Harvard Microrobotics Laboratory
Harvard University

Assembling and Programming a Pheeno V2 Robot System

An ENG-SCI 91r Research Project

Jonas Hansen

Advised by Dr. Justin Werfel
ADUS: Dr. Chris Lombardo

3 May 2023

Edited by Spring Berman

Table of Contents

Introduction

Project Scope and Report

The goal of this project was to assemble a fully functional Pheeno robot (Wilson et al., 2016), and, if time permitting, implement a swarm algorithm to determine its effectiveness. This report describes my assembly and software development process for the Pheeno, with the intention of serving as a thorough guide for anyone interested in building and testing Pheenos of their own.

This guide is my attempt at making the assembly process as clear and bug-free as possible and is divided up into the following five chapters:

1. Chapter 1 describes the assembly of the printed circuit board (PCB), and tips for how to go about the process.
2. Chapter 2 outlines the construction of the actual parts of the Pheeno.
3. Chapter 3 is a walkthrough of the software setup for the onboard Raspberry Pi, as well as how to test simple movements of the Pheeno.
4. Chapter 4 is a setup and installation manual of the "workstation software," or the software to be used by the user's computer to act as the "master" program.
5. Chapter 5 is an overview of how I went about implementing the ACS laboratory Pheeno Markov chain algorithm, and provides a brief analysis of whether it would prove to be an effective swarm algorithm given a large group of Pheenos.

Sprinkled throughout these chapters are things I discovered worth noting and failures I encountered along the way. I reflect on these findings, and suggest what improvements could be made down the road. I hope that in the future this manual could be used by students or any hobbyists as a helpful guide for exploring the realm of swarm robotics using the Pheeno platform.

Chapter 1: PheenoV2 PCB Assembly Instructions

Getting Started

This instruction manual is a guide to assembling the printed circuit board (PCB) for the PheenoV2 robot. The first step of course is to actually have the necessary PCB to solder parts onto! All the schematics and related files for printing boards for this robot can be found on the Autonomous Collective Systems (ACS) Laboratory GitHub repository here. For getting boards printed, I would highly recommend JLCPCB. They are both cheap and efficient, so be sure to get a good amount of extra boards printed in bulk. You might accidentally fry a board or solder through some channels, so it's a good idea to have some backup.

What You'll Need

Equipment/Tools

Fine-tipped soldering iron and solder
Safety goggles/glasses and a fume hood (or some kind of vacuum to safely vent the solder fumes)
Crimp tool (for micro-terminal crimping, like this one)
Pliers
Diagonal cutters
Wire strippers

Parts

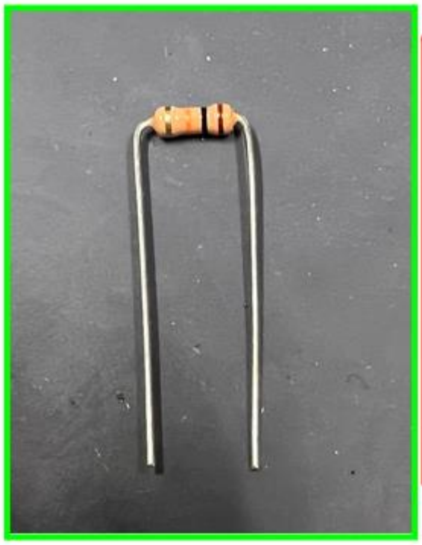
A complete list of parts can be found on the ACS Laboratory website here.

