# PiBot Handbook

- ME292B -

ME292B- Robotic Controls

| Date | Revision | Release Notes |
|---|---|---|
| 5/8/18 | 00 | Initial Release |
| | | |

# Contents

# 1 Introduction

The goal of this class is to introduce robotics control concepts using an AlphaBot2 with a Raspberry Pi Zero. The Raspberry Pi was set up and this manual was written to help step a user through the process. Then, a line follower, obstacle avoidance, Bluetooth connection, and camera code was written and explored to test many components of the robot.

This **link** is a guide to the components of the AlphaBot2 PiZero. There are commands provided to use many of the features, such as using the ultrasonic sensor and the bottom infrared sensors.

# 2 Building AlphaBot2

The following link provides an assembly guide for the robot. The guide is easy to follow and walks through each step for proper assembly. The link is found here: **link**

**Note:** The pins on the Pi must be soldered for adequate connections; otherwise, the Pi will be unable to communicate with the robot.

# 3 Raspberry Pi Setup

## 3.1 Installing Raspbian

Follow the installation guide (**link**) on the official Raspberry Pi website

1. Download "Raspbian Sketch with Desktop" .zip file: **link**
2. The Raspbian Sketch is downloaded as a .zip file. Extract the contents of this .zip file to access the .img file.
3. Download and install Etcher to flash Raspbian onto Pi: **link**
4. Connect the SD card to computer with the SD card USB drive.
5. Using Etcher, place the extracted .img file for Raspbian into Etcher with the SD card selected as the device.

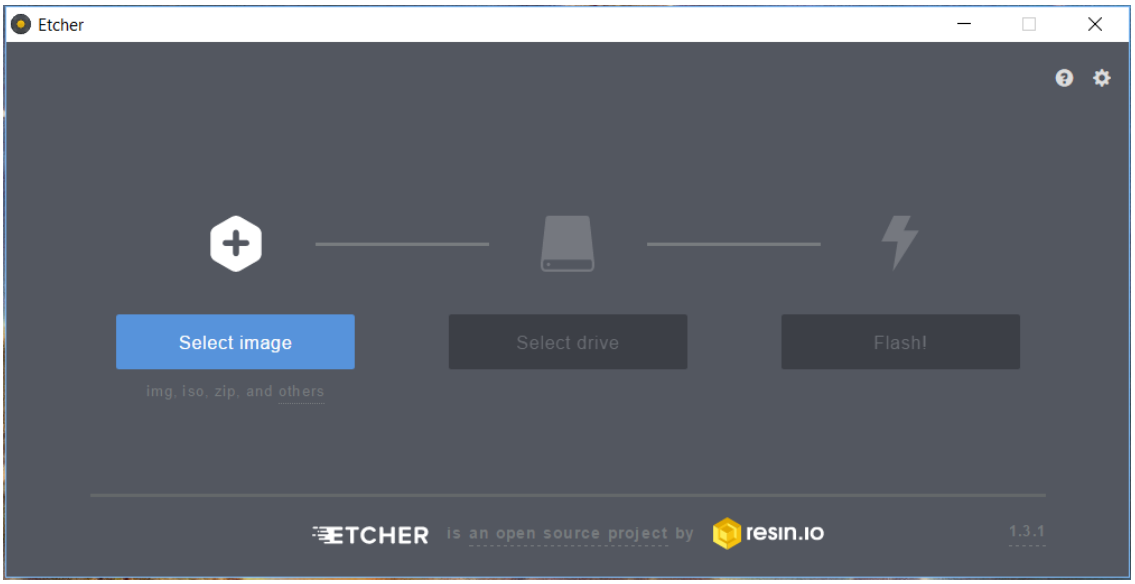


**Figure 1:** Etcher on desktop. Select the image file first, then the SD Card drive and press "Flash!".

6. Flash Raspbian onto the SD card (this may take 10-15 minutes).

**Note:** The Raspbian .img file cannot just be moved onto the SD card—it must be flashed.

### 3.2 Booting up the Pi

1. Connect the Pi to an external monitor using an HDMI to mini HDMI adapter.
2. Once connected, turn on the robot to send power to the Pi.
3. The Pi should begin to boot up. The Raspbian logo will appear in the upper left corner. **Note:** Some of our desktop boot-up processes differed slightly. Sometimes code appears on screen, for example.
4. The screen will go black, a white bar will appear at the top, and finally the desktop of Raspbian will appear - if this happens, the OS has installed correctly.

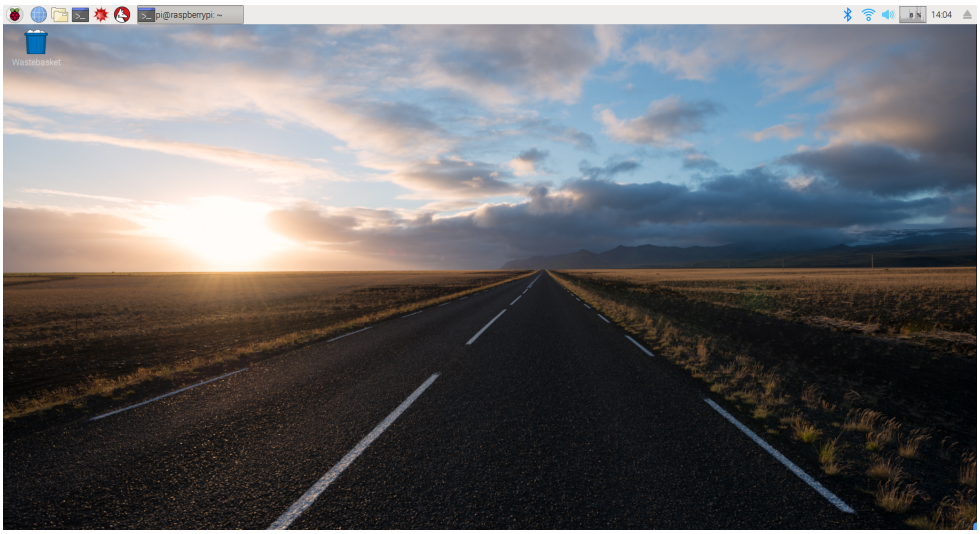

**Figure 2:** Desktop as it appears when Pi is properly booted up.

You should now be presented with a traditional desktop environment that is standard for any computer, as shown above in Figure 2. In order to interface with this environment you will need to plug a USB mouse and keyboard into one of the four USB ports on the robot.

### 3.3 Change Keyboard Layout

You may notice as you begin navigating Raspbian that certain keystrokes do not produce the correct character. This is because of the current default keyboard settings, in order to fix this we must change this to a standard US keyboard layout.

1. Using an external monitor, open the terminal on the Pi. The terminal shortcut is found in the top-left of the desktop.

