A Low-Cost Agile Robot for Outdoor Collectives

Kayla Engelstad1, Erika Gutierrez2, Jesus Hernandez3, Ranye McLendon4,   
Arielle-Cherie Paterson4, Jeff Dusek5, Nicole Carey5, Radhika Nagpal5

1University of Minnesota - Twin Cities, 2California State University, San Bernardino,   
3Swarthmore College, 4University of Maryland, Baltimore County, 5Harvard University

*The objective of this tutorial series is to go over the hardware and software principles we used to construct our swarm robot.*

Table of Contents:

[Step 1: Picking a Low-Cost Platform](#_x1lqvox0te1n) 1

[Step 2: Attaching Computer & MotorHat Hardware](#_bgo18nfplsnx) 2

[Step 3: Other Hardware](#_6lqc1pcluyw2) 4

[Step 4: Installations](#_z9n3okehj0hb) 5

[Step 5: Motor Initialization Code](#_kuti3str6frd) 8

[Step 6: Vision Code](#_abqbx5iepren) 8

[Step 7: Sonar Code](#_o86xl3qq0r66) 9

[Step 8: Random Walk](#_j3zt3smmn41k) 9

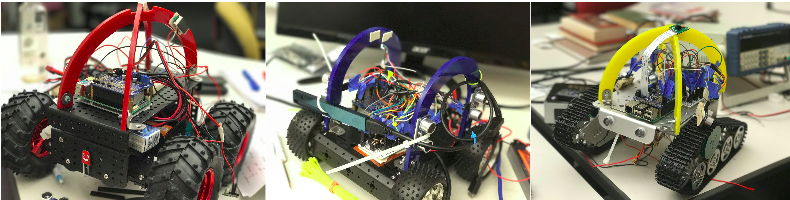
[Step 9: Integration and Hierarchy](#_u3hpm9hd8n93) 10

[Step 10: Architecture](#_mim6avj34xs) 10

[Step 11: Swarm?](#_gb30ty6dvlel) 10

# Step 1: Picking a Low-Cost Platform

The platforms chosen for this project were some of the most cost-efficient options on the market. Choosing these three platforms follows the standard for swarm robotics: each individual robot should be low-cost and easy to replicate.

