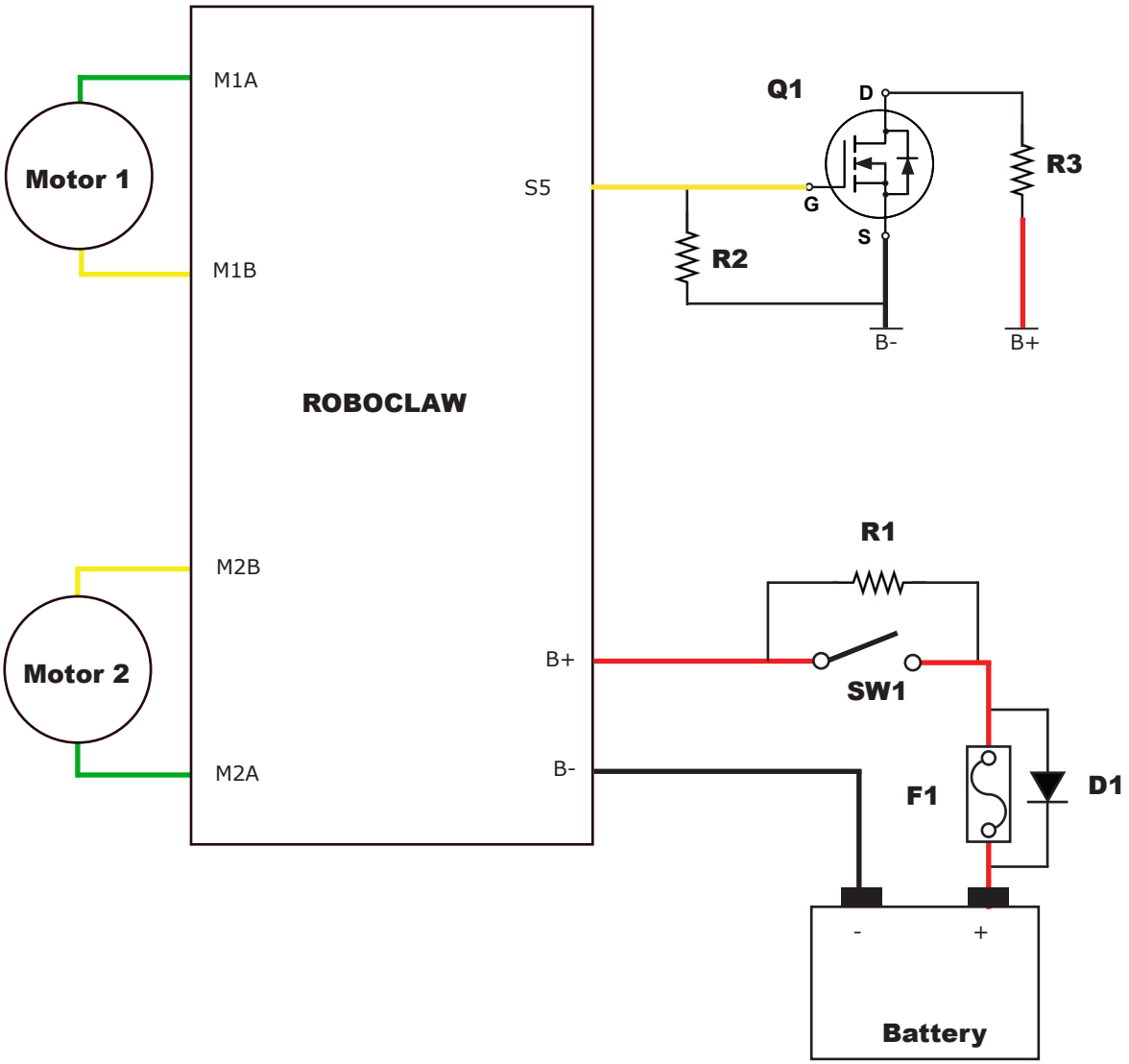
Regenerative Voltage Clamping

Voltage Clamp

When using power supplies regenerative voltage spikes will need to be dissipated. This can be done with a simple circuit shown below and using a V-Clamp pin to activate it. A solid state switch (Q1) and large wattage resistor (R3) are required. The regenerative voltage will be dissipated as heat. An example of a large resistor would be in the range of 10 Ohms at 50 watts for small motors and down to 1 Ohms or lower for larger motors. 50 watts will likely cover most situation smaller wattage resistor may work.

Voltage Clamp Circuit

Wire the circuit as shown below. Q1 should be a 3V logic level MOSFET. An example would be FQP30N06L. Which is rated to a maximum voltage of 60VDC. You may need to change the MOSFET used based on the application. R2 (10K) will keep the MOSFET off when not in use. R3 will need to be adjusted based on the initial test results. Start with a 10 Ohms 50 Watt. If after testing the voltage spike is still too great reduce the resistor to a 5 Ohms and so on.



Voltage Clamp Setup and Testing

Open Ion Studio and set S5 to the Voltage Clamp option in the drop down and save the setting before exiting the application (see user manual).

The circuit shown will need to be tuned for each application to properly dissipate the regenerative voltages. Testing should consist of a running the motor up to 25% of its speed and then quickly slowing down without braking or e-stop while checking the voltage spike. Repeat, by increasing the speed and power by 5% and checking the voltage spikes again. Repeat this process until 100% power is achieved without a major spike or until the voltage clamp is not dissipating the voltage spikes. If over voltages are not completely clamped, either a lower Ohm resistor is required or additional capacitance is required. Add a capacitor of 5000uF to 10000uF or more across B+ and B-.

Bridge Mode

Bridging Channels

RoboClaws dual channels can be bridge to run as one channel, effectively doubling its current capability for one motor. RoboClaw will be damaged if it is not set to bridged channel mode before wiring.

Download and install Ion Studio application. Connect the motor controller to the computer using an available USB port. Run Ion Studio and in general settings check the option to "Bridge Channels". Then click “Save Settings” in the device menu at the top of the window.

When operating in bridged channel mode the total peak current output is combined from both channels. The peak current run time is dependant on heat build up. Adequate cooling must be maintained.

Bridged Channel Wiring

When bridged channel mode is active the internal driver scheme for the output stage is modified. The output leads must be wired correctly or damage will result. Each RoboClaw varies on the correct wiring. See each models data sheet for wiring schematic.

Bridged Motor Control

When RoboClaw is set to bridged mode all motor control commands for M1 will control the attached motor. All commands for M2 will be ignored. In RC and Analog modes S1 will control the motor and S2 will be ignored.

USB Control

USB Connection

When RoboClaw is connected, it will automatically detect it has been connected to a powered USB master and will enable USB communications. USB can be connected in any mode. When the Roboclaw is not in packet serial mode USB packet serial commands can be used to read status information and set configuration settings, however motor movement commands will not function. When in packet serial mode if another device such as an Arduino is connected to S1 and S2 pins and sending commands to the RoboClaw, both those commands and USB packet serial commands will execute.

USB Power

The USB RoboClaw is self powered. This means it receives no power from the USB cable. The USB RoboClaw must be externally powered to function.

USB Comport and Baudrate

The 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 from our website. On Linux or OSX the RoboClaw will be automatically detected as a virtual comport and an appropriate driver will be automatically 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 around 1mb/s.

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 can be used with encoders. Velocity and/or Position PID constants must be calibrated for proper operation first. Once calibrated values have been set and saved into Roboclaws eeprom memory, encoder support using velocity or position PID control can be enabled. Use Ion Studio control software or Packet Serial commands, enable encoders for RC/Analog modes (See Configuration Using Ion Studio).

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.

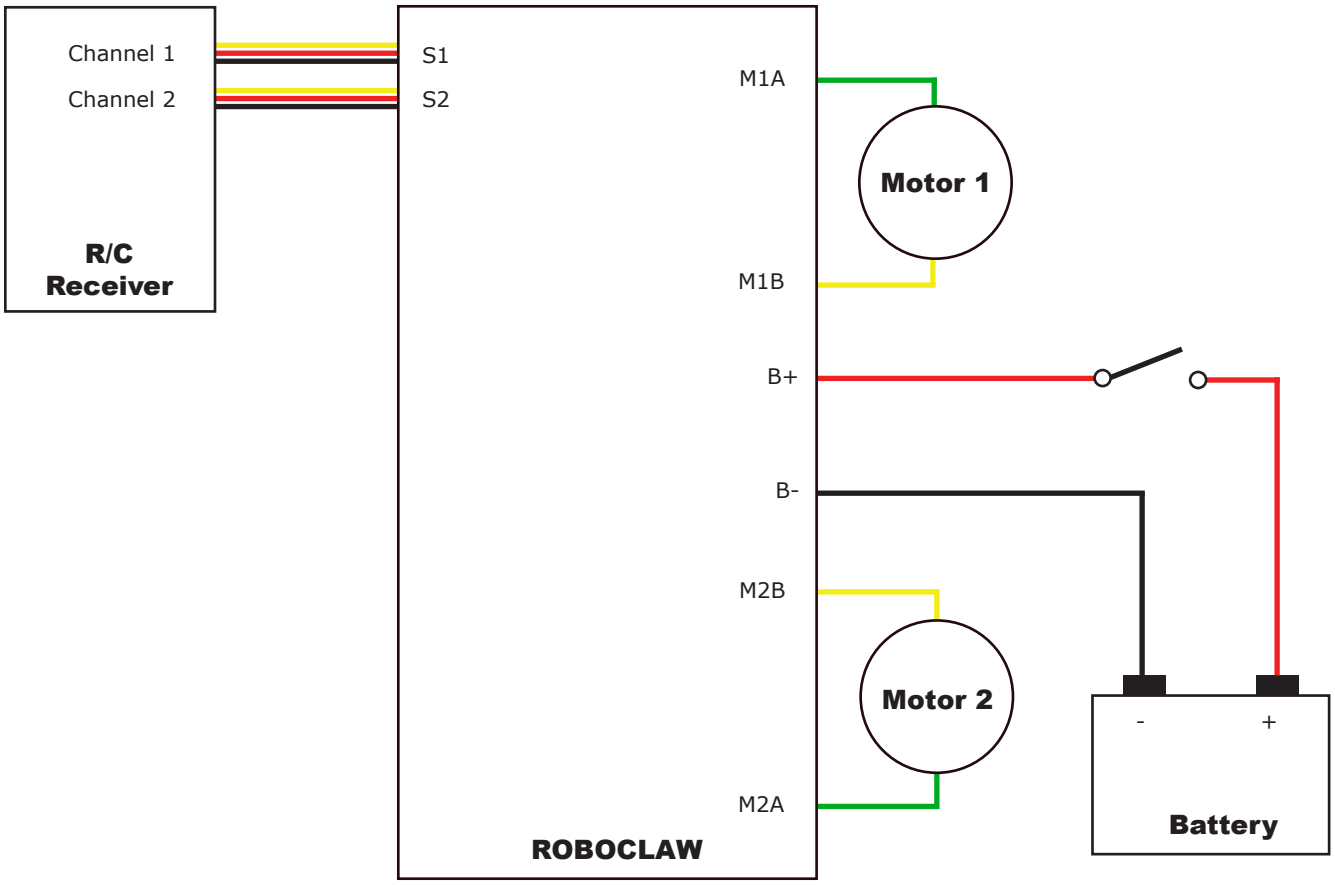
Pulse Ranges

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 the RoboClaw as shown below. Set mode 1 with option 1. Before powering up, center the control sticks on the radio transmitter, turn the radio on first, then the receiver, then RoboClaw. It will take RoboClaw about 1 second to calibrate the neutral positions of the RC controller. After RC pulses start to be received and calibration is complete the Stat1 LED will begin to flash indicating signals from the RC receiver are being received.



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.

```
//RoboClaw RC Mode
//Control RoboClaw with servo pulses from a microcontroller.
//Mode settings: Mode 2(RC mixed mode) with Option 4(MCU with Exponential).

#include <Servo.h>

#define MIN 1250
#define MAX 1750
#define STOP 1500

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

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

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

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

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

  myservo2.writeMicroseconds(MIN);  //full turn left
  delay(1000);

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

  myservo2.writeMicroseconds(MAX);  //full turn right
  delay(1000);
}
```

Analog Control

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. The voltage range is 0V = Full reverse, 1V = Stop and 2V = Full forward.

Analog Mode With Mixing

This mode is the same as 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 forward and backward and S2 to control steering left and right.

Analog Mode with feedback for velocity or position control

Analog Mode can be used with encoders. Velocity and/or Position PID constants must be calibrated for proper operation. Once calibrated values have been set and saved into Roboclaws eeprom, encoder support using velocity or position PID control can be enabled. Use Ion Studio control software or PacketSerial commands to enable encoders for RC/Analog modes (see Configuration Using Ion Studio).

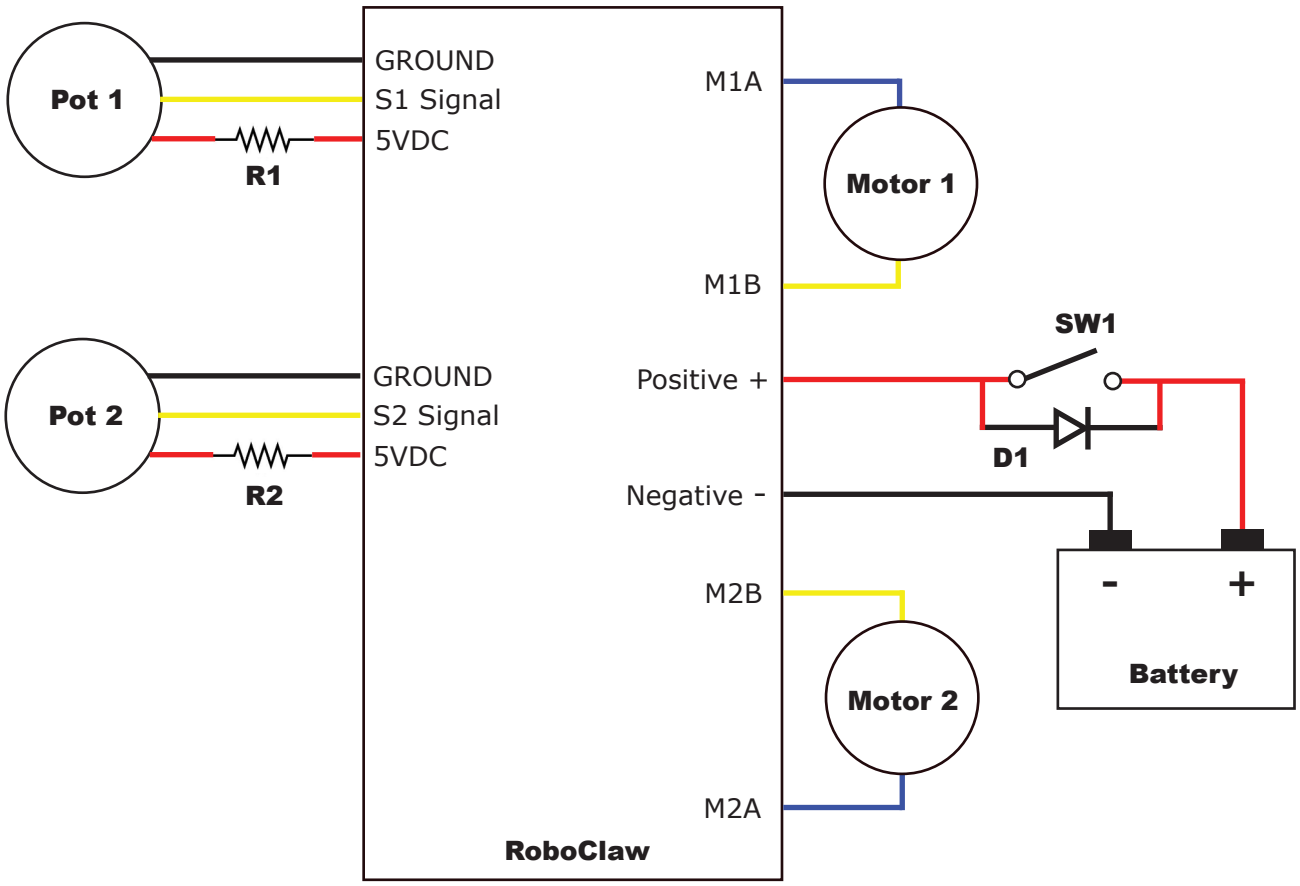
Analog Mode Options

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

RoboClaw uses a high speed 12 bit analog converter. Its range is 0 to 2V. The analog pins are protected and 5V tolerant. The potentiometer range will be limited if 5V is utilized as the reference voltage. A simple resistor divider circuit can be used to reduce the on board 5V to 2V for use with a potentiometer(POT). See the below schematic. The POT acts as one half of the resistor divider. If using a 5k potentiometer R1 / R2 = 7.5k, If using a 10k potentiometer R1 / R2 = 15k and if using a 20k potentiometer R1 / R2 = 30k.

Set mode 3 with option 1. Center the potentiometers before applying power. The S1 potentiometer will control the motor 1 direction and speed. The S2 potentiometer will control the motor 2 direction and speed.



Stand Serial Control

Standard Serial Mode

In this mode S1 accepts TTL level byte commands. Standard 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 (eg: Max232) is required. The baud rate can be changed using the SET button once a serial mode has been selected.

Standard Serial communications has no error correction. It is recommended to use Packet Serial mode instead for more reliable communications.

Serial Mode Baud Rates

Option	Description
1	2400
2	9600
3	19200
4	38400
5	57600
6	115200
7	230400
8	460800

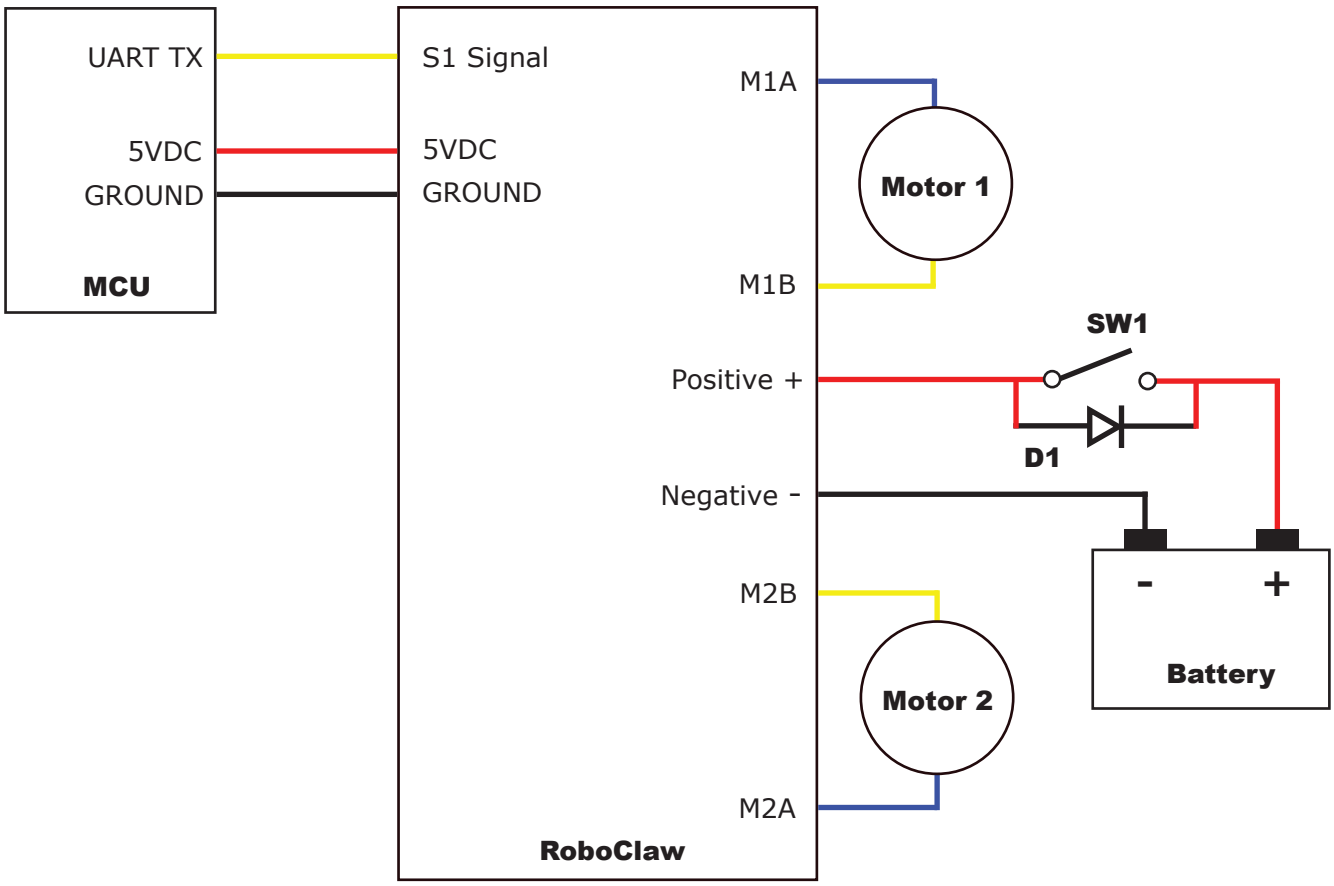
Standard Serial Command Syntax

The RoboClaw standard serial is setup to control both motors with one byte sized command character. Since a byte can be any value from 0 to 255(or -128 to 127) the control of each motor is split. 1 to 127 controls channel 1 and 128 to 255(or -1 to -127) controls channel 2. Command value 0 will stop 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

Standard Serial Wiring Example

In standard serial mode the RoboClaw can only receive serial data. The below wiring diagram illustrates a basic setup of RoboClaw for use with standard serial. The diagram below shows the main battery as the only power source. Make sure the LB jumper is set correctly. The 5VDC shown connected is only required if your MCU needs a power source. This is the BEC feature of RoboClaw. If the MCU has its own power source do not the 5VDC.

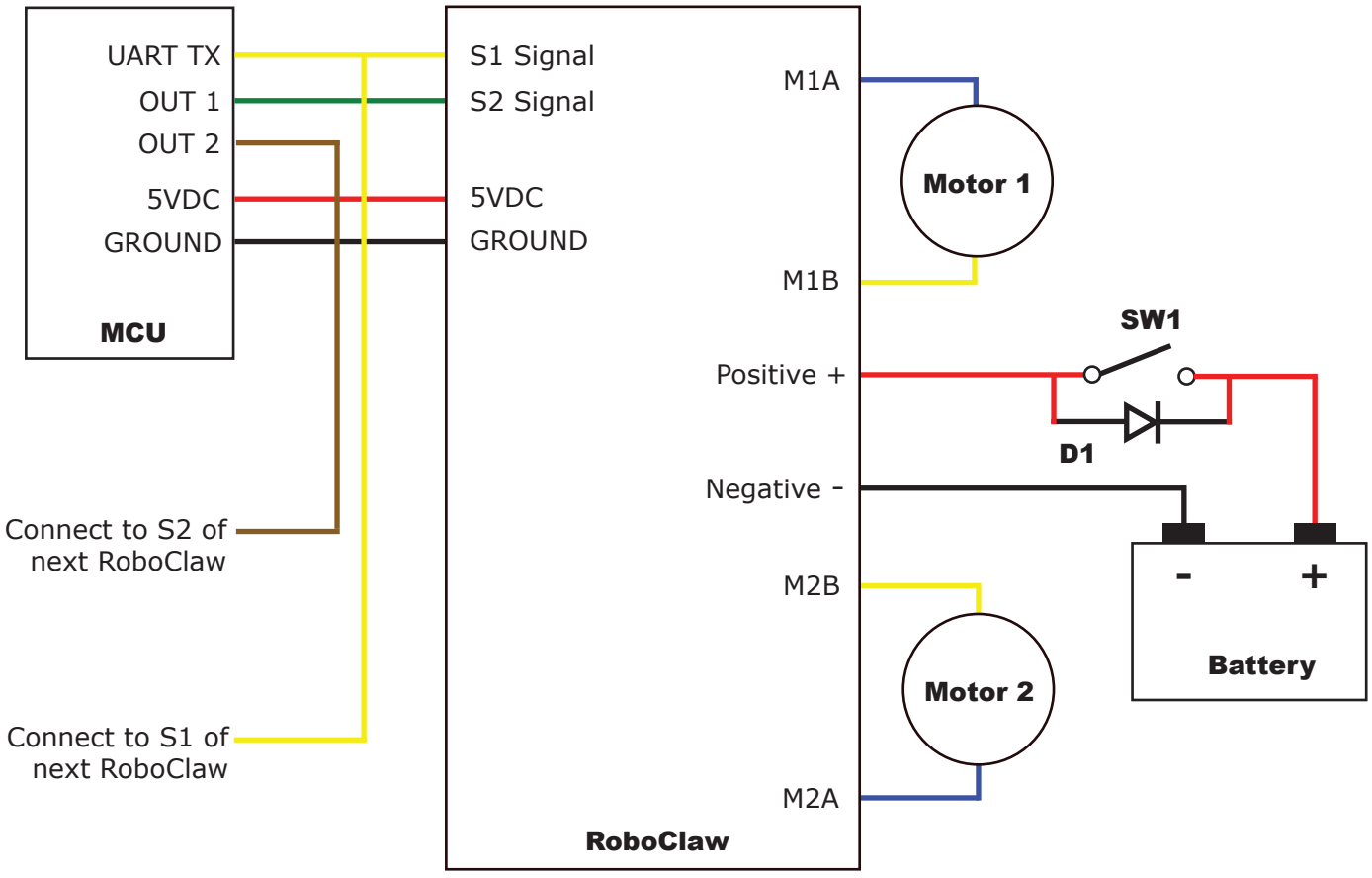


Standard Serial Mode With Slave Select

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 diagram below shows the main battery as the only power source. Make sure the LB jumper is set correctly. The 5VDC shown connected is only required if your MCU needs a power source. This is the BEC feature of RoboClaw. If the MCU has its own power source do not connect the 5VDC.



Standard Serial - Arduino Example

The following example will start both channels in reverse, stop, forward, stop, turn left, stop turn right stop. The program was written and tested with a Arduino Uno and Pin 11 connected to S1 and pin 10 connected to S2.

```
//Roboclaw simple serial example.  Set mode to 5.  Option to 4(38400 bps)
#include "BMSerial.h"

BMSerial mySerial(10,11);

void setup() {
  mySerial.begin(38400);
}

void loop() {
  mySerial.write(1);
  mySerial.write(-127);
  delay(2000);
  mySerial.write(64);
  delay(1000);
  mySerial.write(127);
  mySerial.write(-1);
  delay(2000);
  mySerial.write(-64);
  delay(1000);
  mySerial.write(1);
  mySerial.write(-1);
  delay(2000);
  mySerial.write(0);
  delay(1000);
  mySerial.write(127);
  mySerial.write(-127);
  delay(2000);
  mySerial.write(0);
  delay(1000);
}
```

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 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.

Address

Packet serial requires a unique address when used with TTL serial pins(S1 and S2). With up to 8 addresses available you can have up to 8 RoboClaws bussed on the same RS232 port when properly wired. 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.

NOTE: When using packet serial commands via the USB connection the address byte can be any value from 0x80 to 0x87 since each USB connection is already unique.

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
5	57600
6	115200
7	230400
8	460800

Packet Timeout

When sending a packet to RoboClaw, if there is a delay longer than 10ms between bytes being received in a packet, RoboClaw will discard the entire packet. This will allow the packet buffer to be cleared by simply adding a minimum 10ms delay before sending a new packet command in the case of a communications error. This can usually be accomodated by having a 10ms timeout when waiting for a reply from the RoboClaw. If the reply times out the packet buffer will have been cleared automatically.

Packet Acknowledgement

RoboClaw will send an acknowledgment byte on write only packet commands that are valid. The value sent back is 0xFF. If the packet was not valid for any reason no acknowledgement will be sent back.

CRC16 Checksum Calculation

Roboclaw uses a CRC(Cyclic Redundancy Check) to validate each packet it receives. This is more complex than a simple checksum but prevents errors that could otherwise cause unexpected actions to execute on the Roboclaw.

The CRC can be calculated using the following code(example in C):

```
//Calculates CRC16 of nBytes of data in byte array message
unsigned int crc16(unsigned char *packet, int nBytes) {
    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;
}
```

CRC16 Checksum Calculation for Received data

The CRC16 calculation can also be used to validate received data from the Roboclaw. The CRC16 value should be calculated using the sent Address and Command byte as well as all the data received back from the Roboclaw except the two CRC16 bytes. The value calculated will match the CRC16 sent by the Roboclaw if there are no errors in the data sent or received.

Easy to use Libraries

Source code and Libraries are available on the Ion Motion Control website that already handle the complexities of using packet serial with the Roboclaw. Libraries are available for Arduino(C++), C# on Windows(.NET) or Linux(Mono) and Python(Raspberry Pi, Linux, OSX, etc).

Handling values larger than a byte

Many Packet Serial commands require values larger than a byte can hold. In order to send or receive those values they need to be broken up into 2 or more bytes. There are two ways this can be done, high byte first or low byte first. Roboclaw expects the high byte first. All command arguments and values are either single bytes, words (2 bytes) or longs (4 bytes). All arguments and values are integers (signed or unsigned). No floating point values (numbers with decimal places) are used in Packet Serial commands.

To convert a 32bit value into 4 bytes you just need to shift the bits around:

```
unsigned char byte3 = MyLongValue>>24; //High byte
unsigned char byte2 = MyLongValue>>16;
unsigned char byte1 = MyLongValue>>8;
unsigned char byte0 = MyLongValue;            //Low byte
```

The same applies to 16bit values:

```
unsigned char byte1 = MyWordValue>>8;  //High byte
unsigned char byte0 = MyWordValue;            //Low byte
```

The oposite can also be done. Convert several bytes into a 16bit or 32bit value:

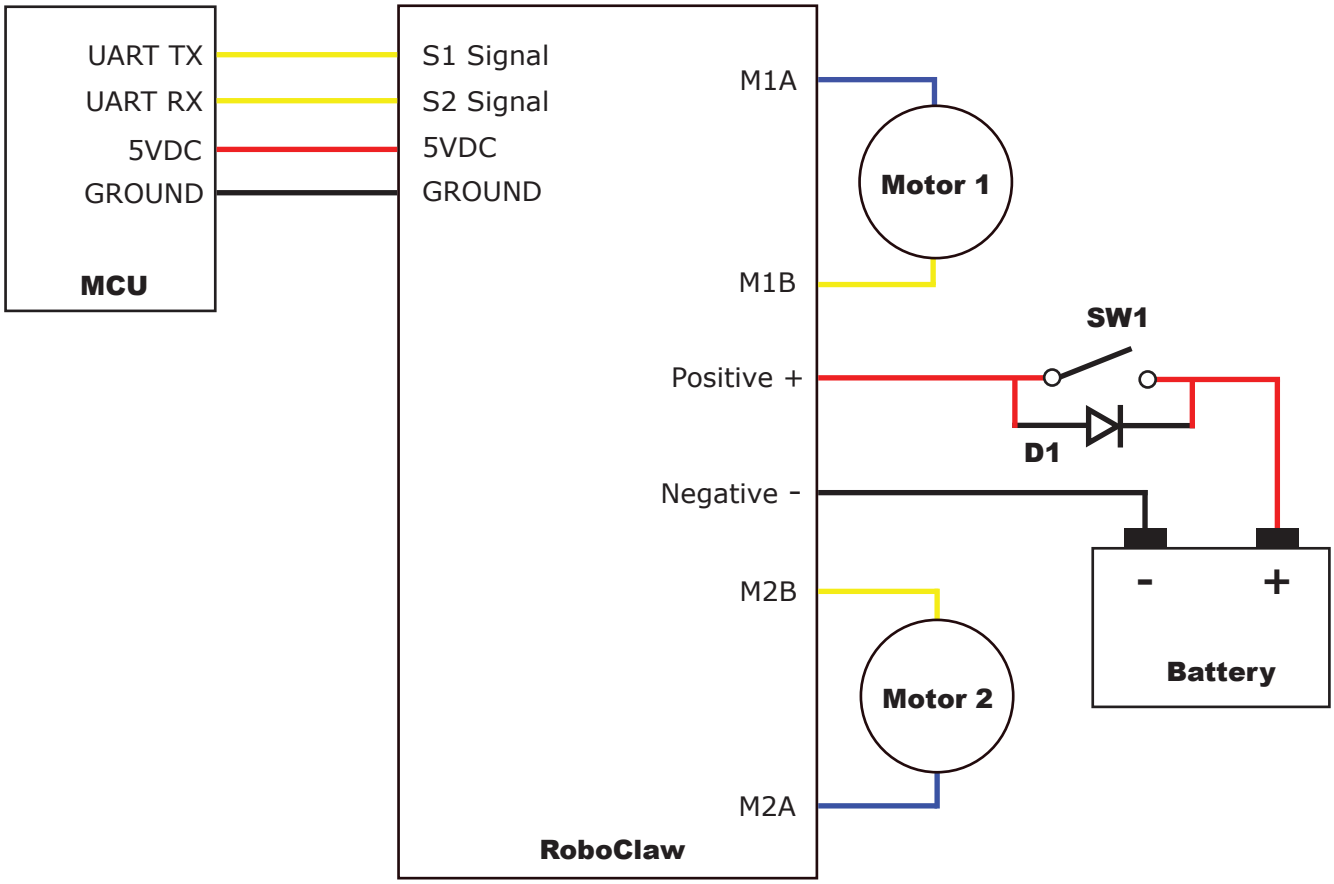
```
unsigned long MyLongValue = byte3<<24 | byte2<<16 | byte1<<8 | byte0;

unsigned int MyWordValue = byte1<<8 | byte0;
```

Packet Serial commands, when a value must be broken into multiple bytes or combined from multple bytes it will be indicated either by (2 bytes) or (4 bytes).

Packet Serial Wiring

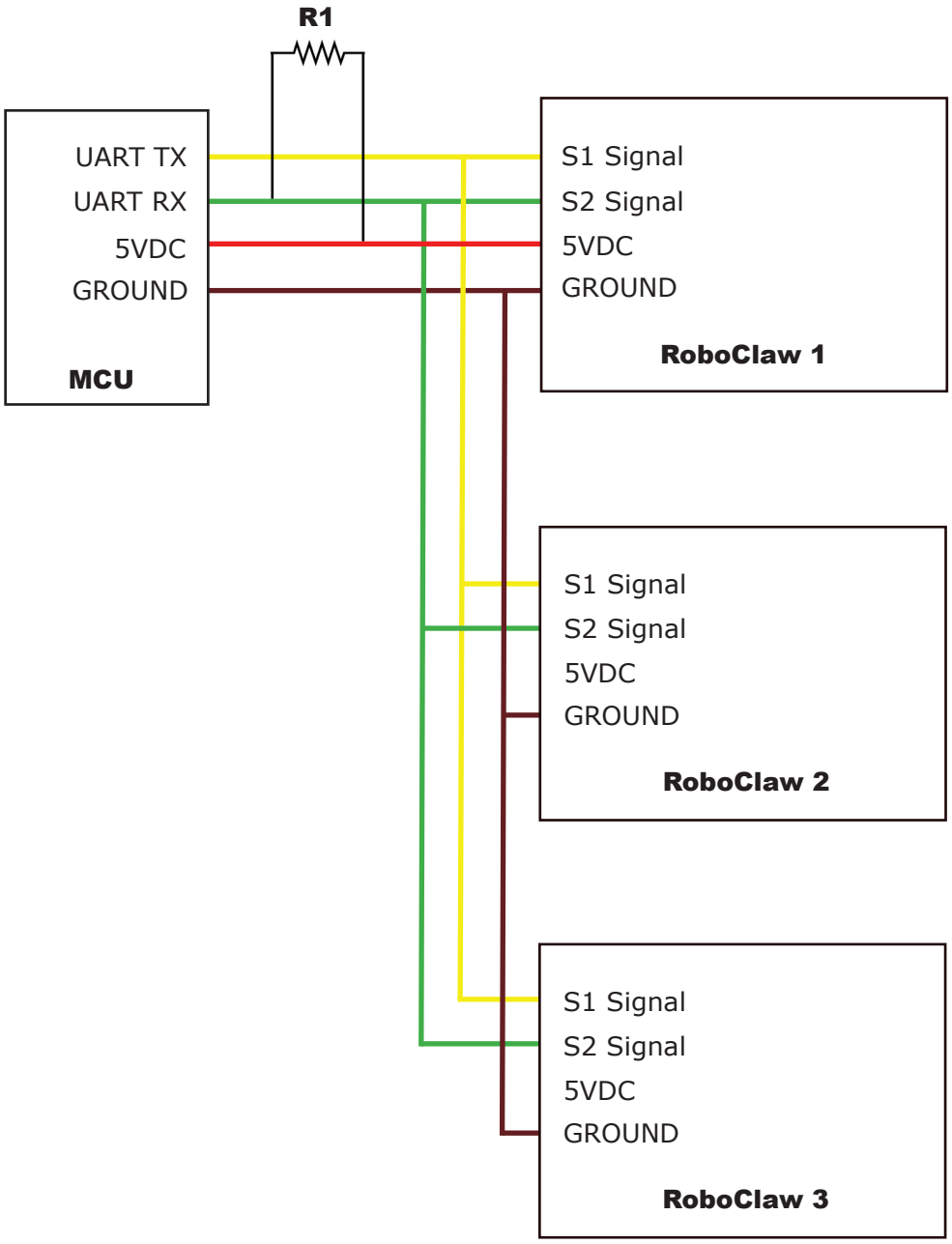
In packet serial mode the RoboClaw can transmit and receive serial data. A microcontroller with a UART is recommended. The UART will buffer the data received from RoboClaw. When a request for data is made to RoboClaw the return data will have at least a 1ms delay after the command is received if the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

The diagram below shows the main battery as the only power source. Make sure the LB jumper is set correctly. The 5VDC shown connected is only required if your MCU needs a power source. This is the BEC feature of RoboClaw. If the MCU has its own power source do not connect the 5VDC.