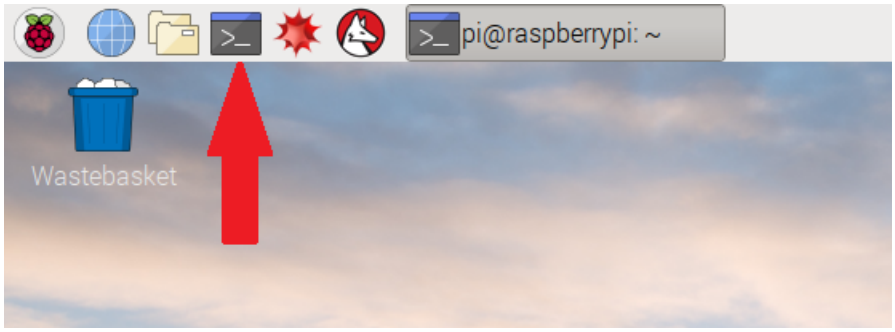


**Figure 3:** Terminal shortcut, found in top-left of desktop. Terminal window is also open currently, shown by block to the far right.

2. Type "**sudo raspi-config**"
3. Navigate using the arrows to "**4 Localization Options**"

4. Select "**I3 Change Keyboard Layout**"
5. Select "**MacBook/MacBook Pro**" or another computer model
6. Select "**English UK**"
7. Select "**The default for the keyboard layout**"
8. Select "**No compose key**"
9. Select "**No**"
10. You will be brought back to the main menu of "**sudo raspi-config**" and select "**Finish**"

### 3.4 Change Password

Later on, we will be implementing SSH communication with the Pi so that we can remotely control the robot without the need for an external monitor. In order to keep this line of communication secure, it is important to change the default password. The default username is “pi” and the default password is “raspberry”. The username will be used to identify the Pi later on, so we do not need to change it.

1. Open the terminal from the new desktop and type "**passwd**"
2. Enter the current password, "raspberry" (default)
3. Enter your new password
4. If completed, the terminal will read "passwd: password updated successfully"

### 3.5 Connect to UVM WiFi

Next we need to ensure that the robot is able to connect to UVM’s WiFi and automatically login with user credentials when powered on. This is a very important step as the SSH communication is done over UVM’s network and without access to the desktop environment, the WiFi must connect during the boot process. From UVM’s CS 121- Computer Organization:

1. Set location:
    (a) Type in "**sudo raspi-config**" to open the Pi Configuration Menu

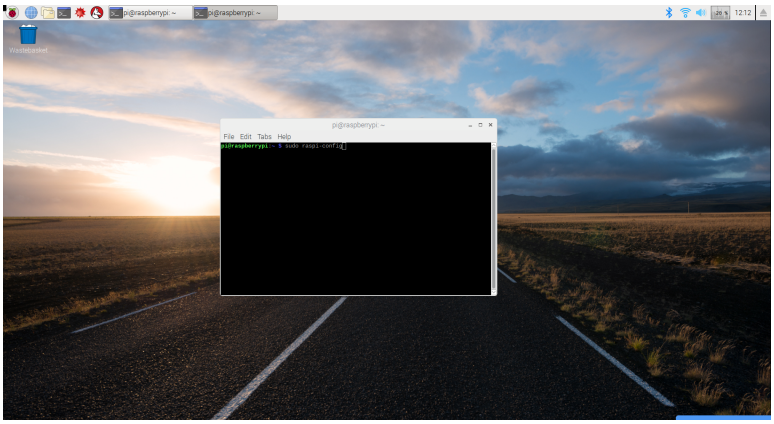

**Figure 4:** Step 2a

(b) Select "**Localization Options**"

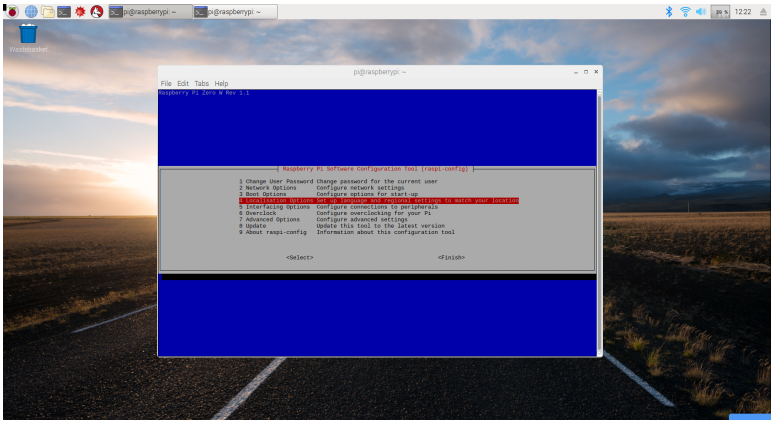

**Figure 5:** Step 2b

(c) Select "**Change the Time Zone**"

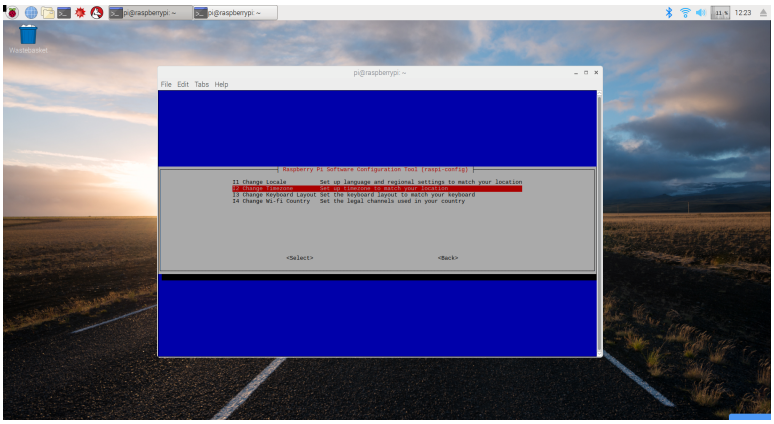

**Figure 6:** Step 2bi

i. "US"

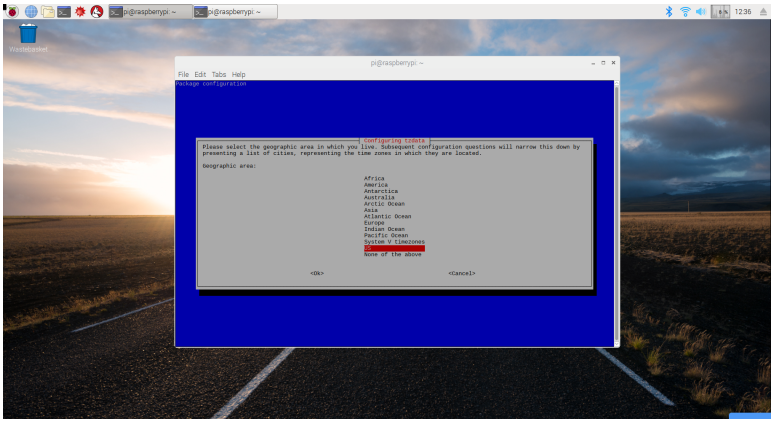

**Figure 7:** Step 2bi1

ii. "Eastern"