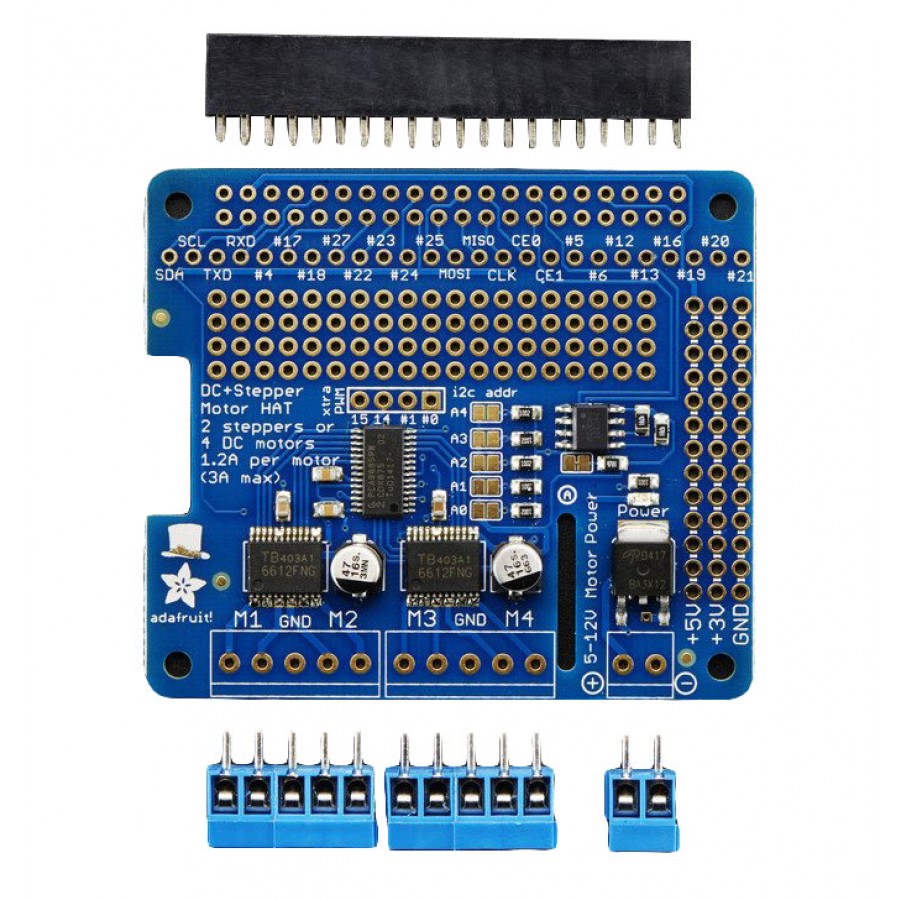
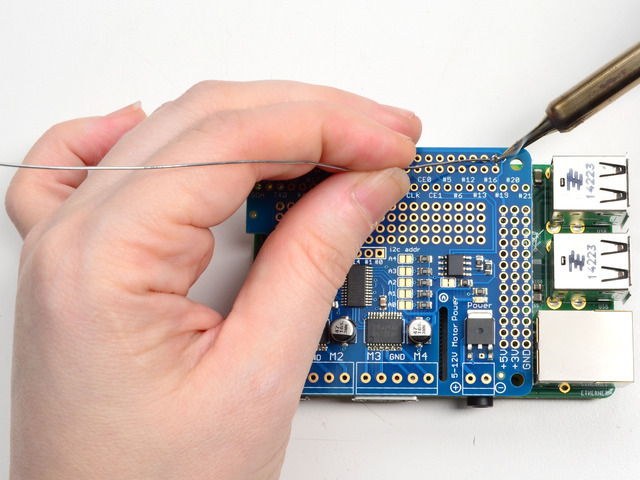
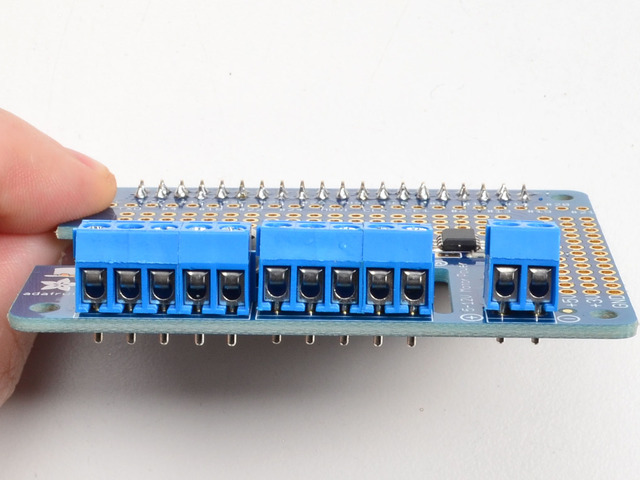
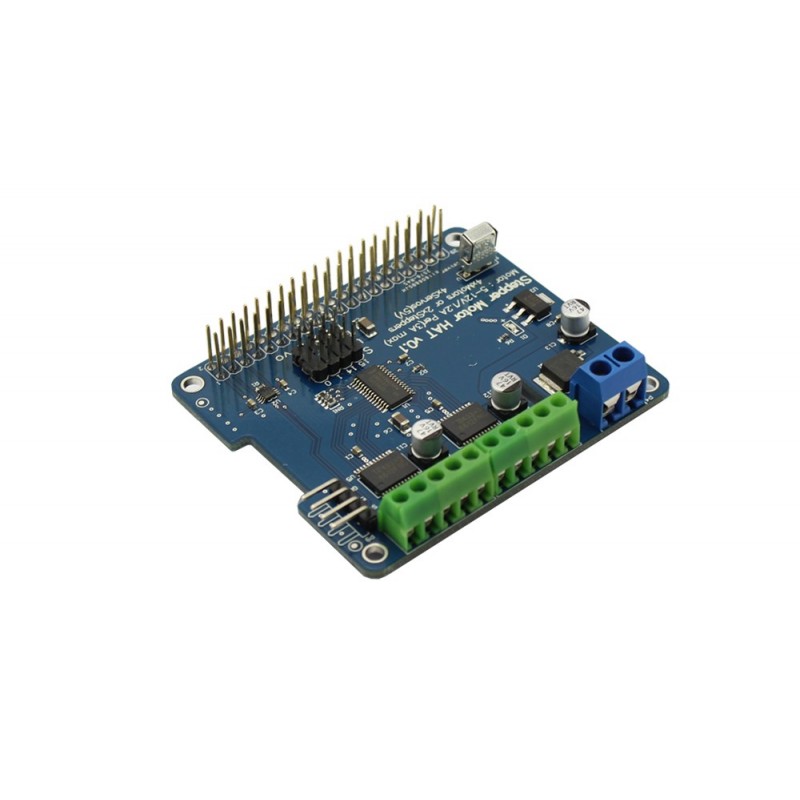
Here are the platforms:

Large Robot Chassis:<https://goo.gl/BxEK8k> Small Robot Chassis: [goo.gl/kWeHBM](https://www.amazon.com/UniHobby-Chassis-Maximum-Aluminum-Projects/dp/B01IRKKGKI/ref=sr_1_6?ie=UTF8&qid=1491499571&sr=8-6&keywords=Smart+Car+Chassis+4WD) Tank Robot: [goo.gl/NgLqBB](http://goo.gl/NgLqBB)

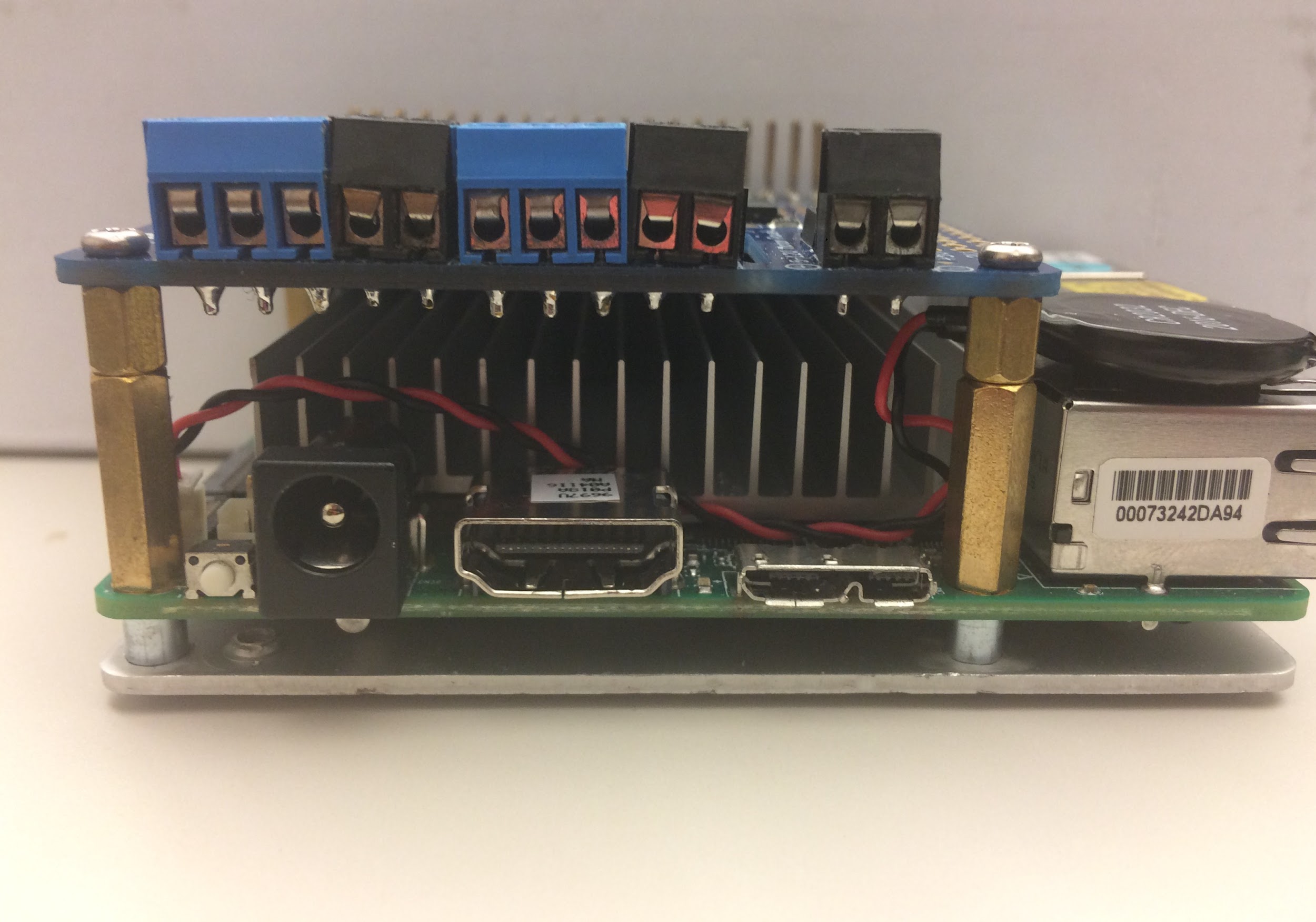
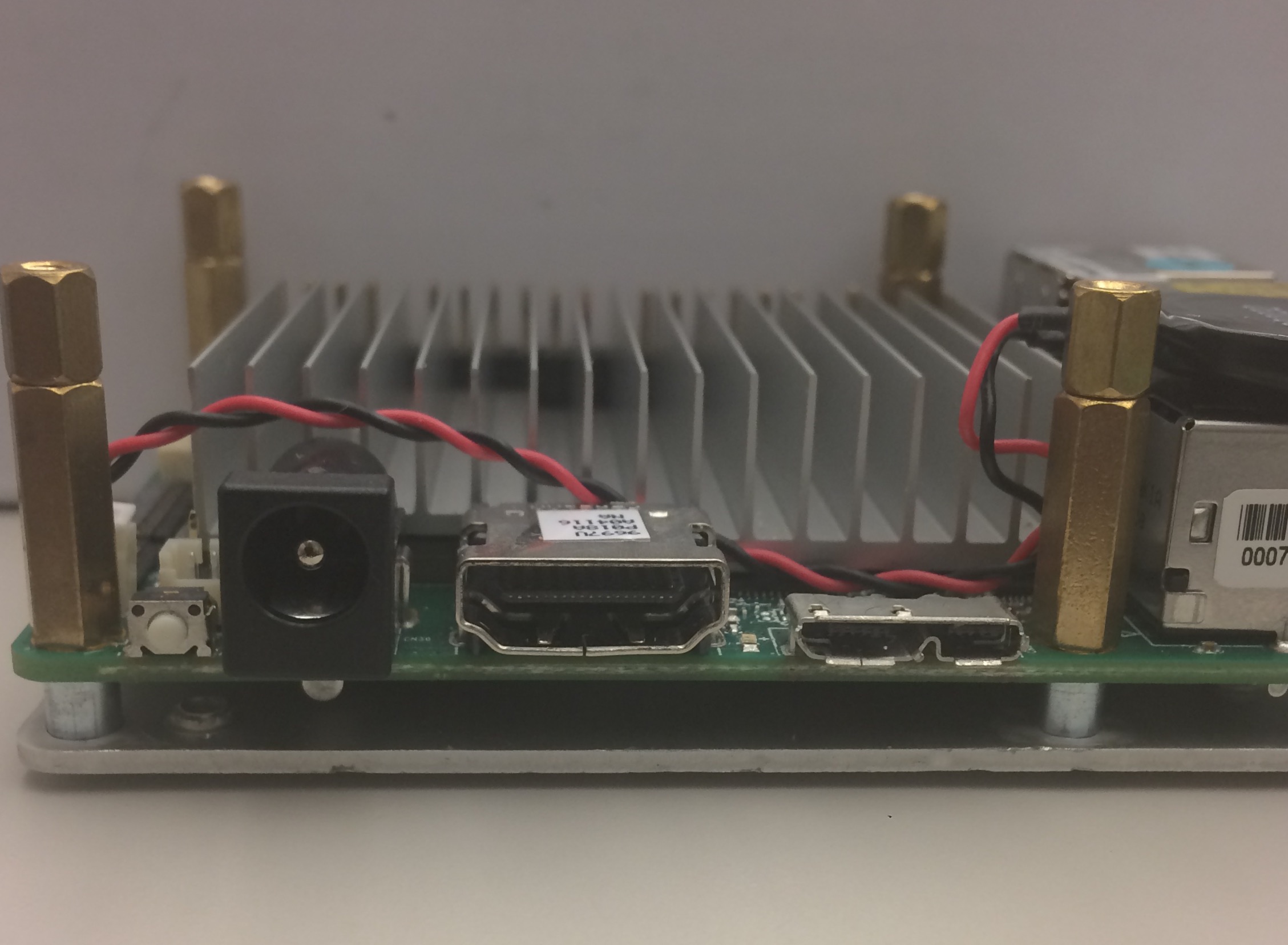
# Step 2: Attaching Computer & MotorHat Hardware