Multi-Unit Packet Serial Wiring

In packet serial mode up to eight Roboclaw units can be controlled from a single serial port. The wiring diagram below illustrates how this is done. Each Roboclaw must have multi-unit mode enabled and have a unique packet serial address set. This can be configured using Ion studio. Wire the S1 and S2 pins directly to the MCU TX and RX pins. Install a pull-up resistor (R1) on the MCU RX pin. A 1K to 4.7K resistor value is recommended.



Commands 0 - 7 Compatibility Commands

The following commands are used in packet serial mode. The command syntax is the same for commands 0 thru 7:

```
Send: Address, Command, ByteValue, CRC16
Receive: [0xFF]
```

Command	Description
0	Drive Forward Motor 1
1	Drive Backwards Motor 1
2	Set Main Voltage Minimum
3	Set Main Voltage Maximum
4	Drive Forward Motor 2
5	Drive Backwards Motor 2
6	Drive Motor 1 (7 Bit)
7	Drive Motor 2 (7 Bit)
8	Drive Forward Mixed Mode
9	Drive Backwards Mixed Mode
10	Turn Right Mixed Mode
11	Turn Left Mixed Mode
12	Drive Forward or Backward (7 bit)
13	Turn Left or Right (7 Bit)

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, CRC(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, CRC(2 bytes)]
Receive: [0xFF]
```

2 - Set Minimum Main Voltage (Command 57 Preferred)

Sets main battery (B- / B+) minimum voltage level. If the battery voltages drops below the set voltage level RoboClaw will stop driving the motors. 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 - 140 (6V - 34V). 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.

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

3 - Set Maximum Main Voltage (Command 57 Preferred)

Sets main battery (B- / B+) maximum voltage level. The valid data range is 30 - 175 (6V - 34V). During regenerative breaking a back voltage is applied to charge the battery. When using a power supply, by setting the maximum voltage level, RoboClaw will, before exceeding it, go into hard braking mode until the voltage drops below the maximum value set. This will prevent overvoltage conditions when using power supplies. 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: [Address, 3, Value, CRC(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, CRC(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, CRC(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, CRC(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, CRC(2 bytes)]
Receive: [0xFF]
```

Commands 8 - 13 Mixed Mode Compatibility Commands

The following commands are mix mode commands used to control speed and turn for differential steering. Before a command is executed, both valid drive and turn data packets are required Once RoboClaw begins to operate the motors turn and speed can be updated independently.

8 - Drive Forward

Drive forward in mix mode. Valid data range is 0 - 127. A value of 0 = full stop and 127 = full forward.

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

9 - Drive Backwards

Drive backwards in mix mode. Valid data range is 0 - 127. A value of 0 = full stop and 127 = full reverse.

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

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.

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

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.

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

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.

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

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.

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

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 an Arduno Uno with P11 connected to S1 and P10 connected to S2. Set mode 7 and option 4. Additional example programs can be downloaded from Ionmc.com.

```
//Set mode to 7(packet serial address 0x80) and option to 4(38400)

//Includes required to use Roboclaw library
#include "BMSerial.h"
#include "RoboClaw.h"

//Roboclaw Address
#define address 0x80

//Setup communcaitions with roboclaw. Use pins 10 and 11 with 10ms timeout
RoboClaw roboclaw(10,11,10000);

void setup() {
  //Communciate with roboclaw at 38400bps
  roboclaw.begin(38400);
  roboclaw.ForwardMixed(address, 0);
  roboclaw.TurnRightMixed(address, 0);
}

void loop() {
  //Using independant motor Forward and Backward commands
  roboclaw.ForwardM1(address,64); //start Motor1 forward at half speed
  roboclaw.BackwardM2(address,64); //start Motor2 backward at half speed
  delay(2000);
  roboclaw.BackwardM1(address,64);
  roboclaw.ForwardM2(address,64);
  delay(2000);
  roboclaw.BackwardM1(address,0);
  roboclaw.ForwardM2(address,0);
  delay(2000);

  //Using independant motor combined forward/backward commands
  roboclaw.ForwardBackwardM1(address,96); //start Motor1 forward at half speed
  roboclaw.ForwardBackwardM2(address,32); //start Motor2 backward at half speed
  delay(2000);
  roboclaw.ForwardBackwardM1(address,32);
  roboclaw.ForwardBackwardM2(address,96);
  delay(2000);
  roboclaw.ForwardM1(address,0); //stop
  roboclaw.BackwardM2(address,0); //stop
  delay(2000);

  //Using Mixed commands
  roboclaw.ForwardMixed(address, 127);  //full speed forward
  roboclaw.TurnRightMixed(address, 0);  //no turn
  delay(2000);
  roboclaw.TurnRightMixed(address, 64);  //half speed turn right
  delay(2000);
  roboclaw.TurnLeftMixed(address, 64);  //half speed turn left
  delay(2000);
```

```
  roboclaw.TurnLeftMixed(address, 0);  //stop turn
  roboclaw.BackwardMixed(address, 127);  //half speed backward
  delay(2000);
  roboclaw.TurnRightMixed(address, 64);  //half speed turn right
  delay(2000);
  roboclaw.TurnLeftMixed(address, 64);  //half speed turn left
  delay(2000);

  roboclaw.ForwardMixed(address, 0);    //stop going backward
  roboclaw.TurnRightMixed(address, 64);  //half speed right turn
  delay(2000);
  roboclaw.TurnLeftMixed(address, 64);  //half speed left turn
  delay(2000);
  roboclaw.TurnRightMixed(address, 0);  //stop turn(full stop)
  delay(2000);
}
```

Advance Packet Serial

Commands

The following commands are used to read RoboClaw status information, version information and to set or read configuration values. All commands sent to RoboClaw need to be signed with a CRC16 (Cyclic Redundancy Check of 2 bytes) to validate each packet it received. This is more complex than a simple checksum but prevents errors that could otherwise cause unexpected actions to execute on the Roboclaw. See Packet Serial section of this manual for an explanation on how to create the CRC16 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
48	Read Motor PWMs
49	Read Motor Currents
57	Set Main Battery Voltages
58	Set Logic Battery Voltages
59	Read Main Battery Voltage Settings
60	Read Logic Battery Voltage Settings
68	Set default duty cycle acceleration for M1
69	Set default duty cycle acceleration for M2
74	Set S3,S4 and S5 Modes
75	Read S3,S4 and S5 Modes
76	Set DeadBand for RC/Analog controls
77	Read DeadBand for RC/Analog controls
80	Restore Defaults
81	Read Default Duty Cycle Accelerations
82	Read Temperature
83	Read Temperature 2
90	Read Status
91	Read Encoder Modes
92	Set Motor 1 Encoder Mode
93	Set Motor 2 Encoder Mode
94	Write Settings to EEPROM
95	Read Settings from EEPROM
98	Set Standard Config Settings
99	Read Standard Config Settings
100	Set CTRL Modes
101	Read CTRL Modes
102	Set CTRL1
103	Set CTRL2

Command	Description
104	Read CTRLs
133	Set M1 Maximum Current
134	Set M2 Maximum Current
135	Read M1 Maximum Current
136	Read M2 Maximum Current
148	Set PWM Mode
149	Read PWM Mode

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, CRC(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.

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), CRC(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, CRC(2 bytes)]
```

26 - Set Minimum Logic Voltage Level

Note: This command is included for backwards compatibility. We recommend you use command 58 instead.

Sets logic input (LB- / LB+) minimum voltage level. RoboClaw will shut down with an error if the voltage is below this level. 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 - 140 (6V - 34V). 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.

```
Send: [Address, 26, Value, CRC(2 bytes)]
Receive: [0xFF]
```

27 - Set Maximum Logic Voltage Level

Note: This command is included for backwards compatibility. We recommend you use command 58 instead.

Sets logic input (LB- / LB+) maximum voltage level. The valid data range is 30 - 175 (6V - 34V). RoboClaw will shutdown with an error if the voltage is above this level. 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: [Address, 27, Value, CRC(2 bytes)]
Receive: [0xFF]
```

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), CRC(2 bytes)]
```

49 - Read Motor Currents

Read the current draw 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), CRC(2 bytes)]
```

57 - Set Main Battery Voltages

Set the Main Battery Voltage cutoffs, Min and Max. Min and Max voltages are in 10th of a volt increments. Multiply the voltage to set by 10.

```
Send: [Address, 57, Min(2 bytes), Max(2bytes, CRC(2 bytes)]
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 to set by 10.

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

68 - Set M1 Default Duty Acceleration

Set the default acceleration for M1 when using duty cycle commands(Cmds 32,33 and 34) or when using Standard Serial, RC and Analog PWM modes.

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

69 - Set M2 Default Duty Acceleration

Set the default acceleration for M2 when using duty cycle commands(Cmds 32,33 and 34) or when using Standard Serial, RC and Analog PWM modes.

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

74 - Set S3, S4 and S5 Modes

Set modes for S3,S4 and S5.

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

Mode	S3mode	S4mode	S5mode
0	Default	Disabled	Disabled
1	E-Stop(latching)	E-Stop(latching)	E-Stop(latching)
2	E-Stop	E-Stop	E-Stop
3	Voltage Clamp	Voltage Clamp	Voltage Clamp
4		M1 Home	M2 Home

Mode Description

Disabled: pin is inactive.
Default: Flip switch if in RC/Analog mode or E-Stop(latching) in Serial modes.
E-Stop(Latching): causes the Roboclaw to shutdown until the unit is power cycled.
E-Stop: Holds the Roboclaw in shutdown until the E-Stop signal is cleared.
Voltage Clamp: Sets the signal pin as an output to drive an external voltage clamp circuit
Home(M1 & M2): will trigger the specific motor to stop and the encoder count to reset to 0.

75 - Get S3, S4 amd S5 Modes

Read mode settings for S3,S4 and S5. See command 74 for mode descriptions

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

76 - Set DeadBand for RC/Analog controls

Set RC/Analog mode control deadband percentage in 10ths of a percent. Default value is 25(2.5%). Minimum value is 0(no DeadBand), Maximum value is 250(25%).

```
Send: [Address, 76, Reverse, Forward, CRC(2 bytes)]
Receive: [0xFF]
```

77 - Read DeadBand for RC/Analog controls

Read DeadBand settings in 10ths of a percent.

```
Send: [Address, 77]
Receive: [Reverse, SForward, CRC(2 bytes)]
```

80 - Restore Defaults

Reset Settings to factory defaults.

```
Send: [Address, 80]
Receive: [0xFF]
```

81 - Read Default Duty Acceleration Settings

Read M1 and M2 Duty Cycle Acceleration Settings.

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

82 - Read Temperature

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

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

90 - Read Status

Read the current unit status.

```
Send: [Address, 90]
Receive: [Status, CRC(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

91 - Read Encoder Mode

Read the encoder mode for both motors.

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

Encoder Mode bits

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

92 - Set Motor 1 Encoder Mode

Set the Encoder Mode for motor 1. See command 91.

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

93 - Set Motor 2 Encoder Mode

Set the Encoder Mode for motor 2. See command 91.

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

94 - Write Settings to EEPROM

Writes all settings to non-volatile memory. Values will be loaded after each power up.

```
Send: [Address, 94]
Receive: [0xFF]
```

95 - Read Settings from EEPROM

Read all settings from non-volatile memory.

```
Send: [Address, 95]
Receive: [Enc1Mode, Enc2Mode, CRC(2 bytes)]
```

98 - Set Standard Config Settings

Set config bits for standard settings.

```
Send: [Address, 98, Config(2 bytes), CRC(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
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), CRC(2 bytes)]
```

100 - Set CTRL Modes

Set CTRL modes of CTRL1 and CTRL2 output pins(available on select models).

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

On select models of Roboclaw, two Open drain, high current output drivers are available, CTRL1 and CTRL2.

Mode	Function
0	Disable
1	User
2	Voltage Clamp
3	Brake

User Mode - The output level can be controlled by setting a value from 0(0%) to 65535(100%). A variable frequency PWM is generated at the specified percentage.

Voltage Clamp Mode - The CTRL output will activate when an over voltage is detected and released when the overvoltage disipates. Adding an external load dump resistor from the CTRL pin to B+ will allow the Roboclaw to disipate over voltage energy automatically(up to the 3amp limit of the CTRL pin).

Brake Mode - The CTRL pin can be used to activate an external brake(CTRL1 for Motor 1 brake and CTRL2 for Motor 2 brake). The signal will activate when the motor is stopped(eg 0 PWM). Note acceleration/default_acceleration settings should be set appropriately to allow the motor to slow down before the brake is activated.

101 - Read CTRL Modes

Read CTRL modes of CTRL1 and CTRL2 output pins(available on select models).

```
Send: [Address, 101]
Receive: [CTRL1Mode(1 bytes), CTRL2Mode(1 bytes), CRC(2 bytes)]
```

Reads CTRL1 and CTRL2 mode setting. See 100 - Set CTRL Modes for valid values.

102 - Set CTRL1

Set CTRL1 output value(available on select models)

```
Send: [Address, 102, Value(2 bytes), CRC(2 bytes)]
Receive: [0xFF]
```

Set the output state value of CTRL1. See 100 - Set CTRL Modes for valid values.

103 - Set CTRL2

Set CTRL2 output value(available on select models)

```
Send: [Address, 103, Value(2 bytes), CRC(2 bytes)]
Receive: [0xFF]
```

Set the output state value of CTRL2. See 100 - Set CTRL Modes for valid values.

104 - Read CTRL Settings

Read CTRL1 and CTRL2 output values(available on select models)

```
Send: [Address, 104]
Receive: [CTRL1(2 bytes), CTRL2(2 bytes), CRC(2 bytes)]
```

Reads currently set values for CTRL Settings. See 100 - Set CTRL Modes for valid values.

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, CRC(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, CRC(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), CRC(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), CRC(2 bytes)]
```

148 - Set PWM Mode

Set PWM Drive mode. Locked Antiphase(0) or Sign Magnitude(1).

```
Send: [Address, 148, Mode, CRC(2 bytes)]
Receive: [0xFF]
```

149 - Read PWM Mode

Read PWM Drive mode. See Command 148.

```
Send: [Address, 149]
Receive: [PWMMode, CRC(2 bytes)]
```

Encoders

Cloose Loop Modes

RoboClaw supports a wide range of encoders for close loop modes. Encoders are used in velocity, position or a cascaded velocity with position control mode. This manual mainly deals with Quadrature and Absolute encoders. However additional types of encoders are supported.

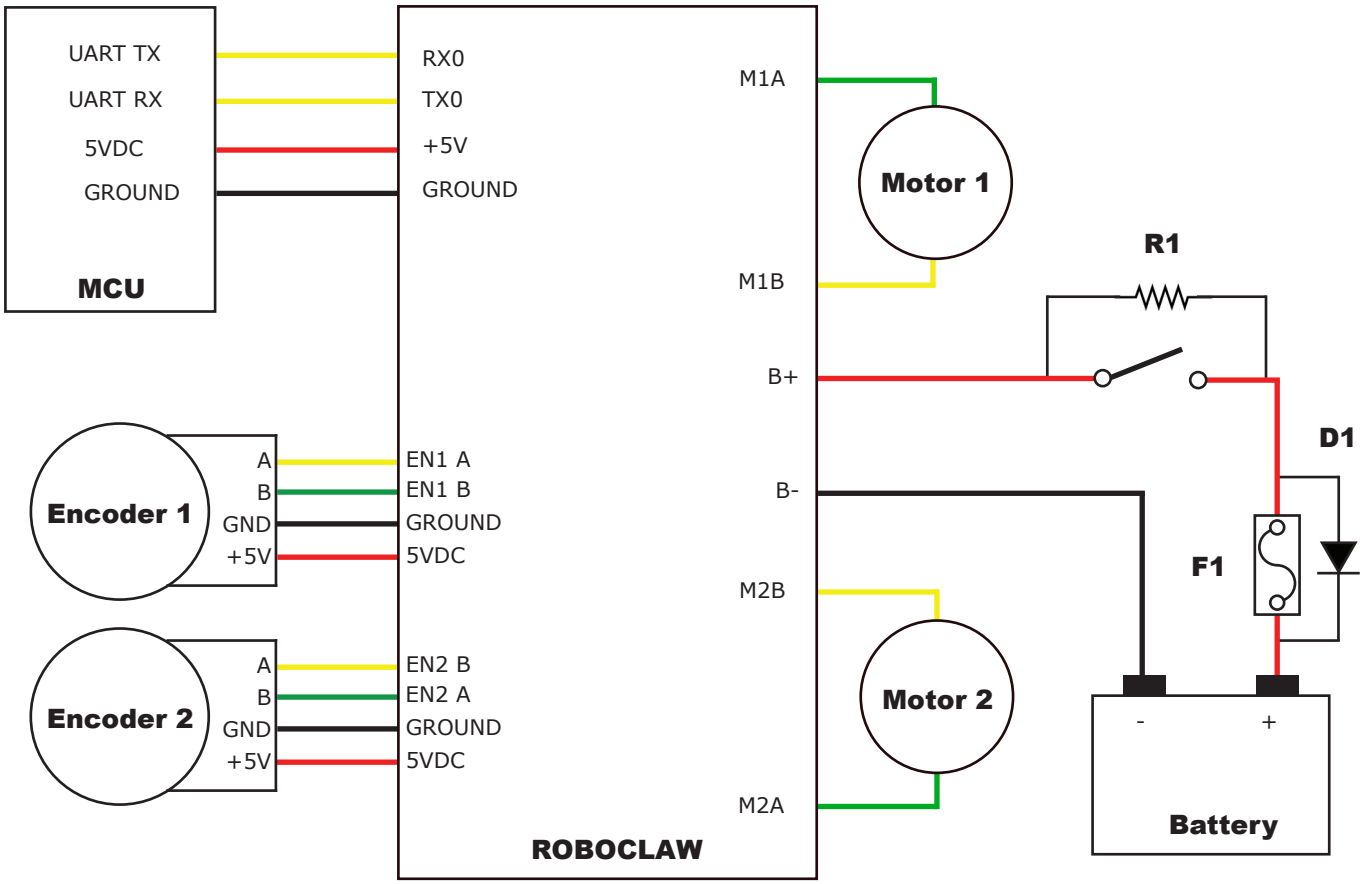
Encoder Tunning

All encoders will require tuning to properly function. Ion Studio incorperates an Auto Tune function which can automatically tune the PID and editable fields for manual tuning of the PID. Encoders can also be tunned using Advance Control Commands which can be sent by a MCU or other control devices.

Quadrature Encoders Wiring

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

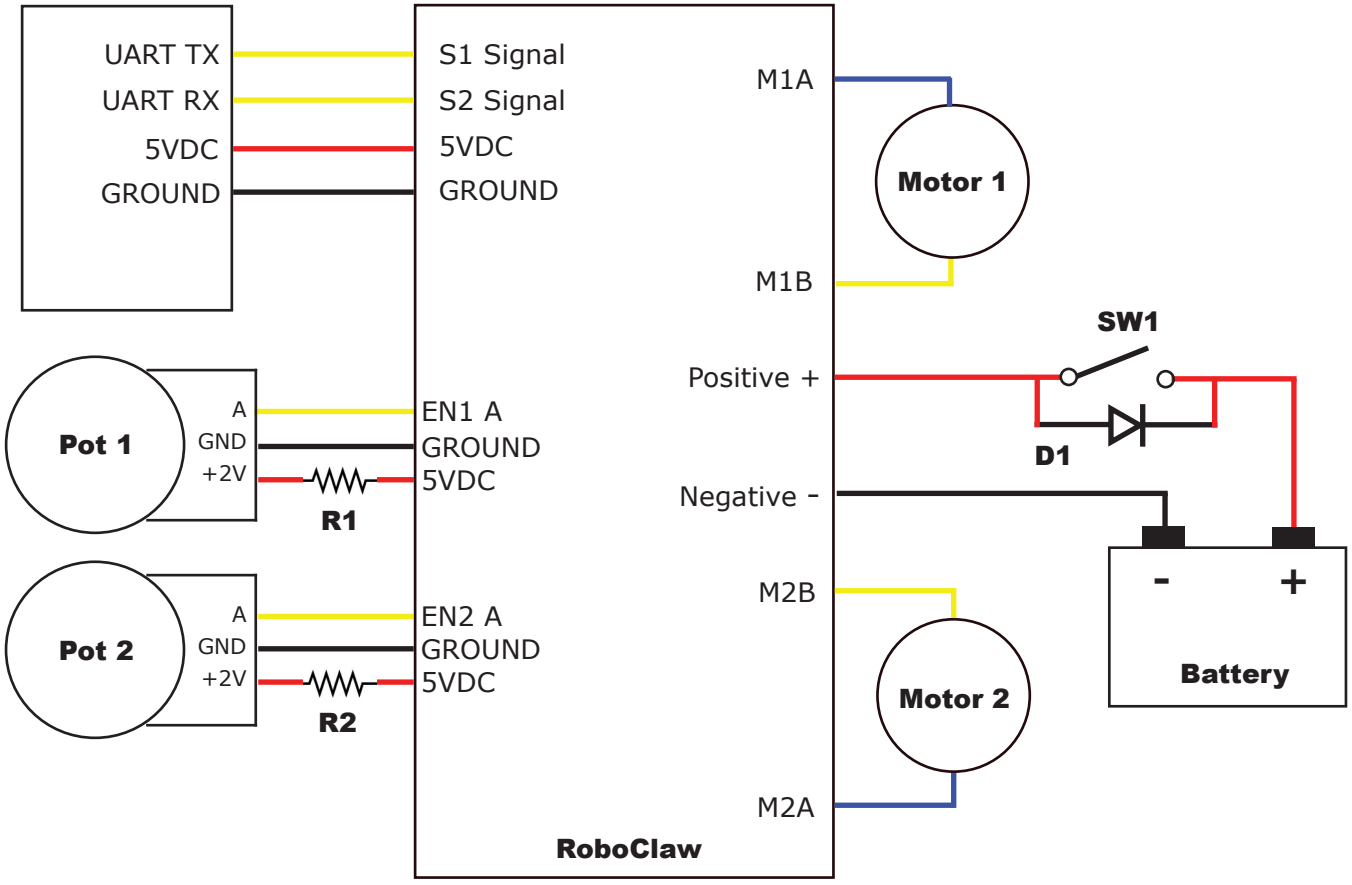
Quadrature encoders are directional. In a simple two motor robot, one motor will spin clock wise (CW) and the other motor will spin counter clock wise (CCW). The A and B inputs for one of the encoders must be reversed to allow both encoders to count up when the robot is moving forward. If both encoder are connected with leading edge pulse to channel A one will count up and the other down. This will cause commands like Mix Drive Forward to not work as expected. All motor and encoder combinations will need to be tuned.



Absolute Encoder Wiring

RoboClaw is capable of reading absolute encoders that output an analog voltage. Like the Analog input modes for controlling the motors, the absolute encoder voltage must be between 0v and 2v. If using standard potentiometers as absolute encoders the 5v from the RoboClaw can be divided down to 2v at the potentiometer by adding a resistor from the 5v line on the RoboClaw to the potentiometer. For a 5k pot R1 / R2 = 7.5k, for a 10k pot R1 / R2 = 15k and for a 20k pot R1 / R2 = 30k.

The diagram below shows the main battery as the only power source. Make sure the LB jumper is installed. The 5VDC shown connected is only required if your MCU needs a power source. This is the BEC feature of RoboClaw. If the MCU has its own power source do not connect the 5VDC.



Encoder Tuning

To control motor speed and or position with an encoder the PID must be calibrated for the specific motor and encoder being used. Using Ion Studio the PID can be tuned manually or by the auto tune fucntion. Once the encoders are tuned the settings can be saved to the onboard eeprom and will be loaded each time the unit powers up.

The Ion Studio window for Velocity Settings will auto tune for velocity. The window for Position Settings can tune a simple PD position controller, a PID position controller or a cascaded Position with Velocity controller(PIV). The cascaded tune will determine both the velocity and position values for the motor but still requires the QPPS be manually set for the motor before starting. Auto tune functions usually return reasonable values but manually adjustments may be required for optimum performance.

Auto Tuning

Ion Studio provides the option to auto tune velocity and position control. To use the auto tune option make sure the encoder and motor are running in the desired direction and the basic PWM control of the motor works as expected. It is recommend to ensure the motor and encoder combination are functioning properly before using the auto tune feature.

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

2. Slide the motor slider up to start moving the motor forward. Check the encoder is increasing in value. If it is not either reverse the motor wires or the encoder wires. The recheck.

Before using auto tune you must first set the motors and encoders maximum speed. For the purpose of auto tune the maximum quadrature pulse per second (QPPS) is the maximum speed the motor and encoder can acheive. When using an absolute encoder the QPPS will be the maximum rotational speed of the absolute encoder. Check the encoders data sheet to ensure the maximum rotational speed is not exceeded. Auto tune for position control can not automatically measure the maximum QPPS due to most position control systems having a limited range of movement.

3. To determine the maximum QPPS value, use the PWM settings screen to run the motor and encoder at 100% duty by moving the slider bar full up or down. Record the value from M1 Speed or M2 Speed fields at the top of the window. This is your maximum QPPS speed. If the motor can not be ran at full speed due to physical constraints, then an estimated maximum speed in encoder counts is required.

4. Enter the QPPS speed obtained from step 3 into the QPPS fields under settings. Ensure the correct QPPS is entered for the corresponding motor channel. Two identical motors and encoders may not function exactly the same so the maximum QPPS may vary.

5. To start auto tune click the auto tune button for the motor channel that is will be tuned first. The auto tune function will try to determine the best settings for that motor channel.

If the motor or encoder are wired incorrectly, the auto tune function can lock up and the motor controller will become unresponsive. Correct the wiring problem and reset the motor controller to continue.

Manual Velocity Calibration Procedure

1. Determine the quadrature pulses per second(QPPS) value for your motor. The simplest method to do this is to run the Motor at 100% duty using Ion Studio and read back the speed value from the encoder attached to the motor. If you are unable to run the motor like this due to physical constraints you will need to estimate the maximum speed in encoder counts the motor can produce.

2. Set the initial P,I and D values in the Velocity control window to 1,0 and 0. Try moving the motor using the slider controls in IonMotion. If the motor does not move it may not be wired correctly or the P value needs to be increased. If the motor immediately runs at max speed when you change the slider position you probably have the motor or encoder wires reversed. The motor is trying to go at the speed specified but the encoder reading is coming back in the opposite direction so the motor increases power until it eventually hits 100% power. Reverse the encoder or motor wires(not both) and test again.

3. Once the motor has some semblance of control you can set a moderate speed. Then start increasing the P value until the speed reading is near the set value. If the motor feels like it is vibrating at higher P values you should reduce the P value to about 2/3rds that value. Move on to the I setting.

4. Start increasing the I setting. You will usually want to increase this value by .1 increments. The I value helps the motor reach the exact speed specified. Too high an I value will also cause the motor to feel rough/vibrate. This is because the motor will over shoot the set speed and then the controller will reduce power to get the speed back down which will also under shoot and this will continue oscillating back and forth form too fast to too slow, causing a vibration in the motor.

5. Once P and I are set reasonably well usually you will leave D = 0. D is only required if you are unable to get reasonable speed control out of the motor using just P and I. D will help dampen P and I over shoot allowing higher P and I values, but D also increases noise in the calculation which can cause oscillations in the speed as well.

Manual Position Calibration Procedure

1. Position mode requires the Velocity mode QPPS value be set as described above. For simple Position control you can set Velocity P, I and D all to 0.

2. Set the Position I and D settings to 0. Set the P setting to 2000 as a reasonable starting point. To test the motor you must also set the Speed argument to some value. We recommend setting it to the same value as the QPPS setting(eg maximum motor speed). Set the minimum and maximum position values to safe numbers. If your motor has no dead stops this can be +-2 billion. If your motor has specific dead stops(like on a linear actuator) you will need to manually move the motor to its dead stops to determine these numbers. Leave some margin infront of each deadstop. Note that when using quadrature encoders you will need to home your motor on every power up since the quadrature readings are all relative to the starting position unless you set/reset the encoder values.

3. At this point the motor should move in the appropriate direction and stop, not necessarily close to the set position when you move the slider. Increase the P setting until the position is over shooting some each time you change the position slider. Now start increasing the D setting(leave I at 0). Increasing D will add dampening to the movement when getting close to the set position. This will help prevent the over shoot. D will usually be anywhere from 5 to 20 times larger than P but not always. Continue increasing P and D until the motor is working reasonably well. Once it is you have tuned a simple PD system.

4. Once your position control is acting relatively smoothly and coming close to the set position you can think about adjusting the I setting. Adding I will help reach the exact set point specified but in most motor systems there is enough slop in the gears that instead you will end up causing an oscillation around the specified position. This is called hunting. The I setting causes this when there is any slop in the motor/encoder/gear train. You can compensate some for this by adding deadzone. Deadzone is the area around the specified position the controller will consider to be equal to the position specified.

5. One more setting must be adjusted in order to use the I setting. The Imax value sets the maximum wind up allowed for the I setting calculation. Increasing Imax will allow I to affect a larger amount of the movement of the motor but will also allow the system to oscillate if used with a badly tuned I and/or set too high.

Encoder Commands

The following commands are used with the encoders both quadrature and absolute. The Encoder Commands are used to read the register values for both encoder channels.

Command	Description
16	Read Encoder Count/Value for M1.
17	Read Encoder Count/Value for M2.
18	Read M1 Speed in Encoder Counts Per Second.
19	Read M2 Speed in Encoder Counts Per Second.
20	Resets Encoder Registers for M1 and M2(Quadrature only).
22	Set Encoder 1 Register(Quadrature only).
23	Set Encoder 2 Register(Quadrature only).
30	Read Current M1 Raw Speed
31	Read Current M2 Raw Speed
78	Read Encoders Counts
79	Read Motor Speeds

16 - Read Encoder Count/Value M1

Read M1 encoder count/position.

```
Send: [Address, 16]
Receive: [Enc1(4 bytes), Status, CRC(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, CRC(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, CRC(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, CRC(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, CRC(2 bytes)]
Receive: [0xFF]
```

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), CRC(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), CRC(2 bytes)]
Receive: [0xFF]
```

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, CRC(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, CRC(2 bytes)]
```

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

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), CRC(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), CRC(2 bytes)]
```

Advance Motor Control

Advanced Motor Control Commands

The following commands are used to control motor speeds, acceleration distance and position using encoders. The PID can also be manually adjusted using Advance Motor Control Commands.

Command	Description
28	Set Velocity PID Constants for M1.
29	Set Velocity PID Constants for M2.
32	Drive M1 With Signed Duty Cycle. (Encoders not required)
33	Drive M2 With Signed Duty Cycle. (Encoders not required)
34	Drive M1 / M2 With Signed Duty Cycle. (Encoders not required)
35	Drive M1 With Signed Speed.
36	Drive M2 With Signed Speed.
37	Drive M1 / M2 With Signed Speed.
38	Drive M1 With Signed Speed And Acceleration.
39	Drive M2 With Signed Speed And Acceleration.
40	Drive M1 / M2 With Signed Speed And Acceleration.
41	Drive M1 With Signed Speed And Distance. Buffered.
42	Drive M2 With Signed Speed And Distance. Buffered.
43	Drive M1 / M2 With Signed 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	Drive M1 / M2 With Signed Speed, Acceleration And Distance. Buffered.
47	Read Buffer Length.
50	Drive M1 / M2 With Individual Signed Speed and Acceleration
51	Drive M1 / M2 With Individual Signed Speed, Accel and Distance
52	Drive M1 With Signed Duty and Accel. (Encoders not required)
53	Drive M2 With Signed Duty and Accel. (Encoders not required)
54	Drive M1 / M2 With Signed Duty and Accel. (Encoders not required)
55	Read Motor 1 Velocity PID Constants
56	Read Motor 2 Velocity PID Constants
61	Set Position PID Constants for M1.
62	Set Position PID Constants for M2
63	Read Motor 1 Position PID Constants
64	Read Motor 2 Position PID Constants
65	Drive M1 with Speed, Accel, Deccel and Position
66	Drive M2 with Speed, Accel, Deccel and Position
67	Drive M1 / M2 with Speed, Accel, Deccel and Position
68	Set default duty cycle acceleration for M1
69	Set default duty cycle acceleration for M2

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), CRC(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), CRC(2 bytes)]Re-
ceive: [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), CRC(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), CRC(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), CRC(2 bytes)]
Receive: [0xFF]
```

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

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), CRC(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), CRC(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), CRC(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), CRC(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), CRC(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), CRC(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 - 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.

```
Send: [Address, 41, Speed(4 Bytes), Distance(4 Bytes), Buffer, CRC(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 - 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.

```
Send: [Address, 42, Speed(4 Bytes), Distance(4 Bytes), Buffer, CRC(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 - Buffered 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, CRC(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 - 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.

```
Send: [Address, 44, Accel(4 bytes), Speed(4 Bytes), Distance(4 Bytes),
             Buffer, CRC(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 - 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.

```
Send: [Address, 45, Accel(4 bytes), Speed(4 Bytes), Distance(4 Bytes),
            Buffer, CRC(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 - Buffered 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, CRC(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.

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, CRC(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.

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), CRC(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, CRC(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.

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), CRC(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), CRC(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), CRC(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).

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), CRC(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), CRC(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), CRC(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), CRC(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), CRC(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), CRC(2 bytes)]
```

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, CRC(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, CRC(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, CRC(2 bytes)]
Receive: [0xFF]
```

Reading Quadrature Encoder - Arduino Example

The example was tested with an Arduino Uno using 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. Additional example programs can be downloaded from Ionmc.com.

```
//Set mode to 7(packet serial address 0x80) and option 4(38400)

//Includes required to use Roboclaw library
#include "BMSerial.h"
#include "RoboClaw.h"

//Roboclaw Address
#define address 0x80

//Definte terminal for display. Use hardware serial pins 0 and 1
BMSerial terminal(0,1);

//Setup communcaitions with roboclaw. Use pins 10 and 11 with 10ms timeout
RoboClaw roboclaw(10,11,10000);

void setup() {
  //Open terminal and roboclaw at 38400bps
  terminal.begin(57600);
  roboclaw.begin(38400);
}

void loop() {
  uint8_t status1,status2,status3,status4;
  bool valid1,valid2,valid3,valid4;

  //Read all the data from Roboclaw before displaying on terminal window
  //This prevents the hardware serial interrupt from interfering with
  //reading data using software serial.
  int32_t enc1= roboclaw.ReadEncM1(address, &status1, &valid1);
  int32_t enc2 = roboclaw.ReadEncM2(address, &status2, &valid2);
  int32_t speed1 = roboclaw.ReadSpeedM1(address, &status3, &valid3);
  int32_t speed2 = roboclaw.ReadSpeedM2(address, &status4, &valid4);

  terminal.print("Encoder1:");
  if(valid1){
    terminal.print(enc1,HEX);
    terminal.print(" ");
    terminal.print(status1,HEX);
    terminal.print(" ");
  }
  else{
    terminal.print("invalid ");
  }
  terminal.print("Encoder2:");
  if(valid2){
    terminal.print(enc2,HEX);
    terminal.print(" ");
    terminal.print(status2,HEX);
    terminal.print(" ");
  }
  else{
    terminal.print("invalid ");
  }
  terminal.print("Speed1:");
```

```
  if(valid3){
    terminal.print(speed1,HEX);
    terminal.print(" ");
  }
  else{
    terminal.print("invalid ");
  }
  terminal.print("Speed2:");
  if(valid4){
    terminal.print(speed2,HEX);
    terminal.print(" ");
  }
  else{
    terminal.print("invalid ");
  }
  terminal.println();

  delay(100);
}
```

Speed Controlled by Quadrature Encoders - Arduino Example

The following example was written using an Arduino UNO using 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. Additional example programs can be downloaded from Ionmc.com.