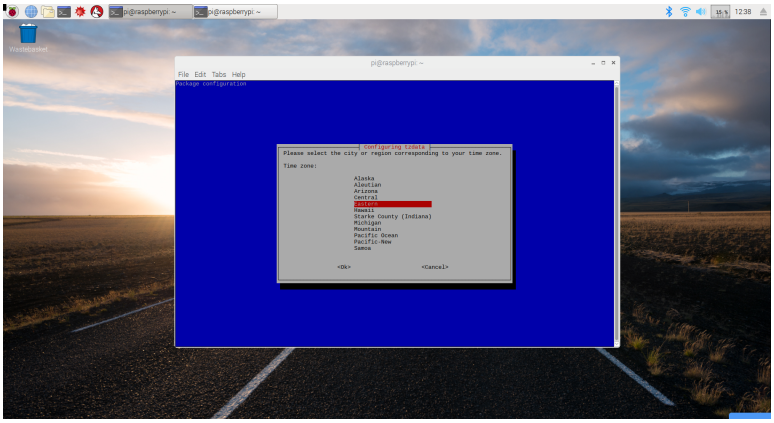

**Figure 8:** Step 2bi2

(d) Select "**Change Wi-Fi Country**"

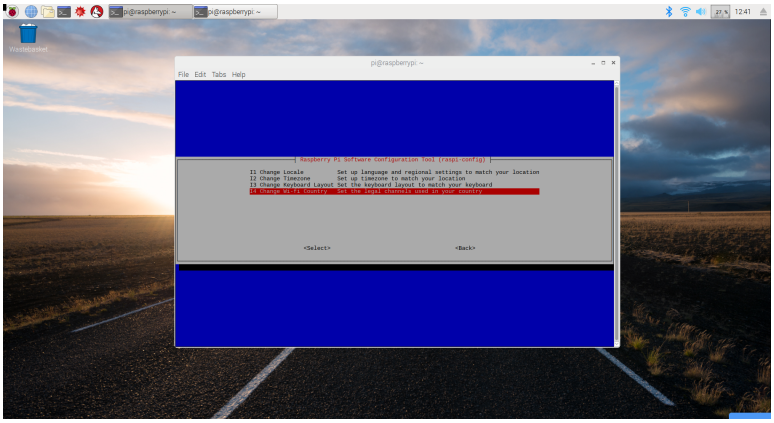

**Figure 9:** Step 2bii

i. "US United States"

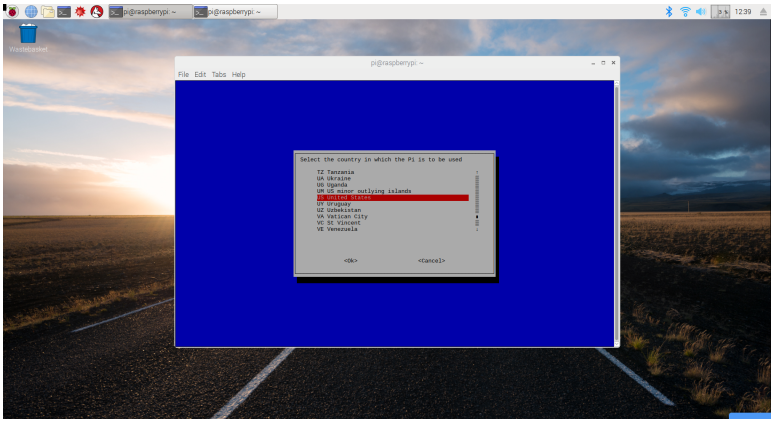

**Figure 10:** Step 2bii1

(e) Select "**Finish**" to save settings

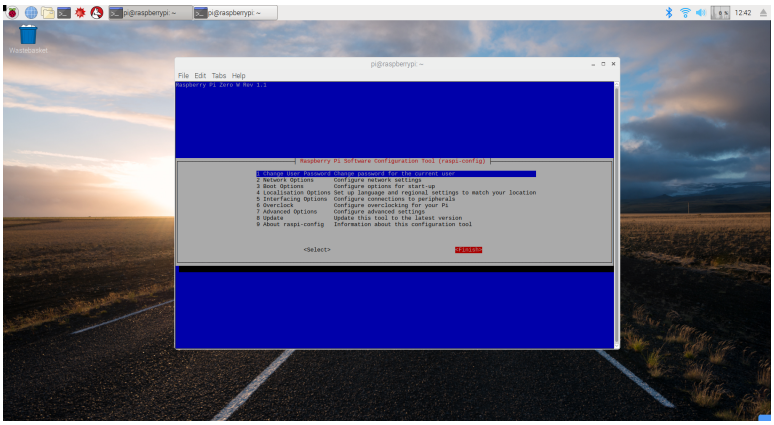

**Figure 11:** Step 2ci

(f) Reboot the Pi

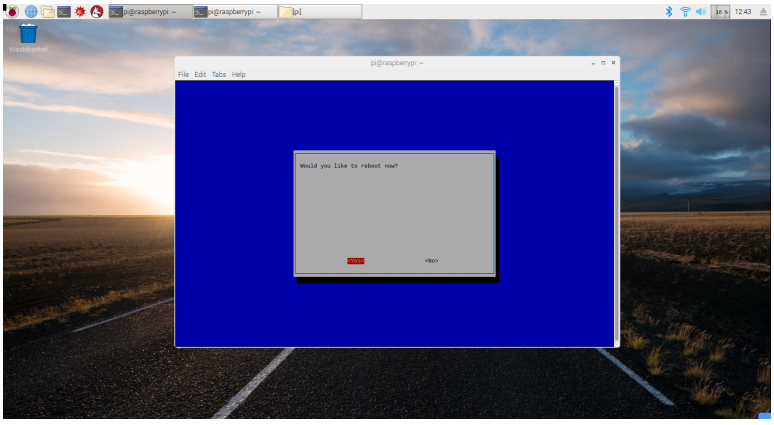

**Figure 12:** Step 2ci2

2. Change the date:

   (a) Open terminal again and type: "**sudo date -s "Thu Mar 3 15:56 2018"**"

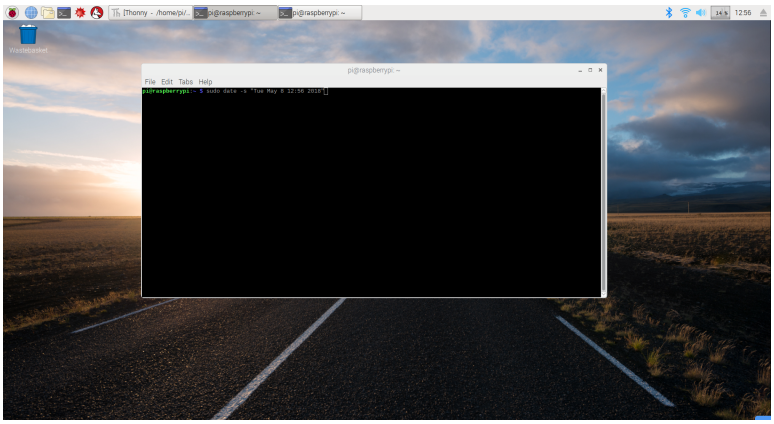

**Figure 13:** Step 3ai

   **Note:** Quotations need to be placed around the date/time where the time is in 24 hour format

   (b) The date should be changed and show in the terminal

3. Connect to WiFi:

   (a) In terminal type: "**sudo nano /etc/wpa_supplicant/wpa_supplicant.conf**"