After the robot platforms were assembled, the next step was to attach additional hardware needed to control the robot. The two types of computers we used were the UP Board and the Raspberry Pi 3, Model B. These computers held the algorithms needed to have the robots act autonomously based on the continuous stream of data they got from their sensors. The algorithms on each computer interacted with the MotorHat hardware, which controlled the speed and direction of each robot’s motors.

To install the MotorHat hardware::

The MotorHat pins are soldered to the designated holes on the MotorHat board. Afterwards, the MotorHat is screwed onto the UP board and Raspberry Pi board.

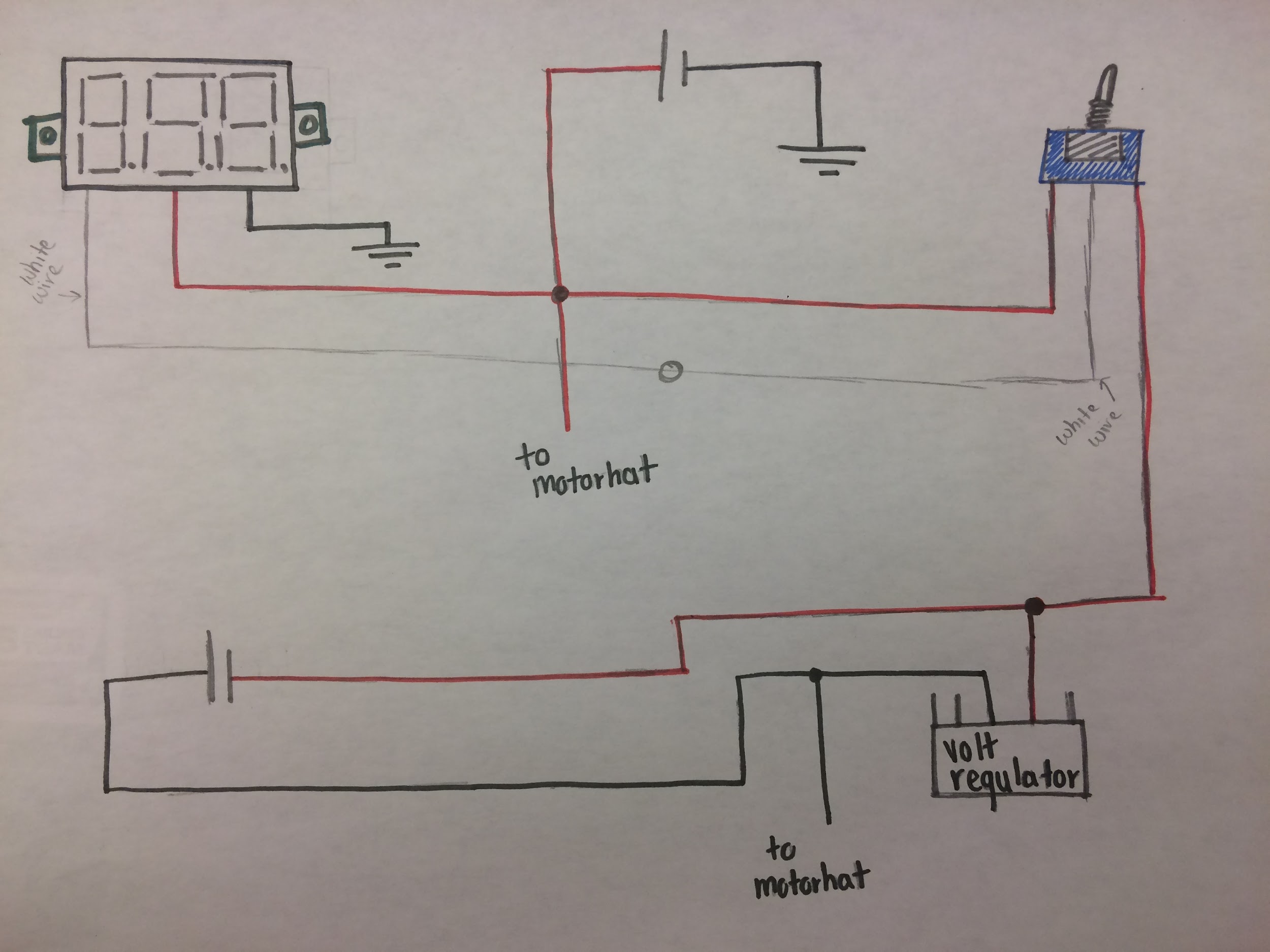


UP board 4GB RAM+ 32 GB eMMC:<https://goo.gl/kR341o>  
Adafruit DC & Stepper Motor HAT for Raspberry Pi: [https://goo.gl/jKn1Z4](https://www.adafruit.com/products/2348)

# Step 3: Other Hardware

The other hardware used in this project includes components for powering the computers and motor hats (batteries and step-down voltage regulators), taking in vision information about the environment (Intel RealSense R200 Camera and Raspberry Pi Camera), and taking in distance information about the environment (HC -SR04 Ultrasonic Sonars).

To wire the switch with the Mini LED Voltmeter and batteries:



The switch has a red wire on each end and a middle wire.. The mini LED voltmeter has a red, black and white wire.

1. The red wire from the voltmeter, the red wire from the switch and the wire of the 11.1 battery must be soldered into one wire.
2. We soldered an extra wire to those three wires to plug into the power of the motor hat.