```
//Set mode 7(packet serial address 0x80) and mode 4(38400)

//Includes required to use Roboclaw library
#include "BMSerial.h"
#include "RoboClaw.h"

#define SPEED 12000
#define SPEED2 12000

//Roboclaw Address
#define address 0x80

//Velocity PID coefficients
#define Kp 1.0
#define Ki 0.5
#define Kd 0.25
#define qpps 44000

//Definte terminal for display. Use hardware serial pins 0 and 1
BMSerial terminal(0,1);

//Setup communcaitions with roboclaw. Use pins 10 and 11 with 10ms timeout
RoboClaw roboclaw(10,11,10000);

long speed;
long speed2;

void setup() {
  //Open terminal and roboclaw serial ports
  terminal.begin(57600);
  roboclaw.begin(38400);

  speed = SPEED;
  speed2 = SPEED2;

  //Set PID Coefficients
  roboclaw.SetM1VelocityPID(address,Kd,Kp,Ki,qpps);
  roboclaw.SetM2VelocityPID(address,Kd,Kp,Ki,qpps);
}
```

```
void displayspeed(void)
{
  uint8_t status1,status2,status3,status4;
  bool valid1,valid2,valid3,valid4;

  int32_t enc1= roboclaw.ReadEncM1(address, &status1, &valid1);
  int32_t enc2 = roboclaw.ReadEncM2(address, &status2, &valid2);
  int32_t speed1 = roboclaw.ReadSpeedM1(address, &status3, &valid3);
  int32_t speed2 = roboclaw.ReadSpeedM2(address, &status4, &valid4);
  terminal.print("Encoder1:");
  if(valid1){
    terminal.print(enc1,DEC);
    terminal.print(" ");
    terminal.print(status1,HEX);
    terminal.print(" ");
  }
  else{
    terminal.print("invalid ");
  }
  terminal.print("Encoder2:");
  if(valid2){
    terminal.print(enc2,DEC);
    terminal.print(" ");
    terminal.print(status2,HEX);
    terminal.print(" ");
  }
  else{
    terminal.print("invalid ");
  }
  terminal.print("Speed1:");
  if(valid3){
    terminal.print(speed1,DEC);
    terminal.print(" ");
  }
  else{
    terminal.print("invalid ");
  }
  terminal.print("Speed2:");
  if(valid4){
    terminal.print(speed2,DEC);
    terminal.print(" ");
  }
  else{
    terminal.print("invalid ");
  }
  terminal.println();
}
```

```
void loop() {
  roboclaw.SpeedM1(address,speed);
  roboclaw.SpeedM2(address,speed);
  for(uint8_t i = 0;i<100;i++){
    displayspeed();
    delay(10);
  }

  roboclaw.SpeedM1(address,-speed);
  roboclaw.SpeedM2(address,-speed);
  for(uint8_t i = 0;i<100;i++){
    displayspeed();
    delay(10);
  }

  roboclaw.SpeedM1(address,0);
  roboclaw.SpeedM2(address,0);
  delay(2000);

  roboclaw.SpeedM1(address,speed2);
  roboclaw.SpeedM2(address,-speed2);
  for(uint8_t i = 0;i<100;i++){
    displayspeed();
    delay(10);
  }

  roboclaw.SpeedM1(address,-speed2);
  roboclaw.SpeedM2(address,speed2);
  for(uint8_t i = 0;i<100;i++){
    displayspeed();
    delay(10);
  }

  roboclaw.SpeedM1(address,0);
  roboclaw.SpeedM2(address,0);
  delay(2000);
}
```

Warranty

Basicmicro warranties its products against defects in material and workmanship for a period of 1 year. If a defect is discovered, Basicmicro will, at our sole discretion, repair, replace, or refund the purchase price of the product in question. Contact us at sales@basicmicro.com. No returns will be accepted without the proper authorization.

Copyrights and Trademarks

Disclaimer

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

Contacts

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

Discussion List

A web based discussion board is maintained at http://www.basicmicro.com

Technical Support

Technical support is available by sending an email to support@basicmicro.com, by opening a support ticket on the Ion Motion Control website or by calling 800-535-9161 during normal operating hours. All email will be answered within 48 hours.

Scanse™

User's Manual and Technical Specifications

sweep v1.0
scanning laser range finder

- ✓ Cost efficient design
- ✓ Operates in full sunlight
- ✓ Low power consumption
- ✓ Wide field of view
- ✓ Small footprint
- ✓ Simple serial connectivity
- ✓ Long Range

⚠ CAUTION

Laser Safety

This device contains a component which emits laser radiation. This laser product is designated Class 1 in accordance with IEC 60825-1:2007 during all operating modes. This means that the laser is safe to look at with the unaided eye, however it is advisable to not look directly into the beam when in use.

Power Safety

When connecting a Sweep sensor to a 5VDC power source, it should be limited to a maximum of 8A as defined in EN 60950-1, sub clause 2.5, Table 2B.

Documentation Revision Information

Rev	Date	Changes
0.991	07/17/2017	Added predicted life section
0.99	05/11/2017	Simplified azimuth conversion formula
0.98	04/18/2017	Updated communication protocol
0.97	03/29/2017	Updated mechanical drawings
0.9	12/19/2016	Initial Release

Table of Contents

Specifications

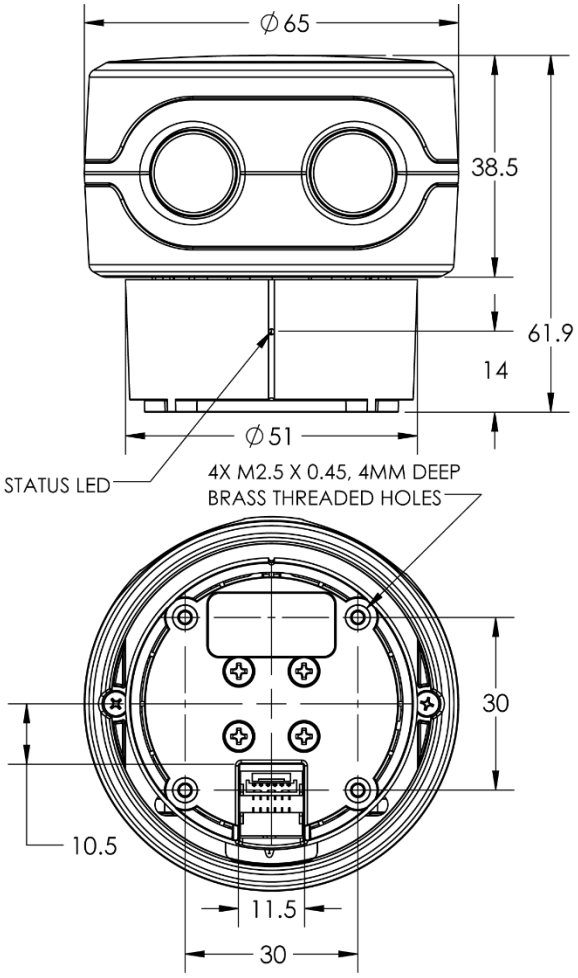


Figure 1, Sweep Dimension Drawing

Physical

Specification	Value
Weight	120 g (4.23 oz.)
Operating Temperature	-10 to 60° C (14 to 140°F)
Storage Temperature	-40 to 80° C (-40 to 176°F)

Electrical

Specification	Value
Power	5VDC ±0.5VDC
Current Consumption	Up to 650mA 450mA nominal

Measurement Performance

Specification	Value
Range (75% reflective target)	40 m (131ft)
Resolution	1 cm (0.4 in)
Update Rate (75% reflective target)	Up to 1075Hz (see "Theory of Operation")

Field of View

Specification	Value
Horizontal Field of View	360 degrees
Vertical Field of View	0.5 degrees

Sweep is a single plane scanner. This means that as its head rotates **counterclockwise**, it records data in a single plane. The beam starts out at approximately 12.7mm in diameter and expands by approximately 0.5 degrees as show in Figure 2.

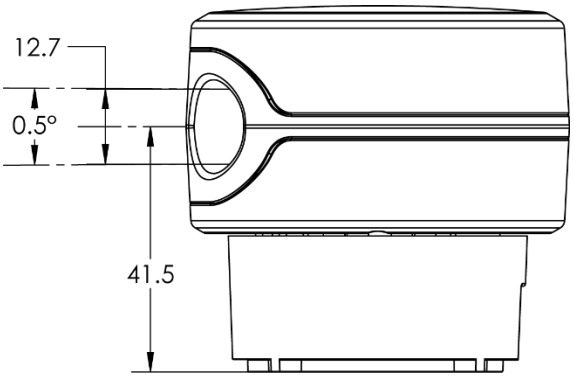
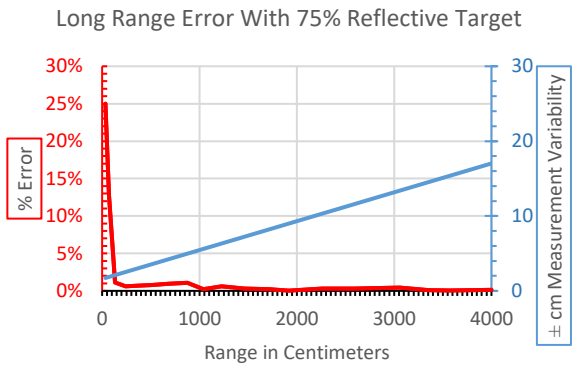


Figure 2, Sweep Field of View

Measurement Error Test Data



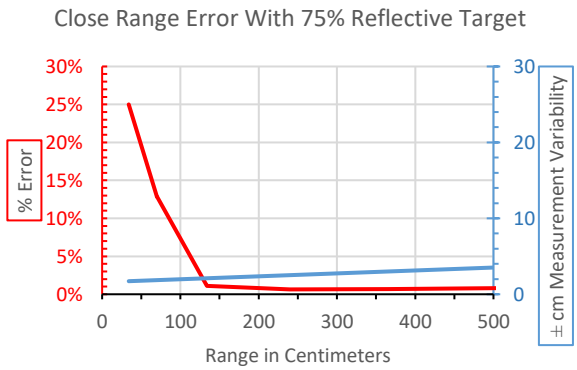


Figure 3, Sweep Accuracy Graphs

Overview of Interfaces

Sweep can be connected to low level micro controllers directly using its serial port, or to a PC using the provided USB to serial converter.

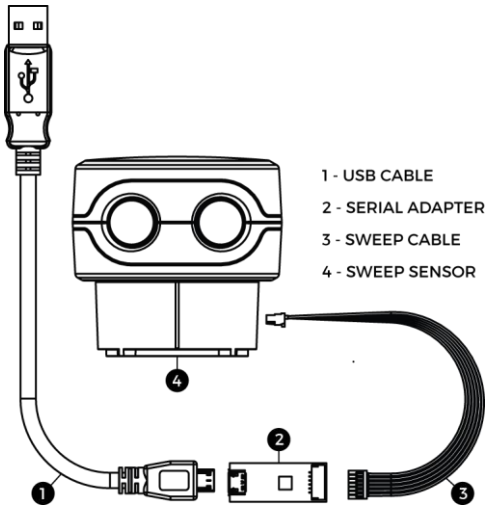


Figure 4, Sweep Cable Diagram

Connector

Sweep has two serial port connectors with identical signals. This allows for more mounting options.

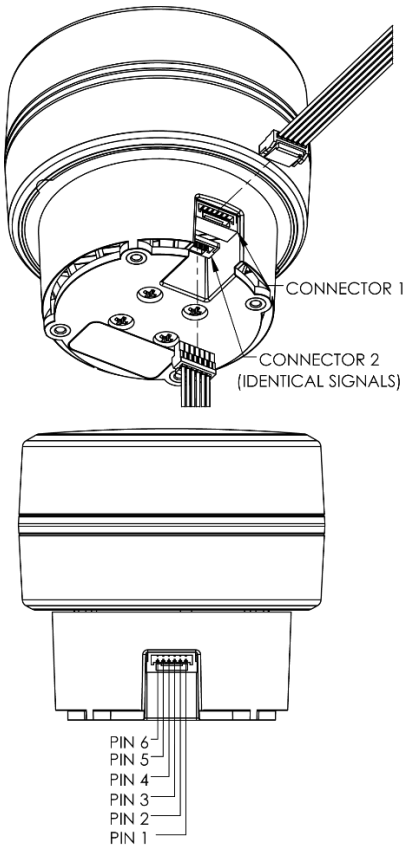


Figure 5, Sweep Connector Diagram

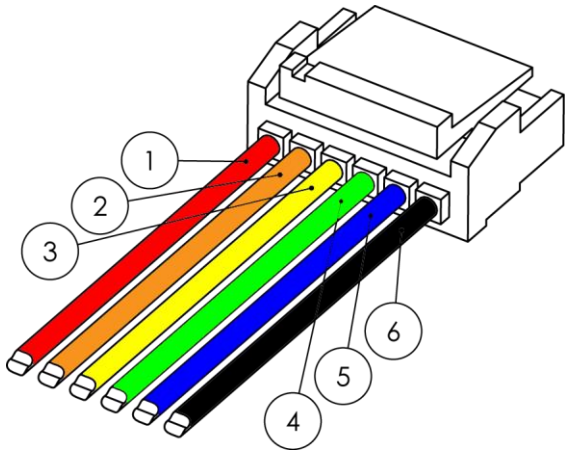


Figure 6, Sweep Pigtail Cable Connector Detail

Table 1, Pin Definition

Pin	Color	Function
1	Red	5VDC ±0.5VDC
2	Orange	Power enable (internal pull-up). Pull down to put device in sleep mode.
3	Yellow	Sync/Device Ready Goes high when first range measurement of new scan is completed, then goes low when second range measurement is completed. Remains low until next rotation. Line is only active when scanning and low when not.
4	Green	UART RX 3.3V (5V compatible)
5	Blue	UART TX 3.3V (5V compatible)
6	Black	Ground (-)

You can create your own cable if needed for your application. These components are readily available:

Part	Description	Mfg.	Part No.
Connector Housing	6-Position, rectangular housing, latch-lock connector receptacle with 1.25 mm (0.049 in.) pitch.	JST	GHR-06V-S
Connector terminal	26-30 AWG crimp socket connector	JST	SSHL-002T-P0.2
Wire	UL 1061 26 AWG stranded copper	N/A	N/A

Status LED

The status LED (Light Emitting Diode) located on the base of the sensor can give valuable feedback.

Display	Meaning
Blinking Green	Initial startup routine. No data is output at this time.
Solid Blue	Normal operation.
Solid Red	Internal communication error.

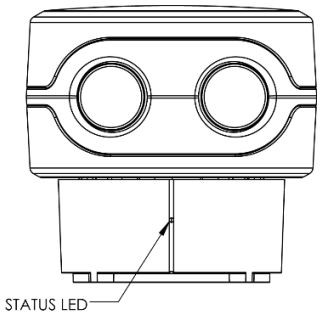


Figure 7, Status LED Location

Mounting Features and Orientation

Sweep has four brass threaded inserts designed to fit M2.5X0.45 screws in its base. These threaded holes are the way to mount Sweep to your device. The threaded holes are aligned with the scanner's measurement angles. The scanner's zero-degree measurement starting angle is aligned with the status LED, as shown in Figure 8.

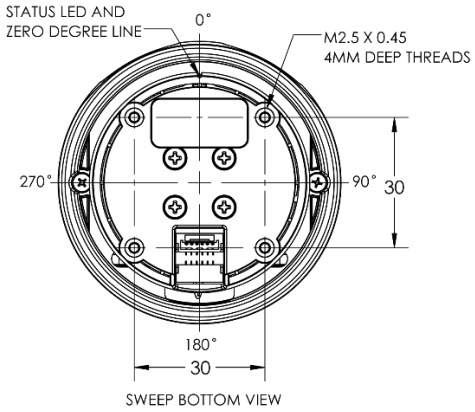


Figure 8, Sweep Standard Mounting Features

Mounting and Vibration Considerations

Sweep can be mounted in any orientation. Sweep's rotating head is dynamically balanced, which means it is immune to linear vibration, but it can be affected by rotational vibration. Sudden rotational shocks can cause the head to either slow down or speed up, which can affect angular measurements. If Sweep is rotationally jerked hard enough, it can cause the motor to lose sync, which will trigger a momentary motor pause, and then restart.

Mounting Sweep with Bottom Removed

If space is limited, the scanner's plastic bottom piece can be removed, and the internal motor mounting holes can be used to mount the device instead, as shown in Figure 9. Room must be provided for airflow around the scanner's head, and care must be taken not to damage the delicate circuit board components. Using the scanner in this configuration should only be considered by professionals. A 3D model that makes these mounting dimensions clear can be found on our downloads page.

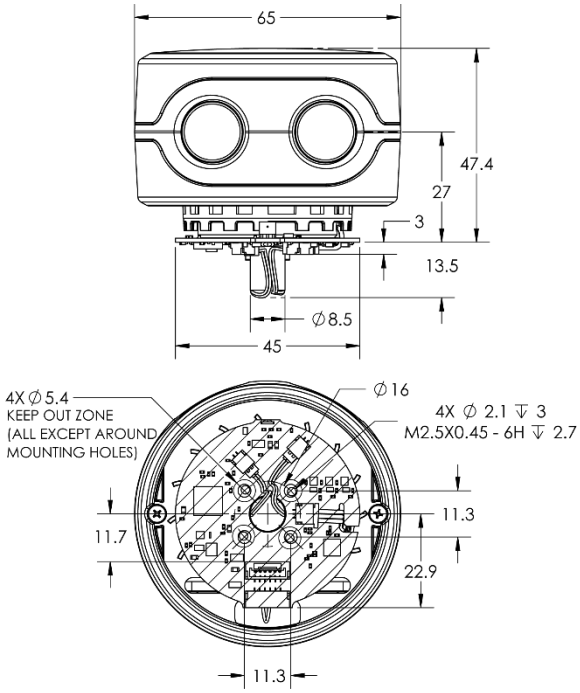


Figure 9, Sweep Internal Mounting Features

ALL DIMENSIONS ARE IN MM, DRAWINGS ARE NOT TO SCALE

Ingress Protection Rating

Sweep is rated as IP51, which is to say, it is not dust or water tight. **It is recommended that Sweep be placed inside a protective transparent enclosure if it will be used in dusty or wet environments.**

Enclosure Window Design

Sweep uses 905nm laser light, which passes through several kinds of clear glass and plastic very well. Based on our testing, clear Polycarbonate plastic is one of the best choices, as it can be molded to fit the profile of the application's enclosure, is very inexpensive, and in most cases, is more than 95% translucent to Sweep's light beam. Factors that can affect the performance of a window are:

- Thickness of the window. Thicker windows will block more light, as well as bend the light more if the beam is not hitting the window normal to the surface.
- Scratches and dust. The presence of scratches and dust on the window will scatter the laser light, and may reflect some of the light back into the sensor's detector, causing measurement errors.
- Surface coatings. There are a variety of coatings that can help with the performance of windows. One is an anti-reflective (AR) coating, which can help reduce the amount of laser light that is reflected as it passes through the window's surface.

Theory of Operation

Distance Measurement

Sweep employs a time of flight ranging method. This technique involves transmitting a packet of micro pulses of light in a unique pattern. When this light bounces off an object and returns to the receiving detector, a correlation algorithm is used to identify the unique light pattern from ambient noise. **Each light packet is different from the last, which allows multiple Sweep sensors to operate adjacent to each other without interference**.

The light packets that Sweep uses can vary in length, which can affect accuracy of range measurements, as well as the maximum range and update rate. Under normal operation, Sweep limits the maximum time per measurement to a value determined by the sample rate set using the **LR** command (see LR packet structure description). If not enough light is returned from the environment, the measurement fails, and a 1 is returned as the range value. On the other hand, if a lot of light is returned from the environment, the correlation algorithm can reach its maximum accuracy early, and can return a range value more quickly. This is what makes the update rate of Sweep variable. The value of setting a slower sample rate using the **LR** command, is that more light will be gathered from a target, and the range measurements will be more accurate. The exact accuracy is determined by many factors, including the target surface characteristics and ambient noise, so we cannot give an exact number for relative accuracy between the different **LR** settings.

Angle Measurement

Sweep uses an optical encoder to measure the angle of the rotating sensor head. The angle that is recorded for a range data point is the angle the sensor is at when the measurement is completed. The beginning of the scan, and zero degrees is located where the status LED projects out of the base of the sensor, as indicated in Figure 7.

Predicted Life

Sweep has a predicted life of up to 45 million scan rotations. This life rating assumes the sensor is running at the default 5Hz scan rate and is operated intermittently for up to 1 hour at a time. It is highly recommended to shut off the sensor whenever it is not in use. This can be done by pulling the enable line on the main connector to ground, or commanding the sensor to operate at 0Hz using the motor speed command. Things that can reduce the life of the sensor are high vibration or high temperature environments, running the sensor continuously for long periods of time(>1hr), or running the sensor at higher speeds (>5Hz). A repair kit can be purchased from Scanse to replace the wear component and extend the life of your sensor.

Visualizer Overview

You can download the Sweep visualizer at www.scanse.io/downloads. The purpose of the Scanse visualizer is to provide a way to quickly evaluate Sweep's performance in your application/environment. It also contains a programming tool for updating Sweep's firmware. A full tutorial for using the visualizer can be found in software support section at support.scanse.io.

Serial Protocol Specification

Specification	Value
Bit Rate	115.2 Kbps
Parity	None
Data Bit	8
Stop Bit	1
Flow Control	None

Data Encoding and Decoding

All characters used for commands and responses are ASCII code in addition to CR and LF, except for the measurement packet.

Communication Format

All communication packets between the host computer and the sensor begin with ASCII letter command codes.

General Communication Packet Structure

(HOST -> SENSOR)

Command with no parameter

Command Symbol (2 bytes)	Line Feed (1 byte)
Example: DS, DX, MI, IV…	Line Feed (LF) or Carriage Return (CR)

or

Command with parameter

Command Symbol (2 bytes)	Parameter (2 bytes)	Line Feed (1 byte)
Example: MS, LR…	Example: '03'	Line Feed (LF) or Carriage Return (CR)

(SENSOR -> HOST)

Response with no parameter echoed

Command Symbol (2 bytes)	Status (2 bytes)	Sum of Status (1 byte)	Line Feed (1 byte)

or

Response with parameter echoed

Command Symbol (2 bytes)	Parameter (2 bytes)	Line Feed (LF)	Status (2 bytes)	Sum of Status (1 byte)	Line Feed (1 byte)

Definition of terms:

Command Symbol: 2 byte code at the beginning of every command (ASCII)

ASCII Code (2 bytes)	Function
DS	Start data acquisition
DX	Stop data acquisition
MZ	Motor Ready
MS	Adjust Motor Speed
MI	Motor Speed Info
LR	Adjust LiDAR Sample Rate
LI	LiDAR Sample Rate Info
IV	Version Info
ID	Device Info
RR	Reset Device

Parameter: Information that is needed to change sensor settings (ASCII).. Example: a motor speed code 05 transmitted as ASCII parameter '05', which has byte values [48, 53] in decimal.

Line Feed (LF) or Carriage Return (CR): Terminating code. Command can have LF or CR or both as termination code but receipt will always have LF as its termination code.

Status: 2 bytes of data used to convey the normal/abnormal processing of a command. ASCII byte values of '00' or '99' indicate that the sensor received and processed the command normally. Value of '11' specifies an invalid parameter was included in the command. Any other byte values are reserved for communicating other errors or failure that are command specific. Usually these indicate a failure to process the command, often for valid reasons.

Sum of Status: 1 byte of data used to check for corrupted transmission. See Appendix for instructions for authenticating receipts.

DS - Start data acquisition

- Initiates scanning
- Sensor responds with header containing status.
- Sensor begins sending constant stream of Data Block receipts, each containing a single sensor readings. This stream continues indefinitely until the host sends a DX command.

(HOST -> SENSOR)

D	S	LF

(SENSOR -> HOST)

Header response

D	S	Status (2 bytes)	SUM (1 byte)	LF

The DS command is not guaranteed to succeed. There are a few conditions where it will fail. In the event of a failure, the two status bytes are used to communicate the failure.

Status Code (2 byte ASCII code):

- '00': Successfully processed command. Data acquisition effectively initiated
- '12': Failed to process command. Motor speed has not yet stabilized. Data acquisition NOT initiated. Wait until motor speed has stabilized before trying again.
- '13': Failed to process command. Motor is currently stationary (0Hz). Data acquisition NOT initiated. Adjust motor speed before trying again.

(SENSOR -> HOST)

Data Block (7 bytes) Data Block

Sync/Error (1 byte)	Azimuth - degrees(float) (2 bytes)	Distance - cm(int) (2 bytes)	Signal Strength (1 byte)	Checksum (1 byte)

Data Block Structure:

The Data Block receipt is 7 bytes long and contains all the information about a single sensor reading.

- **Sync/Error Byte:** The sync/error byte is multi-purpose, and encodes information about the rotation of the Sweep sensor, as well as any error information. Consider the individuals bits:

e6	e5	e4	e3	e2	e1	e0	sync

  - **Sync bit**: least significant bit (LSB) which carries the sync value. A value of 1 indicates that this Data Block is the first acquired sensor reading since the sensor passed the 0 degree mark. Value of 0 indicates all other measurement packets.
  - **Error bits**: 7 most significant bits (e0-6) are reserved for error encoding. The bit e0 indicates a communication error with the LiDAR module with the value 1. Bits e1:6 are reserved for future use.
- **Azimuth:** Angle that ranging was recorded at (in degrees). Azimuth is transmitted as a 16 bit fixed point value with a scaling factor of 16 (4bit after radix). Note: the lower order byte is received first, higher order byte is received second. Use instructions in the Appendix.
- **Distance:** Distance of range measurement (in cm). Distance is a 16 bit integer value. Note: the lower order byte is received first, higher order byte is received second. Use instructions in the Appendix.
- **Signal strength :** Signal strength of current ranging measurement. Larger is better. 8-bit unsigned int, range: 0-255
- **Checksum:** Calculated by adding the 6 bytes of data then dividing by 255 and keeping the remainder. (Sum of bytes 0-5) % 255 ... Use the instructions in the Appendix.

DX - Stop data acquisition

Stops outputting measurement data.

(HOST -> SENSOR)

D	X	LF

(SENSOR -> HOST)

D	X	Status (2 bytes)	SUM	LF

MZ – Motor Ready

Returns a ready code representing whether or not the device is ready. A device is ready when the calibration routine is complete, and that the motor speed has stabilized to its current setting. Intended use involves checking whether device is ready before sending a "DS" or "MS" command.

(HOST -> SENSOR)

M	Z	LF

(SENSOR -> HOST)

M	Z	Ready Code (2 bytes)	LF

Ready Code (2 byte ASCII code, ie: '01' = 0x3031):

- '00' : Device is ready
- '01' : Device is NOT ready

Whenever Sweep changes motor speed, it performs a calibration routine to account for inconsistencies in encoder which helps the sweep produce accurate measurements. This calibration routine is initiated after:

- powering on the device
- after receiving any form of "Adjust Motor Speed - MS" command, regardless of the size of the adjustment (ie: even calling "MS" with the current motor speed will still trigger a calibration)

During this calibration routine, the LED on the face of the device will blink blue. Once the blue LED has stopped blinking, the calibration routine is complete and the motor is ready. This wait time also helps enforce that the motor speed has stabilized at the new setting before anything else.

Currently, the device cannot process certain types of commands while the calibration routine is underway. These types of commands include:

- Data Start - DS
- Adjust Motor Speed - MS

The MZ command allows the user to repeatedly query the motor speed state until the return code indicates the motor speed has stabilized. After the motor speed is noted as stable, the user can safely send commands like DS or MS.

MS – Adjust Motor Speed

Adjusts motor speed setting to the specified code indicating a motor speed between 0Hz and 10Hz. This sets the target speed setting, but the motor will take time (~6 seconds) to perform a calibration routine and stabilize to the new speed setting. The blue LED on the device will flash until the calibration is complete and the motor speed has stabilized. Once a speed is set, the sensor will always return to this speed, even after a power cycle (except when setting the speed to 0Hz – in which case it will go back to 5Hz after a power cycle).

(HOST -> SENSOR)

M	S	Speed Code (2 bytes)	LF

(SENSOR -> HOST)

M	S	Speed Code (2 bytes)	LF	Status (2 bytes)	Sum	LF

Speed Code (2 byte ASCII code, ie: '05' = 0x3035):

- '00' = 0Hz
- '01' = 1Hz
- '02' = 2Hz
- '03' = 3Hz
- '04' = 4Hz
- '05' = 5Hz
- '06' = 6Hz
- '07' = 7Hz
- '08' = 8Hz
- '09' = 9Hz
- '10' = 10Hz

MS command is not guaranteed to succeed. There are a few conditions where it will fail. In the event of a failure, the two status bytes are used to communicate the failure.

Status Code (2 byte ASCII code, ie: '11' = 0x3131):

- '00': Successfully processed command. Motor speed setting effectively changed to new value. Device still requires time to stabilize, and calibrate.
- '11': Failed to process command. The command was sent with an invalid parameter. Use a valid parameter when trying again.
- '12': Failed to process command. Motor speed has not yet stabilized to the previous setting. Motor speed setting NOT changed to new value. Wait until motor speed has stabilized before trying to adjust it again.

MI – Motor Information

Returns current motor speed code representing the rotation frequency (in Hz) of the current target motor speed setting. This does not mean that the motor speed is stabilized yet.

(HOST -> SENSOR)

M	I	LF

(SENSOR -> HOST)

M	I	Speed Code (2 bytes)	LF

See previous "MS – Adjust Motor Speed" command for a breakdown of the possible Speed Codes.

LR – Adjust LiDAR Sample Rate

Default Sample Rate: 500-600Hz. See Theory of Operation section for explanation of why there is a range of sample rate values.

(HOST -> SENSOR)

L	R	Sample Rate Code (2 bytes)	LF

(SENSOR -> HOST)

L	R	Sample Rate Code (2 bytes)	LF	Status (2 bytes)	Sum	LF

Sample Rate Code (2 byte ASCII code, ie: '02' = 0x3032):

- '01' = 500-600Hz
- '02' = 750-800Hz
- '03' = 1000-1075Hz

LR command is not guaranteed to succeed. There are a few conditions where it will fail. In the event of a failure, the two status bytes are used to communicate the failure.

Status Code (2 byte ASCII code, ie: '11' = 0x3131):

- '00': Successfully processed command. Sample Rate setting effectively changed to new value.
- '11': Failed to process command. The command was sent with an invalid parameter. Use a valid parameter when trying again.

LI – LiDAR Information

Returns current LiDAR Sample Rate Code in ASCII:

(HOST -> SENSOR)

L	I	LF

(SENSOR -> HOST)

L	I	Speed(Hz) (2 bytes)	LF

Sample Rate Code (2 byte ASCII code, ie: '02' = 0x3032):

- '01' = 500-600Hz
- '02' = 750-800Hz
- '03' = 1000-1075Hz

IV - Version Details

Returns details about the device's version information.

- Model
- Protocol Version
- Firmware Version
- Hardware Version
- Serial Number

(HOST -> SENSOR)

I	V	LF

(SENSOR -> HOST)

I	V	Model (5 bytes)	Protocol (2 bytes)	Firmware Version (2 bytes)	Hardware Version (1 byte)	Serial Number (8 bytes)	LF
Example: IVSWEEP01011100000001							
I	V	SWEEP	01	01	1	00000001	LF

ID - Device Info

Returns details about the device's current state/settings.

- Bit Rate
- Laser State
- Mode
- Diagnostic
- Motor Speed
- Sample Rate

(HOST -> SENSOR)

I	D	LF

(SENSOR -> HOST)

I	D	Bit Rate (6 bytes)	Laser state	Mode	Diagnostic	Motor Speed (2 bytes)	Sample Rate (4 bytes)	LF
Example: IV115200110050500								
I	D	115200	1	1	0	05	0500	LF

RR - Reset Device

Resets the device. Green LED indicates the device is resetting and cannot receive commands. When the LED turns blue, the device has successfully reset.

(HOST -> SENSOR)

R	R	LF

(SENSOR -> HOST)

No Response

Appendix:

Authenticating Receipts (Does not apply to Data Block)

Authentication of a valid receipt is accomplished by a checksum, which uses the Status (2 bytes) and Sum of Status (1 byte) from the receipt. To perform the checksum, the received Sum of Status bytes is checked against a calculated sum of the 2 Status bytes. The protocol design allows for performing the checksum visually in a terminal, which requires all bytes in a receipt to be legible ASCII values (ex: '00P'). Therefore, performing the checksum in code is not intuitive. It works like this:

- The status bytes are summed
- The lower 6 bits (Least Significant) of that sum are then added to 30Hex
- The resultant value is compared with the checksum byte

```
//statusByte#1 + statusByte#2
let sumOfStatusBytes = status1_byteValue + status2_byteValue;
//grab the lower (least significant) 6 bits by performing a bit-wise AND with 0x3F (ie: 00111111)
let lowerSixBits = sumOfStatusBytes & 0x3F;
//add 30Hex to it
let sum = lowerSixBits + 0x30;
return ( sum === checkSumByteValue );
```

Example: Consider the common case of `'00P'` (decimal -> `[48, 48, 80]`, hex -> `[0x30, 0x30, 0x50]`)

```
0x30 + 0x30 = 0x60 // sum of the status bytes
0x60 & 0x3F = 0x20 // retrieve only the lower 6 bits
0x20 + 0x30 = 0x50 // calculate the ASCII legible sum
0x50 = 'P'         // translate to ASCII
```

Parsing Data Block 16-bit integers and floats

The Data Block receipt includes int-16 and fixed-point values (distance & azimuth). In the case of distance, the value is a 16-bit integer. In the case of the azimuth, the value is a fixed-point with a scaling factor of 16.

A 16-bit int is sent as two individual bytes. The lower order byte is received first, and the higher order byte is received second. For example, parsing the distance:

```
//assume dataBlock holds the DATA_BLOCK byte array
//such that indices 3 & 4 correspond to the two distance bytes
let distance = (dataBlock[4] << 8) + (dataBlock[3]);
```

For fixed-point values (azimuth), start by using the same technique to acquire a 16-bit int. Once you have it, you can perform the conversion to fixed-point value like so:

```
//assume dataBlock holds the DATA_BLOCK byte array,
//such that indices 1 & 2 correspond to the two azimuth bytes
let angle_int = (dataBlock[2] << 8) + (dataBlock[1]);
let degree = angle_int/16.0;
```

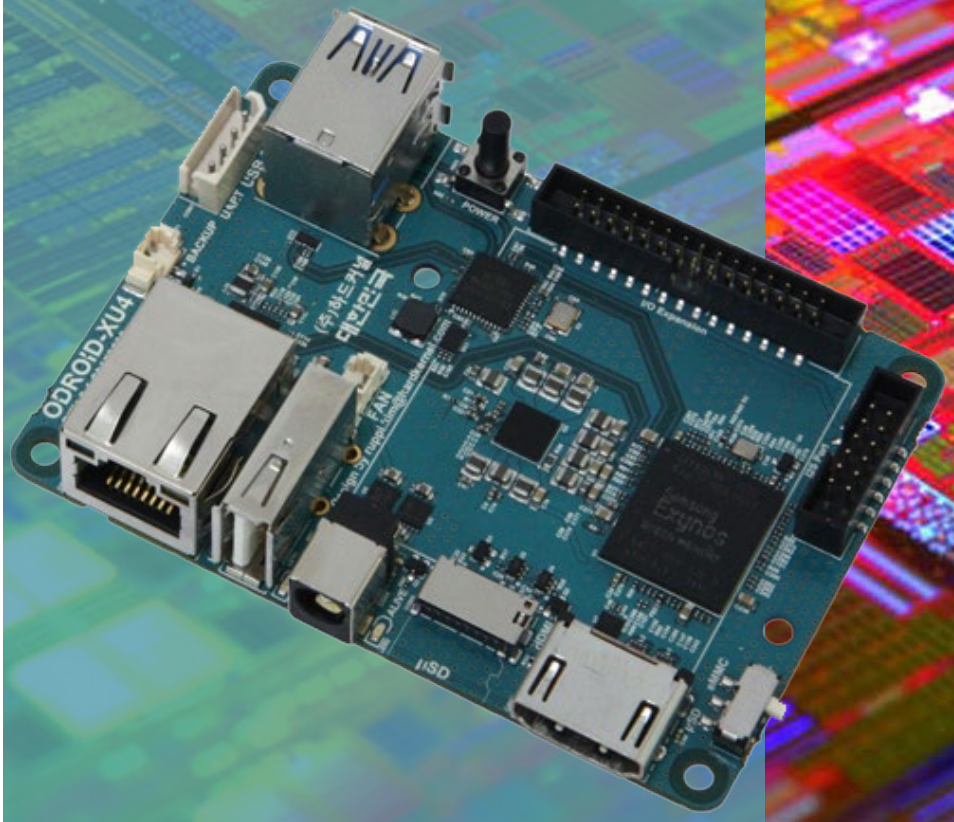
Performing Data Block Checksum

The last byte of a Data Block receipt is a checksum byte. It represents the sum of all bytes up until the checksum byte (ie: the first 6 bytes). Obviously, a single byte caps at 255, so the checksum can't reliably hold the sum of 6 other bytes. Instead, the checksum byte uses the modulo operation and effectively contains the remainder after dividing the sum by 255 (this is the same as saying sum % 255). Validating the checksum looks like:

```
//ONLY applies to receipts of type ReceiptEnum.DATA_BLOCK
//calculate the sum of the specified status or msg bytes
let calculatedSum = 0;
for(let i = 0; i < 6; i++){
    //add each status byte to the running sum
    calculatedSum += dataBlock[i];
}
calculatedSum = calculatedSum % 255;
let expectedSumVal = dataBlock[6];
return calculatedSum === expectedSumVal;
```

HARDKERNEL

USER MANUAL



ODROID-XU4

rev. 20170310

Index

Credits

Authors:	Rob Roy Venkat Bommakanti
Art Editor:	Bruno Doiche
Technical Editors:	Tobias Schaaf Saleem Almajed

What we stand for.

We strive to symbolize the edge of technology, future, youth, humanity, and engineering.

Our philosophy is based on Developers.
And our efforts to keep close relationships with developers around the world.

For that, you can always count on having the quality and sophistication that is the hallmark of our products.

Simple, modern and distinctive.
So you can have the best to accomplish everything you can dream of.

HARDKERNEL

Welcome

Congratulations on purchasing the ODROID-XU4! It is one of the most powerful low-cost Single Board computers available, as well as being an extremely versatile device. Featuring an octa-core Exynos 5422 big.LITTLE processor, advanced Mali GPU, and Gigabit ethernet, it can function as a home theater set-top box, a general purpose computer for web browsing, gaming and socializing, a compact tool for college or office work, a prototyping device for hardware tinkering, a controller for home automation, a workstation for software development, and much more.

Some of the modern operating systems that run on the ODROID-XU4 are Ubuntu, Android, Fedora, ARCHLinux, Debian, and OpenELEC, with thousands of free open-source software packages available. The ODROID-XU4 is an ARM device, which is the most widely used architecture for mobile devices and embedded 32-bit computing.

Chapter 1

Differences between a typical PC and a Single Board Computer (SBC)

If you are used to using a standard PC such as an OSX or Windows machine, there are a few small differences to note when transitioning to an ARM device. To begin with, the speed of an ARM processor is not directly comparable to the speed of an Intel processor. Because of the efficiency of the CPU, the XU4 can give great response time that feels just as fast as using a more expensive computer. The operating systems available for the XU4 are also highly optimized, and benefit from the expertise of many open-source contributors that continually review each others' work that bring daily improvements to the OS.

In addition, nearly all of the applications available for the XU4 also have their source code publicly available, which means that you can freely modify and update the applications to fit your specific needs. Program authors often maintain a GitHub repository, where suggestions can be submitted, reviewed and distributed to all of the application's users.

The XU4 also uses Solid State technology for its storage media, although a conventional hard disk may be used as an auxiliary device. The boot partition can be stored on either a microSD card or the much faster eMMC module, and Hardkernel's products have the unique distinction of supporting removable eMMC modules, so that operating systems may be switched out conveniently and easily. An eMMC module is a type of storage typically used in a smart phone, and is one of the more advanced compact media devices available.

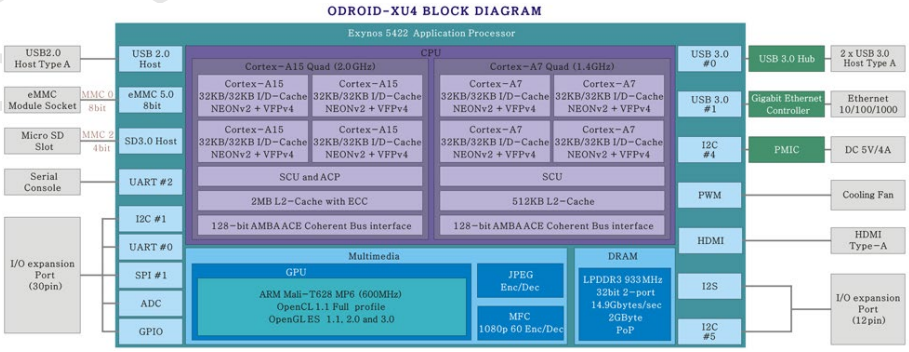
The power consumption of a typical personal computer can be anywhere between 100W and 1000W or more, depending on the peripherals, processor and type of power supply used. However, the ODROID-XU4 uses between 10W and 20W, greatly reducing your electricity bills, as well as allowing unique power configurations such as compact solar power cells and long-running batteries.

Components Included on an SBC

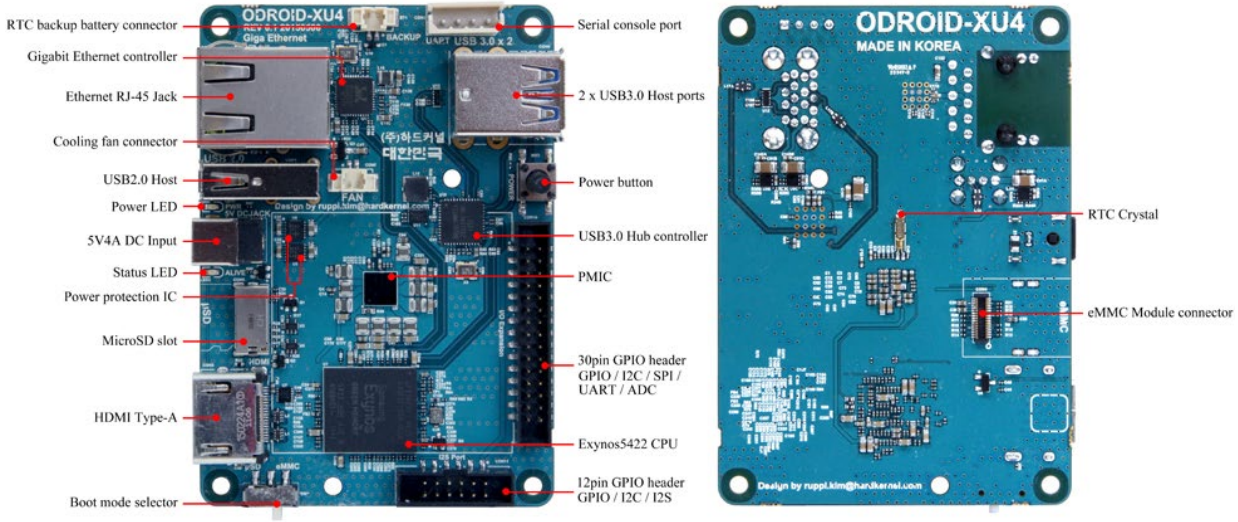
The ODROID-XU4 contains many of the same connections as a typical computer, with 1 USB 2.0 port, 2 USB 3.0 ports, an Ethernet port that supports Gigabit transfer speeds, an HDMI connector for 720p and 1080p monitors, and a 5V/4A DC power connector. In addition to these standard inputs, the XU4 also includes a 40-pin GPIO port, an external RTC battery connector, a USB-UART serial console port, an eMMC module connector, and a dedicated slot for a microSD card. For more details, refer to the ODROID-XU4 introductory videos at `https://youtu.be/wtqfC9v0xB0` and `https://youtu.be/lUchfyTpOjU`.

Block Diagram

The following diagram illustrates conceptually how the components of the XU4 fit together:



XU4 Block Diagram and Annotated Board Image



Power Supply

The XU4 board requires a 5V/4A DC power source. The dedicated power connector (miniature barrel jack) can accept a DC plug cable with a plug that has an outer diameter of 5.5mm and an inner diameter of 2.1mm. The plug inner core (center) is positive (attached to the RED wire in the cable) and outer cylinder is negative (attached to BLACK wire in the cable). The XU4 can be powered using different options, which are outlined below.

5V DC 4A Power Supply

Attach the plug to the power connector on the XU4. Plug the 2-pin PSU into the power outlet. The pins are Asian standard, and you may need an adapter to use in your region, such as the Americas. The PSU pictured above is available from Hardkernel.

DC plug cable

If you have a 5V DC 4A PSU which does not have the required plug, you can cut off the plug from such a power supply. Expose about ½" of the red/black wires on the psu cable and attach them to the same colored cables of this cable, then solder the joints. You can cover the joint using electrical tape or a heat-shrink wrap. Attach the plug to the XU4 and insert the PSU pins into a power outlet.

This cable is also available from Hardkernel, and may be paired with the SmartPower peripheral, which is an excellent bench power supply with variable voltage.

Monitor

The XU4 offers an HDMI port for connecting an HDMI-compliant monitor. It is recommended to use the Hardkernel supplied HDMI cable, but many other high quality standard cables should also work. There are some reported issues with cables that lack HDMI grounding wires inside the cable, so it is best to order this cable directly from Hardkernel or one of its certified distributors.

The image below shows the use of an ODROID-VU HDMI touchscreen monitor supporting 10-point touch control.

ODROID-VU HDMI Monitor

Keyboard and Mouse

Nearly all USB HID-compliant keyboard and mouse will work when connected to one of the four USB ports. The use of a bluetooth mouse or keyboard requires a bluetooth dongle and, for first pairing, either an USB mouse/keyboard, an SSH access from remote or a working touchscreen.

To pair a bluetooth keyboard or mouse via the Linux console, run the following command in a Terminal window:

```
$ sudo hcitool scan
```

Push the Connect button on the bluetooth keyboard or mouse to initiate a connection with the ODROID, and the following output should appear in the console:

```
Scanning ...
XX:XX:XX:XX:XX:XX     Rapoo E6700
$ sudo bluez-simple-agent hci0 XX:XX:XX:XX:XX:XX
```

Push the Connect button again, and enter the shown pin on the keyboard, followed by the Enter key. If no pin is shown, try 000000. Then, type the following to trust the device and restart the bluetooth service:


```
$ sudo bluez-test-device trusted XX:XX:XX:XX:XX:XX yes
$ sudo /etc/init.d/bluetooth restart
```

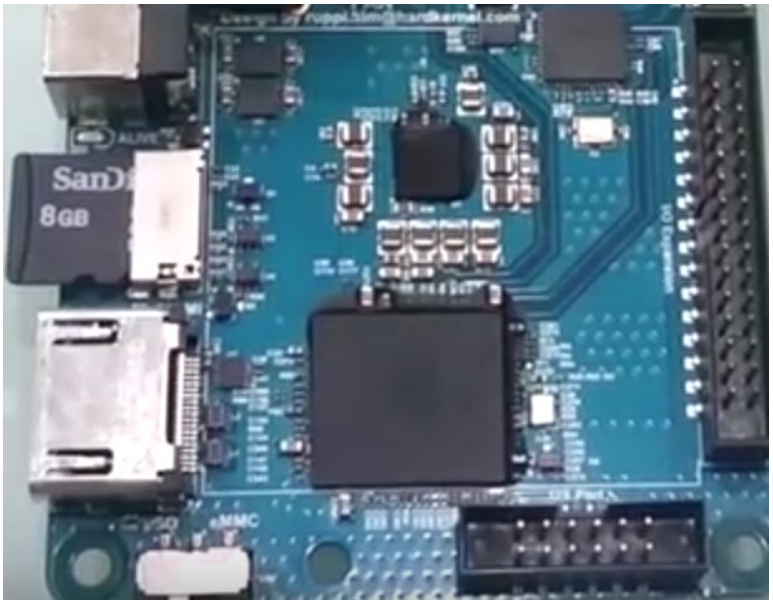
In Android, the Settings application may be used to connect to the bluetooth keyboard or mouse. More details on using bluetooth with Android are covered in Chapter 2.

Ethernet and WiFi

The Ethernet port accepts a standard Ethernet cable, and is capable of up to 1 GB (1000 MB) per second transmission rate. The WiFi USB adapter fits in any of the 3 USB ports.

MicroSD Card

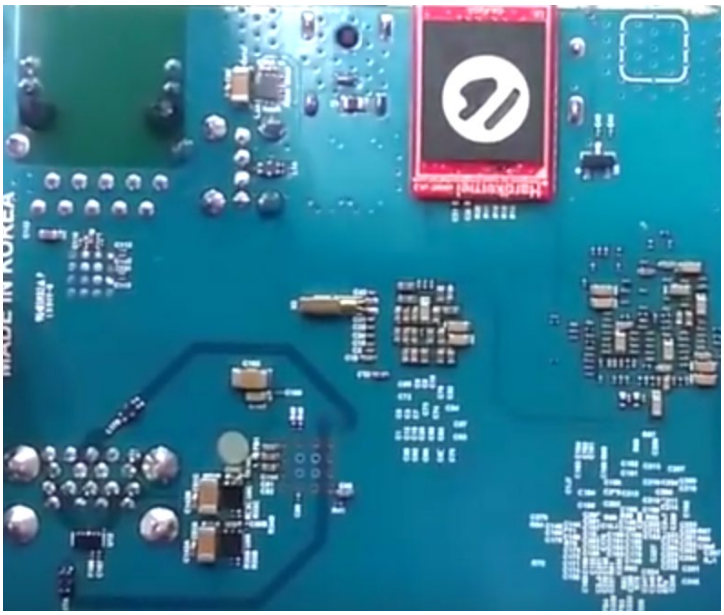
Align the metal strips of the microSD card with the pins of the microSD card connector, and slowly push it in until it clicks in place. Be gentle. If you are unable to push it in, it may be misaligned. Recheck and flip the microSD card if you notice a wrong insertion direction. The image below shows a properly mounted microSD card.



XU4 closeup with microSD card attached

eMMC Module

Align the eMMC module and the eMMC connector on the XU4 board, using the white rectangle on the PCB as a guide. The female portion of the eMMC module should line up with the male connector on the board. Slowly push it in until it clicks in place. Be gentle. If you are unable to push it in, it may be misaligned. Recheck and turn the eMMC module if you notice a wrong insertion direction. The writing on the card will be exposed after insertion. The image below depicts how the eMMC module appears after it is mounted.



XU4 closeup with eMMC module attached

LED status

The ODROID-XU4 includes several LED lights that indicate the status of the device:

The red LED
• Is on when power is available
The blue LED
• Is on (solid light) when the bootloader is running
The blue LED
• Blinks slowly when the kernel is running, like a heartbeat
The blue LED
• Blinks quickly when the kernel is in panic mode

Chapter 1

Technical specifications

Processor
Samsung Exynos5422 Cortex™-A15 2Ghz and Cortex™-A7 Octa core CPUs with Mali Mali-T628 MP6 GPU

Storage
There are two different methods of storage for the operating system. The first is by using a microSD Card and another is by inserting an eMMC module, which is normally used for storage for smartphones and digital cameras.

eMMC 5.0 module socket
8GB/64GB : Toshiba
16GB/32GB : Sandisk iNAND Extreme

The eMMC storage access time is 2-3 times faster than the SD card. You can purchase 4 size options: 8GB, 16GB, 32GB and 64GB. Using an eMMC module will increase speed and responsiveness, similar to the way in which upgrading to a Solid State Drive (SSD) in a typical PC also improves performance over a mechanical hard drive (HDD).

Micro Secure Digital (microSD) card slot
The ODROID-XU4 can utilize a newer UHS-1 SD model, which is about twice as fast as a class 10 card. There are some microSD cards which cause an additional boot delay time of around 30 seconds. According to our testing, most Sandisk microSD cards don't cause a long boot delay. The ODROID-XU4 model is compatible with a wide array of microSD cards, but class 10 cards or above are highly recommended.

5V 4A DC input
The DC input is for 5V power input, with an inner diameter of 2.1mm, and an outer diameter of 5.5mm.

USB host ports
There is one USB 2.0 host port and two USB 3.0 ports. You can plug a keyboard, mouse, WiFi adapter, storage or many other devices into these ports. You can also charge your smartphone with it! If you need more than 3 ports, you can use an external USB hub. A self-powered hub will also reduce the power load on the main device.

HDMI port

The XU4 model uses a standard Type-A HDMI connector.

Ethernet RJ-45 jack

The standard RJ45 Ethernet port for LAN connection supports 10/100/1000 Mbps speed. The green LED Flashes when there is 100 Mbps connectivity, and the yellow LED Flashes when there is 1000 Mbps connectivity.

General Purpose Input and Output (GPIO) ports

The 30-pin GPIO port can be used as GPIO/IRQ/SPI/ADC, and the 12-pin GPIO port can be used as GPIO/I2S/I2C for electronics and robotics. The GPIO pins on an ODROID-XU4 are a great way to interface with physical devices like buttons and LEDs using a lightweight Linux controller. If you're a C/C++ or Python developer, there's a useful library called WiringPi that handles interfacing with the pins, which is described in Chapter 4. Note that all of the GPIO ports are 1.8Volt, and the ADC inputs are limited to 1.8Volt. If a sensor or peripheral needs higher voltage, the GPIO ports may be level-shifted to 3.3V or 5V using the XU4 Level Shifter Shield.

Serial console port

Connecting to a PC gives access to the Linux console. You can monitor the boot process, or to log in to the XU4 to perform root maintenance. Note that this serial UART uses a 1.8 volt interface, and it is recommended to use the USB-UART module kit available from Hardkernel. A Molex 5268-04a (2.5mm pitch) is mounted on the PCB, and its mate is Molex 50-37-5043 Wire-to-Board Crimp Housing.

RTC (Real Time Clock) backup battery connector

If you want to add a RTC functions for logging or keeping time when offline, just connect a Lithium coin backup battery (CR2032 or equivalent). All of the RTC circuits are included on the ODROID-XU4 by default. It connects with a Molex 53398-0271 1.25mm pitch Header, Surface Mount, Vertical type (Mate with Molex 51021-0200).

Gigabit Ethernet

The Realtek RTL8211F is a highly intergrated 10/100/1000M Ethernet transceiver that complies with 10Base-T, 100Base-TX, and 1000Base-T IEEE 802.3 standards.

USB MTT hub controller

The Genesys GL3521 is a 2-port, low-power, and configurable USB 3.0 SuperSpeed hub controller.

USB VBUS controller

A NCP380 Protection IC for USB power supply from OnSemi.

Boot media selector

The eMMC/SD card switch on the side of the board selects the boot media.

Power supply circuit

Discrete DC-DC converters LDOs are used for CPU/DRAM/IO power supply.

Power protector IC

The power protected is a NCP372 over-voltage, over-current, reverse-voltage protection IC from OnSemi.

Heat Sink and fan

Electronic components all generate heat while operating, and Electronic components all generate heat while operating, and different components generate different levels of heat. Some components do not require any cooling, while others do.

Complex components such as the XU4 processor may reach temperatures as high as 95°C. At high temperatures, the processor will throttle itself and operate slower so that temperatures do not continue to increase. Some owners prefer that the temperatures do not reach such high levels and install a heat sink, which is available from the Hardkernel store. Heat transfer from components to the surrounding air is related to the surface area available to transfer heat to the surrounding air. The processor of the XU4 provides a relatively small area to dissipate heat. The heat sink is much larger and is therefore able to dissipate more heat into the surrounding air than the processor itself.

The fan provides additional cooling by drawing air across the heat sink, and is controlled through software to vary the amount of cooling depending on the temperature of the heat sink.

Chapter 1

Frequently Asked Questions

What's an ODROID?
ODROID means Open + Android. It is a development platform with hardware as well as software.

Why is the company named Hardkernel?
Hardkernel produces both hardware and the associated Linux kernels.

What SoC are you using?
The SoC is a Samsung Exynos5422 Octa.

What GPU does it include?
An ARM Mali-T628 6 Core.

How does it boot?
You may boot from either microSD card or eMMC module.

Can I buy OS pre-loaded SD card or eMMC?
Yes, they are available from Hardkernel as well as other distributors.

Can I add extra RAM memory?
No. The RAM is not removable or swappable.

How can I get the Ubuntu or Android BSP source code?
The U-boot, Kernel and OS source code are released via Github from the first shipping date at `http://github.com/hardkernel`.

Does it play 1080p content well on Ubuntu Linux?
Yes, H.264/H.265/VXU4/MPEG4/MPEG2 video clips are playable with Kodi (formerly XBMC) in most cases.

How can I root Android?
Android for the ODROID is unlocked and rooted by default for development.

How can I install the Google Play Store?
It is very simple. Just download an installer from `http://bit.ly/1gkv4PM`, click the APK, and follow the instructions inside the application.

Can I get a PCB layout file and gerber file?
The ODROID project is not a full open source hardware, and only the schematics are released to the public.

What peripherals are available?
The following peripherals are available from the Hardkernel store and many of the Hardkernel certified distributors:

- WiFi Module
- Bluetooth Module
- HDMI Cable
- MicroSD 8GB, 16GB cards (with a pre-installed OS)
- eMMC 8GB, 16GB, 32GB, 64GB module (with a pre-installed OS)
- Shifter Shield
- ODROID-SHOW2
- USB-UART Module Kit
- USB-CAM 720p
- USB GPS Module
- Weather Board
- Expansion Board
- USB IO Board
- myAHRS+
- ODROID-SPDIF
- Cloudshell
- USB Audio Adapter
- Backup Battery for RTC
- ODROID Smart Power Supply
- ODROID-VU5, VU7 and VU8 capacitive touchscreen kits

What are the minimum peripherals to run Ubuntu or Android with an ODROID-XU4?

The following peripherals are not included with the basic ODROID-XU4 board, but may be purchased separately from the Hardkernel store, or from a certified distributor:

- HDMI monitor for output device & HDMI Cable
- Mouse for input device
- MicroSD card for the operating system and user data (8GB or higher is required) or an eMMC module
- MicroSD card reader to install the operating system image
- Ethernet cable
- Power supply: DC 5V/4A

I am a platform/OS developer. What should I buy with the ODROID-XU4?
You should obtain the minimum peripherals, along with the USB-UART module kit for debugging and accessing the system console.

I have a USB-Serial converter. Should I buy your USB-UART module kit?
We strongly recommend using our USB-UART module kit because it includes the proper connector and voltage.

How do I access the Internet with an ODROID-XU4?
Use a 10/100/1000 LAN Ethernet connection, or purchase the WiFi module kit for a wireless connection.

What display can I use?
There is an HDMI Type-A output port on the ODROID-XU4. Below is a list of resolutions that are currently supported:

- 1920x1200 *(WUXGA)*
- 1920x1080 *(1080p)*
- 1280x720 *(720p)*
- 720x480 *(480p)*
- 720x576 *(576p)*
- 1280x800 *(800p for ODROID-VU)*
- 1280x1024 *(SXGA)*
- 1024x768 *(XGA)*
- 800x600 *(SVGA)*
- 800x480 *(WVGA)*
- 640x480 *(VGA)*

Can I use an HDMI to DVI converter?
An HDMI-DVI converter may work with many DVI monitors, but a few of them will not work due to compatibility issues. We recommend our HDMI LCD kit (ODROID-VU) that includes a capacitive touch screen if you want to develop a modern user interface.

Is there a touch screen on the HDMI LCD Kit?
Yes. It supports 10-point multi-touch via a standard USB interface.

Is sound over HDMI supported?
Yes.

Is SPDIF and optical pass-through supported?
It is possible with our USB to SPDIF interface, available at the Hardkernel store, but it only works with Kodi (XBMC) on Ubuntu. The Android platform does not yet support the 5.1 channel pass-through.

Is there any analog audio output or input?
Not on the board itself, but you can use our USB Audio Adapter for analog audio.

Is there an I2S port to connect a HiFi DAC?
No.

What is an eMMC module?
The eMMC module is a NAND flash-based storage IC which is mounted on a custom PCB for easier upgrade/replacement, and is much faster than SD card. The transfer rate of a typical eMMC is approximately 65MB/sec, while a microSD UHS-1 card is approximately 30MB/sec.

Do you have a SATA port?
No, but you can use a SATA-to-USB bridge device.

Which power adapter should I use?
The ODROID-XU4 consumes less than 1A in most cases, but it can climb to 4A if many passive USB peripherals are attached directly to the main board. It is recommended to use the Hardkernel 5V/4A PSU or USB-to-DC Plug cable with a 5V/4A charger. Due to the limited power output from a computer's USB port, we suggest only powering the ODROID-XU4 with a good quality 5V/4A PSU.

What operating system (OS) does it use?
We recommend Android and Ubuntu as our default distribution. The OS is stored on the SD-card / eMMC.

Which Android and Ubuntu version are included?
Android 4.4.x and Ubuntu 16.04, which both run on the Linux kernel 3.10 LTS. Newer OS and kernel versions will be made available on the XU4 wiki at `http://bit.ly/1kMUC27` as they are developed.

Which OpenGL and OpenCL versions are included in Android and Ubuntu?
OpenGL ES 1.0, 2.0, 3.0 and 3.1 are included. OpenCL is also fully supported.

Where is the OpenGL ES SDK?
Refer to the ARM Mali Developers site at http://bit.ly/1FRJEi0 for information on OpenGL ES.

Is the full source code open, and can I build it by myself?

Yes, the bootloader, kernel and OS platform source code are available. However, the GPU userland drivers are in binary format due to ARM's policies.

Do I need to patch for optional accessories that are purchased from the Hardkernel store?
No, they will work out-of-box with the kernels supplied with the operating system.

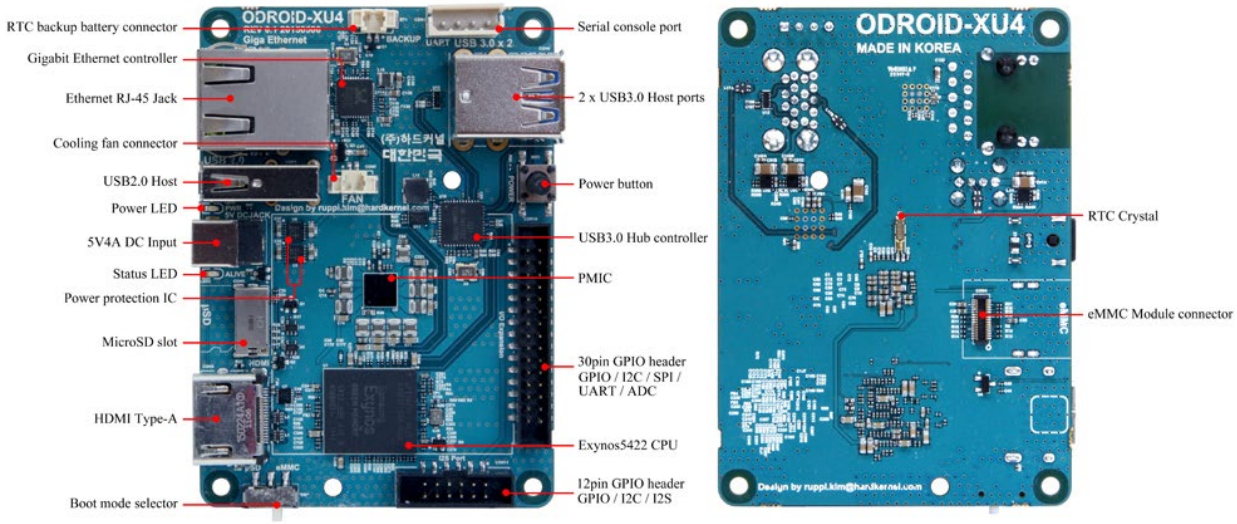
Does the device support networking?
Yes, there is 10/100/1000 RJ45 Ethernet port.

Is there WiFi?
WiFi is available via an optional USB dongle.

I still have more questions!
You may ask any ODROID-related questions in our user support forums at `http://forum.odroid.com`.

Getting Started

Given the introduction to the XU4 Single Board Computer (SBC) development board and the fact that you have all of the necessary peripherals, you must be excited to get started with your XU4 now. To be able to use the computer on the Internet, we first need an idea of how the XU4 will fit into a home computing network.

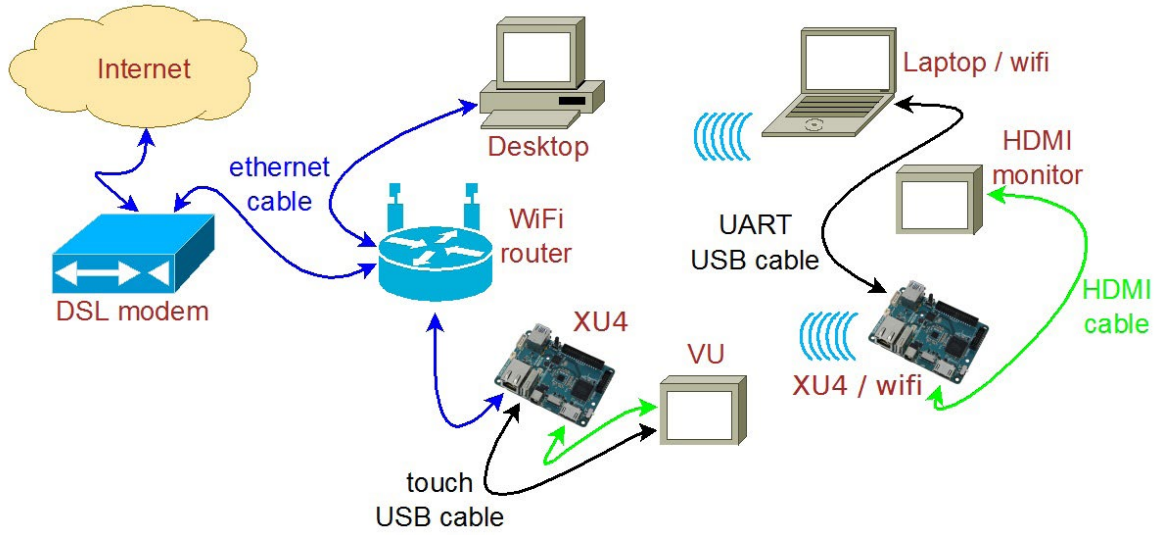


Basic XU4 board

Home Computing Network

A typical simplistic home computing network environment includes the following, as shown in the diagam below:

- A Digital Subscriber Line (DSL) modem, typically provided by your Internet Service Provider (ISP), which connects your network to the Internet,
- A single (2.4GHz) / Dual (2.4 and 5GHz) band 802.11b/g/n WiFi router, with at least four (4) Gigabit Ethernet (gigE) ports and an additional Wide Area Network (WAN) port, where the WAN port connects to the DSL modem,
- An Intel/AMD based laptop (running MS Windows, OSX or Linux), that connects via wired ethernet or WiFi to the router, and
- At least one XU4, that connects (wired ethernet or WiFi) to the router, which uses a High-Definition Multimedia Interface (HDMI) monitor (possibly touch-capable, such as an ODROID-VU).



Typical home computing network

After preparing the boot media, it is theoretically possible to use a XU4 in a network disconnected from the Internet. However, many use cases require an always-ON functioning secure connection to the Internet. In either case, it is presumed you have a fully functioning intranet (i.e., the internal network within the home). There are numerous online guides to setting up a secure network and as such, the setup of such a network is beyond the scope of this guide.

Preparing the Work Area

Your work area should be similar to the following:

- A well-lit and ventilated area,
- A flat-top desk spanning at least 2m x 1m with a height of 1m or appropriate to optimize posture and minimize overall fatigue,
- A freely available area of non-conductive surface of at least 1m x 1m, preferably covered by a well-grounded anti-static mat with you wearing the grounding wristband/cable attached to the mat. This safeguards the handling and placement of unprotected electronics such as XU4 board, etc., on this area,
- A desktop with accessible monitor, keyboard/mouse or a laptop to prepare the boot-media for the XU4 and debug the XU4 bringup process,
- If needed, a 4-port USB3 (USB2 compatible) hub attached to the computer within the reach of the free area - for easy image flashing using a USB-based SD/eMMC module reader/writer and attaching the USB UART debug cable, and
- A well-grounded surge-protector with 6 or more outlets, within reach of the free area.

The above criteria will go a long way in ensuring a safe workspace for you and your projects involving boards such as the XU4.

Flashing an Image

All SBCs require an operating system (OS) for booting. The operating system will be transferred to and reside on boot media, either built on to the board or attachable to the board. The process of placing the operating system on the boot media is termed flashing. Given this process, it is obvious that another computer system will be needed to flash the image onto the boot media, before the XU4 can boot up.

The flexible XU4 design allows you to use a boot media device that will be attached to it through an appropriate receptacles on the board. If you refer to the annotated board image, you can see that the XU4 supports two boot media types - microSD card or an eMMC module. They can be obtained from Hardkernel directly, either with Linux or Android, pre-installed. If you purchased one such device and if it has a factory-installed image, you can skip the flashing steps in this section and proceed to the next section.

Boot Media

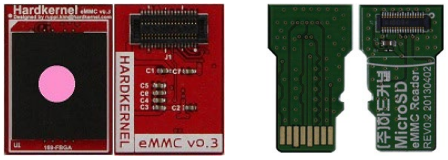
microSD/microSDHC card

- Slower, less longevity
- 8GB, 16GB, 32GB
- Linux, Android
- Requires USB reader/writer

16GB Linux MicroSD card and USB reader/writer

eMMC version 5.0 module

- Faster, more longevity
- 8GB, 16GB, 32GB, 64GB
- Linux, Android
- Requires eMMC/USB adapter
- Requires USB SD card reader/writer



16GB Linux eMMC and adapter

§ = Available from Hardkernel

Development boards such as the XU4, by definition, will at some point either require an OS upgrade or a simple reflash to revert to a known good base state. The section below describes how to flash the media.

USB microSD card adapter

No matter which boot media you select for your XU4, you will need an additional device to perform the flashing process, which is called a USB SD card reader / writer. It should be compatible with your host desktop or laptop (flashing computer) and with the OS running on it. The image above shows a typical microSD card adapter. If you recall in the previous section, we recommended the placement of a USB3 hub close to your free work area. This hub, if present, is where you would connect the USB SD card reader / writer.

Now, you may see the point of the suggested hub location - basically, it allows for frequent flashing of the boot media and its convenient attachment to the XU4. If all points in the connection are USB3 type, you will get the fastest possible read / write speeds. USB2 compatibility will allow one or more points to be of type USB2, but will result in slower speeds.

Start using the designated non-conductive free work area for all subsequent activities. In the microSD receptacle of the USB SD card reader / writer, look for the metal contact pins. Then, if you are using a microSD card to flash the image, align the metal strips of the microSD card to make contact with the pins. Insert the microSD card into the USB SD card reader / writer. The image below indicates the alignment when using the Transcend USB3 microSD card reader / writer model. The alignment may be different with your reader / writer model.

However, if you are using an eMMC module to flash the image, you first need to attach the eMMC module to the eMMC/microSD USB adapter. Place the eMMC module and the adapter such that the 34 pin female/male connectors overlap and gently press them together. Align the metal strips of the microSD adapter to make contact with the pins in the microSD receptacle of the USB SD card reader / writer. Insert the assembly by pressing firmly on the eMMC module until it clicks into place.

Preparing the eMMC module

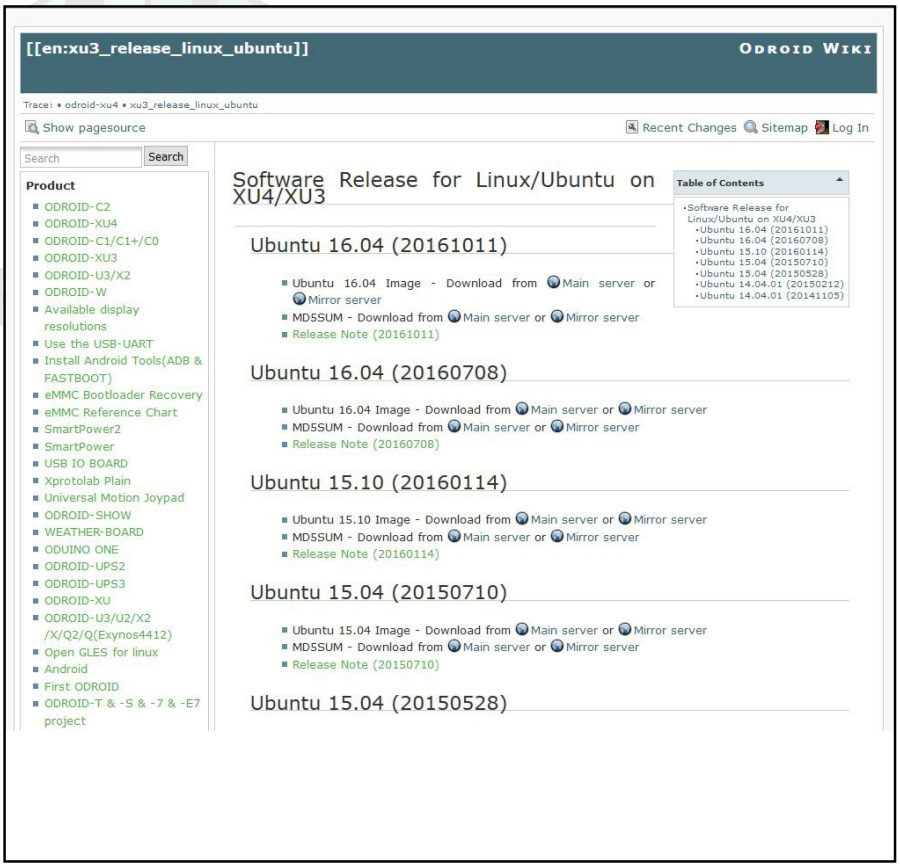
Shown above is the alignment when using the Transcend USB3 microSD card reader / writer model. The alignment may be different with your own microSD USB adapter. Next, ensure that the computer you are about to use to flash the image, has a functioning connection to the Internet. Now that you have the USB SD card reader / writer with the boot media inserted, attach the male USB end of the USB SD card reader / writer to the appropriate (USB3 or USB2) female port on the computer/USB3-hub.

The boot times are shortest for eMMC 5.0 modules, midway for class 10 microSD (microSDHC / microSDXC UHS-I) cards and longest for class 4 microSD cards.

Flashing an Image on a Microsoft Windows Computer

As soon as you attach the USB SD card reader / writer for the first time, the Windows OS will sense the presence of a new USB device through its plug-and-play (PNP) infrastructure. It will obtain the device identifiers from the device and search through its local database. If found, it will install it for the first time. If not found, it will request you to provide a location for the driver or seek to find it over the Internet. Once the driver is obtained for the first time, it will install it and request permission to reboot the system, which should be allowed. Once installed, it will use the driver on all subsequent sessions.

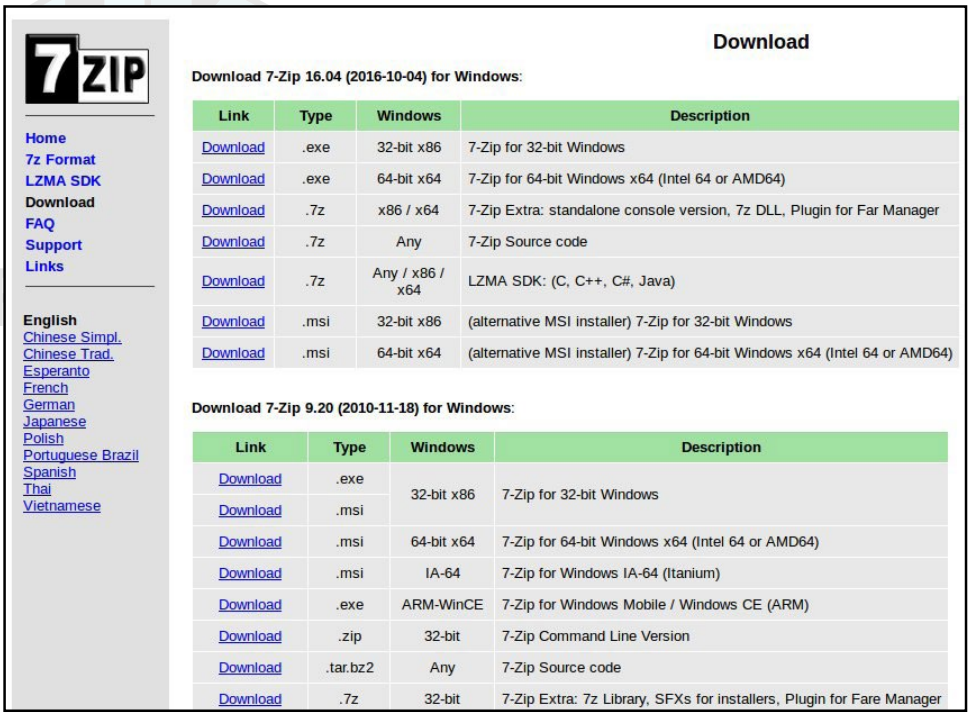
After the system has booted, launch a web-browser such as Firefox and navigate to `http://bit.ly/1kMUC27`, which will list all of the available Ubuntu images. Click on the link to download the latest Ubuntu 16.04 image, for example, `http://bit.ly/2kINgjl`.



List of Ubuntu images for the ODROID-XU4

Because the ODROID-XU4 is fully software compatible with the ODROID-XU3, many images listed on the Hardkernel website and the ODROID forums are labeled for the ODROID-XU3. Any image that was built for the ODROID-XU3 will work properly on the ODROID-XU4.

You will need to uncompress this file before flashing it to the boot media. To do so, you will need a specialized utility to uncompress the downloaded file. A reliable utility to uncompress such files is the 7-zip program. Launch a web-browser such as Firefox and navigate to `http://www.7-zip.org/download.html`, which will list all of the available installation files for this utility. Click on a link that corresponds to your host computer's OS. Details of your OS can be found by running either the msinfo32.exe or systeminfo.exe command in a command window. The Windows OS used for this guide is a 64-bit type. Click the download link that corresponds to the 64-bit Windows OS, in this case, 7z1509-x64.msi.



7ZIP

Home
7z Format
LZMA SDK
Download
FAQ
Support
Links

English
Chinese Simpl.
Chinese Trad.
Esperanto
French
German
Japanese
Polish
Portuguese Brazil
Spanish
Thai
Vietnamese

Download

Download 7-Zip 16.04 (2016-10-04) for Windows:

Link	Type	Windows	Description
Download	.exe	32-bit x86	7-Zip for 32-bit Windows
Download	.exe	64-bit x64	7-Zip for 64-bit Windows x64 (Intel 64 or AMD64)
Download	.7z	x86 / x64	7-Zip Extra: standalone console version, 7z DLL, Plugin for Far Manager
Download	.7z	Any	7-Zip Source code
Download	.7z	Any / x86 / x64	LZMA SDK: (C, C++, C#, Java)
Download	.msi	32-bit x86	(alternative MSI installer) 7-Zip for 32-bit Windows
Download	.msi	64-bit x64	(alternative MSI installer) 7-Zip for 64-bit Windows x64 (Intel 64 or AMD64)

Download 7-Zip 9.20 (2010-11-18) for Windows:

Link	Type	Windows	Description
Download	.exe	32-bit x86	7-Zip for 32-bit Windows
Download	.msi		
Download	.msi	64-bit x64	7-Zip for 64-bit Windows x64 (Intel 64 or AMD64)
Download	.msi	IA-64	7-Zip for Windows IA-64 (Itanium)
Download	.exe	ARM-WinCE	7-Zip for Windows Mobile / Windows CE (ARM)
Download	.zip	32-bit	7-Zip Command Line Version
Download	.tar.bz2	Any	7-Zip Source code
Download	.7z	32-bit	7-Zip Extra: 7z Library, SFXs for installers, Plugin for Fare Manager

List of 7-zip installation files

After the file is downloaded, run it and select the default options, and wait for 7-zip to finish installing. Next, launch the Windows Explorer application and browse the download directory. Select the downloaded compressed image file and right-click the file with the mouse. Select the 7-zip menu option and under that select the Extract Here option. After the uncompress process, it will result in the image file that can be flashed. In this example, it will be `ubuntu-16.04-mate-odroid-xu3-20161011.img`. Note that, although this image is an XU3 image, it is compatible with the XU4 board.

Another essential utility is one that will be used to flash the uncompressed image file, such as the Win32DiskImager utility. Hardkernel has an updated version of it that adds a useful step to the flash process, which clears the boot media prior to writing the image. This version of Win32DiskImager.exe can be downloaded from `http://bit.ly/1LVPcbF`. Uncompress the compressed image file using the 7-zip application, which generates the directory win32diskimager2-binary that includes the Win32DiskImager2.exe application. Move the entire directory to `C:\Program Files (x86)`:

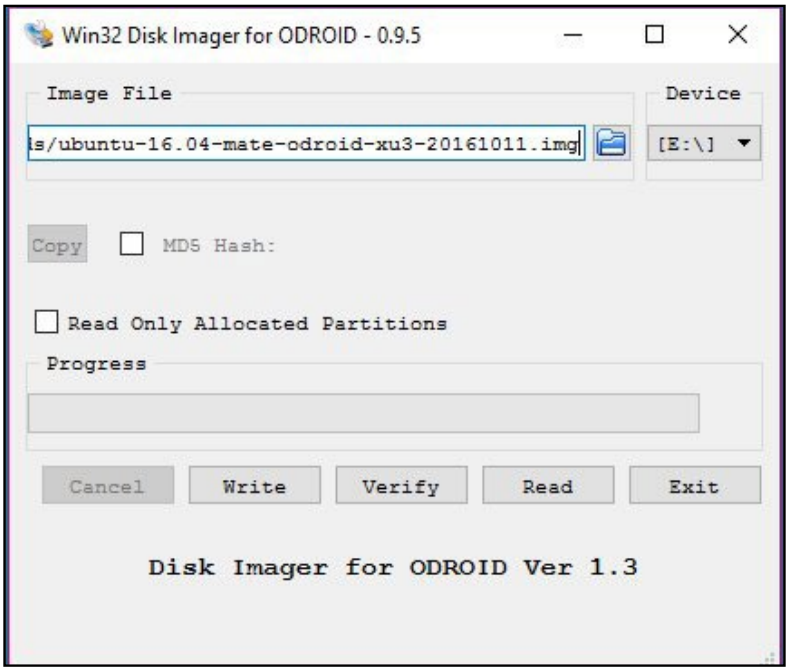
```
C:\Program Files (x86)\win32diskimager2-binary\
```

You will need to be logged in to Windows OS as an administrator to move this directory. Then, create a shortcut for the application on the desktop using the following link:

```
C:\Program Files (x86)\win32diskimager2-binary\Win32DiskImager2.
exe
```

Next, change the properties of the shortcut (right-click of mouse) so that it will run with Administrator privileges. Finally, double-click this shortcut in order to launch the Win32DiskImager2.exe application. Click the folder (directory) icon to browse and select the directory that has the uncompressed image file.

Select the uncompressed image file. Make sure that the drive letter of the selected device (in the example below, E:) corresponds to the boot media that was placed in the USB SD card reader / writer. The screenshot above shows the Win32DiskImager2 interface. Click on the Write button to start the flash process.



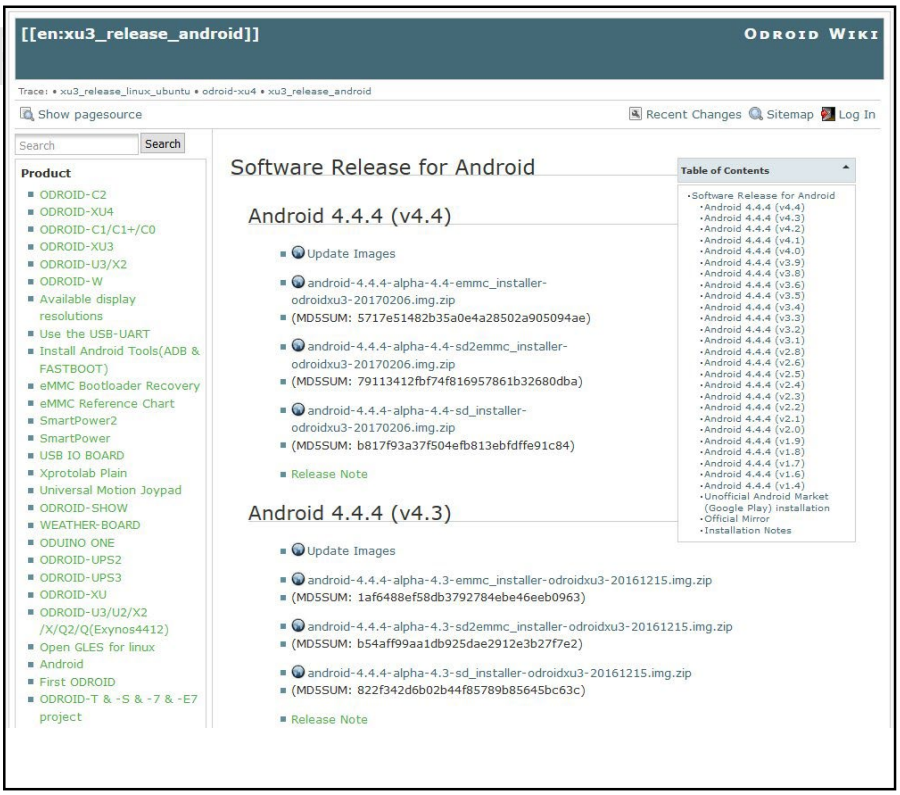
Flashing an image using Win32DiskImager2

You will be notified of the completion of the flash process. Launch the Windows Explorer application and right-click the device that was used in the flash process by Win32DiskImager2.exe with the mouse and select the Eject option. When permitted, remove the USB SD card reader / writer from the USB port of the computer. Remove the boot media and place it on the non-conductive surface.

The above steps are to flash the Linux image. What if you wish to flash an Android image? While you would download a different compressed file, all subsequent steps would be that same as those listed above.

You would need to to launch the browser once again and point it

to `http://bit.ly/1XwOatZ`. Look for the latest eMMC installer Android compressed image file link and click the link to download it. The screenshot below shows the webpage that lists the Android images. Once the compressed file has been downloaded, uncompress it and flash it, using the steps listed earlier in this section.



List of Android images for the ODROID-XU4

Flashing an Image on a Linux Ubuntu Computer

In your Ubuntu Linux desktop, launch a terminal window. Create a working directory:

```
$ cd ~ && mkdir linux-img && cd linux-img
```

You can download the compressed image using a browser like Firefox for Ubuntu using the same (Windows OS) steps described earlier. If you follow this method, you should use the “mv” command to move the downloaded compressed file to the working directory listed above.

Instead, if you wish, you can use a Linux utility like “wget” right from within the working directory:

```
$ wget http://odroid.in/ubuntu_16.04lts/\
ubuntu-16.04-mate-odroid-xu3-20161011.img.xz
```

Wait for the download process to complete, then uncompress the file using the following command:

```
$ unxz ./ubuntu-16.04-mate-odroid-xu3-20161011.img.xz
```

This will result in an uncompressed image file called ubuntu-16.04-mate-odroid-xu3-20161011.img. Attach the USB SD card reader/writer with the boot media into an appropriate USB port. After a few moments, check for an entry that corresponds to the SD card or eMMC module using the following command (your results may differ):

```
$ df -h
Filesystem      Size  Used  Avail      Use%  Mounted on
/dev/mmcblk1p2  58G   3.7G  53G        7%    /
...
/dev/sdb1       30G   224K  30G        1%    /media/odroid/BLANK
```

This tells us that the write destination path to the boot media is /dev/sdb. Given this information, we can carefully construct the image copy command using appropriate input and output paths:

```
$ cd ~/linux-img
$ sudo dd \
  if=./ubuntu-16.04-mate-odroid-xu3-20161011.img \
  of=/dev/sdb \
  bs=1M conv=fsync
$ sync
```

The “dd” command is very powerful, so use it with a lot of care. If incorrect parameters (especially the of parameter) are used here, you could potentially ruin the OS installation of the computer. The “dd” command is often referred to as the disk destroyer command.

This step could take a while. So, wait for it to complete before proceeding. Once completed, remove the USB SD card reader / writer from the USB port of the computer. Remove the boot media and place it on the non-conductive surface.

The above steps are to flash the Linux image. To flash an Android image, simply download a different compressed file, and all subsequent steps would be that same as those listed above. As noted earlier, the Android images can be found at `http://bit.ly/1XwOatZ`.

Flashing an Image on an OSX Computer

The procedure for creating an image file using OSX is similar to Linux, with some small differences. First, download and install the

xzutils package from http://tukaani.org/xz/, making sure to select the OSX binaries, and use it to uncompress the image:

```
$ xz -d <path-to-compressed-image-file>
```

Other differences are that the block size (bs) parameter for the "dd" command is in lowercase, and the USB adapter's device name is in the format /dev/diskX. Before writing the image to SD Card or eMMC module using the "dd" utility, run the command "diskutil" to determine the device name, then unmount it:

```
$ diskutil list
$ diskutil unmountdisk /dev/diskX
$ sudo dd of=/dev/diskX bs=1m if=<path-to-uncompressed-image-file>
```

Inserting the eMMC Module or SD Card

Now that the desired image has been flashed on to the boot media, it's time to insert the boot media into the appropriate port on the XU4. Ensure that the work area is clear of unnecessary items. Wear the grounded wristband. Touch a metallic surface, like that of your desktop. Then, open the XU4 packaging box. Place the anti-static bag containing the XU4 board, on to the non-conductive work surface. Open the bag and remove the board, holding the edges. Avoid touching any of the components, peripheral connectors, ports, exposed circuits or on board solder. Place the board on the work surface, with the bottom side up. Check the annotated diagrams and follow the instructions from Chapter 1 for inserting the eMMC module or SD card.

Powering Up

Before powering the board, attach all peripherals as described in Chapter 1. Insert the power cable, which will be indicated by the red LED lighting up, and wait for the image to boot to desktop. It will take anywhere from 20 seconds to 2 minutes for the desktop to appear, depending on the operating system and the type of boot media used.

Troubleshooting

Red LED not glowing

If the red LED is not glowing, power is not being supplied to the board. Check the connections, and verify that the plug is inserted properly. Also, make sure that a supported 5V/4A power supply is being used, and that it is plugged in properly. Depending on the peripherals attached, you may need a more powerful power supply, such as one rated at 5V/6A+. If the first power supply doesn't work, try another supported model. If neither work, seek to RMA the XU4.

Blue LED not flashing or glowing

If the blue LED is not flashing or glowing at all, it is likely there is something wrong with the image being booted. Check the following:

• Ensure the boot media is inserted properly in the receptacle on the XU4 and that the boot media switch (SD/eMMC) is in the proper position.

• Retry flashing the boot media with the downloaded image.

• If it still does not boot up, repeat the whole process, starting with re-downloading the image from the website.

• If these attempts are not successful, create a post on the ODROID forums at `http://forum.odroid.com`, which may result in an approved RMA.

Blue LED solid glowing

If the blue LED is not flashing or glowing at all, something went wrong with the image writing or the card is not correctly inserted. Try to download, extract and write the image again. If that doesn't solve the issue, try downloading and flashing a different image.

Mouse/Keyboard not working

Ensure that it's correctly plugged in, or try a different USB port. You can change it while the XU4 is running.

No desktop image

Be patient, since some SD cards require several minutes to boot, and anywhere from 30 seconds to 2 minutes to show the first image. Verify that the HDMI cable is correctly plugged in, and edit the boot.ini on a PC and select the correct resolution (see Chapter 1 for details).

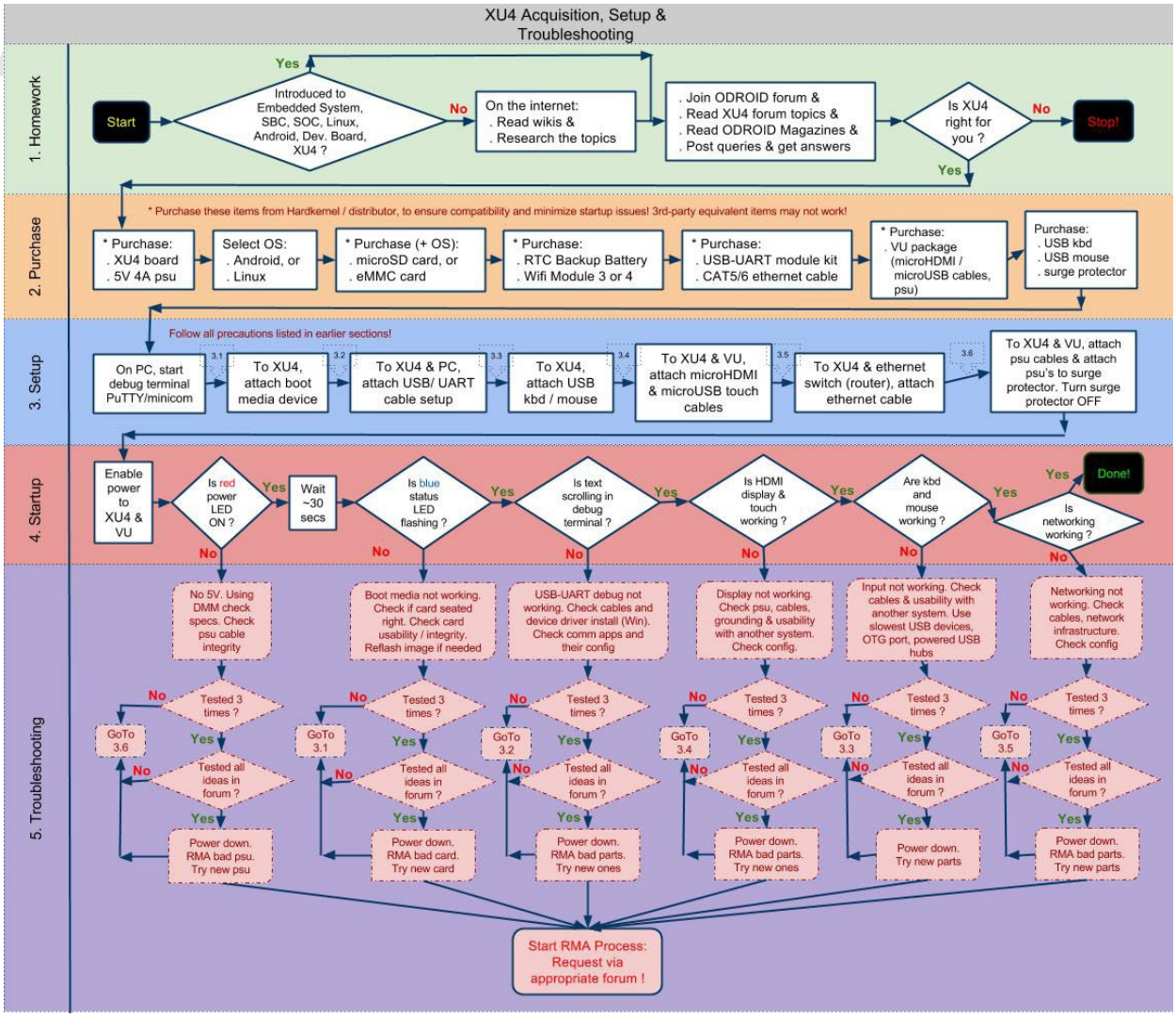
Some HDMI cables have compatibility issues, so make sure to use an official Hardkernel HDMI cable. Some monitors have non-standard EDID functions. In this case, you need to select the non-EDID mode in the ODROID configuration utility.

Some cables have a shorter plug, and it may be necessary to remove the plastic case for a better connection. If there is a USB host port on the monitor, connect a USB cable from the ODROID to the monitor, which creates a ground path between the TV and the device. Add a short jumper wire between the outer casing of the HDMI and USB ports to create a ground path.

On the next page the series of steps that should be followed in order to troubleshoot issues is outlined in detail. Before requesting an RMA, research the relevant forums and verify that a solution or work-around doesn't already exist for your issue.

If your issue has not been previously encountered by anyone,

create a new issue and post it on the forums at `http://forum.odroid.com`, then carefully read any feedback and follow the recommendations. Provide as many details as possible, indicating not only your software and hardware environment, but also the exact steps to recreate the issue.



Acquisition, setup and troubleshooting

Configuring Linux

After booting the ODROID to a Linux desktop, run the included ODROID Utility, which is linked on the desktop, and expand the file system to have full use of the microSD card or eMMC module. Then, run the ODROID-Utility again to update the system and reboot.

To update the HDMI resolution, edit the file /media/boot/boot.ini. Update the section titled “Screen Configuration for HDMI” by un-commenting only one entry of the HDMI setting. For some older versions of Linux, you may be able to use the ODROID Utility to update the

HDMI setting. In those systems, run the ODROID-Utility, select the option: HDMI Configuration. Select the desired HDMI resolution and exit the utility.

The SSH daemon is enabled by default in the Ubuntu template, and the username:password is odroid:odroid. The root password is also odroid.

Finally, the language and timezone settings may be configured with the following command, after installing the desired language in the “Language Support” settings application:

```
$ sudo dpkg-reconfigure locales
```

After the command completes, drag the selected language to the top of the list in the “Language Support” settings. The timezone may be selected by typing the following command:

```
$ sudo dpkg-reconfigure tzdata
```

Configuring Android

Once the Android desktop has loaded, run the ODROID Utility app, and select the desired CPU frequency, monitor resolution and orientation. To set the timezone and other configurable options, use the built-in Settings panel that comes with the Android installation.

Powering Down

Shutting down the ODROID-XU4 is very important in order to prevent damage to the microSD card or eMMC module. In Android, powering down is done by selecting the power button icon at the bottom of the desktop. In Linux, powering down may be done by either selecting the shutdown option from the Applications menu, or by typing the following into a Terminal window:

```
$ sudo shutdown -h now
```

Wait until the blue LED goes out, which indicates that all system activity has completed. It is now safe to unplug the ODROID-XU4 power supply, remove the boot media, and detach the peripherals.

Operating Systems

The ODROID-XU4 can run many free, full-featured, Linux-based, open-source operating systems. Two of the most popular, Ubuntu and Android, are available for download from Hardkernel's website, while many other flavors of Linux may be compiled from source, such as ARCH Linux, FreeBSD, Fedora, CentOS, OpenSUSE, Slackware, and Mint. All of them have a common customized kernel which allows the operating system to communicate with the ODROID hardware. Hardkernel publishes kernels that are specific to the ODROID architecture, and maintains a repository where they may be downloaded and installed as improvements are made.

Ubuntu/Debian

Ubuntu is based on the Debian operating system, and both offer a desktop as well as a large library of applications that may be freely installed, used, and modified. The main GUI tool for obtaining applications on Ubuntu is Synaptic Package Manager, although the command line tool called apt-get is often used by advanced users and scripts in order to streamline the installation process. Applications are stored on servers called repositories, which allow you to receive updates and new versions of software automatically.

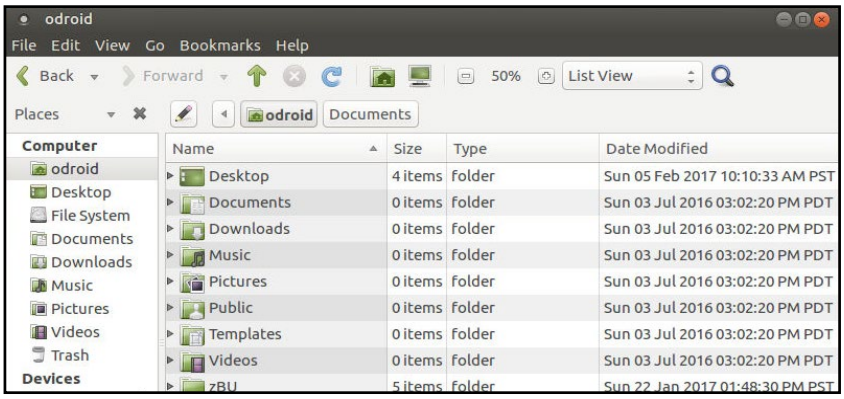
Booting Up

Ubuntu (MATE desktop) can be ordered as a pre-installation on the eMMC or microSD card that comes with the ODROID-XU4. Please refer to Chapter 2 for instructions on downloading and flashing Ubuntu onto your XU4's boot media. Once the XU4 has been powered on, it will boot to a desktop similar to the one shown in the following image.

Chapter 3

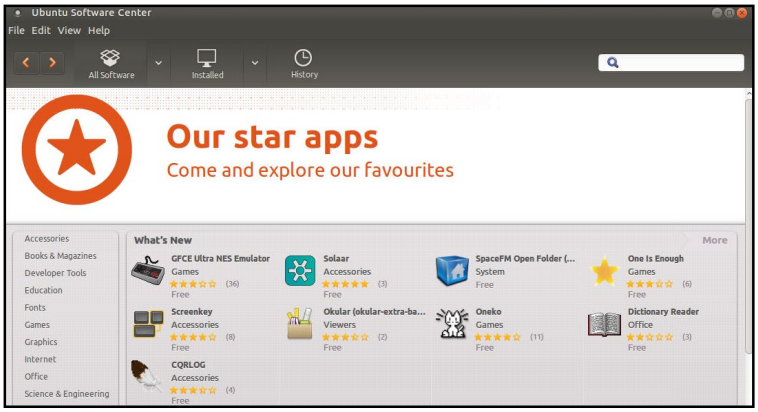
Linux Basics

Ubuntu and Debian, along with most Linux distributions, have a home folder structure where documents, files and media may be stored for each user of the system. For example, downloaded files from the Internet are automatically saved to the Downloads folder, and the other folders may be used to organize various public and private files such as pictures, video, and word processing documents. There are also many special folders that are normally hidden from view that start with a dot (.), where settings for various applications are stored. To view the Home folder, start the File Manager application from the Applications menu in the bottom left corner of the screen, which will look like the screenshot below.



Both Ubuntu and Debian offer many different types of desktop environments, such as MATE, Xubuntu, LXDE (Lubuntu), Gnome, KDE (Kubuntu), and Blackbox, to name a few. Each one includes specific applications that are useful for different types of users. For instance, a typical desktop user who wishes to browse the Internet, use LibreOffice to create, edit and read documents, and listen to music or watch videos might choose Lubuntu, Kubuntu, or Xubuntu, since they include a robust Applications menu and Software Center for downloading new applications, as shown in the image below. The Software Center may be installed using the following command:

```
$ sudo apt-get install software-center
```



If the ODROID-XU4 is being used as part of robotics project, a lightweight environment such as Blackbox may be more suitable, in order to free up the amount of memory available for the main controller application. Other specialized environments are also available from Synaptic Package Manager, including Ubuntu Studio, which includes many applications that are primarily used for creating and producing music, videos, artwork, and photography.

Kernel

At the heart of the Linux system is the kernel, which is responsible for allowing the desktop environment to communicate with the hardware through a common interface. Each ODROID has its own customized kernel code that is modified, tested and published by Hardkernel. Programmers may also download the source code from the official repository at `http://github.com/hardkernel` and make any modifications or contributions that they wish.

The Linux kernel is compatible with all versions of Linux, so any operating system that has been compiled for the ARM hard-float architecture (ARMHF) may be converted to run on the ODROID by installing an ODROID-XU4 kernel and modifying the boot partition. This makes Linux one of the most versatile operating systems available, since it can be configured to run on almost any device, including laptops, desktop PCs, smartphones, and ODROID microcomputers.

Graphical User Interface (GUI)

All versions of Linux offer a command line interface (CLI), and sometimes include a window manager and desktop environment that is launched on startup. Most modern Linux systems such as Ubuntu use a library called X11 to create a windowing environment and provide graphics libraries so that users may interact with applications using a mouse. The ODROID-XU4 includes a Mali T628 Graphics Processor Unit (GPU) which is controlled by X11, in conjunction with the Open Graphics Library (OpenGL), in order to render graphics on a 720p or 1080p monitor. ODROIDs use a subset of the popular OpenGL library called OpenGL ES, which is specifically designed to work with ARM processors, especially smartphones. Applications that are written for OpenGL ES can use low-level graphics functions on the GPU chip itself in order to quickly and efficiently render graphics, resulting in a much faster and smoother user experience than using the CPU alone.

720p vs 1080p

ODROIDs support both 720p and 1080p monitor configurations, which are video resolutions commonly supported on LCD TV's and on many LCD computer monitors. 720p is sometimes referred to as HD,

and 1080p as Full HD. The following comparison table lists the native display resolutions along with some sources that use these resolutions.

Standard	Resolution	Aspect Ratio	Refresh Rate	Typical Sources
1080p	1920 x 1080	16:9(1.78:1)	24Hz, 50Hz, 60Hz	HDTV, Blu-ray
1080i	1920 x 1080	16:9(1.78:1)	50Hz, 60Hz	HDTV, Blu-ray
720p	1280 x 720	16:9(1.78:1)	50Hz, 60Hz	HDTV, miniDV

As a comparison, the standard DVD Video format uses a display resolution of 720 x 480 with a 4:3 aspect ratio (non-widescreen). This resolution is also referred to as D-1 video and is the standard used or miniDV digital camcorders.

Progressive vs Interlaced Video

1080p and 720p are progressive video formats. For these formats, a complete video image is transferred for each frame of the video signal, so that a 60fps video displays 60 complete, full frames per second. This format is visually superior to interlaced video, and in general should be used when supported by your display.

1080i and 720i are interlaced video formats, and video interlacing is the historical standard for broadcast video. In interlacing, two frames (or more correctly, fields) are transferred for each frame of the video signal, so that a 60fps video has 30 complete, full frames per second. The display device sends the two fields to the screen in the horizontal odd and even (or interlaced) pixel positions. Given the visual persistence of the human eye, this reconstruction of full frames is undetectable to the viewer.

In general, faster frame rates result in a visually smoother video playback, and interlacing results in a halving of the native frame/field transfer rate because two fields are required for the full image. Frame rate and visual experience are subjective, however. Traditionally, movies produced with conventional film use a 24 frames per second rate, and many viewers prefer the cinematic experience that 24fps video provides. In fact, much of the professional video produced with high-end gear is actually shot at 24fps, and later converted to video frame rates to provide the look and feel of cinematic film.

Video Downconversion

When selecting a screen resolution for your XU4, you should first consult the specifications for the display that you plan to use. In general, using the native resolution of your display will provide the best performance and quality.

For example, if you use a display that only supports native 720p, and you set your XU4 to a 1080p screen resolution, it will probably

work, but is not the ideal configuration. The mismatch in resolutions indicates that your display is automatically performing a downconversion of the incoming signal. Despite appearing to work OK, the XU4 will be performing unnecessary work in order to create the 1080p output.

If a 720p video is viewed with this configuration, the XU4 will upconvert the original 720p video by extrapolation to produce a 1080p output, only to have that 1080p video downconverted by interpolation back into 720p by the video display. In this case, setting the XU4 screen resolution to 720p is the logical choice.

Video Upconversion

When the XU4 is set to a higher output resolution than the source video, an upconversion is performed by extrapolation to produce the higher resolution frame in the framebuffer. The average video player can display videos with a number of different source resolutions. These videos are upconverted by the XU4 in order to fit the screen, and the user therefore has a uniform viewing experience.

However, if the XU4 is used solely for such purposes as video editing, video effects, or historical restoration, and the source material resolution is lower than the maximum resolution of the video display, the user may wish to setup the video settings differently. In this case, matching both the XU4 video settings and the video display settings to the source material's native format would be appropriate. With this configuration, the video playback would provide an image that is true to the original source material, as well as freeing up more of the XU4's CPU bandwidth for other processing tasks.

HDMI Overscan

If the display used on your XU4 shows a slight cropping of the visible image on the screen, you may be experiencing overscan. This is not an uncommon problem, and especially so for LCD TV monitors. The fix is usually a simple one, and the underlying issue is most likely due to a setting with the LCD monitor. Some PC Monitors with HDMI inputs will also apply overscan to the HDMI input, assuming that a broadcast TV signal is being used.

Why Do Monitors Have Overscan?

Monitors that are used for broadcast television usually have overscan enabled by default. This is a normal feature of TV monitors and has been present from the very beginning of television. Overscan is used to crop the edges of the video frame in order to remove any erratic or distorted edges that often exist with broadcast video. To the viewer, this results in a cleaner picture, and the overscan simply isn't

noticed. For a computer display however, this can be an issue. For this reason, computer LCD monitors usually do not provide overscan, and if they do have this feature, it is disabled by default.

Disabling Monitor Overscan

Prior to attempting to fix overscan using the methods detailed below, first verify that the monitor resolution setting and the XU4 screen resolution setting match. Unfortunately, there isn't a standard method or terminology for disabling overscan used by monitor manufacturers, so you may have to search through the monitor menus a bit until you find the overscan setting.

Display Setting Button on Remote Control

Using the LCD monitor remote control, first look for a display mode button labeled Display, Screen Mode, marked with a display icon, or marked with |<>| (for wide mode). If your remote has one of these dedicated display mode keys, pressing this key should allow you to cycle through the display modes.

Finding Setting in Monitor Menus

If you cannot find a display mode button, you will need to enter the display's menu setting mode, usually marked Menu or Settings. You will need to search for the menu item that controls the overscan setting, and on some displays you will need to enable the Advanced mode. The table below describes several possible menu locations for the overscan setting on different displays:

Menu > Picture Mode > Aspect ratio
Change to "Just Scan"

Menu > Picture > Screen adjustment > Picture Size
Change to "Screen Fit"

Menu > View Mode
Change to "Dot by Dot"

Menu > Tools
Change HDMI Source to "PC"

Menu > Picture > Screen adjustment > Picture Size Screen
Auto Config

Menu > AV Preset > HDMI
Change to "PC"

As you can see, finding the setting for your LCD monitor may take some time to locate. Other terms manufacturers may use for overscan disable include HD size, full pixel, unscaled, native, and 1:1.

On some monitor models, one of the HDMI inputs is intended to work with a PC and will disable overscan for that input (often HDMI 2). You can also look at the monitor's HDMI connectors for a label such as PC Input, or read the monitor users manual to find the correct input to use if applicable.

If you have used one of the above methods to change your LCD monitor's display mode and it does not help, or if your monitor does not provide an overscan adjustment, use one of the following methods to adjust the XU4 video output directly.

Command Line Interface

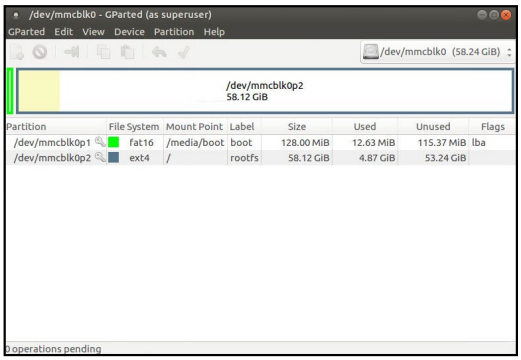
The Linux terminal CLI is the basic method of interacting with the Linux system, normally using a shell called BASH. When a graphical system is running, a Terminal window may be launched in order to issue commands using the keyboard.

Many Linux systems also come with the Secure Shell (SSH) server, which allows a command line interface to be invoked from a remote computer via Ethernet. Other protocols, such as Virtual Network Computing (VNC) have the ability to project the graphical environment to a remote computer as well.

BASH is a powerful scripting language as well as a means of reading, creating, modifying, and launching files and programs that are stored in the Linux file system. A collection of BASH commands, called a script, can be saved to a file with the extension ".sh" and used as an executable file. A shell script can perform many tasks, such as configuring program variables, launching an application, and copying or moving files.

Disk Partitions

Linux can read and write to a variety of disk formats, with the most common called EXT3, EXT4, and FAT. If disk compatibility with Windows and OSX is a priority, it is recommended to format external



drives as FAT32 which can be read by nearly all operating systems. The root filesystem of Linux is usually in EXT3 or EXT4 format, which can only be read by other Linux systems.

Disk partitions may be easily resized in order to use all of the available space on a disk using a utility called Gparted. The image below shows the Gparted application being used to resize a root partition. It is recommended to use the ODROID Utility to resize the main partition in Ubuntu in order to have as much storage space as possible on the microSD card or eMMC module.

Web Browsing

One of the main uses of modern computers is to browse the World Wide Web, and the octa-core ODROID-XU4 delivers a smooth, enjoyable browsing experience. Using the Ubuntu Software Center, a browser such as Firefox and/or Chromium may be installed, along with open-source versions like Iceweasel.

When watching streaming videos, it is recommended to use the Kodi/XBMC YouTube plugin in order to launch videos in full-screen, since mainstream browsers are not specifically written to use the video decoding chip available on ODROIDs. However, the ODROID-XU4 CPU is powerful enough to play standard videos within a browser window at normal size using software rendering.

Kodi (formerly XBMC)

Kodi is an application which can turn an ODROID-XU4 into an amazingly powerful home media center. In fact, using the XU4 to run Kodi is the reason why many, if not most people purchased their XU4 in the first place. If you're not familiar with Kodi, here are links to the official Kodi Intro FAQ at `http://bit.ly/1G8wDjD` and the Kodi Wikipedia page at `http://bit.ly/1Ir2z3R`.

The current versions of Ubuntu Linux and Android provided by Hardkernel already have Kodi installed. However, you may wish to re-install or upgrade Kodi in the future when a new release becomes available. Pre-release (beta) versions of Kodi are made available for test, and you can join a group of users that test out new features before the formal release.

Troubleshooting

If you experience problems with playback of some video formats in Kodi, the first troubleshooting step would be to set video acceleration to "software" by selecting the System menu, the pressing **Settings >Video >Acceleration >Decoding Method >Software**.

Kodi Upgrade

If you are installing a newer version of Kodi, you do not need to uninstall the current version first. However you should create a Kodi backup just to be safe, which is covered later in this section.

Kodi Installation

To install Kodi, use the ODROID Utility published by Hardkernel. If the ODROID Utility is not already installed on the image, it may be downloaded using the following Terminal commands:

```
$ sudo wget -O /usr/local/bin/odroid-utility.sh \
  https://raw.githubusercontent.com/mdrjr/\
  odroid-utility/master/odroid-utility.sh
$ sudo chmod +x /usr/local/bin/odroid-utility.sh
$ sudo odroid-utility.sh
```

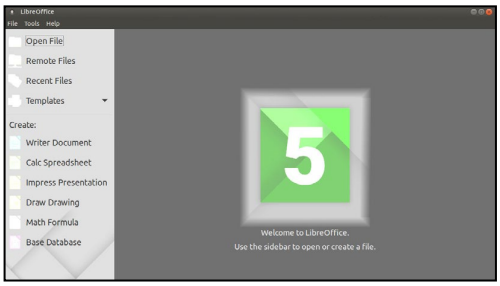
Kodi Backup

Backing up your data is always a good idea, and Kodi is no exception. Kodi configurations and databases can become quite detailed, but backup and restoration is quite simple. You can perform full or partial backups/restores, as well as copy your complete Kodi setup to another system - even to your PC or laptop.

In order to perform a Kodi backup or restore, you will first need to install the Kodi “Backup AddOn”. More information may be found at `http://bit.ly/1JER8XL`, along with details on performing Kodi backups at `http://bit.ly/1KRNF6K`, and on the Kodi forum at `http://bit.ly/1QGOfGf`.

Office and Productivity Applications

LibreOffice is a popular and powerful office productivity suite that includes word processing, spreadsheets, presentations, drawing, and flowcharting applications. It is a free alternative to Microsoft Office, and can be installed on Ubuntu via the Ubuntu Software Center or Synaptic Package Manager. The loading screen of LibreOffice is shown below:



Music and MIDI

Not long after the first personal computers became available, creative individuals began to find inventive ways to use them for music composition and performance. Before long, the need for I/O standard-

ization became clear, and in 1983 the MIDI specification was developed. MIDI (Musical Instrument Digital Interface) gained rapid industry acceptance, and within a few years, a MIDI Interface was found on virtually every sound card and on most PCs with sound chips on the motherboard. This is no longer the case, but the MIDI interface, MIDI instruments, and external MIDI synthesizers are still widely used by musicians and sound engineers. In fact, MIDI is still the de-facto standard instrument interface, and does not show any signs of falling into obscurity as have many early computing standards.

What is MIDI?

MIDI encompasses not only the MIDI connector hardware specification, but also includes the communication protocol used for instruments and synthesizers, and a MIDI file format specification for MIDI recording and playback. The standard MPU-401 physical MIDI interface is a 5-pin DIN connector. Internal sound cards no longer include this connector, but it can be found on some high-end external USB sound modules. However USB-to-MIDI adapters are inexpensive and widely available. The MIDI Manufacturers Associated website at `http://bit.ly/1Gn2cYw` is a good source for additional information.

Linux Support for Music and MIDI

MIDI adapters, USB Sound Devices, and Music Applications have wide support on Linux. For example, the Ubuntu Studio project is an excellent open-source Linux build that has extensive device support with pre-configured applications. The official Ubuntu Studio site page has an http://bit.ly/1FFKyvh which will give you an idea of the range of Music and MIDI applications possible on Linux.

Ubuntu Studio includes some of the most popular audio apps available - including tools for DAW (Digital Audio Workstations) for multi-track mixing, sequencers for MIDI music, drum machines, software synthesizers, and even music creation via programming. Ubuntu Studio is available from the Ubuntu Software Center and Synaptic Package Manager.

How do I add a MIDI Interface to the XU4 ?

Three basic options for MIDI are available for the XU4:

USB sound module: If you wish to use a USB sound module, look for a device that has solid Linux and/or Android device driver support which includes the MPU-401 interface.

USB-to-MIDI-Adapter: A number of basic devices are available which have been proven to work on both Android and Linux. Compatible de-

vices will be advertised as a “no device driver required” model.
Direct UART to MIDI interface: Since MIDI is a asynchronous serial interface, standard serial ports can often be easily converted to a MIDI interface.

Sound Cards and Devices for MIDI File Playback on the XU4

Not all sound cards and modules provide native MIDI file playback capability, including the ODROID USB sound card. Many other USB sound cards do however, so make sure to consult the device’s specifications. Professional-level USB MIDI synthesizers are also available, including sampling synths by Roland, Ensoniq, Kurzweil, Roland, and Korg.

Experimental Music with the XU4

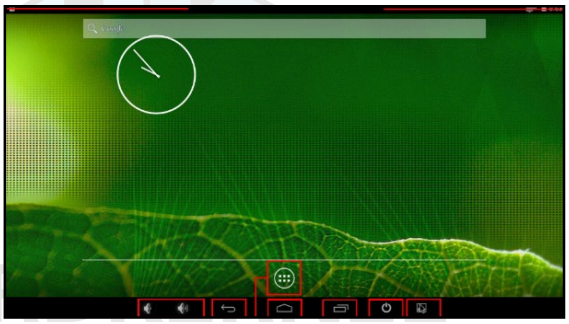
Using an ODROID-XU4 for computer music opens up possibilities beyond those of traditional computer music performance, productions, and engineering. A MIDI Interface and any number of sensors could be used to create unique musical instruments. For example, a pressure transducer could be used as a MIDI breath controller, ultrasonic transducers could be used to make a Vulcan Lute, a string-less harp, or a super-theremin, or a capacitive touch controller such as the Freescale MPR121. Using conductive paint for the capacitive inputs could inspire even more interesting inventions.

Android

Android is an easy-to-use, yet powerful operating system, intended for smartphones, tablets and other portable devices It also runs very well on the ODROID-XU4, not only as an inexpensive testing platform for building Android applications, but also as a set-top box for video and music streaming and playback, a general-purpose operating system capable of web browsing, social networking, remote control of other computers, and much more.

Desktop Environment

The Hardkernel Android image offers several Android tablet features, including two pulldown menus at the top of the screen. As shown below, the top left menu shows notifications and application information. The top right menu offers direct access to the operating system settings. To open the applications menu, click on the circle with the six dots, which displays an alphabetical list of all installed apps. The Recent menu shows the recently opened applications. Power Options allows you to reboot, shutdown the device and put it in airplane mode, which disables all wireless functions, including Bluetooth.



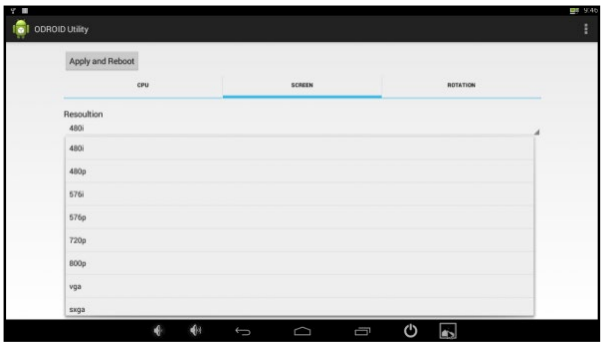
ODROID Utility and Updater

To run the ODROID Utility application, open the applications menu and click on the ODROID Utility icon. When you start the ODROID Utility for the first time, it will ask for superuser permissions, which should be confirmed. This application contains several tabs which can be used to configure various aspects of the hardware:

CPU changes the CPU governor settings and can set the Kodi media center to launch at system startup. The Performance governor will give the best speed but consume the most power.

Screen allows you to change the resolution as well as stretch and move the screen.

Rotation allows you to rotate the screen to either Portrait or Landscape mode.



Setting the Display Resolution

On the Screen tab of the ODROID Utility application, select the preferred monitor resolution, then click on Apply and Reboot. The new resolution will be active after the reboot has completed.

An overscan adjustment tool is also provided in the ODROID Utility application. To adjust the screen, click on the Screen Tab. You will then see a group of arrows that represents the four screen edges. For the edges that you wish to adjust, click on the corresponding arrow in order to set the number of offset pixels. After adjusting the settings, click on the “Apply and Reboot” button.

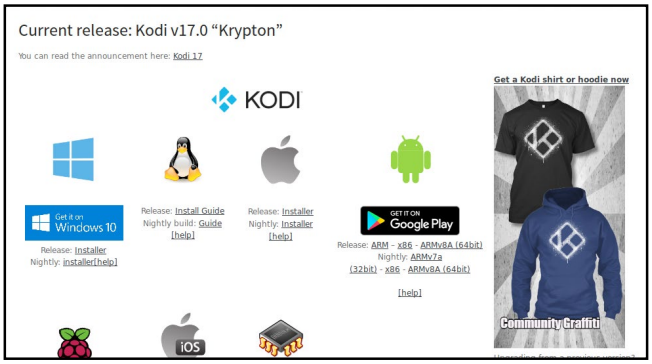
Installing Google Play and applications

The official Hardkernel Android image comes without any Goo-

gle applications installed. The best way to install them, is to use the "Universal 1-Click GApps Installer for ODROID" application, available at http://bit.ly/1gkv4PM.

Kodi

To install Kodi, you will first need to download the most recent .apk file from the Kodi Download page at http://bit.ly/1yIrUDd, as shown in the image below. Find the link to the ARM build, then click on the link to download the .apk installation package. After download, simply click on the .apk file to install. The official Kodi website provides more detail on the Android installation at http://bit.ly/1FFK7B9.



Netflix

Install Netflix from Google Play, then click on its icon, which should be automatically added to the Android desktop. Login with your existing Netflix account or create a new one. When you start a video, it always begins playing at low quality, then switches to higher quality after about 30 seconds, depending on the speed of your Internet connection.

Gaming

Most Android games are compatible with the ODROID-XU4, but some applications that use multitouch gestures won't be playable without a touchscreen. The ODROID-VU, available from the Hardkernel Store, will allow all Android games to be played without the need for an auxiliary controller.

Music and MIDI

There are a number of Android Apps available for music and MIDI. In order to work correctly with Android, USB MIDI adapters and sound cards typically need to be class-compliant USB MIDI devices that do not require a device driver, and need to be used on a USB connector that supports host mode.

The MIDI Manufacturers Association webpage at http://bit.ly/1IzGkvb is a good source for additional links on this topic. Oth-

er good webpages for MIDI and computer music on Android include http://bit.ly/1FFKoUH and http://bit.ly/1IzGumw.

TouchDAW is an excellent and mature app which allows the XU4 to be a MIDI Controller for DAW and general-purpose MIDI control on secondary workstations. TouchDAW supports the leading computer music production applications on Windows, IOS, and Linux systems.

Using Bluetooth Devices with Android

Bluetooth device detection and pairing are very easy with Android using the Settings application. After plugging in a USB bluetooth adapter and powering up the XU4, you will need to enable the bluetooth receiver. Open the Settings application, find Bluetooth in the settings list under the Wireless and Networks category, and select ON to enable your USB bluetooth adapter and bluetooth services. To begin pairing your bluetooth device to the XU4, turn on your device and put the device into pairing mode. The Bluetooth manager will display a list of detected devices, as well as devices that have been successfully paired.

If your device was detected, but not shown to be a paired device, click on the sliders icon on the right side of the listed device to open the bluetooth settings for this device. Complete the pairing process by entering the pairing code or passkey for the device. If the bluetooth device was not detected, clicking on search for devices will perform a manual scan for any detectable bluetooth devices within range of the XU4.

Adding an ODROID-VU Touchscreen

Hardkernel has designed multiple HDMI touch screens that work with the XU4:

ODROID-VU5: 5” 800 x 480 5-finger touch display
ODROID-VU7: 7” 800 x 480 5-finger touch display
ODROID-VU7 Plus: 7” 1024 x 600 5-finger touch display

To use these displays with the XU4, connect the USB cable from them to the ODROID-XU4’s USB port via an HDMI Type A cable. Use an appropriate power supply, provide the required power to the display. Hardkernel’s Android images support these displays without any additional configuration.

Hardware Tinkering

Now that you have been introduced to the XU4 and have become aware of booting it up with a image, it is time to get acquainted with its main purpose of creation – to be able to interact with the external world through the age-old activity of tinkering. While its design and production is a marvel in itself, it really shines in its use to interact with the external world, through a slew of breakout-boards and sensory devices – all under the control of powerful operating systems like Linux or Android.

Here are the devices covered in this section:

- **USB UART Module Kit**
- **Bluetooth Module 2**
- **ODROID SHOW2**
- **Weather Board 2**
- **DC Plug Cable Assembly 5.5mm**
- **USB audio adapter**
- **USB-SPDIF**
- **ODROID USB-CAM 720P**
- **USB3 to SATA Bridge Board**
- **USB3 / SATA2 HDD/SSD RAID 0/1 enclosure**
- **USB GPS Module**
- **myAHRS+ board**
- **Cloudshell**
- **Expansion Board**
- **Shifter Shield**
- **ODROID VU7, VuShell for VU7, ODROID VU7 Plus**
- **Micro USB-DC Power Bridge**
- **Heat Sink**
- **SmartPower2**
- **oCam camera, oCam Global Shutter, M12 Lens Set**
- **WiFi Module 0, WiFi Module 3, WiFi Module 4, WiFi Module 5**

Always shut down the XU4 and power it off before attaching or detaching peripheral devices. The same applies for peripherals that have their own power supplies. Some unsupported powered peripherals, though known to work with the XU4, may not be safe to use if they leak current back through the USB port, which could damage the XU4.

Make sure to backup your image before attempting to work with any new peripheral, especially if it requires installation of additional software. You can always revert to a known working state, should your efforts cause issues.

Prerequisites

Install the latest official Hardkernel Linux image from http://bit.ly/1Y9EZhJ onto an eMMC module or compatible microSD card. The version information is found by typing the following command:

```
$ uname -a
Linux odroid 3.10.104-131 #1 SMP PREEMPT Sat Feb 18 01:04:01 UTC
2017 armv7l armv7l armv7l GNU/Linux
```

After ensuring Internet connectivity, update this image using the following commands:

```
$ sudo apt-get update && sudo apt-get upgrade && \
  sudo apt-get dist-upgrade
```

Restart the system. The version image should match the following, or have a higher version:

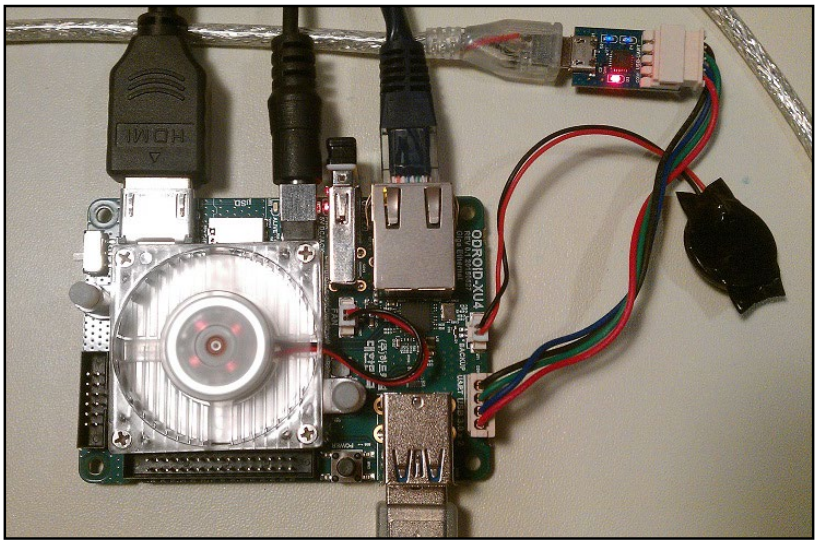
```
$ uname -a
Linux odroid 3.10.92-64 #1 SMP PREEMPT Mon Nov 23 15:13:42 BRST
2015 armv7l armv7l armv7l GNU/Linux
```

You can optionally install a VNC server, which allows you to control the XU4 from another device on the local network via VNC Viewer:

```
$ sudo apt-get install x11vnc
```

More information about configuring VNC may be found at http://bit.ly/1OYeFVb.

USB UART Module Kit



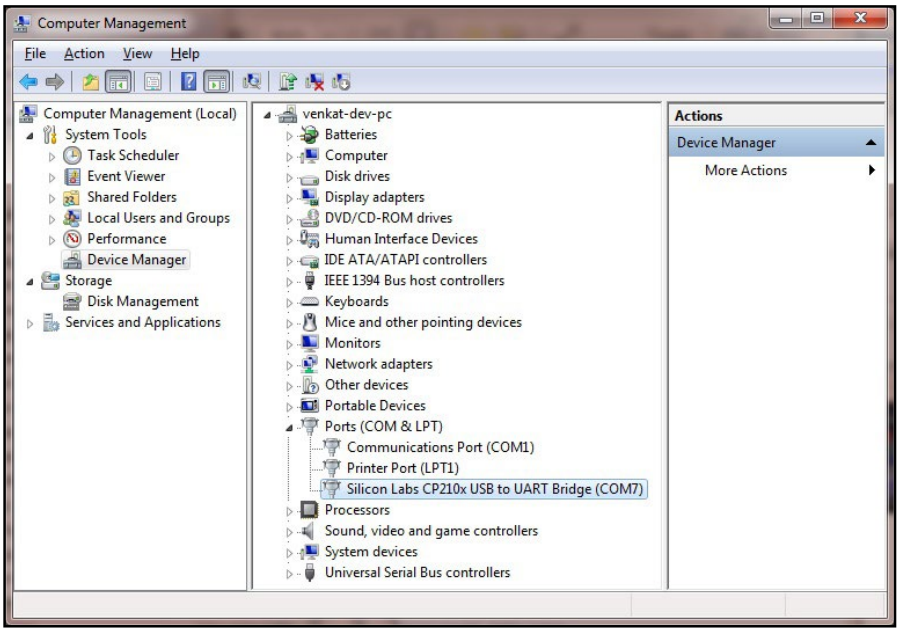
It is often crucial to get a view of the early boot process, especially

if you are working with newly added external devices or creating special purpose boot images. Typically these activities can be debugged by viewing their corresponding log entries using the dmesg application. This can be achieved using the USB UART module and a microUSB male to USB male cable, which is typically used to charge smartphones and tablets. Together, they form the kit, as shown above.

When using Microsoft Windows as a host PC system, you will need to install a Terminal application such as PuTTY. This can be used to set up common configurations in order to access the XU4 either via a serial connection or SSH.

Prior to setting up the kit, power off the XU4. Attach one end of the USB UART module to the XU4. Attach the other (micro USB) end of the USB UART module to the USB cable. Attach the free end of the cable to an available USB port of the host PC system, then power on the XU4.

The Windows host PC system will recognize the module as a Silicon Labs CP210x USB to UART Bridge and install the appropriate device driver. After installation has completed, you should note which COM port the module is associated with. To do so, launch the Computer Management utility of Windows, highlight the Device Manager option, and expand the Ports list. In this particular case, as shown below, the module is installed on COM7. This is the COM port that should be used in the PuTTY configuration.

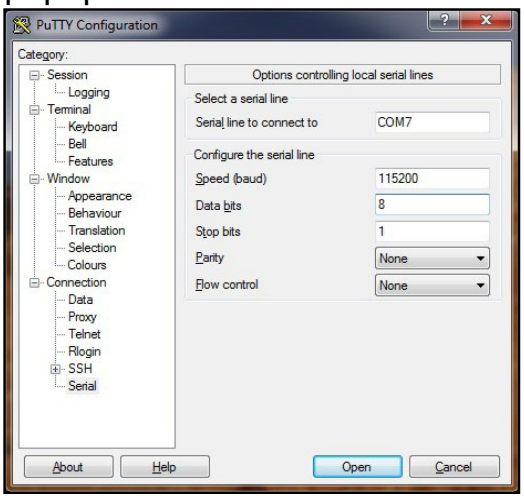


USB UART module on COM7

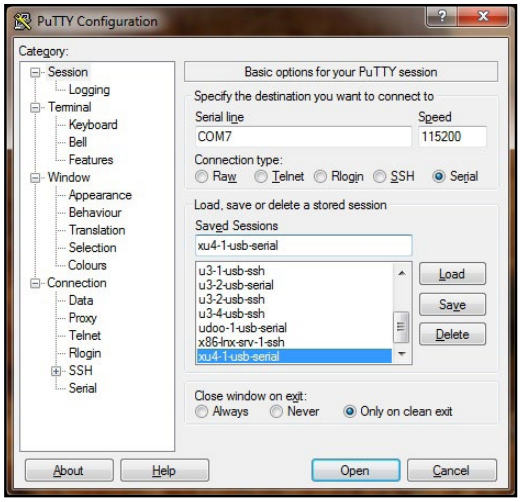
Next, launch PuTTY and select the Serial option on the left pane as shown next page.

```
Serial line to connect to: COM7
Speed (baud): 115200
Data bits: 8
Stop bits: 1
Parity: None
Flow Control: XON/XOFF
```

Enter configuration information as shown in the screenshots below. Then, select the Session option in the left pane. You will see a popup window as indicated below.



Serial configuration in PuTTY



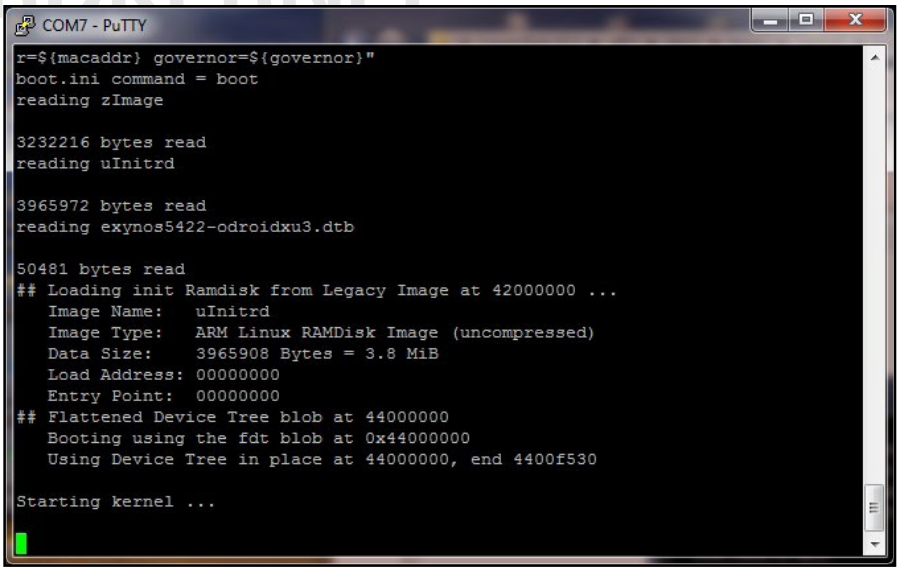
Saved serial configuration in PuTTY

```
Serial line: COM7
Speed: 115200
Connection Type: Serial
```

Give this configuration a name such as “XU4-1-usb-serial”, then save it. This saved configuration can be reloaded and used anytime,

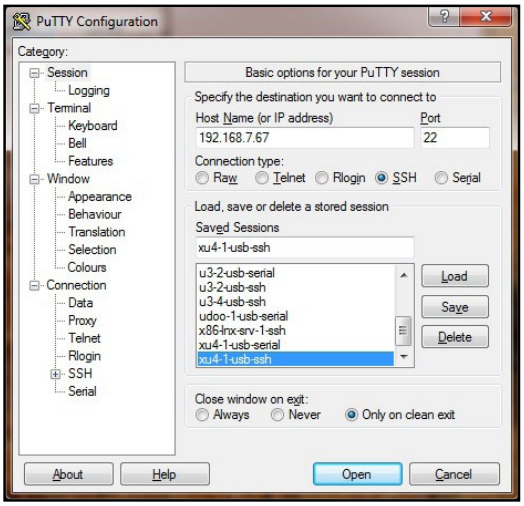
as long as the COM port on the host PC system has not changed. Click on the Open button to start a session.

Next, reboot the XU4. After a short delay, you will see messages scroll by in the PuTTY console, as shown below, related to the mounting of external devices and loading of software modules. These messages can help with debugging issues that you may encounter. Most of these messages will also be listed via the dmesg command.



PuTTY serial console

Close the serial console, then set up the SSH connection option by relaunching the PuTTY application and selecting the Session option on the left pane. For this SSH option to work, the host PC system and the XU4 need to be visible to each other on the same local area network (LAN). Refer to the screenshot below for an example PuTTY configuration.

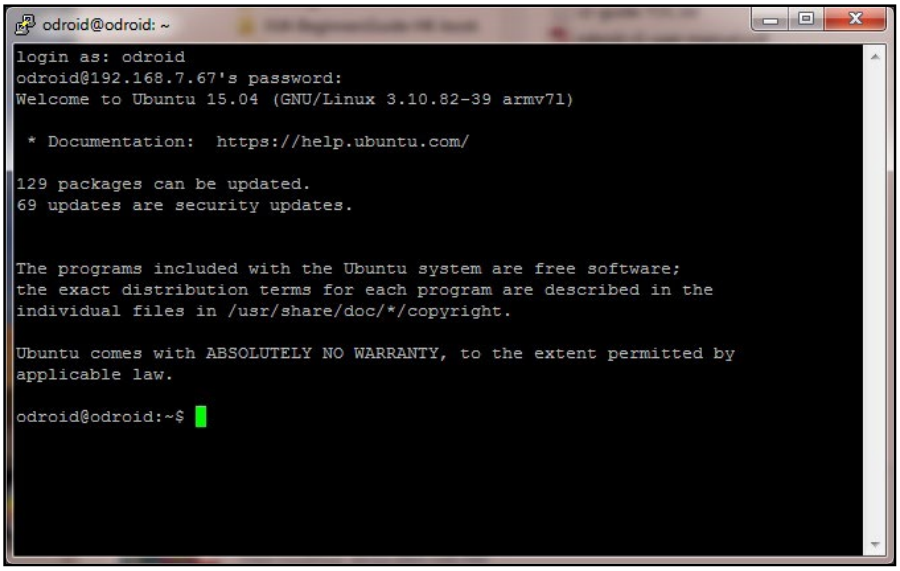


Saved SSH configuration in PuTTY

```
Host name: 192.168.7.67
Port: 22
Connection type: SSH
```

Enter the XU4's IP address or hostname (if it can be resolved from the host PC system) and a numerical value of 22 for the Port number. Select the Connection Type to be SSH. Enter "XU4-1-usb-ssh" for the configuration name and save it. This saved configuration can be reloaded and used anytime, as long as the XU4's IP address and hostname remain unchanged. Click on the Open button to start a session.

Next, reboot the XU4 and wait approximately 2 minutes. If this is the first time you are accessing SSH, you may be prompted to trust the XU4's RSA key. Select Yes, and you should be prompted for a username and password, which are typically "odroid" and "odroid". You will see a screen as shown below.



PuTTY SSH console

Using the USB UART connection, you can access the XU4 system as if you had locally opened a terminal instance. While you cannot view the boot-time log in real-time, you can run a large number of Linux commands, including dmesg, from this SSH session.

So far, we've addressed the case where the host PC system is a Windows system. Next, we will present information on the use of a Ubuntu desktop Linux host system. Most of the steps apply to a host system running any Linux version, including a server OS.

Linux Host Setup

Access the Linux desktop of the host system and launch a ter-

minal session. Attach the USB cable from the USB UART setup to the host system. Use the following commands to verify the installation:

```
$ lsusb
...
Bus 001 Device 005: ID 10c4:ea60 Cygnal Integrated Products, Inc.
CP210x UART Bridge / myAVR mySmartUSB light
...
```


Based on the output above, you get obtain additional details of the USB UART module using the following command:

```
$ sudo lsusb -D /dev/bus/usb/001/005
Device: ID 10c4:ea60 Cygnal Integrated Products, Inc.  CP210x UART
Bridge / myAVR mySmartUSB light
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x10c4 Cygnal Integrated Products, Inc.
  idProduct          0xea60 CP210x UART Bridge / myAVR mySmartUSB light
  bcdDevice            1.00
  iManufacturer           1 Silicon Labs
  iProduct                2 CP2104 USB to UART Bridge Controller
  iSerial                 3 00513B0C
  bNumConfigurations      1
…
```

The TTY port on which the module is detected can be obtained with the following command:

```
$ sudo ls -lsa /dev/tty* | grep USB
0 crw-rw---- 1 root dialout 188,   0 Feb  18 23:48 /dev/ttyUSB0
```

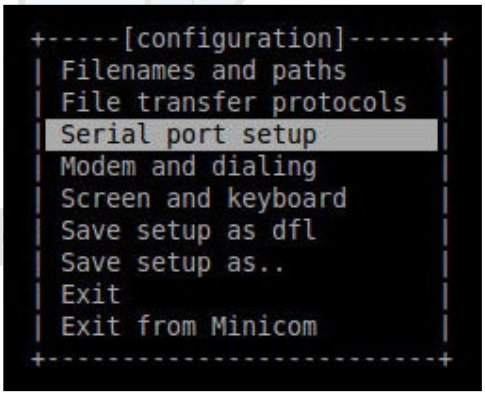
The lockfile can also be checked using the following command:

```
$ sudo ls -lsa /var/lock*
0 lrwxrwxrwx 1 root root 9 Apr 26  2014 /var/lock -> /run/lock
```

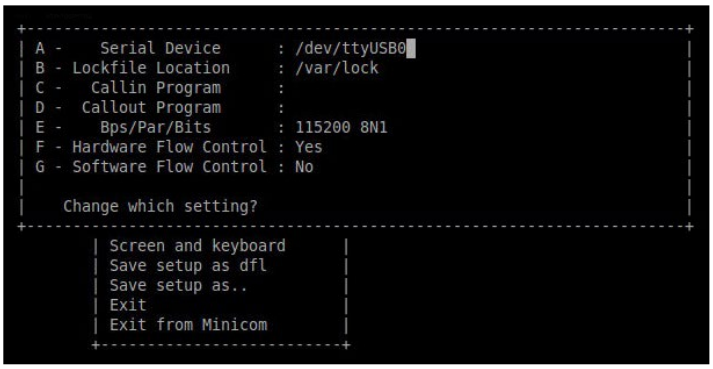
This information is used to setup the communications between the host system and USB UART cable attached to the XU4. The application used to enable communication is called Minicom. Launch Minicom using the following commands:

```
$ minicom -version
minicom version 2.6.1 (compiled May  1 2012)
Copyright (C) Miquel van Smoorenburg.
...
$ sudo minicom -o -s
```

Here are the four screens necessary to configure and use Minicom:

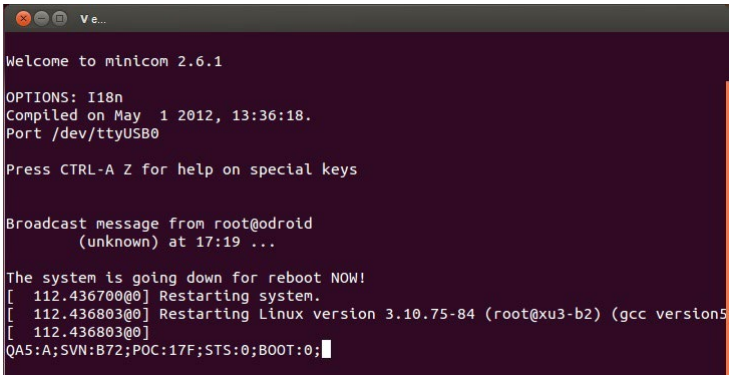


Port option



Serial port data

Save setup



Minicom session

The four steps include:

• Select the Serial port setup option, which will display screen 2.
• In screen 2, type A to select option A. Edit the serial device to that used on your system. In this case it is: /dev/ttyUSB0. Ensure the lockfile location is what was obtained earlier. Likewise, check that option E reflects the information shown in the figure: 115200 8N1. Ensure Hardware Flow Control is set to YES, then hit the ENTER key to go to the next screen.
• In screen 3, select the Save setup as … option to save this config for future use. Enter a filename at the new prompt. Hit ENTER to accept the filename. Then select Exit option to complete the config process.
• You will now return to the terminal windows, as shown in step 4. Reboot the XU4 and you will observe boot-time information scroll through the minicom (terminal) session.

Bluetooth Module 2

Hardkernel offers a bluetooth adapter called the Bluetooth (4.0) Module 2 which is certified to work with the XU4 under both Android and Linux in the following modes:

• Classic Bluetooth v2.0: Android, Linux
• Bluetooth High Speed v3.0: Linux
• Bluetooth low energy v4.0 (BLE): Linux

Bluetooth Module 2 adapter

While some of the required software modules may already exist in your XU4's image, it is helpful to install the following additional modules and utilities using the following commands:

```
$ sudo apt-get install bluez-dbg bluez-utils bluez-tools
$ sudo apt-get install bluewho blueman python-bluetooth
```

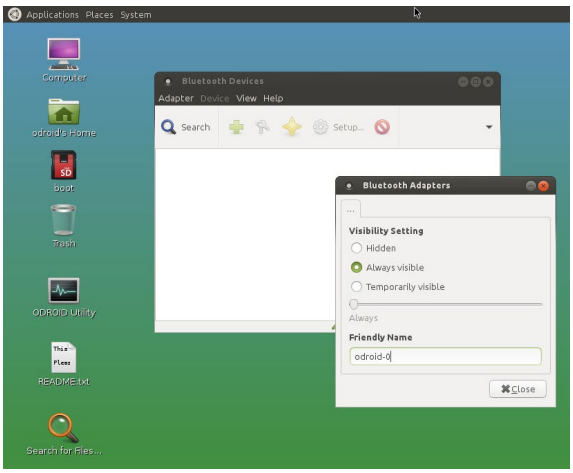
Some of these utilities may be used to debug your Bluetooth setup if you are using a third-party bluetooth adapter. Next, check the USB information related to the adapter using the lsusb command.

You can check the Bluetooth adapter's support for additional features by examining the dmesg logs, using the following command:

```
$  dmesg | grep Blue
[0.182920] Bluetooth: Core ver 2.18
[0.182951] Bluetooth: HCI device and connection manager init
[0.182965] Bluetooth: HCI socket layer initialized
[0.182977] Bluetooth: L2CAP socket layer initialized
[0.182999] Bluetooth: SCO socket layer initialized
[3.650243] Bluetooth: RFCOMM TTY layer initialized
[3.650254] Bluetooth: RFCOMM socket layer initialized
[3.650265] Bluetooth: RFCOMM ver 1.11
[3.650272] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[3.650274] Bluetooth: BNEP filters: protocol multicast
[3.650279] Bluetooth: BNEP socket layer initialized
[3.650281] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[3.650286] Bluetooth: HIDP socket layer initialized
```

You can then check the list of the installed Bluetooth modules, using the command "dpkg -l | grep blue". We can then check the presence of the Bluetooth device, which is useful for connection configuration later:

```
$ hciconfig
hci0:    Type: BR/EDR  Bus: USB
    BD Address: 00:1A:7D:DA:71:13 ACL MTU: 310:10  SCO MTU: 64:8
    UP RUNNING PSCAN
    RX bytes:685 acl:0 sco:0 events:50 errors:0
    TX bytes:4159 acl:0 sco:0 commands:50 errors:0

$ sudo rfkill list all
0: hci0: Bluetooth
    Soft blocked: no
    Hard blocked: no
    Hard blocked: no
```

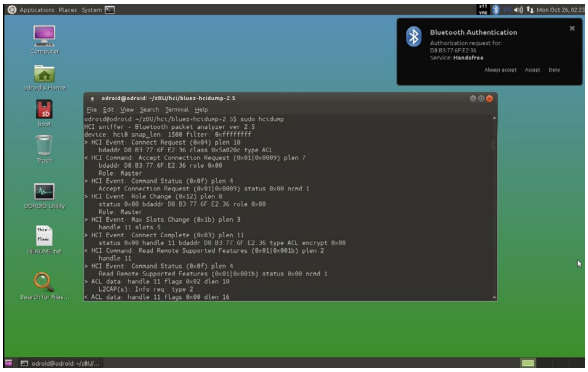


Bluetooth Manager configuration screen

To configure the adapter from the Ubuntu desktop, launch the Bluetooth Manager configuration utility from the System → Preferences menu item. Then, select the Adapter → Preferences menu item to configure the application. Keep the friendly name of the Bluetooth adapter as “odroid-0”. You can make the device always visible when other Bluetooth devices scan for this adapter. The image above shows the configuration screen. After configuring, save the changes and reboot.

If there is a need to test or debug issues related to the adapter, you can use the command line Bluetooth packet analyzer/sniffer tool called “hcidump” to analyze the Bluetooth traffic. This tool does not have a prebuilt package, so it needs to be built from source using the following steps:

```
$ sudo apt-get install autoconf
$ cd ~/
$ mkdir hci
$ cd hci/
$  wget  -c  http://www.kernel.org/pub/linux/bluetooth/bluez-hcidump-2.5.tar.xz
$ tar xvfJ bluez-hcidump-2.5.tar.xz
$ cd bluez-hcidump-2.5
$ autoconf
$ automake --add-missing
$ ./configure
$ make
$ sudo make install
$ sudo hcidump
HCI sniffer - Bluetooth packet analyzer ver 2.5
device: hci0 snap_len: 1500 filter: 0xffffff
…
```



Debug Bluetooth traffic using hcidump

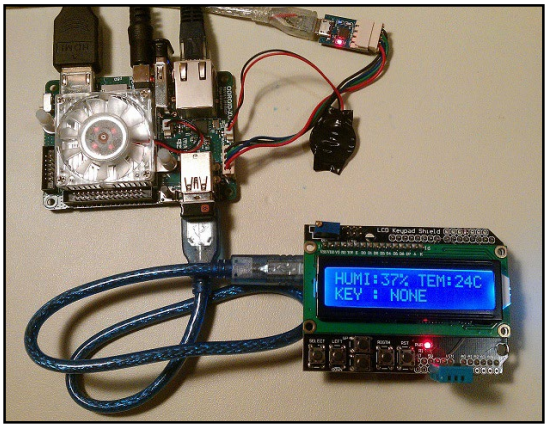
Using a device that has Bluetooth support, enable it and scan for the XU4’s Bluetooth device called “odroid-0”. Select it in order to pair it with the XU4. Debug information will appear in the terminal instance where the hcidump utility was started, as shown in the image above.

ODUINO ONE

The ODROID-compatible Arduino called the ODUINO ONE includes the following components, all packaged into one experimentation unit:

- Arduino Uno R3 + a shield containing,
- a 16x2 LCD display,
- a DHT11 one-wire humidity / Temperature sensor, and
- a keypad (four directional keys)

The ONE package can be powered by the XU4 using the USB Std A/Std B cable. Connect the cable to the ODUINO ONE and the XU4. The ONE goes through the power-up process and displays the ambient humidity and temperature, as shown in the image below. It can also display the key that gets clicked on the keypad, and can be reset from the same keypad.



ODUINO ONE setup using Arduino UNO

Some additional information on the ONE package can be obtained using the following commands:

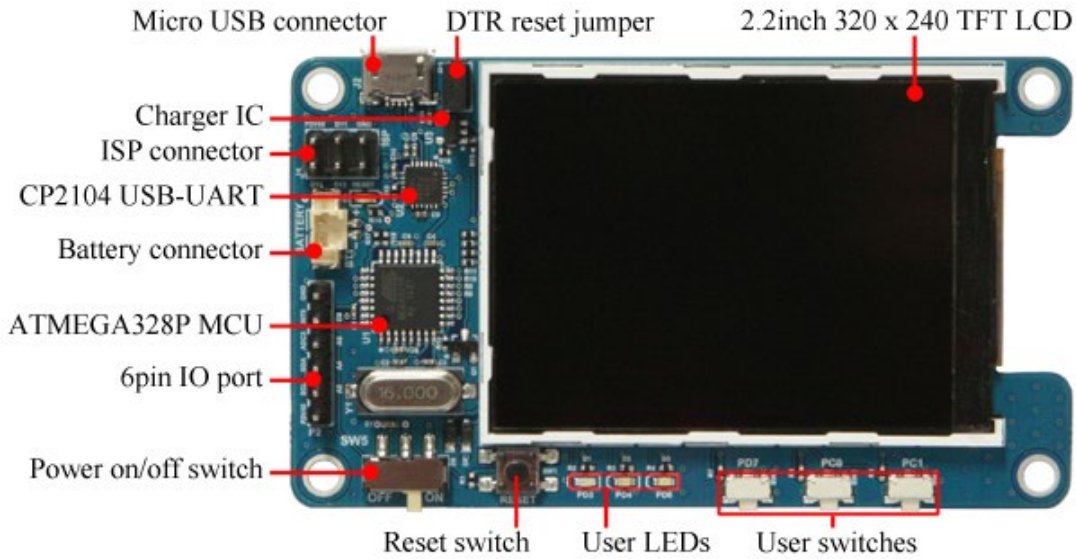
```
$ ls -lsa /dev/ttyA*
0 crw-rw---- 1 root dialout 166, 0 Oct 26 03:39 /dev/ttyACM0

$ lsusb
…
Bus 003 Device 005: ID 2341:0043 Arduino SA Uno R3 (CDC ACM)
…

$ lsusb -D /dev/bus/usb/003/005
Device: ID 2341:0043 Arduino SA Uno R3 (CDC ACM)
Couldn't open device, some information will be missing
Device Descriptor:
  bLength                 18
  bDescriptorType         1
  bcdUSB                  1.10
```

```
  bDeviceClass            2 Communications
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0         8
  idVendor                0x2341 Arduino SA
  idProduct               0x0043 Uno R3 (CDC ACM)
  bcdDevice               0.01
  iManufacturer           1
  iProduct                2
  iSerial                 220
  bNumConfigurations      1
…
```


The activity on the ONE package may also be transmitted to the XU4. Hardkernel has provided C sample source code to display this information transmitted by the ONE at `http://bit.ly/1Q1K3p2`. Note that it has been tested on some older platforms, but in order for it to work on the XU4, the sample code needs to be modified as described at `http://bit.ly/1p8uitU`.

ODROID-SHOW2

Hardkernel offers an Arduino compatible 2.2” 240×320 TFT-LCD display, called the ODROID-SHOW2, that can be used with the XU4, any other Hardkernel board, or even a PC.



ODROID-SHOW2 detail closeup

After attaching the SHOW2 to the XU4, use the following commands to obtain its information:

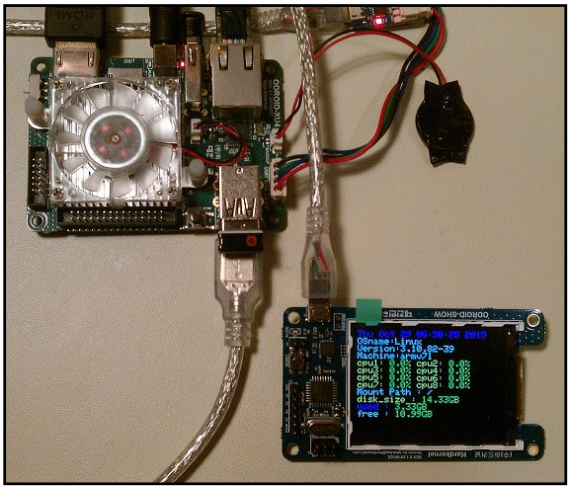
```
$ lsusb
…
Bus 003 Device 003: ID 10c4:ea60 Cygnal Integrated Products, Inc.
CP210x UART Bridge / myAVR mySmartUSB light
```

```
$ sudo lsusb -D /dev/bus/usb/003/003
[sudo] password for odroid:
Device: ID 10c4:ea60 Cygnal Integrated Products, Inc.  CP210x UART
Bridge / myAVR mySmartUSB light
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x10c4 Cygnal Integrated Products, Inc.
  idProduct          0xea60 CP210x UART Bridge / myAVR mySmartUSB
light
  bcdDevice            1.00
  iManufacturer           1 Silicon Labs
  iProduct                2 CP2104 USB to UART Bridge Controller
  iSerial                 3 00875559
  bNumConfigurations      1
…

$ ls -lsa /dev/ttyUSB*
0 crw-rw---- 1 root dialout 188, 0 Oct 29 06:07 /dev/ttyUSB0
```

There are ODROID-SHOW2 programming examples at `http://bit.ly/1NcQwd7`. Obtain the SHOW2 sample source code and build a sample using the commands:

```
$ cd ~
$ sudo apt-get install git
$ git clone https://github.com/hardkernel/ODROID-SHOW
$ cd ODROID-SHOW/example/linux
$ gcc -o status status.c && sudo ./status
```

You should see the LCD display similar to the image below.



ODROID-SHOW2 displaying some system statistics

Additional details, such as burning new firmware for the SHOW, along with sample applications, may be found at `http://bit.ly/1toe7Pl`. One important aspect of the SHOW2 board, is the presence of a Data Terminal Ready (DTR) reset jumper/switch. Its role is important during burning of new firmware. The following precautions need to to be followed, regarding the DTR jumper:

- It must be installed when you upload the firmware, and
- It must NOT be installed during normal usage mode.

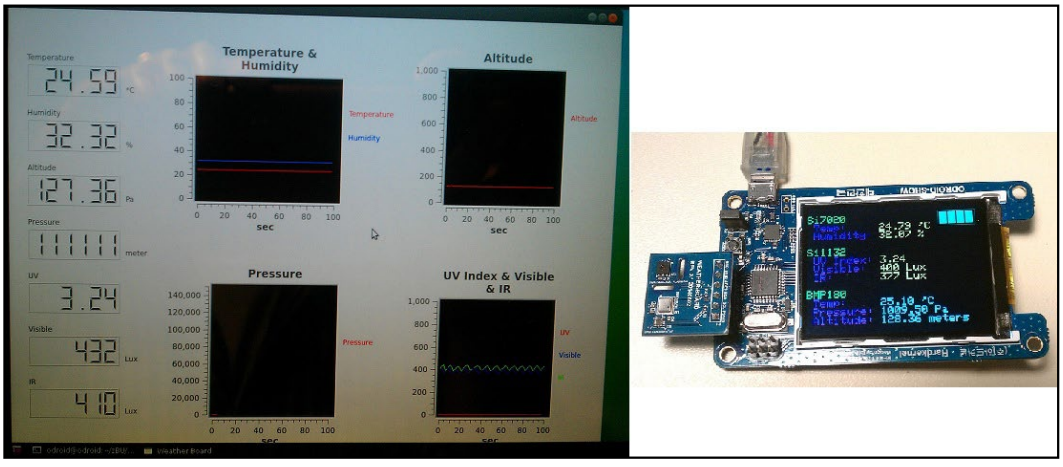
Weather Board

The Weather board is an add-on developed by Hardkernel that may be used with either the SHOW2 or a 16x2 LCD device. Follow the following steps to get the Weather board to work with the SHOW2 and the ODROID-XU4.

First, power down the XU4. Attach the Weather board to the SHOW2 as described in the wiki at `http://bit.ly/1IG1LvF`. Then, attach the assembly using the USB cable to the XU4. Turn on the power to the XU4. The link above also lists the steps necessary to install the Arduino-compatible firmware for displaying the temperature, humidity, pressure, light levels, and altitude on the SHOW2.

Next, install Hardkernel's Ubuntu sample application for displaying the same information in the QT-based user-interface on the XU4:

```
$ sudo apt-get install qt4-default qt4-designer libqwt-dev
$ export GIT_SSL_NO_VERIFY=1
$ git clone https://github.com/hardkernel/ODROID-SHOW
$ cd ODROID-SHOW/qt_weather
$ uic weather_board.ui > ui_weather_board.h
$ qmake
$ make
$ sudo ./WEATHER_BOARD
```



Weather board on SHOW2 and monitor

The image above shows both the assembly and the QT-based application's user-interface displaying the real-time weather data.

DC Plug Cable Assembly 5.5mm

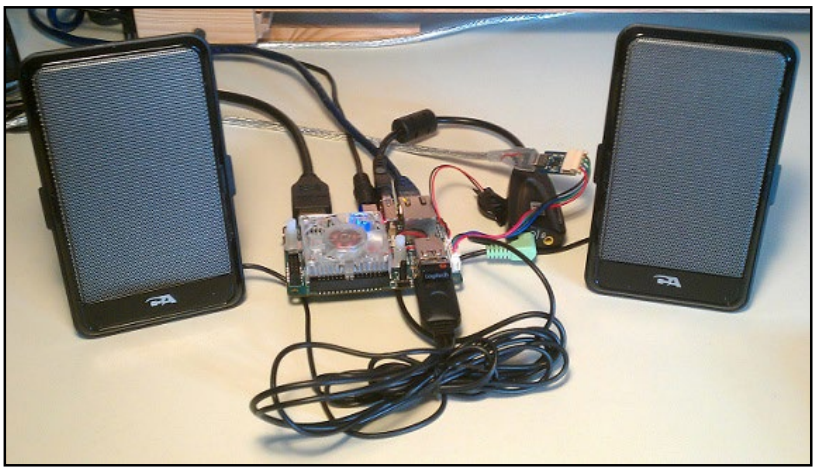
The DC Plug cable accessory, shown below, is typically used to provide power to the XU4 using a special purpose power supply. These power supplies could include bench-top models or Hardkernel's SmartPower. Ensure that the power supply provides clean power, rated at 5V, 2.0+ Amps. Use the cable colors to ensure polarities match at either end.

DC plug cable assembly

USB Audio Adapter

Attach the USB-powered audio adapter to the XU4. After a few moments, check to see if the device is detected:

```
$ lsusb
…
Bus 002 Device 002: ID 0d8c:000c C-Media Electronics, Inc.  Audio
Adapter
```



USB audio adapter with USB-powered speakers

This adapter is based on the CM108AH single-chip USB audio solution, and is capable of stereo output and includes the dual DAC/ headphone amplifier, ADC, microphone booster, PLL, regulator, and USB transceiver. While the setup can be tested with headphones con-

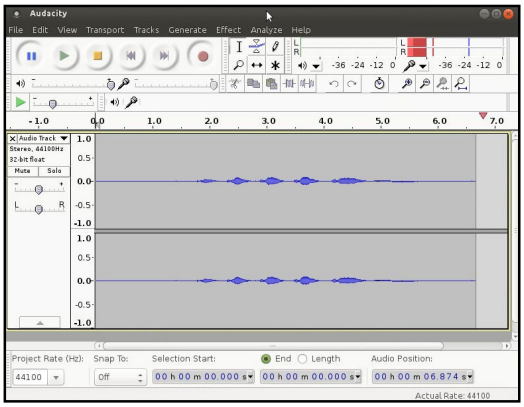
nected to the headphone port on the adapter, one can also test it with a USB-powered stereo speaker set as shown in the image above.

Adjust the volume using the speaker icon on the top right of the desktop and set to ~25% for safety. Start a web browser like Firefox and launch a Youtube link in order to play some video with audio.

This adapter also includes a mono microphone jack. Attach a 3.5mm microphone to the adapter, then install Audacity and PulseAudio Volume Control applications using the following commands:

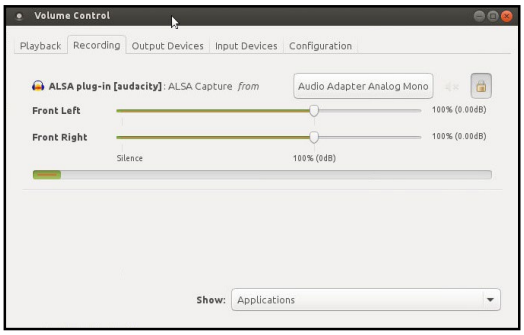
```
$ sudo apt-get install audacity
$ sudo apt-get install pavucontrol
```

Launch Audacity from the Applications → Sound & Video menu, then start recording a sample audio stream. The image below illustrates a recording in progress.



Audacity screen during a recording

Click the small green Play icon on the 3rd row of icons. You should be able to listen to your recording over the attached speakers/headphones. You can also verify the recording in progress using the PulseAudio Volume Control application. To do so, launch the application from the Applications menu. Select the Recording tab and start recording. You should observe the recording level in the green bar located midway on the screen as shown in the image below.



PulseAudio Volume Control showing recording progress

USB-SPDIF

Hardkernel makes an XU4-compatible S/PDIF (Sony/Philips Digital Interface Format) kit with a USB interface, as shown below. This allows you to hook up the audio from the XU4 via USB to an amplifier (A/V receiver), when the HDMI audio out option is not used.

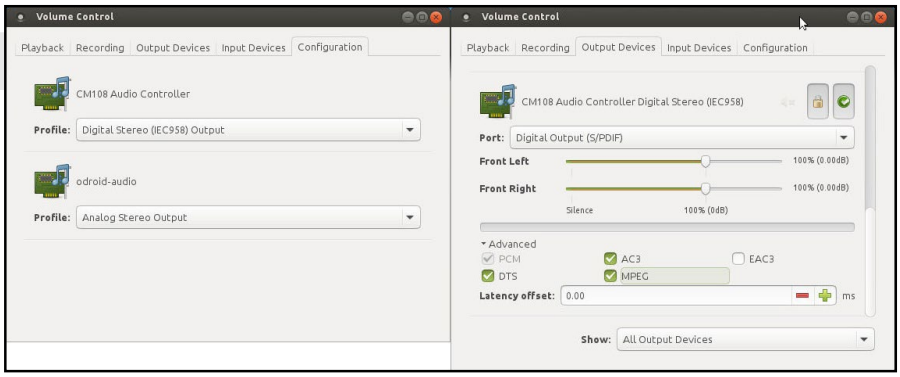
USB-SPDIF kit

Attach the cable provided in the kit to the device on one end and the XU4's USB port on the other end. Reboot the system. You can check to see if the device is detected using the commands:

```
$ lsusb
…
Bus 003 Device 007: ID 0d8c:013c C-Media Electronics, Inc.  CM108
Audio Controller
…

$ sudo lsusb -D /dev/bus/usb/003/007
[sudo] password for odroid:
Device: ID 0d8c:013c C-Media Electronics, Inc.  CM108 Audio Con-
troller
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               1.10
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0         8
  idVendor           0x0d8c C-Media Electronics, Inc.
  idProduct          0x013c CM108 Audio Controller
  bcdDevice            1.00
  iManufacturer           1 C-Media Electronics Inc.
  iProduct                2 USB PnP Sound Device
  iSerial                 0
  bNumConfigurations      1
…
```

Once the device is verified to be recognized by the system, re-

boot the system, then configure it by launching PulseAudio Volume Control from the Applications menu. Then, in the Configuration tab, select the Digital Stereo Output profile, and select the Output Devices tab and update the configuration for the CM108 device as shown below. Note that TrueHD or DTS-MA pass-through is not supported.



PulseAudio Volume Control settings for the SPDIF

Connect a TOSLINK optical cable to this SPDIF peripheral and an AV Receiver capable of accepting the optical TOSLINK cable. The AV Receiver should be connected to an appropriate set of speakers. Power up the AV Receiver and select the appropriate input option on the AV Receiver. Launch the browser on the XU4 and access a YouTube video link that has audio included. Play it in order to verify that the audio is audible.

USB-CAM 720P

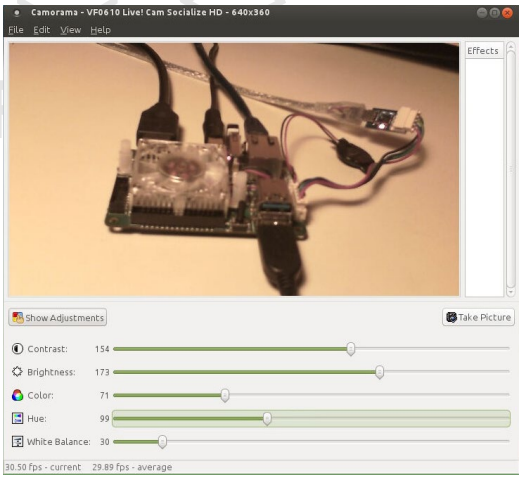
Hardkernel offers a USB-CAM rated at 720p with up to 30 fps. The image below shows a camera that is certified to work with the XU4:

720p 30fps USB-CAM

Attach the USB-CAM to an available USB port on the XU4, and wait for a few moments. Install the camorama application using the following command:

```
$ sudo apt-get install camorama
```

Launch the camorama application from the Applications → Graphics → Camorama Webcam Viewer menu of the desktop, then point the camera at a well-lit object. You should see a screen similar to the image below.



Camorama application

USB3 / SATA3 HDD/SSD Interface Kit and SATA Bridge Board

The USB3/SATA3 HDD/SDD interface kit offered by Hardkernel can be used to attach a SATA HDD or SSD to the XU4. The package includes the docking system, a power supply (12V/2A) and a USB3.0 cable. This package is compatible with Android and linux on the XU4.

From a Terminal window, obtain details about the USB3/SATA3 bridge component used in the docking system using the following command:

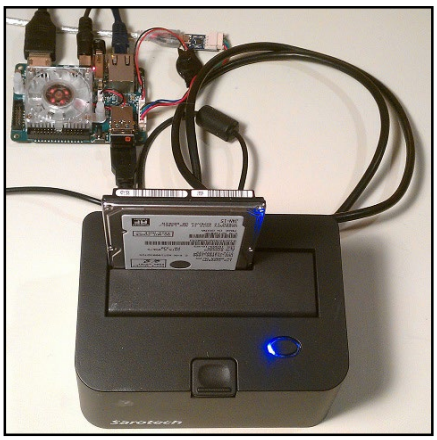
```
odroid@xu4-2:~$ lsusb
…
Bus 004 Device 003: ID 174c:55aa ASMedia Technology Inc.  ASMedia
2105 SATA bridge    …

$ sudo lsusb -D /dev/bus/usb/004/003
[sudo] password for odroid:
Device: ID 174c:55aa ASMedia Technology Inc.  ASMedia 2105 SATA
bridge
Device Descriptor:
  bLength             18
  bDescriptorType          1
  bcdUSB               3.00
  bDeviceClass             0 (Defined at Interface level)
  bDeviceSubClass          0
  bDeviceProtocol          0
  bMaxPacketSize0          9
  idVendor           0x174c ASMedia Technology Inc.
  idProduct          0x55aa ASMedia 2105 SATA bridge
  bcdDevice            1.00
  iManufacturer            2 Asmedia
```

```
  iProduct                3 ASM1051
  iSerial                 1 0123456789ABCDEF0124
  bNumConfigurations      1
…
      Device can operate at Full Speed (12Mbps)
      Device can operate at High Speed (480Mbps)
      Device can operate at SuperSpeed (5Gbps)
      bFunctionalitySupport   1
      Lowest fully-functional device speed is Full Speed (12Mbps)
      bU1DevExitLat           10 micro seconds
      bU2DevExitLat           2047 micro seconds
…
```

The docking system uses the ASMedia's ASM1051E single-chip based SuperSpeed USB (USB 3.0) to Serial ATA3 bridge, and the Bridge Board uses a Genesys Logic bridge. Follow these steps to use it:

- Shutdown the XU4 and turn the power off
- Attach the docking system to the XU4 with the included USB3 cable
- Insert the SATA storage device (HDD/SSD) with its SATA interface aligned with the receptacle in the docking system
- Gently press down the storage device so the SATA connectors connect well and the storage device sits firmly in the docking system
- Turn on the power to the XU4 and let it complete the boot up process.

The image below shows the installation of new 1TB HGST Travelstar 2.5" SATA3 HDD suitable for laptops. It worked well with SATA3 SSDs too, such as the PNY 240GB XLR8 model. Some older desktop SATA2 desktop-class HDDs may also work.



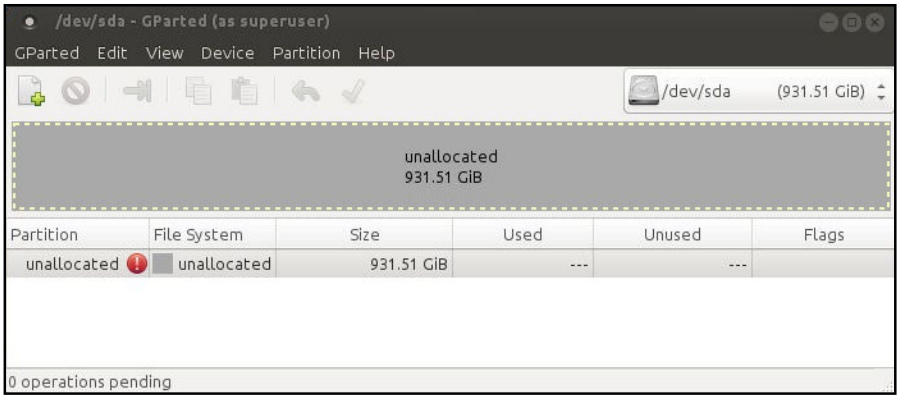
Docking system with SATA HDD storage device attached to XU4

A brand new storage device, if unformatted, will not be automat-

ically detected by the docking system. You can follow the steps below to use the storage device on an XU4. First, install gparted using the command:

```
$ sudo apt-get install gparted
```

Next, start the gparted application from the desktop's System → Administration menu. Select the device, typically named /dev/sda if no other storage devices exist on the system. You should see the screen shown below.

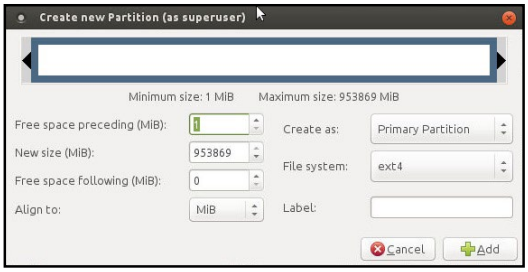


Initial gparted view of new HDD

Select the unallocated partition and select the Device → Create Partition Table menu item. You will be presented with a warning as shown below.

Partition creation warning

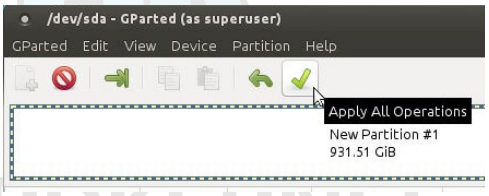
Click the Apply button, and you will be presented with a screen to enter the new partition's information, as shown below.



New partition information

Enter a label name such as "xu4-hdd-01" and click the Add button. You will be presented with the next screen. Select the new parti-

tion and Click on the check icon in order to apply/save all changes, as shown below.

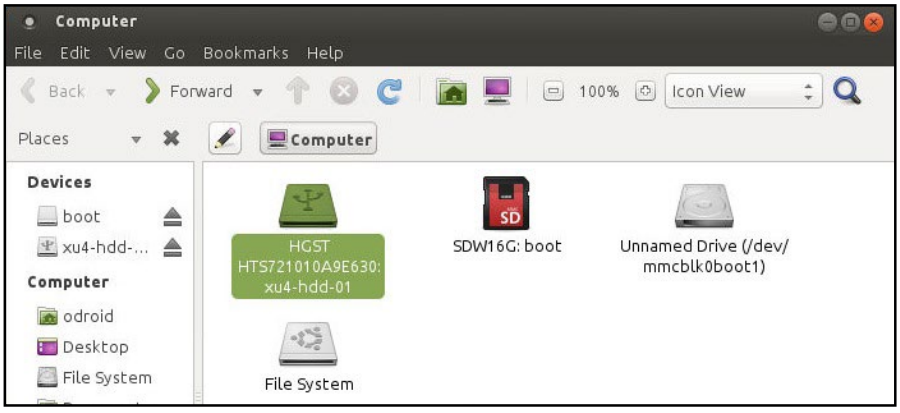


Save partition information

Reboot the system, then use the following command to validate the creation of the new partition:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            746M  0   746M   0% /dev
tmpfs           200M  6.9M  193M   4% /run
/dev/mmcblk0p2   15G  3.8G   10G  28% /
tmpfs           998M  160K  998M   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           998M  0   998M   0% /sys/fs/cgroup
/dev/mmcblk0p1  128M  7.1M  121M   6% /media/boot
cgmfs           100K  0   100K   0% /run/cgmanager/fs
tmpfs           200M  8.0K  200M   1% /run/user/118
tmpfs           200M   28K  200M   1% /run/user/1000
/dev/sda1       917G   72M  871G   1% /media/odroid/xu4-hdd-01
```

You can also verify using the Console application by selecting the Places → Computer menu from the desktop. You can see that the HDD has been installed and mounted, ready for use, as shown below.



New hard drive mounted and ready for use

USB3 / SATA2 HDD/SSD RAID 0/1 Enclosure

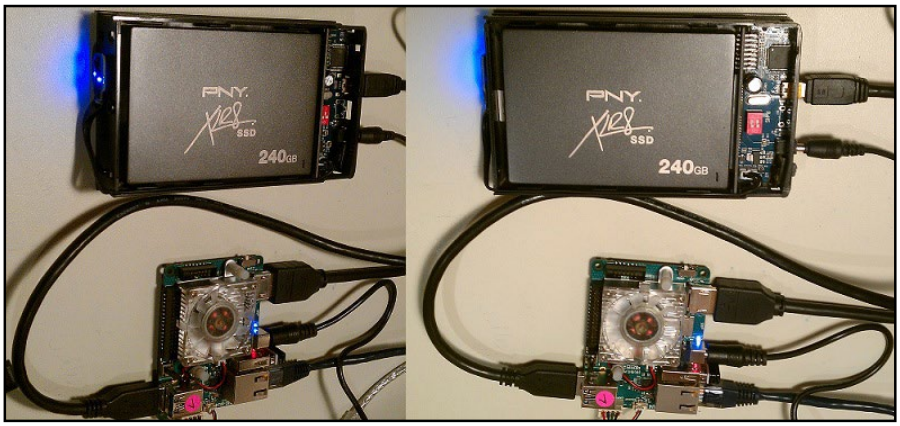
As of February 2017, Hardkernel does not yet provide a kit that officially supports RAID 0/1 support for HDDs/SSDs. However, you may be successful in using some 3rd-party peripherals, such as the CineRAID CR-H212 2-bay 2.5” HDD/SDD RAID enclosure, with the

most recent (3.10.92+) Linux images. No new software needs to be installed. It has not been tested under any Android version. The enclosure required its own 5V/4A power source (purchased separately), since the XU4 cannot provide the peak startup power requirements of the enclosure via the USB3 port.