   (b) This will open up a file, change the contents of the file to the following:

```
country=US
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
        network={
        ssid="UVM"
        proto=RSN
        key_mgmt=WPA-EAP
        pairwise=CCMP
        auth_alg=OPEN
        eap=PEAP
        identity="netid"
        password="netid_password"
        }
```

   **Note:** Be sure to change "netid" and "netid_password" to your respective UVM credentials.

   (c) Press Control-X to save the file

   (d) Press Enter to exit back to the terminal

4. Reboot the Pi by typing: **reboot**

5. After the reboot, the UVM WiFi should be connected in the upper right corner

Troubleshooting:

1. If you cannot connect to the UVM network, manually run the wpa_supplicant command:
   (a) Type: **sudo wpa_supplicant -iwlan0 - c/etc.wpa_supplicant/wpa_supplicant.conf**
   (b) This should display output—if things are connecting, that is good and something else is causing an issue. If not, read the output and search for what may be causing the issue.
   (c) **Ctrl+C** to kill the command
2. Assuming there is a connection when manually running, you can use the -B option to run it in the background
   (a) Type: **sudo wpa_supplicant -B -iwlan0 - c/etc.wpa_supplicant/wpa_supplicant.conf**
3. This allows you to check your IP address to make sure everything is valid
   (a) Type: **ifconfig wlan0**
4. If the Pi does not connect automatically at boot, you will need to look at the log files /var/log/syslog. Use grep to filter out network related events:
   (a) Type: **sudo cat /var/log/syslog | grep network**
   (b) Check logs for errors and research a solution

# 4 Remote Communication

## 4.1 Determine IP Address

Assuming that the WiFi has been setup correctly, finding the Pi's IP address is a straightforward process. Type "**hostname -I**" (last character is a capital "i") into terminal. The IP address is shown as the first string of numbers.

Alternatively, out of the terminal, you can also hover over the WiFi symbol in the upper right corner and your Pi's IP should appear on screen as shown below.

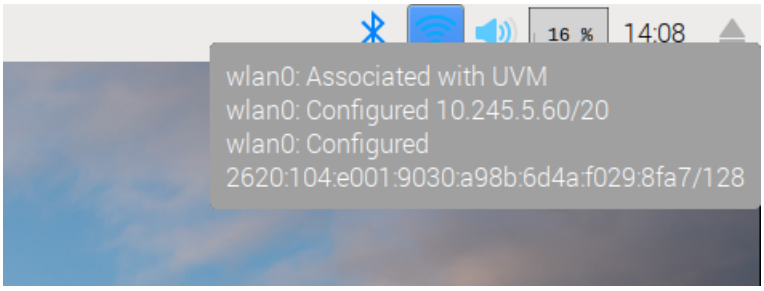


**Figure 14:** Finding the IP address by hovering over the WiFi symbol. The IP is shown after the "wlan0: Configured" in the second line. Discard the "/XX" ("/20" in this case) at the end of the IP.

## 4.2 Connect Pi to Computer for Remote Access

### 4.2.1 SSH Communication

To allow for SSH Communication, first ensure that the Pi is setup for SSH:

1. In terminal type "**sudo raspi-config**" to bring up Pi Configuration Menu
2. Select "**Interfacing Options**"
3. Select "**P2 SSH**"
4. Select "**Yes**" when prompted to enable SSH
5. Select "**OK**"
6. Power on the Pi. Going into the Configuration Menu should show that SSH is enabled.

### 4.2.2 Remote Access (Windows)

Opening a line of communication for remote access (Windows):

1. Navigate to "**https://www.chiark.greenend.org.uk/ sgtatham/putty/latest.html**" and download PuTTY for Windows: **link**
2. Install PuTTY and run from the start menu
3. Type your Pi's IP address into the "**Host Name (or IP address)**" field and click "**Open**"
4. If you eventually see a message saying "**Network error: Connection timed out**", it is likely that you have entered the wrong IP address for the Pi
5. Once you see a dialog box that says "**PuTTY security alert**", click "**Yes**". This means that the IP address is working, and this is the only time that this message will appear

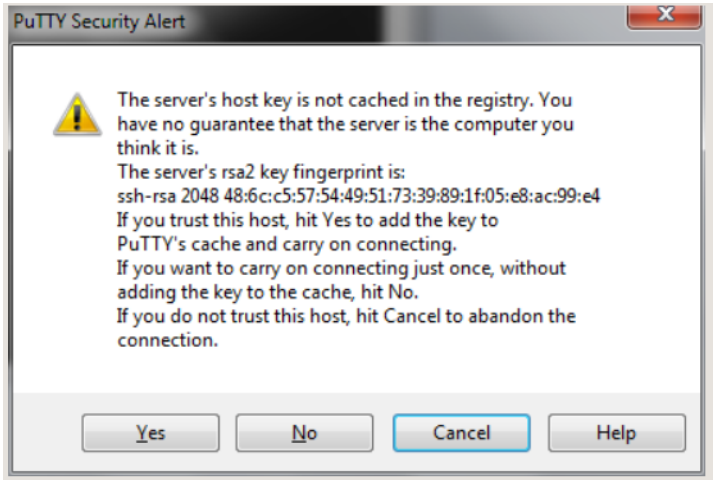


**Figure 15:** Message that appears when accessing Pi remotely for the first time through PuTTY

6. The usual login prompt then appears. The default username is "**Pi**" and the default password is "**raspberry**".

   **Note**: You might have changed your password earlier, in Section 3.4

7. You should now be in a command prompt identical to the Raspberry Pi's.

### 4.2.3 Remote Access (Mac)

Opening a line of communication for remote access (Mac):

1. Open Terminal
2. Type in "**ssh pi@**" followed by the IP address ("**ssh pi@###.##.#.##**") ("**pi**" is the username for the Pi).

   **Note:** "**connection timed out**" indicates the wrong IP address was entered

   (a) If this is the first time connecting to the pi under a new IP you will be prompted whether you would like to permit the connection, enter **yes** to continue

3. Enter the password for the Pi (default is "**raspberry**").

   **Note**: You might have changed your password earlier, in Section 3.4

4. You now should be connected remotely with a terminal that is exactly the same as the one on your Raspberry Pi.

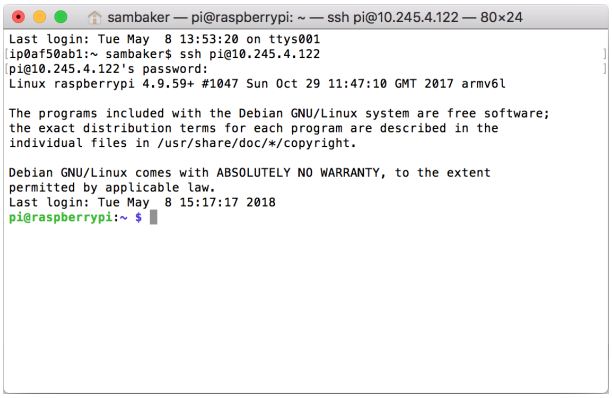


**Figure 16:** Mac terminal for ssh communication with raspberry pi

The Terminal window on your computer now mirrors the Terminal running on your pi. This allows us to run the pi Headless, meaning that we do not require the desktop environment.