IN ADDITION, I discovered some extremely helpful modifications that require these female header pins:

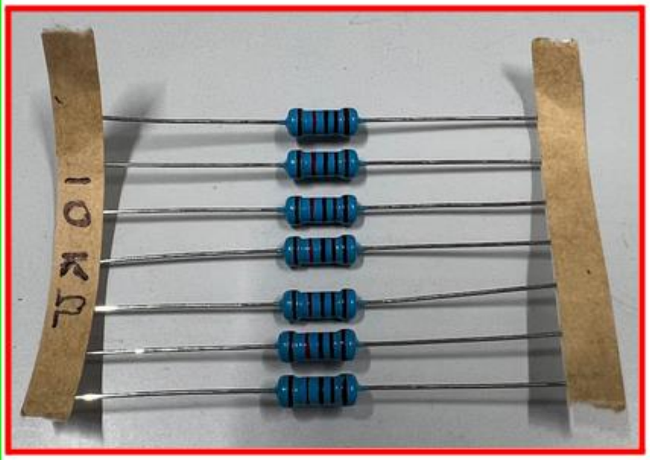
Female Headers.

➔ These headers are for the Teensy3.2 board and Adafruit motor drivers so you can easily remove them from the PCB if they need to be replaced or if you need to debug some issues. They can also be used for any other part you want to be easily removable!

➔ Another note that I discovered regarding the 10kΩ resistors:



✓ Use these 10kΩ resistors



⃠ **DO NOT** use these 10kΩ resistors

- The blue 10kΩ resistors are too large for the space allocated on the PCB, so be sure to use the light brown 10kΩ resistors.

Getting to Know the Board

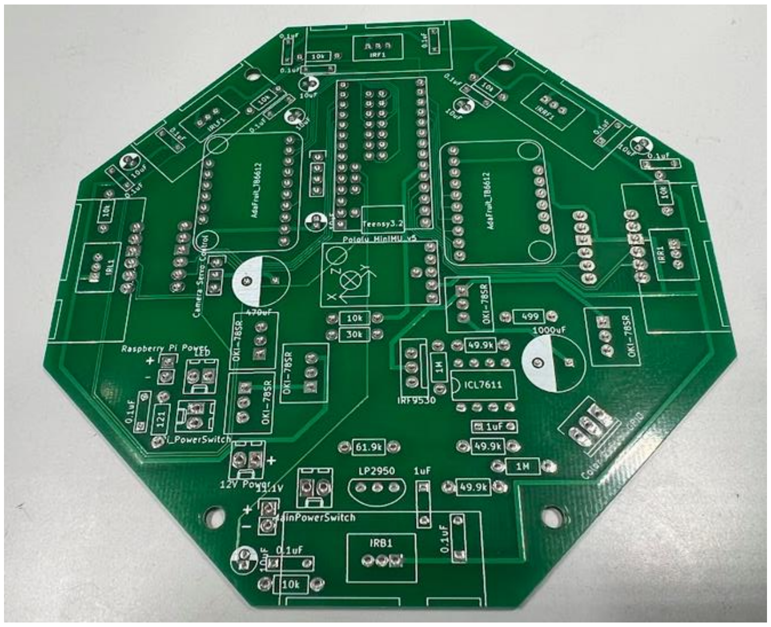


Figure 1.1: PheenoV2 PCB board without components.

➔ Figure 1.1 above shows a PheenoV2 PCB board without any components attached. Each region for a component is labeled with the part name/number, and for asymmetric parts, the bounding box is drawn in such a way to indicate the orientation in which the designated component should be placed. So for example, when attaching a LP2950 transistor to the board, be sure that the curved side of the transistor lines up with the curved border of the bounding box, as shown in Figure 1.2 below:

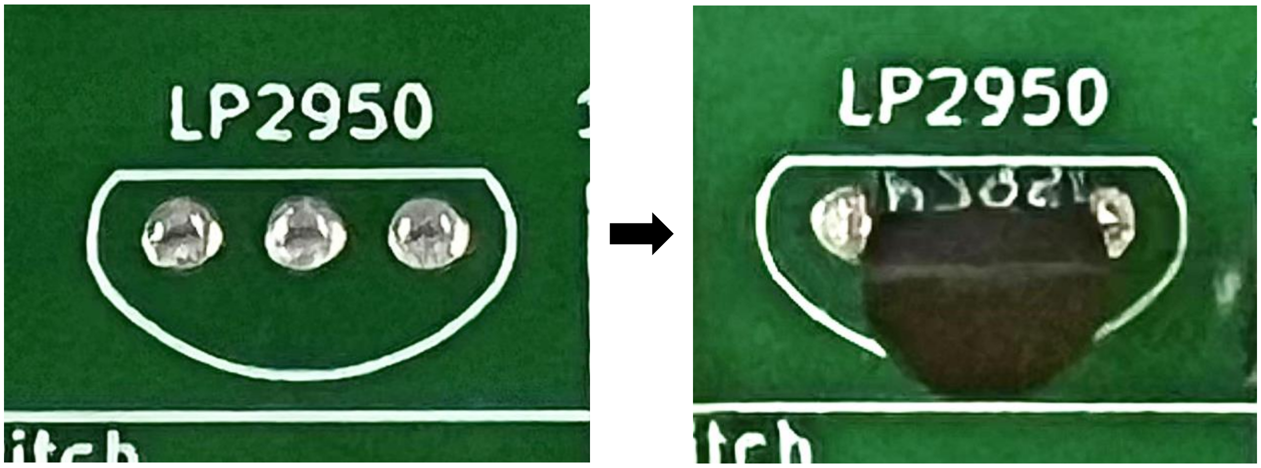


Figure 1.2: Lining up orientation of a part with its designated space on the PCB.

➔ Another thing to note: if you look across the board, you'll often see parts with square pads in addition to the usual circular pads. These square pads denote the *positive pin* (or "pin 1") of the part. For example, let's take a look at this spot that will eventually house a 1000µF capacitor:

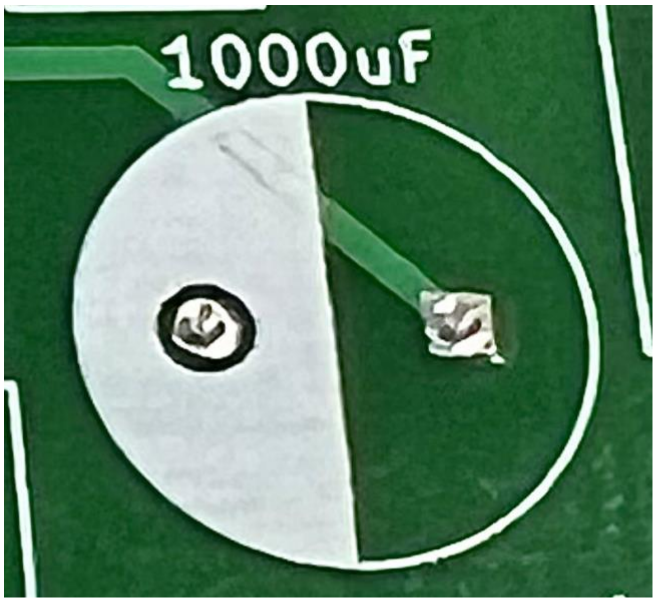


Figure 1.3: PCB location for 1000µF capacitor.

We can see that the contact pad on the right is square, and the one on the left is circular. This means that the *positive* pin of the capacitor should be inserted into the square pad and the *negative* pin into the circular pad. So if you ever get tripped up on which way to attach a part, just remember this rule!

Assembling the Board

This section describes the process I took to put together the PCB, along with some tips and tricks I discovered along the way. Feel free to make adjustments as you see fit, especially if you are already more comfortable with doing things a certain way (e.g. if you have a specific way you like to solder, if you prefer attaching the larger components first and then the smaller components, etc.). This is simply a guide with some of my personal reflections on what worked for me and what did not turn out so great.