3. The black wire from the 11.1 battery and voltmeter must be connected to a ground.
4. The white wire from the voltmeter must connect to the middle wire of the switch.
5. The other red wire of the switch must connect to the red wire of the 7.4 battery and connect to the voltage regulator.
6. The black wire from the 7.4 battery must connect to the voltage regulator.
7. A wire must be soldered onto the black wire to connect to the motor hat.

Here are those components:

Mini LED Voltmeter display: <https://www.amazon.com/Digital-Display-2-5-30V-Voltmeter-Eachbid/dp/B00P7QC8PW/ref=pd_lpo_vtph_328_lp_img_4/145-8835236-9028834?_encoding=UTF8&psc=1&refRID=FD12EB4JGKD1HZHA68B6>

Floureon 7.4V 5200mAh High Power 2S 30C Lipo Battery (motors): [https://goo.gl/NVwCKq](https://www.amazon.com/gp/product/B00HWQ1JAU/ref=oh_aui_detailpage_o03_s00?ie=UTF8&psc=1)

Gens ace LiPo Battery Pack 2200mAh 25C 3S 11.1V (computer): [https://goo.gl/Rsx9dL](https://www.amazon.com/gp/product/B00WJN4LG0/ref=oh_aui_detailpage_o04_s00?ie=UTF8&psc=1)

The actual battery that runs everything now

Pololu 5v/5a Step-down Voltage Regulator:<https://www.pololu.com/product/2851>

Intel RealSense R200:<https://software.intel.com/en-us/realsense/r200camera>

Raspberry Pi Camera: <https://goo.gl/krVNzx>

HC-SR04 Ultrasonic Sensors: <https://www.sparkfun.com/products/13959>

The following hardware components helped get the above components up and running. The offsets were used to attach the motorhat on the computer and the USB 3.0 adapter and cable were used to hook up the Intel RealSense R200 Camera to the computer.