### 4.3 Running Python Remotely

To create a new Python file:

1. Connect remotely to Pi through computer terminal.
2. Type in "**sudo nano filename.py**" naming the Python file what you want in place of "**filename**".
3. This creates a file on the Pi.
4. Screen enters the Pi's file interface
5. Type in code for Python script
6. When finished, press Control-X (even for mac) to save the file
7. Will be prompted to "**Save Modified Buffer**", enter "**y**" for yes to save it
8. Press Enter to return to terminal window

Running existing file:

1. In the terminal on the computer, navigate to directory containing desired file:

   "**cd ~/pathtofolder**" opens a folder within the current directory.

   **Note:** The "**~**" is "**/home/pi**" and is the default folder for the Pi

   "**cd ../**" goes up one level in current directory.

2. Enter "**sudo python filename.py**" to run the file

   **Note:** While it is possible to run code without the "**sudo**" command, some python code requires additional permissions and may fail to run if this is excluded.

### 4.4 Setting Up File Transfer

To work on Python on your computer in IDE and then transfer it to Pi:

1. Install Pure-FTPd by following instructions at: **link**
2. Only follow the "Basic Configurations" listed on the website.
3. Navigate to FileZilla to download "FTP Server - FileZilla" for Mac/Windows: **link**
4. Turn on the robot and wait about a minute for the start-up process to complete.
5. Open FileZilla. Type the IP address into the "**Host**" field, the username ("pi") into the "**Username**" field, and the password ("raspberry" or your own as chosen in section 3.4) into the "**Password**" field

6. Click "**Quickconnect**". The following screen should appear:

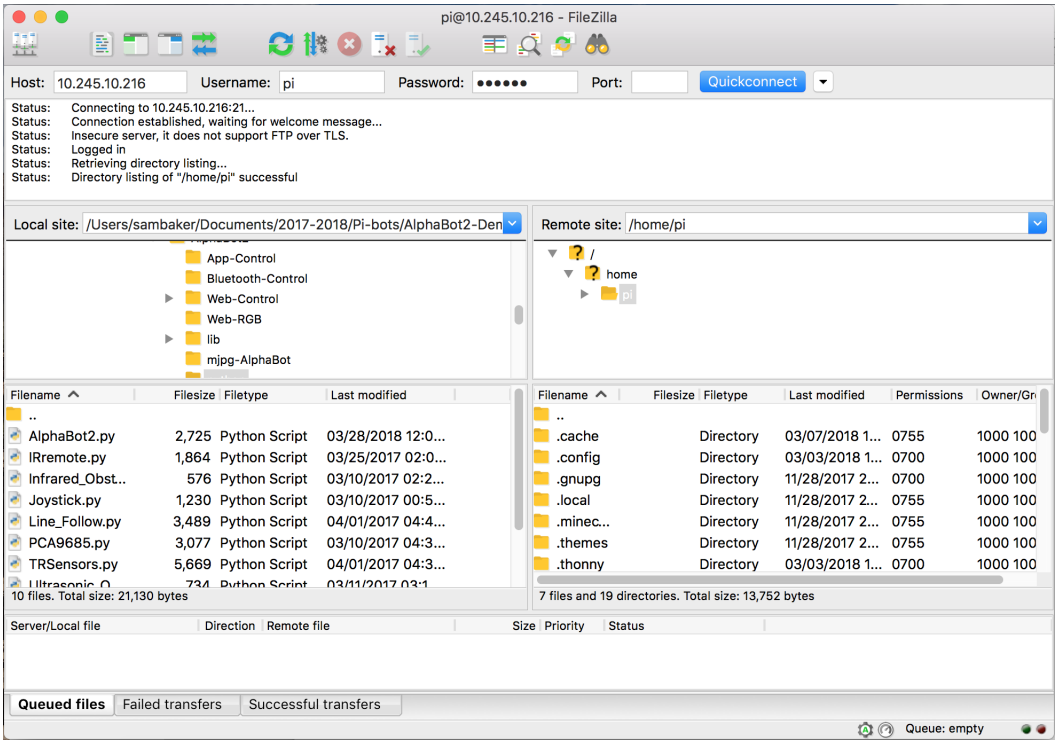


**Figure 17:** Screen as it appears when accessing Pi remotely through FileZilla. Taken on a Mac—may differ slightly on a Windows device.

7. To transfer files, simply navigate to the proper directory and drag and drop files between file windows in bottom right and left.

Now that we have access to the Pi's directories, you can simply drag and drop desired files from your personal computer to the Pi. We will test that this works by creating a new directory for the AlphaBot and storing the python demo code in this file in the next section. We will attempt to run some of the scripts provided to test the robot's sensors.

# 5 Running Demo Code

To make sure that the robot and communication protocols are working properly, please load the python demo code found on the website to a desired directory. To do this:

1. Follow this **link** to the Alphabot website.
2. Click on the top link in the table (should say "current" next to it, in the left column. This begins the Python demo script .zip download.
3. Unzip the file in the desired directory on your computer.
4. Using FileZilla, transfer the demo scripts to the Pi in a directory of your choice.

## 5.1 AlphaBot2.py:

Run this script first by navigating to the chosen demo script directory within your remote terminal (explained in Section 4.3) and typing:

```
sudo python AlphaBot2.py
```

The robot should drive forward, in a relatively straight line. If the line is not straight, do not be alarmed, as the motor values will be adjusted later. Calibration is battery and device dependent. Be careful of the device falling off the surface it is on, as it sometimes starts driving very fast. Stop the script by pressing Control-X (Windows or Mac). The script can also be run through the Pi's desktop if plugged into an external monitor, as shown below in Figure 18.

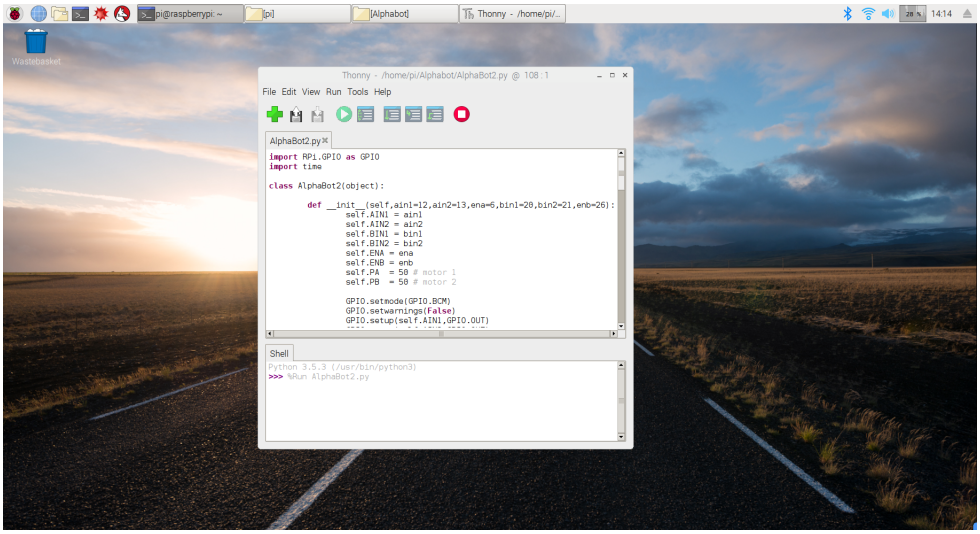


**Figure 18:** Example of running the script AlphaBot2.py through the Pi's desktop environment. The red stop button within the UI can be used to stop the script.