Step 1 – IR Transceiver Mounts

What You'll Need

- 6x 3-pin female headers

- 18x male breakaway pins
- 18x 24-30 Molex connection terminals

Assembly

1. The first task is to prepare the IR transceiver mounts so we can solder them to the board. Use some pliers to pull out 18 metal pins from their black plastic encasing in a row male breakaway pins, like this (grip the black encasing with one pair of pliers/wire cutters and pull out the metal pins with another pair):

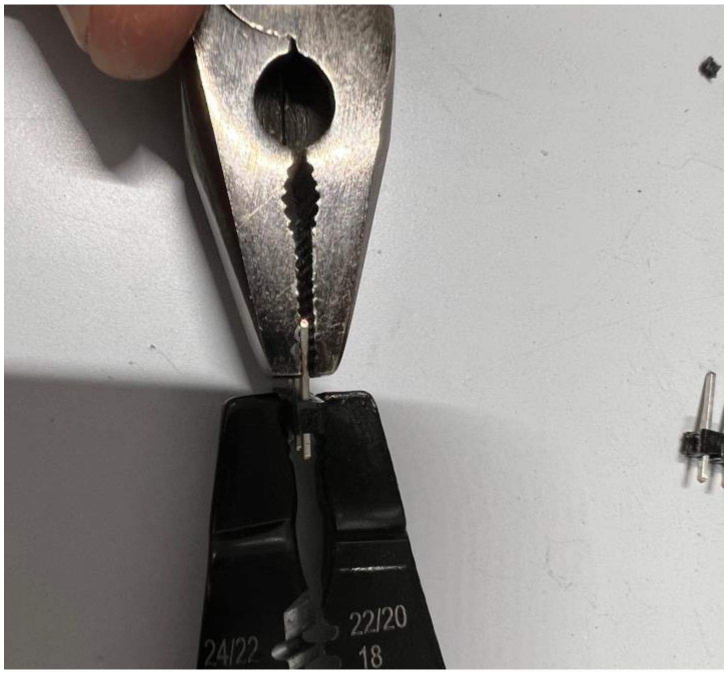


Figure 2.1: Pulling out the metal pins from male breakaway pins.

2. For each 3-pin female header, gather together 3 metal pins and 3 24-30 Molex connection terminals, like so:

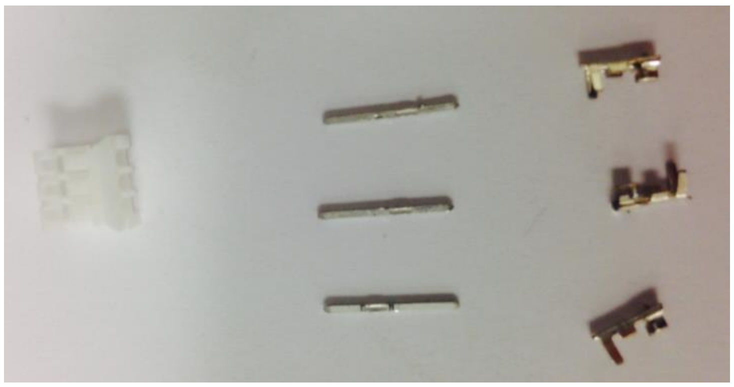


Figure 2.2: Parts needed for the IR transceiver mounts.

3. Now use your crimp tool to crimp each metal pin into a Molex connector. After doing so, give the metal pin a tug to check that it is in securely.
4. Use your pliers to insert the connectors into the 3-pin female headers, being sure that the side of the Molex connector with the little overhang is the one that faces the holes of the 3-pin female header. This means that the connector should click into place into the female header, as shown in **Figure 2.3** below.
    a. **Tip:** if you can't get the Molex connector to fit in the female header, use your pliers to squeeze the connector more snugly around the metal pin.