Deans Style Connectors (batteries): <https://goo.gl/sgsNPF>

Board Offsets (M2.5): [https://goo.gl/VPmBsX](https://www.amazon.com/gp/product/B06XXV8RTR/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1)

USB 3.0 Adapter: [https://goo.gl/c1VSga](https://www.amazon.com/gp/product/B01BVR7I9G/ref=oh_aui_detailpage_o01_s01?ie=UTF8&psc=1)

USB 3.0 Adapter Cable: [https://goo.gl/4ebr1F](https://www.amazon.com/gp/product/B00II0H7T6/ref=oh_aui_detailpage_o01_s00?ie=UTF8&psc=1)

# Step 4: Installations

The UP Boards require quite a bit of installations. The first installation is the Ubuntu operating system. After that is the library for the Intel RealSense camera, librealsense. After that is installed, a Python wrapper for librealsense is necessary to be able to code in Python and use the RealSense camera. The Python wrapper is called pyrealsense.

Here are the GitHub repositories for librealsense and pyrealsense.

To setup the UP board, follow these steps in order.

1. Install Ubuntu 16.04 using a bootable usb drive

2. Install Librealsense with necessary patches. Librealsense is a cross platform API for the Real Sense depth camera.

* The GitHub repository for Librealsense can be found here:<https://github.com/IntelRealSense/librealsense>
* To install librealsense, clone or download the repository. Then, follow the instructions in the installation guide here:<https://github.com/IntelRealSense/librealsense/blob/master/doc/installation.md>

3. Install pyrealsense, which is a python wrapper for the Librealsense API.

* The GitHub repository for pyrealsense is here:<https://github.com/toinsson/pyrealsense>
* Best way to install pyrealsense is to use pip as root
  + *sudo pip install pyrealsense*
  + pip should already be installed with python, but if not you may have to install setuptools and wheel

4. Install OpenCV computer vision software

* Easiest way to install OpenCV is to use an install script
* create a file "install-opencv.sh"
  + *emacs install-opencv.sh*
  + Copy the script here into the new file and save:<https://github.com/milq/milq/blob/master/scripts/bash/install-opencv.sh>
* Run the install script- this will take some time
  + *bash install-opencv.sh*

5. Install the UP board linux kernel for Ubuntu 16.04

* Installation instructions can be found on the UP wiki here:<https://up-community.org/wiki/Ubuntu>

6. Remove generic linux kernels

* This ensures the board boots using the UP board kernel
* Instructions for removing the generic kernel are here:<https://up-community.org/wiki/Ubuntu>

Using the AdaFruit DC & Stepper Motor Hat:

1. Download and install the AdaFruit python code that has been modified by Emutex for the Up Board

* Install the "Adafruit\_GPIO" library following the instructions at the top of the page here under "overview":<https://up-community.org/wiki/Adafruit>
* Install the "Adafruit Motor HAT Python library" following instructions here:<https://up-community.org/wiki/Adafruit#Using_Adafruit_DC_.26_Stepper_Motor_HAT_on_UP>

2. Solder headers and terminal blocks as shown here:<https://learn.adafruit.com/adafruit-dc-and-stepper-motor-hat-for-raspberry-pi>

3. Connect to power supply or battery:<https://learn.adafruit.com/adafruit-dc-and-stepper-motor-hat-for-raspberry-pi/powering-motors>

4. Connect appropriate DC motor (NOTE: example code DCTest.py works with block M3, not M1):<https://learn.adafruit.com/adafruit-dc-and-stepper-motor-hat-for-raspberry-pi/using-dc-motors>

5. To run an example, follow the directions here:<https://up-community.org/wiki/Adafruit#Using_Adafruit_DC_.26_Stepper_Motor_HAT_on_UP>

* NOTE: examples must be run as root: *sudo python DCTest.py*

Using the AdaFruit 16 Channel PWM/Servo Hat:

1. Download and install the AdaFruit python code that has been modified by Emutex for the Up Board

* Install the "Adafruit\_GPIO" library following the instructions at the top of the page here under "overview":<https://up-community.org/wiki/Adafruit>
* Install the "Adafruit\_PCA9685" Python library following instructions here:<https://up-community.org/wiki/Adafruit#Using_Adafruit_16-Channel_PWM_.2F_Servo_HAT_on_UP>

2. Install MRAA which is needed to manage some I2C communication tasks

* MRAA GiHub repository is here:<https://github.com/intel-iot-devkit/mraa>
* Follow the instructions under "Installing on Ubuntu"