## 5.2 IRremote.py:

While still within the demo code directory, type the following command into the terminal:

```
sudo python IRremote.py
```

This code is a fun test to make sure that your robot's motors are functioning correctly. Using the remote included with your AlphaBot, press "2" to go forwards, "8" to go backwards, "4" to turn left, "6" to turn right, and "5" to stop. "+" and "-" adjust the speed of the robot, and "EQ" returns the speed to the default setting.

## 5.3 Ultrasonic_Rangeing.py:

While still within the demo code directory, type the following command into the terminal:

```
sudo python Ultrasonic_Rangeing.py
```

The terminal should now display the distance that the ultrasonic sensor is reading. This will be used as part of the obstacle avoidance code later.

### 5.4 Line_Follow.py:

In order to run the demo code Line_Follow.py, it is necessary to address the uninstalled python packages. In this case, neoPixels is undefined in the python code. Please follow instructions in Section 5.6 to install this package.

Once installed, type the following command into the terminal:

```
sudo python Line_Follow.py
```

This code references the demo script "TRSensors.py", which contains functions for the five bottom IR sensors. These functions are referenced by other demo codes that need to read values from those sensors. When running the demo code, the robot will spin in a circle as it calibrates itself. IR sensor values will be streamed to the terminal for live monitoring. To initiate the line follow after calibration, the button on top of the robots chassis needs to be pressed. Make sure that the line is dark and wide enough to be read by the sensors.

### 5.5 PCA9685.py

This script requires SMBus to be installed and I2C to be enabled. Please follow the instructions in Section 5.6 before proceeding.

Once installed, type the following command into the terminal:

```
sudo python PCA9685.py
```

This script should test the bottom servo motor that connects the camera to the AlphaBot. The servo will pan back and forth through its entire range of motion.

### 5.6 Required Packages

**neoPixels:**

1. In your browser, navigate to: **link**.
2. Follow the steps under the section Compile & Install rpi_ws281x Library. **Note:** Updating may take a couple of minutes. There are links above each section of code to download.

**SMBus**

1. In your browser, navigate to: **link**.
2. Follow the instructions listed to install SMBus and enable I2C.

# 6 PiCamera

## 6.1 Taking a Picture

To take a picture, you must first enable the camera on the Raspberry Pi. To do so, open up the terminal and run

```
sudo raspi-config
```

Navigate to and select **Interfacing Options**. Select **P1 Camera** to enable the camera on the Pi. A system reboot will be required after this.

To install the camera package on the Pi, type

```
sudo apt-get update
sudo apt-get install python-picamera
```

In a new python file, create the following three lines of code:

```
import picamera
camera = picamera.PiCamera()
camera.capture('image.jpg')
```

To capture an image, run the file on the Pi. The image will be saved as *image.jpg* in the same directory that the python code is located. Many times, the first image taken by your robot could be blurry. To fix this, tighten or un-tighten the screw on top of the lens attachment and try screwing/unscrewing the lens itself while retaking pictures.

**Note:** If the file does not show up in FileZilla, change folders and then come back to the same folder to allow for it to refresh. It may take a little while for the photo to show up in the directory. If the image does not show after a minute or two, run the code again. In some cases, this took almost 10 minutes.

## 6.2 Taking a Video

To record a 5 second video, create and run a new python file with the following code:

```
import picamera

camera = picamera.PiCamera()
camera.resolution = (640, 480)

camera.start_recording('my_video.h264')
camera.wait_recording(5)
camera.stop_recording()
```

**Note:** The *time* package may also be used to delay between calling functions in place of **camera.wait_recording(5)**.

This will create a video file named **video.h264** in the same directory as the code. This is not the desired format for viewing the video.

To convert the file into a **.mp4**, run the following code on your Pi. You will only need to run this the first time you are trying to take a video.

```
sudo apt-get update
sudo apt-get install -y gpac
```

Once installed, run this code to convert the video file into a **.mp4**.

```
MP4Box -fps 30 -add my_video.h264 video.mp4
```

The video should now show up in the same directory as **video.mp4**. It will now be able to be viewed using QuickTime or another video playing program.

**Note:** If the file does not show up in FileZilla, change folders and then come back to the same folder to allow for it to refresh. The video may take a few minutes to show up in the directory. Another option is to use FileZilla to transfer the .h264 file over to your local computer and use an online converter to convert it to a .mp4.

**Additional Camera Settings:**

```
camera.sharpness = 0
camera.contrast = 0
camera.brightness = 50
camera.saturation = 0
camera.ISO = 0
camera.video_stabilization = False
camera.exposure_compensation = 0
camera.exposure_mode = 'auto'
camera.meter_mode = 'average'
camera.awb_mode = 'auto'
camera.image_effect = 'none'
camera.color_effects = None
camera.rotation = 0
camera.hflip = False
camera.vflip = False
camera.crop = (0.0, 0.0, 1.0, 1.0)
```

Complete camera documentation can be found: **here**

## 6.3 Streaming Camera To Computer

There are many diffrent methods that you can start and share a live stream from your Pi's camera. While streaming, your bot can preform other tasks; however, the process of streaming requires a large amount of the Pi's RAM so do not expect peak preformance while streaming.