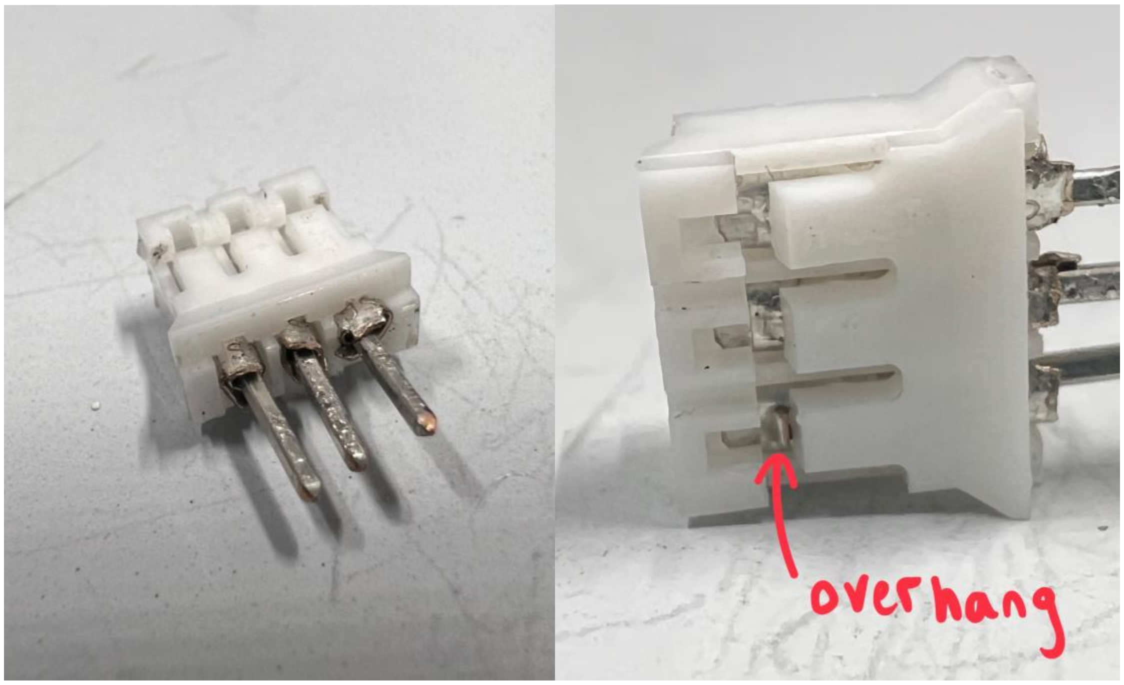


Figure 2.3: Assembled IR transceiver mount**.**

5. Once you have 6 IR transceiver mounts assembled, go ahead and solder them into the PCB board into their designated spots, as shown in **Figure 2.4** below.

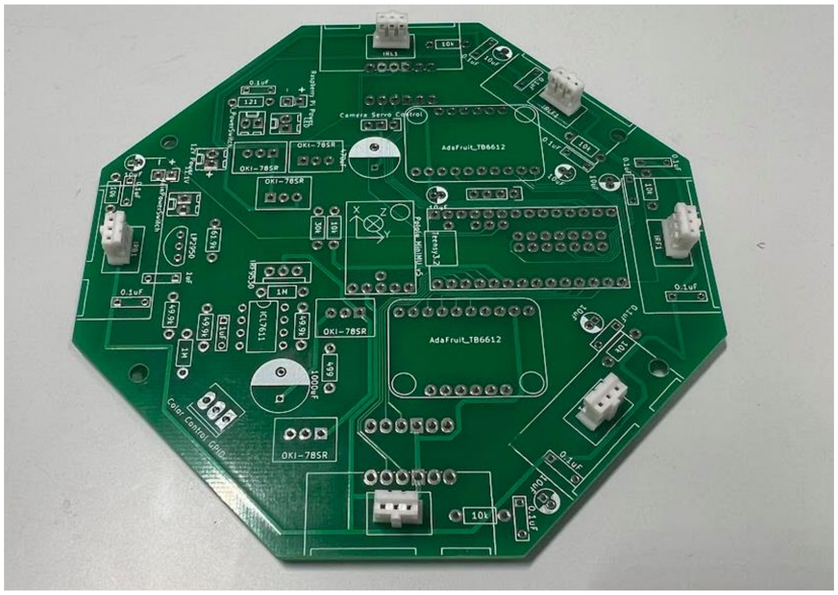


Figure 2.4: PCB board with IR transceiver mounts attached.