3. Solder headers and terminal blocks as shown here:<https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi/>

4. Connect to power supply or battery:<https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi/powering-servos>

5. Connect appropriate servo:<https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi/connecting-servoss>

6. To run an example, follow the directions here:<https://up-community.org/wiki/Adafruit#Using_Adafruit_16-Channel_PWM_.2F_Servo_HAT_on_UP>

* NOTE: examples must be run as root: *sudo python simpletest.py*

Raspberry Pi Camera Setup: <https://www.raspberrypi.org/learning/getting-started-with-picamera/>

(& sonar installations: Kayla)

# Step 5: Motor Initialization Code

The motor init code is currently used in both the vision and sonar code. The code has the following functions:

* *getMotorValue*
* *isCorrectionNeeded*
* *getError*
* *getCorrection*
* *SetAndDriveRight*
* *SetAndDriveLeft* and
* *turnOffMotors*

The get motor values function takes in a percentage and converts that into a motor speed. The correction needed function calculates the error, and depending on the magnitude of the error, returns a boolean signifying whether or not a correction is needed. The get error function takes in the x-coordinate of the center of mass and calculates the error to the centerline and returns that error. The get correction function takes in the error, the previous error, and the change in time and implements a PD controller; it returns that correction. The set and drive left/right functions set the speed and direction of the left and right motor pairs. The turn off function turns off all motors.

# Step 6: Vision Code

The Intel RealSense camera was used to give the robot the ability to “see” its surrounding environment. Currently, the vision code runs separately from the sonar and random walk code. Three files are necessary to run the cone detection code:

* *nav\_to\_cone.py*
* *coneDetWithShape.py* and
* *motor\_init.py*

The nav code is the top level file. It initializes the camera and runs the detection code in a loop. The detection code returns the contours of the cone, a boolean of whether or not there is a cone present, and the center of mass of the cone. The nav code uses the boolean to decide whether it should navigate towards a cone or spin to locate one. Once it spots a cone, it navigates towards the center of mass using motor commands and PD control in the motor file.

The cone detection file still has one major bug in it. We’ve implemented shape recognition so that it only navigates towards orange conical shapes, but if it sees anything that is orange but is *not* a conical shape, the code crashes. I think this can be fixed pretty simply with an “and”.

Possible future work on this code includes using the depth capabilities of the camera to stop a certain distance from the cone. Currently, the robot will simply run over the cone.

The functions in the cone detection code include:

* *convexHullPassesAspectRatioTest*
* *convexHullIsPointingUp*
* *find\_contours* and
* *find\_cone*

The find cone function is called from the nav code. It takes in the RGB stream and returns an image (with the contours and center of mass drawn on a black background), a boolean, and the center of mass. Within the find cone function, we call the find contours function, which does just that, and the aspect ratio test function and the pointing up function. The aspect ratio test checks to make sure the top half of the cone is narrower than the bottom half of the cone, aka a cone shape. It takes in a hull and returns a boolean value. The pointing up function takes in a hull and determines whether or not the cone is pointing upwards. It returns a boolean value. We only navigate towards cones that are upright.

## 

# Step 7: Sonar Code

Each robot has four HC-SR04 Ultrasonic Sensors, two on each side of the robot, that give the robot the ability to “feel” its surrounding environment.

The sensors calculated distance using sound waves; the sensor releases a sound wave and once it hits a nearby object, the sound wave bounces back. The sonar code uses that distance to help the robot avoid collisions. The sonar code requires these files to run: Kayla

# Step 8: Random Walk

The robot does encounter situations in which it cannot find a cone or a nearby wall. In these situations the robot performs a random walk. A random walk is when the robot roams throughout the pavement until it finds a wall or cone so it can perform one of the other files of code. It essentially pivots in place for a random amount of time in order to choose a random direction, and then goes forward in that direction for a random amount of time. It does this for a set number of loops. To access the random walk, the files that need to run are: Kayla

# Step 9: Integration and Hierarchy

Currently, the vision, sonar, and random walk code all run separately. Our goal is to integrate these functionalities and implement them with a hierarchy. The highest priority on this hierarchy would be a cone. If no cone is detected, it will try to follow a wall until it reaches a cone. But if there is no wall and no cone, then it will do the random walk until it locates one of the two.

This implementation is in progress.

# Step 10: Architecture

# 

# Step 11: Swarm?

# 

## 