### 6.3.1 VLC Streaming

The First method of streaming is using VLC media player, this is an open source cross-platform multimedia player that can deal with various video file types and streaming protocols. Start by downloading VLC by typing the following in to your SSH program:

```
sudo apt-get install VLC
```

You should be able to run and receive the live stream via VLC media player. First run the live stream on you pi bot adjusting the command below depending on your needs:

```
raspivid -o - -t 0 -n | cvlc -vvv
stream:///dev/stdin --sout '#rtp{sdp=rtsp://:8554/}' :demux=h264
```

-o - will cause the output to be written to stdout, -t 0 sets the timeout to disabled the stream. -n stops the video being previewed, this being thought the HDMI output, so if you are plugged in to an monitor you can view your stream by removing it -n. cvlc is the console VLC player.-vvv and its argument specifies where to get the stream from, and finally -sout and its argument specifies where to output it to.
other additions to raspivid include changing the width, height and fps of the stream as seen below:

```
raspivid -o - -t 0 -n -w WW -h HH -fps FPS | cvlc -vvv
stream:///dev/stdin --sout '#rtp{sdp=rtsp://:8554/}' :demux=h264
```

Where WW is the width of your live stream, HH is the height, and FPS is the fps of your live stream, note that the higher FPS values are more likely to increase the lag between your camera and the live stream, so adjust accordingly.
Once you begin live streaming the pi will begin buffering images, to view the stream you must now open VLC on another computer. Once opened navigate to media>open network stream> type:

```
rtsp://###.###.###.###:8554/
```

Where ###.###.###.### is the IP address of your pi bot. other helpful commands for streaming include -b that controls the bit rate in MBits, -k, or –keypress this cycle between capturing and pausing the stream on the ENTER key. For more information check the documentation of raspivid in your ssh or online at **Raspivid Documentation**

# 7 Auto-Calibration

As you have probably discovered from running demo code, setting motor values to be equal does not necessarily cause the robot to drive straight. This is a physical limitation of the motors themselves since there are slight variances in resistance in the gearing. Therefore, an offset is required to drive in a straight line, which is desired for most standard tasks and operations.

## 7.1 Single Operational Speed

For many tasks, operating the robot at one set speed is all that is required so we have to ensure that it can drive in a straight line. There are many different ways to calibrate the robot and determine the offset, the most obvious way is through trial and error, adjusting motor speeds until the resulting values cause the robot to drive straight. However, this is not a very efficient way of finding the motor offset and requires a lot of manual work. Instead we want to create a program to automate this process.

In order to determine the offset required for calibration, we are going to force the robot to drive along a straight reference line. This can be easily accomplished using a line follower. Assuming that the reference line is perfectly straight, the average motor values during the trial should cause the robot to drive straight. Since we already have demo code for the robot, we are going to use some of these files to make our lives easier. For this program, we are going to need the AlphaBot2.py, Line_Follow.py, and TRSensor.py demo scripts.

1. Begin by creating a new directory for the auto calibration code.
2. Copy over demo code files for **AlphaBot2.py**, **Line_Follow.py**, and **TRSensors.py**. Rename **Line_Follow.py** to something that indicates that this is the calibration code, like **auto_cal.py**
3. Modify the **auto_cal.py** to include the following:
   (a) Two arrays that hold all the set motor values during the trial, these are appended to for each time through the loop
   (b) Condition that exits the line follow while loop when all IR sensors read white (robot no longer on line)
   (c) Calculate the average motor speeds from the two motor value arrays
   (d) Output these values to a .txt file so that these calibrated values can be referenced by other files
4. Modify **AlphaBot2.py** to set *self.PA* and *self.PB* to the calibrated values loaded in from the .txt file generated
5. Run **AlphaBot2.py** to ensure that the robot is now driving in a straight line

This calibration code can be placed in any working directory so that any program you are writing can reference calibrated motor values. While this works well for a single calibration, our method can be made more robust by fitting an offset function for any desired motor speed.

## 7.2 Variable Operational Speed

For other tasks we may want a way to include a way to vary the operating speed of the robot. First, test to see whether the motor offset found with the previous calibration code holds true to other speeds. It is likely that it does not and that this offset is a function of the motor speeds. From experimentation it becomes clear that this function is non linear, therefore must

be solved for using a curve fitting model. Fortunately there are pre-existing python packages that will allow us to do just that. We'll start by using numpy's built in polynomial curve fitting tool.

Start by checking to make sure that we understand how the function itself works, create a new python file (does not have to be on the pi) and copy the following code:

```
import numpy as np

x = np.array([0, 1, 2]) # function input
y = np.array([0, 1, 4]) # function output

z = np.polyfit(x, y, 2) # creates array of polynomial coeeficients to specified order
p = np.poly1d(z) # create an object that acts as the function

print('z:',z)
print('p(4) =', p(4))
```

The solver requires two arrays of the same length, one that is the input to the function and one that is the output. You should be able to recognize the input and output array as representative as a simple exponential equation. We print out the array $z$ for the coefficients and the result of the function with the input of 4. Here we can confirm that the curve fitting function is working. When adapting this for calculating motor offset, the desired motor speed is the input and the offset is the output.

Next we want to modify the previous calibration code to include this curve fitting model. In order to do so we want to:

1. Run the line follower a minimum of three times at three different speeds
2. Create two new arrays for the input (set motor speed) and output (offset between calibrated motor values)
3. Test that the function is working by using a for loop to iterate over a set number of potential motor values and checking that the output of the function is realistic
4. Export a file with the function object so that it can be referenced in other programs, the best way to do this is by using the built in python package *pickle*:

```
f = open('poly.p', 'wb') # create/overwrite file "poly.p"
pickle.dump(offset_func, f) # write "offest_func" object to file
f.close # close file
```

Done correctly, this method is more powerful than the previous single operational speed calibration, however, is more difficult to accomplish in practice. Similarly, this calibration code can be placed in any working directory so that any program you are writing can reference the calibrated motor offset function.

# 8 Obstacle Course

For this challenge, the robot must be able to maneuver through an obstacle course made of wooden walls and black electrical tape on the floor. This will enable the use of the infrared sensors on the bottom of the robot, the ultrasonic sensor, and the infrared distance centers on the front. The tape can be used to create curves in the track, and the wooden walls create a varying height and length of each wall. While continuing to change the course, the robot will have to maneuver in a different way each time.

The code for the obstacle avoidance using the five bottom infrared sensors and front two infrared sensors is in a separate zip file. MrRobot.py is the main file for the obstacle avoidance.

There are boolean variables for each sensor, which are false when there is no obstacle and true when there is an obstacle. There are defined functions at the top of the code that return the boolean values for each sensor. These functions use the demo code to be able to return values from the robot. The code loops through an infinite while loop (line 92) so the robot always is moving. Inside this while loop, there is another, inner while loop (line 93). This inner loops calls the function to move the robot forward, and checks to make sure the sensors still return false. Once a sensor is flagged and its function returns true, the inner while loop is exited.

Then, there are a series of if statements to check to see which sensor was flagged (lines 101-117). Each if statement calls a different movement for the robot, to be able to stay within the bounds of the obstacle course. At the end of the code, the sensors are checked again (line 122 and 123). If the sensor is still flagged at the end of the while loop, the next iteration will not go into the inner loop and will instead go through the series of if statements again.

The ultrasonic code to check the sensors on lines 99 and 124 are commented out, so the ultrasonic boolean value are never updated and always are false. When using the ultrasonic in combination with the infrared sensors, the code would get stuck in the inner while loop. This was attempted to be fixed but no solution was found. For example, a counter variable was printed and incremented every iteration through the inner loop. When the code stopped working, the counter would no longer be printed, indicating the code is stuck on a line in the while loop. Different robots were used to attempt to find a solution, but the same thing happened on the other robots.

To run the code, follow the steps below.

1. Connect to the robot over WiFi and use the computer's terminal to access the Pi's terminal.

2. The main file to run the obstacle avoidance is "**MrRobot.py**." Navigate using terminal commands to the directory the file is located in.