Please note that Hardkernel does not endorse or support this enclosure and cannot be expected to provide help with its use on the XU4. This enclosure is mentioned here only to satisfy your experimental curiosity and no guarantee is offered. Your luck in getting to use unsupported devices such as this enclosure with an XU4 may vary. The details of the enclosure may be discovered with the following command:

```
$ lsusb
...
Bus 004 Device 003: ID 067b:2775 Prolific Technology, Inc.
Bus 004 Device 002: ID 05e3:0616 Genesys Logic, Inc.
...
```

The first entry reflects the RAID controller, and the second entry reflects the USB3 controller used in the enclosure. The image below shows the setup using two (2) PNY XLR8 240GB SSDs in a RAID1 setup, with the enclosure cover removed to show internal details.



XU4 with two SSDs in a RAID1 setup

Make sure to follow the instructions provided along with the enclosure packaging in order to configure it for a RAID1 setup. As always, follow the standard procedure of shutting down power to the XU4 and peripherals before attachment and detachment. Power up the peripherals (if they have their own power supplies) before turning the power to the XU4, so they can be detected. Improper usage could damage the XU4.

After power up, follow the HDD/SSD partition setup instructions in the previous section. In this example, the partition was labeled

"xu4-01-raid1". The XU4 sees the enclosure setup as just one drive called RAID1. Power off the entire setup and power them back up, then run the following command in order to validate the setup:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            746M     0  746M   0% /dev
tmpfs           200M  6.8M  193M   4% /run
/dev/mmcblk0p2   15G  3.8G   10G  28% /
tmpfs           998M   88K  998M   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           998M     0  998M   0% /sys/fs/cgroup
/dev/mmcblk0p1  128M  7.1M  121M   6% /media/boot
cgmfs           100K     0  100K   0% /run/cgmanager/fs
tmpfs           200M   20K  200M   1% /run/user/1000
/dev/sda1       237G   63M  225G   1% /media/odroid/xu4-01-raid1
```

Note that the file-system entry of /dev/sda1 is the same as the entry for the HDD storage device used in the official Hardkernel docking system kit described in the previous section. No two entries can actually have the same filesystem names. They are the same here since these sections were written with only one external storage device attached at a time. These kits and enclosures can used at the same time provided that they use their own power supplies, since the XU4 cannot deliver the high power requirements through the on-board USB3 ports. They will have different device names if used simultaneously.

Since the RAID1 functionality is provided by the hardware in the enclosure, the disk access speeds under optimal test conditions are quite reasonable:

```
$ sudo hdparm -t /dev/sda1
/dev/sda1:
 Timing buffered disk reads: 296 MB in  3.01 seconds =  98.33 MB/sec

odroid@xu4-2:~$ sudo hdparm -T /dev/sda1
/dev/sda1:
 Timing cached reads:   1724 MB in  2.00 seconds = 863.05 MB/sec
```

The actual performance speeds may vary based on your setup.

USB GPS Module

Hardkernel produces a 5V/0.1A GPS receiver with an USB interface that supports the standard National Marine Electronics Association (NMEA) GPS protocol, as shown below. The device uses the Ublox 6010 chipset, with support already built into the official Hardkernel Ubuntu image. One should be able to attach the device to an available USB port on the XU4 and start using it right away.

USB GPS module

```
$ lsusb
…
Bus 003 Device 003: ID 1546:01a6 U-Blox AG
…
$ sudo ls -lsa /dev/ttyA*
0 crw-rw---- 1 root dialout 166, 0 Nov 18 05:26 /dev/ttyACM0
```

Once the relevant TTY port is obtained, you can check to see if the GPS receiver is working properly with the following command:

```
$ sudo cat /dev/ttyACM0 | grep GPRMC
$GPRMC,161053.00,A,3719.54074,N,12201.49867,W,0.079,,110415,,,A*65
$GPRMC,161054.00,A,3719.54074,N,12201.49867,W,0.085,,110415,,,A*65
$GPRMC,161055.00,A,3719.54074,N,12201.49867,W,0.024,,110415,,,A*66
…
```

The RMC - NMEA has its own version of essential GPS pvt (position, velocity, time) data, which represents the following information:

```
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

RMC                 Recommended Minimum sentence C
123519              Fix taken at 12:35:19 UTC
A                   Status A=active or V=Void.
4807.038,N          Latitude 48 deg 07.038' N
01131.000,E         Longitude 11 deg 31.000' E
022.4               Speed over the ground in knots
084.4               Track angle in degrees True
230394              Date - 23rd of March 1994
003.1,W             Magnetic Variation
*6A                 The checksum data, always begins with *
```

To test the higher level functionality of the GPS dongle, we can use the services of gpsd, a service daemon, that monitors one or more GPS modules and makes the pcv (position, course, velocity) data available via the TCP port 2947 of the host system.

Install gpsd and relevant utilities using the following command:

```
$ sudo apt-get install gpsd gpsd-clients foxtrotgps
```

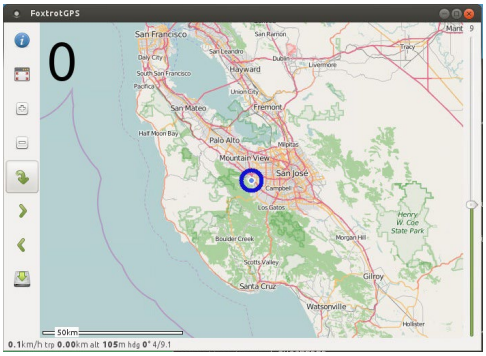
Then, configure gpsd using the following command and options and reboot:

```
$ sudo dpkg-reconfigure gpsd
```


From a Terminal window, launch the foxtrotgps application using the following command:

```
$ foxtrotgps
```

Note that the display of a live map, as shown below, requires the presence of a working internet connection. If you wish to see a real-time map while you are driving in a vehicle with this setup, you will need to use your smartphone as a hot-spot and have the setup talk to it via WiFi.



Foxtrotgps display

myAHRS+ Board

If you are looking for an Attitude Heading Reference System (AHRS) that is minimally influenced by acceleration and magnetic disturbances, the low-cost USB2-based myAHRS+ board offered by Hardkernel is a good choice. It also sports an I2C interface useful in an embedded application like Arduino-based projects.

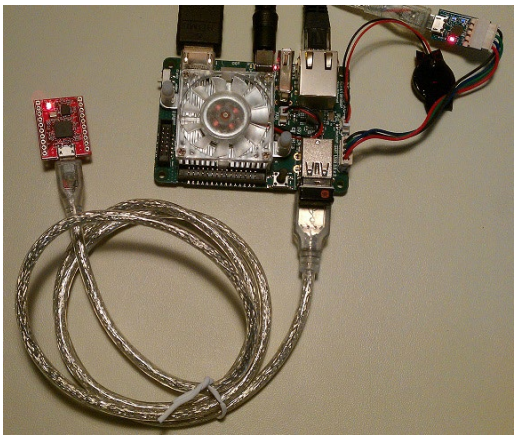
The sensors it includes are:

- triple axis 16-bit gyroscope : ± 2000 dps
- triple axis 16-bit accelerometer : ± 16 g
- triple axis 13-bit magnetometer : ± 1200 μT

Attach the board to the USB2 port on the XU4. After a few moments, you should observe the following two LEDs:

• Red LED:
When ON, it implies that the myAHRS+ is connected to PC via USB ok
• Yellow LED:
When blinking, it means that myAHRS+ is in normal state.

The image below shows the board attached to the XU4.



myAHRS+ on XU4

To get the device information, run the following commands:

```
$ lsusb
…
Bus 003 Device 003: ID 0483:5740 STMicroelectronics STM32F407
…

$ sudo lsusb -D /dev/bus/usb/003/003
Device: ID 0483:5740 STMicroelectronics STM32F407
Device Descriptor:
  bLength                 18
  bDescriptorType          1
  bcdUSB                   2.00
  bDeviceClass             2 Communications
  bDeviceSubClass          0
  bDeviceProtocol          0
  bMaxPacketSize0          64
  idVendor                 0x0483 STMicroelectronics
  idProduct                0x5740 STM32F407
  bcdDevice                2.00
  iManufacturer            1 STMicroelectronics
  iProduct                 2 STM32 Virtual COM Port
  iSerial                  3 000001010000
  bNumConfigurations       1
...
```

The board should be configured using the steps described at http://bit.ly/1TGve9g. Run the following command to view typical out-

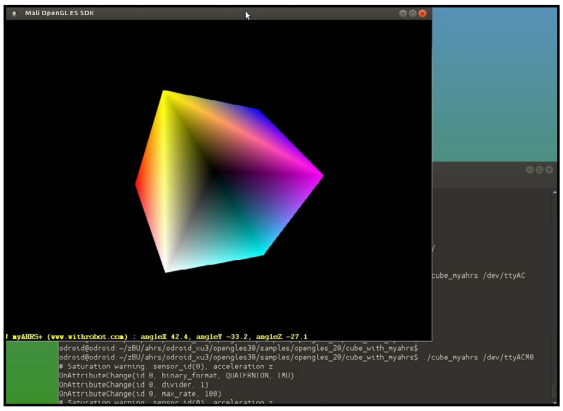
put from the board while moving the board around:

```
$ sudo cat /dev/ttyACM0
$RPY,58,7.95,1.95,79.85*59
$RPY,61,75.92,66.04,132.32*6A
$RPY,62,88.22,59.02,148.94*6B
$RPY,70,-12.06,59.27,48.14*7E
$RPY,85,4.25,22.07,41.41*68
…
```


XU4-specific sample applications can be obtained at http://bit.ly/1jU6VZj. Run the following commands to test an OpenGL ES sample:

```
$ cd ~ && mkdir ahrs && cd ahrs/
$ sudo apt-get install subversion libapache2-svn
$ svn export https://github.com/withrobot/myAHRS_plus/trunk/\
  odroid_xu3
$ cd odroid_xu3/opengles30/
$ chmod 777 *.sh
$ ./config.sh && make -j8
$ cd samples/opengles_20/cube_with_myahrs
$ ./cube_myahrs /dev/ttyACM0
```

The image below shows the OpenGL ES sample application's display.



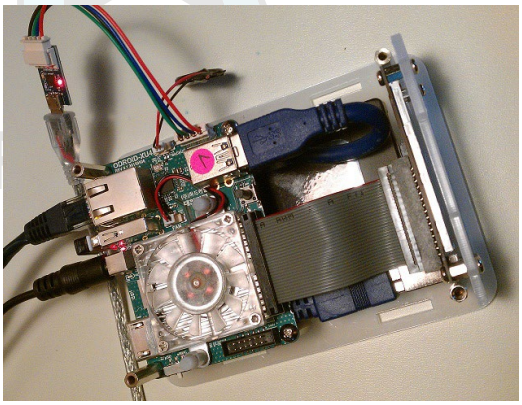
myAHRS+ sample application

Cloudshell

The powerful XU4, along with its I/O, makes for a very cost effective DIY Network Attached Storage (NAS) Solution. Hardkernel has developed a compact NAS solution called the Cloudshell. It is essentially a modern case that includes the following:

- a color 2.2" 320 x 240 TFT LCD console display
- a USB3.0 to SATA bridge
- mounting area for a 2.5inch HDD/SSD
- an IR receiver and required USB3 cable

The assembly instructions can be found at http://bit.ly/1N3xNm7. The image below shows an assembled cloudshell with the top cover removed.



Assembled cloudshell

You can configure the LCD using the steps at http://bit.ly/1Laq7IS, which are detailed below:

```
$ sudo -s
$ echo “options fbtft_device name=hktft9340 busnum=1 rotate=270” >
/etc/modprobe.d/odroid-cloudshell.conf
$ echo “spi_s3c64xx” >> /etc/modules
$ echo “fbtft_device” >> /etc/modules
```

Then edit /etc/modprobe.d/blacklist-odroid.conf, and remove the blacklist on SPI:

```
# IO Board
blacklist ioboard_bh1780
blacklist ioboard_bmp180
blacklist ioboard_keyled

# SPI
# blacklist spidev
# blacklist spi_s3c64xx

# 3.2” LCD Touchscreen driver
blacklist ads7846
```

The commands above will ensure that at every boot, the saved LCD configuration is used. Reboot the board with the HDMI cable disconnected and you should see already information going to the 2.2” LCD. To ensure smartmontools (Genesys Logic USB3 / SATA support in Cloudshell) Ver. 6.5.4132 or higher is installed, run the following command:

```
$ sudo apt install smartmontools
```

Hardkernel also offers an infrared remote control that can be configured to work the Cloudshell using the following commands:

```
$ wget https://raw.githubusercontent.com/mdrjr/\
cloudshell_ir/master/install_ir.sh
$ chmod +x install_ir.sh && sudo ./install_ir.sh
```


The Xorg settings (etc/X11/xorg.conf) to enable the LCD and disable HDMI are shown below:

```
Section "Device"
        Identifier      "ODROID"
        Driver          "fbdev"
        Option          "fbdev"         "/dev/fb0"
        Option          "Debug"         "false"
        Option          "DPMS"          "false"
EndSection
Section "Screen"
        Identifier      "Default Screen"
        Device          "ODROID"
EndSection
Section "ServerLayout"
        Identifier      "Default Layout"
        Option          "BlankTime"     "0"
        Option          "StandbyTime"   "0"
        Option          "SuspendTime"   "0"
        Option          "OffTime"       "0"
EndSection
Section "DRI"
        Mode            0666
EndSection
```

Because of the small size of the the LCD display, it can be difficult to access the icon for shutting down the system. However, you can add a new panel with the shutdown icon on it, as shown below.

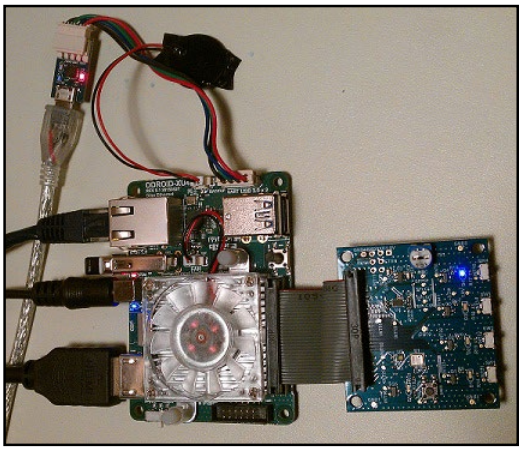
Cloudshell with Linux desktop

Expansion Board

Hardkernel offers a convenient expansion board compatible with the XU4. It has the following inputs and outputs:

- 4 x buttons (GPIO)
- 1 x button (Power On)
- 5 x LEDs (GPIO)
- 1 x SPI Flash 2Mbit (Upto 20Mhz SPI clocking)
- 1 x I2C Temperature/Pressure sensor BMP180
- 1 x I2C Ambient Light sensor BH1780GLI
- 1 x Trimpot(variable resistor) for ADC access

The image below shows the expansion board after being connected to the XU4.



XU4 with expansion board

The latest version of Linux has the driver support for this expansion board. The article at `http://bit.ly/1Y3viqf` details the steps that can be used to test the board:

```
$ su
# modprobe i2c-gpio-custom bus0=10,33,23,10,10
# modprobe ioboard-bmp180
# echo ioboard-bmp180 0x77 > /sys/class/i2c-dev/i2c-10/\
device/new_device
# modprobe ioboard-bh1780
# echo ioboard-bh1780 0x29 > /sys/class/i2c-dev/i2c-10/\
device/new_device
# echo 1 > /sys/class/i2c-dev/i2c-10/device/10-0077/enable
# echo 1 > /sys/class/i2c-dev/i2c-10/device/10-0029/enable
```

Run the following commands to obtain the temperature, pressure, and light levels respectively:

```
# cat /sys/class/i2c-dev/i2c-0/device/0-0077/temperature
240
# cat /sys/class/i2c-dev/i2c-0/device/0-0077/pressure
100985
# cat /sys/class/i2c-dev/i2c-0/device/0-0029/lux
335
```

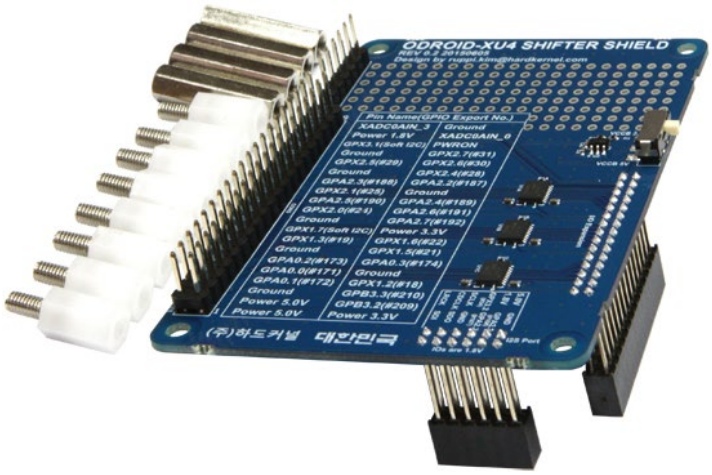
The output values imply the following measurements:

```
240 = 24.0C
100985 = 1009.85Pa
335 = 335Lux
```

Shifter Shield

Many devices already available for integration support either 3.3V or 5.0V. However, the native GPIO pins on the XU4 operate at 1.8V. Because of this difference, a level shifter is needed to allow inter-operation between the XU4 and these devices.

This board contains 3 x TSX0108E bi-directional level shifter ICs. The 40-pin GPIO port on the XU4 is converted to the more common 0.1inch (2.54mm) pitch 40-pin header which is compatible with the spacing on the C1+ and Raspberry Pi/Pi 2. The signals can be level shifted to 3.3V or 5V, whereas the operating voltage can be set using the small hardware slide switch on the board. The image below shows the components provided with the kit.



Shifter shield components

The display is a white on blue LCD display with tactile switches and I/O pins. The WiringPi library provides a convenient programming interface, which may be built using the following commands:

```
$ git clone https://github.com/hardkernel/wiringPi
$ cd wiringPi
$ ./build
```

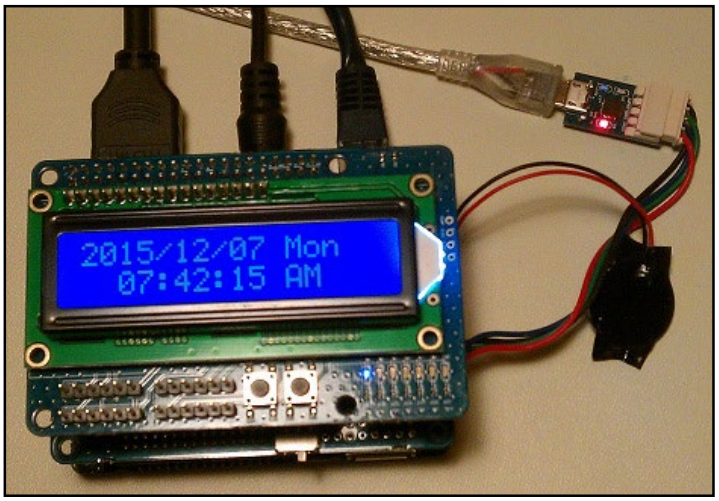
The sample source can be downloaded by visiting the URL `http://bit.ly/1fbtFlE`. Copy the sample project using the following commands:

```
$ cd ~ && mkdir lcdio && cd lcdio && mv ~/Downloads/lcd_cpuInfo.c .
```

Then, build and launch the application using the following commands, which should display the date and time:

```
$ gcc -o lcd_cpuInfo lcd_cpuInfo.c -lwiringPi \
  -lwiringPiDev -lpthread
$ sudo ./lcd_cpuInfo
```

Refer to the C1+ wiki to look at other C1+ peripherals and sample code that can be used with the XU4/Shifter-shield combination, which includes devices such as the Tinkering Kit. The shifter can be tested with the 16x2 LCD compatible with the XU4. The image below shows the assembly.



Shifter shield with 16x2 LCD

ODROID-VU7

If you want to add a 7” HDMI display that supports 5-point multi-touch to your XU4 with low power requirements, Hardkernel has developed the VU7 kit for this very purpose. Linux is supported with minimal setup, and Android compatibility is expected in early 2016. It supports the 800x480 resolution, and offers the ability to enabled or disable the backlighting. Along with the 7” screen, the kit contains the following items:

- Micro USB link board
- HDMI link board

- Micro to Type A USB Cable
- Micro to Micro USB Cable
- Type A to Type A HDMI cable
- Screws and nuts required for assembly

Some of the parts are provided to interface to the XU4, while others, such as the link boards, are useful for the devices such as the C1+. Using Linux kernel 3.10.92-63+ or higher, while attached to another display or via SSH, access the Terminal and type the following commands:

```
$ cd /media/boot
$ nano boot.ini
```

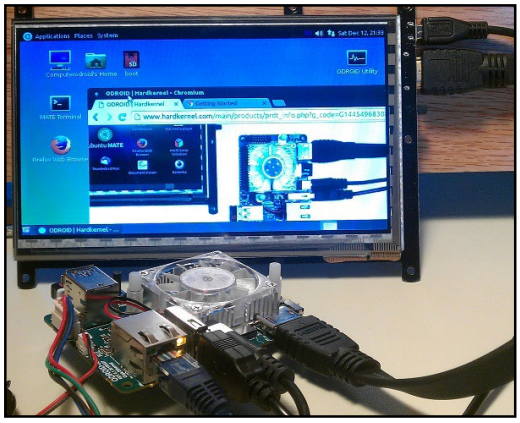
Then, enable the following entries in the boot.ini file:

```
setenv videoconfig “drm_kms_helper.edid_firmware=edid/800x480.bin”
setenv vout “dvi”
```

Make sure that no other videoconfig or vout value is set. The rest of the related settings should be commented out. Save these new settings and shut the system down.

Reconnect the XU4 to the VU7 display, with the touch interface USB cable attached to the USB port on the XU4. Power up the assembly and login to the desktop. The assembly will appear as shown below.

In Linux, the Chromium browser supports pinch and zoom gestures. Additional details on using and setting up the ODROID-VU7 are available at `http://bit.ly/1NWxgDx`.



XU4 with the VU7 display

VuShell for ODROID-VU7

If you wish to create a functional desktop PC that employs the XU4, the VU7 and other appropriate accessories, you could use the

laser-cut acrylic enclosure of the VuShell. It allows two viewing positions, and comes in two colors - smoky blue and smoky white. Details of the assembly process can be obtained at `http://bit.ly/2b8lk6a`.

XU4 assembled with VuShell

ODROID-VU7 Plus

If you want to add a 7” HDMI 1024 x 600 display that supports 5-point multi-touch to your XU4 with low power requirements, Hardkernel has developed the VU7 Plus kit for this very purpose. It supports the 1024x600 resolution, and offers the ability to enable or disable the backlighting.

Along with the 7” screen, the kit contains the following items:

- Micro USB link board
- HDMI link board
- Micro-to-TypeA USB Cable
- Micro-to-Micro USB Cable
- TypeA-to-TypeA HDMI cable
- Screws and nuts required for assembly

Some of the parts are provided to interface to the XU4, while others, such as the link boards, are useful for the devices such as the C1+. Using Linux kernel 3.10.92-63+ or higher, while attached to another display or via SSH, access the Terminal and type the following commands:

```
$ cd /media/boot
$ nano boot.ini
```

Then, enable the following entries in the boot.ini file:

```
setenv videoconfig “drm_kms_helper.edid_firmware=edid/1024x600.bin”
setenv vout “dvi”
```

Make sure that no other videoconfig or vout value is set. The rest of the related settings should be commented out. Save these new settings and shut the system down.

Reconnect the XU4 to the VU7 Plus display, with the touch interface USB cable attached to the USB port on the XU4. Power up the assembly and login to the desktop. The assembly will appear as shown in the image below.

XU4 with VU7 Plus display

The VuShell can also be used with the VU7 Plus to create a functional desktop computer.

Micro USB-DC Power Bridge

When the XU4 is under heavy stress, its power requirements can increase sharply. As a result, the power supplied to the VU7/VU7-Plus displays can drop, which could lead to a flickering display. To ensure

proper isolated power supply to the display at all times, Hardkernel has developed a micro USB-DC power bridge board. It requires its own 5V 2A power supply with a 2.5mm plug.

ODROID-VU5

If you wish to use a display that consumes less power than the VU7 and VU7 Plus, Hardkernel has developed a 5" HDMI 800 x 480 display kit that supports 5-point multi-touch to your XU4. It supports 800x430 resolution, and offers the ability to enable or disable the back-lighting. Along with the 5" screen, the kit contains the following items:

- Micro USB link board
- HDMI link board
- Micro-to-TypeA USB Cable
- Micro-to-Micro USB Cable
- TypeA-to-TypeA HDMI cable
- Screws and nuts required for assembly

Some of the parts are provided to interface to the XU4, while others, such as the link boards, are useful for the devices such as the C1+. Using Linux kernel 3.10.92-63+ or higher, while attached to another display or via SSH, access the Terminal and type the following commands:

```
$ cd /media/boot
$ nano boot.ini
```

Then, enable the following entries in the boot.ini file:

```
setenv videoconfig "drm_kms_helper.edid_firmware=edid/800x480.bin"
setenv vout "dvi"
```

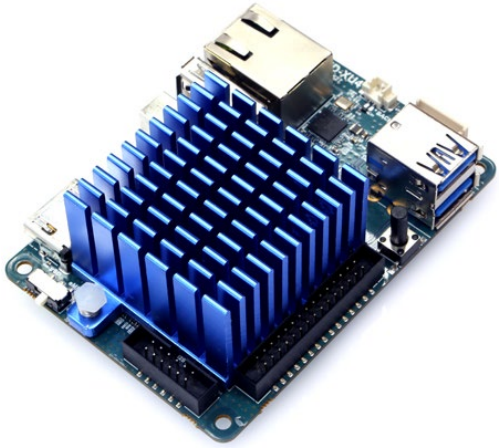
VU5 display

Make sure that no other videoconfig or vout value is set. The rest of the related settings should be commented out. Save these new settings and shut the system down.

Reconnect the XU4 to the VU5 display, with the touch interface USB cable attached to the USB port on the XU4. Power up the assembly and login to the desktop. The assembly will appear as shown in the image above.

Heat Sink

While the standard fan-based active cooler is very effective in dissipating heat from the CPU, it can be a noisy option. Noise increases as the fan speeds up under heavy load. If you want a low noise option, you could use a heat sink. While it is installed, you may notice the system throttling due to excessive heat, in which can you can reduce the maximum CPU clock speed to 1.4 GHz.



XU4 with heat sink

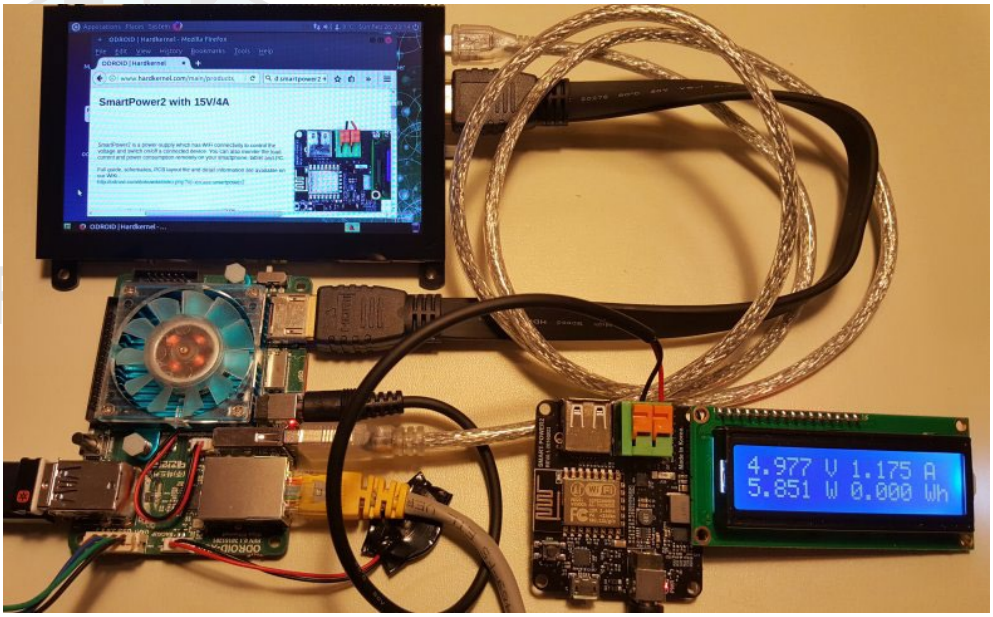
The steps required to install the heat sink are available at http://bit.ly/2l288mY.

SmartPower2

Hardkernel has developed the next generation SmartPower2 accessory with an input of 15V 4A and an output of 5.3V 1A (USB host port) and 5A (terminal block). The latter can be used to power an XU4. The SmartPower2 can be accessed via WiFI and used to control the output voltage and power ON/OFF of a connected device, like the XU4. You can monitor the power profile via a smartphone, tablet or PC.

It can be used in three operation modes:

- WiFi Standalone
- WiFi Connected
- Telnet Connected



XU4 with SmartPower2

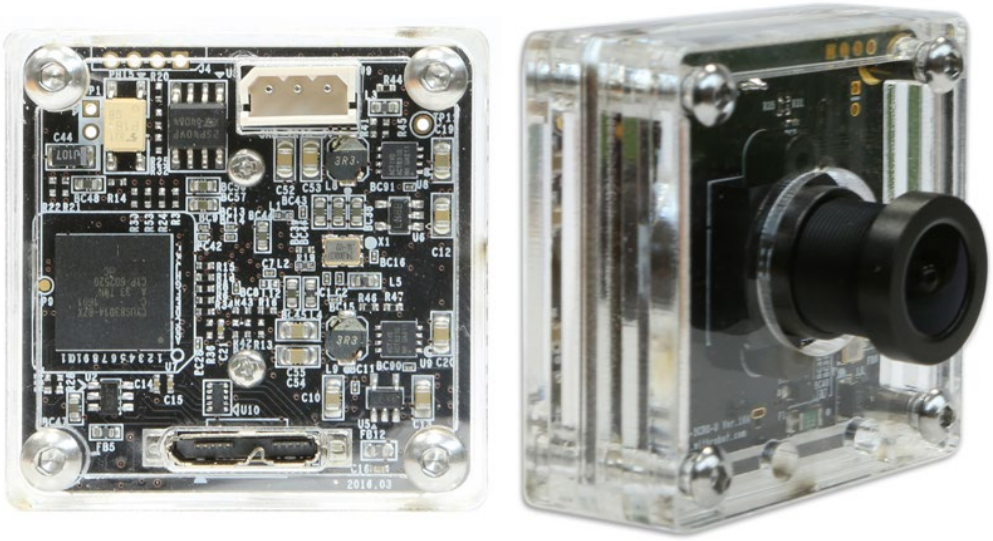
A detailed user guide (smartphone app, and telnet use) and schematics can be found at http://bit.ly/2jVXvOC. The guide describes how the firmware can be modified and rebuilt while attached to an x86 system. Please use caution in altering the firmware, since improper changes could result in the malfunction of the SmartPower2 and/or the attached device, resulting in the warranty being voided.

oCam : 5MP USB 3.0 Camera

If you are looking for a feature-rich capable camera to be used with the XU4, you can use the oCam-5CRO-U. It is offered by Hardkernel and in partnership with WITHROBOT Co., LTD. Some of its specifications include:

Sensor: OmniVision OV5640 CMOS image sensor
Lens: Standard M12 Lens with focal length of 3.6mm
Depth: 30-35 mm (Variable length for Manual Focusing)
FOV: 65 Degree
Shutter: Electric Rolling Shutter
Interface: USB 3.0 Super-Speed
Camera Control: Brightness, Contrast, Hue, Saturation, White Balance
Frame Rate (YUV): 2592x1944@7.5fps, 1920x1080@15fps, 1280x720@30fps, 640x480@120fps, 320x240@120fps
Frame Rate (MJPEG): 1920x1080@30fps, 1280x720@45fps, 640x480@30fps

This camera and its accessories perform the best in applications where motion artifacts have to be minimized, such as on drones (camera in motion) and capturing fast action images (stationary camera shooting fast moving automobiles).



oCam camera

A viewer application has been developed by the manufacturer, and instructions to install the prerequisites and the viewer are available at `http://bit.ly/2lMJMMI`. The commands are listed below:

```
$ sudo apt-get install qt4-default libv4l-dev libudev-dev
$ cd ~ && mkdir ocam && cd ocam
$ svn export https://github.com/withrobot/oCam/trunk/\
Software/oCam_viewer_Linux
$ cd ./oCam_viewer_Linux
$ mkdir build
$ cd ./build
$ qmake ..
$ make release
$ ./oCam-viewer
```

Several articles have been published that describe the versatile use of this camera with the XU4, available at `http://bit.ly/2lMG2e8`:

• Face Detection Using OCAM and ODROID-XU4: How To Recognize Human Features
• Camera Calibration Using OCAM and ODROID-XU4: A Technical Tutorial
• Augmented Reality: Using the OCam and ODROID-XU4
• Object Tracking Using OCam and ODROID-XU4: An Easy Step-By-Step Guide
• Super Eyes: Hand Tracking and Surveillance with the OCam

oCam : 5MP USB 3.0 Global Shutter Camera

Another digital camera with a global shutter for use with the XU4 is the oCam-1MGN-U model. Like the oCam-5CRO-U, it is offered by Hardkernel and developed & manufactured by WITHROBOT Co., LTD.

oCam with global shutter

Some of its specifications include:

Sensor: OnSemi MT9M031 CMOS image sensor
Lens: Standard M12 Lens with focal length of 3.6mm
FOV: 65 Degree
Shutter: Electric Global Shutter
Interface: USB 3.0 Super-Speed
Camera Control: Brightness, Exposure
Frame Rate: 45fps@1280x960, 60fps@1280x720, 80fps@640x480, 160fps@320x240

A common problem while using rolling shutter cameras is the shutter artifact, where straight edges appear curved in the captured image. This global shutter camera produces images where the artifact is corrected and straight edges appear as they should. For details, refer to the ODROID magazine article at `http://bit.ly/2bu0Owj`. The oCam-viewer described earlier also works with this camera.

oCam : M12 Lens Set

The oCam manufacturer also provides a set of four M12 lenses that work with either oCam models listed above. The focal lengths of the four lenses include: 8mm, 6mm, 3mm and 2.65mm. Some of its specifications include:

- Standard M12 mount
- High speed up to 160 fps at the 320 x 240 resolution
- UVC compliance
- Changeable standard M12 lens
- Optics: Glass with IR cut filter (650nm) with fixed iris
- Mount(thread) : M12 x P0.5

oCam M12 Lens Set

WiFi Module 5

The WiFi Module 5 is one of the most feature-rich WiFi modules offered by Hardkernel. It is an EEE 802.11ac/a/b/g/n WLAN module with dual-band (2.4Ghz and 5Ghz) support. The specifications include:

- Realtek RTL8812AU chipset (ID = 0bda:8812)
- Dual-Band MIMO 2x2 Solution (11ac 2x2 MAC/BB/RF+PA)
- USB 3.0 Super-Speed interface (Compatible with USB2.0 too)
- LED for WiFi Link Activity and button of WPS

Support for this module is built into the latest version of Ubuntu Mate for the XU4. Attach the WiFi module and proceed to configure the device. Shown below is the screen used to set up the WiFi Module.

Conclusion

The ODROID-XU4 is compatible with many types of hardware gadgets, and many USB sensors may be used as long as they have Linux drivers available. The gadgets sold by Hardkernel at `http://bit.ly/1fbE9ld` have the advantage of having pre-configured drivers included with the official Hardkernel disk images. We hope you enjoy tinkering and building your own projects using some of the techniques described here.

Additional Resources

ODROID forums: `http://forum.odroid.com`
ODROID Magazine: `http://magazine.odroid.com`
ODROID-XU4 wiki: `http://bit.ly/1IF3Kyh`
Android images: `http://bit.ly/1XwOatZ`
Linux images: `http://bit.ly/1kMUC27`
Improved Win32 DiskImager: `http://bit.ly/1lYQ7MF`
Hardkernel store: `http://bit.ly/1fbE9ld`