3. Place the robot on a black line. The code will initialize the bottom infrared sensors by spinning in a circle.

4. Run the file using the command "**python MrRobot.py**"

5. The robot will spin around. Once it is done, place the robot inside the obstacle course. Press the button located on the right side, in the middle of the "A," "B," "C," and "D" labeled control pad.

6. The robot will navigate the obstacle course. To stop it, press Control-X (even on a Mac) in the terminal.

# 9 Pi to Pi Communication

## 9.1 Bluetooth

To enable the Bluetooth setting on the Pi, you first need to download the software. Once connected to the Pi, in your terminal, type:

```
sudo get-install pi-Bluetooth
sudo reboot
```

Next, to get the information about your Pi, type:

```
hciconfig
```

This will provide you with the BD Address that will allow you to pair to other devices. Now type the following to get the Bluetooth program running:

```
sudo bluetoothctl
```

You terminal will now have text that looks like **[bluetooth]#** at the beginning of your terminal line and you will enter all the commands next to this. Begin by typing the following:

```
agent on
default-agent
pairable on
```

At this point, you will be asked for the BD Address of the device with which you are trying to pair. You can type "**scan on**" into the terminal and all pairable devices within range will show up (this does not seem to be the easiest way, however). It seems to be easier to pair when you know the other device's number, since undesired devices can overpopulate the terminal if many such devices are within range. Once you have found the other Pi's BD Address, type the following:

```
pair XX:XX:XX:XX:XX:XX
trust XX:XX:XX:XX:XX:XX
```

Rather than typing XX:XX:XX:XX:XX:XX, you will want to replace the X's with the BD Address of the other device. If done correctly, the Pis should now be connected to each other. Below is a list of helpful commands when in the Bluetoothctl program:

```
[bluetooth]# help
Available commands:
  list                       List available controllers
  show [ctrl]                Controller information
  select <ctrl>              Select default controller
  devices                    List available devices
  paired-devices             List paired devices
  power <on/off>             Set controller power
  pairable <on/off>          Set controller pairable mode
  discoverable <on/off>      Set controller discoverable mode
  agent <on/off/capability>  Enable/disable agent with given capability
  default-agent              Set agent as the default one
  advertise <on/off/type>    Enable/disable advertising with given type
  set-advertise-uuids [uuid1 uuid2 ...] Set advertise uuids
  set-advertise-service [uuid][data=[xx xx ...] Set advertise service data
  set-advertise-manufacturer [id][data=[xx xx ...] Set advertise manufacturer data
  set-advertise-tx-power <on/off> Enable/disable TX power to be advertised
  set-scan-filter-uuids [uuid1 uuid2 ...] Set scan filter uuids
  set-scan-filter-rssi [rssi] Set scan filter rssi, and clears pathloss
  set-scan-filter-pathloss [pathloss] Set scan filter pathloss, and clears rssi
  set-scan-filter-transport [transport] Set scan filter transport
  set-scan-filter-clear      Clears discovery filter.
  scan <on/off>              Scan for devices
  info [dev]                 Device information
  pair [dev]                 Pair with device
  trust [dev]                Trust device
  untrust [dev]              Untrust device
  block [dev]                Block device
  unblock [dev]              Unblock device
  remove <dev>               Remove device
  connect <dev>              Connect device
  disconnect [dev]           Disconnect device
  list-attributes [dev]      List attributes
  set-alias <alias>          Set device alias
  select-attribute <attribute> Select attribute
  attribute-info [attribute] Select attribute
  read                       Read attribute value
  write <data=[xx xx ...]>   Write attribute value
  notify <on/off>            Notify attribute value
  register-profile <UUID ...> Register profile to connect
  unregister-profile         Unregister profile
  version                    Display version
  quit                       Quit program
```

**Figure 19:** Helpful commands for bluetoothctl

Upon getting one Raspberry Pi connected to another, it was difficult to make any progress from this point in terms of actually getting any sort of function to work from Pi to Pi. One problem that comes with using Bluetooth is that only one Pi can be connected to another at a time, meaning that if you have a group of Pis, they cannot all be communicating at once through this system. From doing research into Bluetooth problems, it seems that using WiFi to have a group of Pis communicate is the best approach for Pi to Pi communication because they can all be connected to the same server at once and can all theoretically communicate with one another.

A helpful link for setting up Bluetooth can be found here: **link**.

## 9.2 WiFi Communication

If the previous steps were taken correctly, communicating with another Pi on the same WiFi is a trivial task—the possibilities of this communication are much more vast and complex. To test the connection between two AlphaBots, all that is needed is the IP address of one of the AlphaBots and for both to be set up on the same WiFi. The command "**sudo ping**" will give you a list of possible options when pinging another bot.

```
sudo ping -cX ##.###.##.##
```

Entering this, with "X" replaced by a number, in to one of the raspberry pi's, via SSH or separate monitor, will ping a robot at the IP ##.###.##.## X times.

Following the steps in this video **link** can be helpful for simple WiFi communication testing. This is where one Pi acts similar to that of a server. Both Pis have to be connected to the same WiFi in order to communicate properly. Since each Pi has a different IP address, these should be known and recorded for the code that is to follow.

Start by creating a new folder on the Pi that is going to send code by typing into the terminal:

```
cd Desktop
sudo mkdir send
cd send
sudo nano send.py
```

This creates a new folder on the desktop of the Pi called send which the python code send lies. Once inside of the file, you will want to type in the following code with the modified changes based on your Pi's IP and port address.

```
import socket

UDP_IP = "sender raspberry pi IP"
UDP_PORT = 5005
MESSAGE = "ANY TEXT"

print "UDP target IP:", UDP_IP
print "UDP target Port:", UDP_PORT
print "Message:", MESSAGE

sock = socket.socket(socket.AF_INET, #Internet
                     socket.SOCK_DGRAM) #UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

Now that one Pi is configured to send a simple text message, we are going to set up the receiving Pi's code. Again, the following code is going to be needed to modified with the receiving Pi's IP and Port values.

```
import socket

UDP_IP = "receiver raspberry pi IP"
UDP_PORT = 5005

sock = socket.socket(socket.AF_INET, #Internet
                     socket.SOCK_DGRAM) #UDP
sock.bind((UDP_IP, UDP_PORT))
```

```
while True:
    data.addr = sock.recvfrom(1024) #buffer size is 1024 bytes
    print "Received message:", data
```

Now that everything is set up, you should be able to type in:

```
sudo python receive.py
```

to run the file. Go to the Pi with the send.py file and type into the terminal:

```
sudo python send.py
```

to run the file. You should see the UDP IP, Port, and Message in the terminal. Looking at the terminal with the receive.py file, the message should have now popped up on screen.

## 10 Notes/Helpful Additions

1. Make sure your camera has even turning capabilities to its right and left during assembly so that its camera has symmetrical turning.
2. Order pre-soldered boards to ensure proper pin functionality and avoid soldering training.
3. Pre-order multiple mini-HDMI cables.
4. Have tool-kits for assembly purposes.
5. Buy more batteries and faster/more charging packs.
6. Storage lockers for robots, tape, wood, etc.