6. Finally, use your diagonal cutters to trim away the excess metal pins that stick out from the bottom of the PCB.

Step 2 – Resistors

What You'll Need

- 1x 61.9kΩ resistor
- 3x 49.9kΩ resistors
- 2x 1MΩ resistors
- 1x 30kΩ resistor
- 7x 10kΩ resistors
- 1x 121Ω resistor (I found that a 120Ω resistor works fine)
- 1x 499Ω resistor

Assembly

1. This step is simply a matter of soldering a bunch of resistors onto the PCB board. Gather all of the resistors in piles sorted by their resistance, and bend their leads tightly at 90° angles so that they fit snugly into the board, like this:

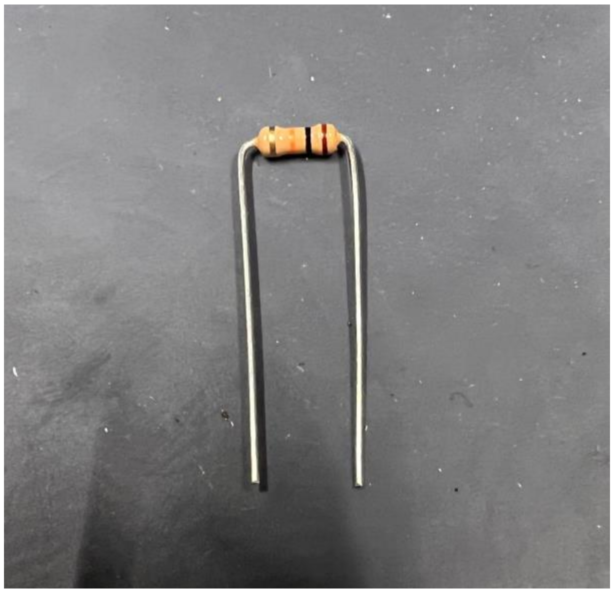


Figure 3.1: Bending resistor leads to insert into the PCB.

2. Once you insert a resistor into its slot, bend one of its leads along the back surface of the board in one direction and the other lead in the opposite direction, like this:

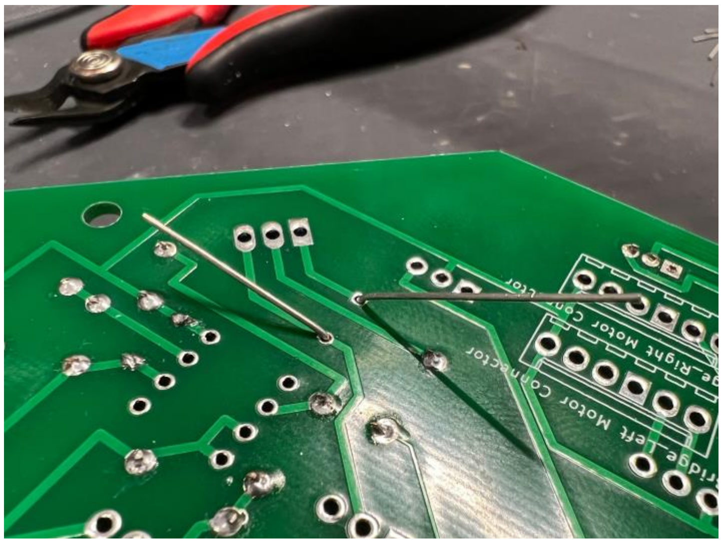


Figure 3.2: Picture of backside of PCB showing how to bend inserted leads to secure part for soldering.

   This will secure the resistor in place and make it *much* easier to solder!

3. Finally, trim away the excess leads sticking out of the bottom of the PCB using your diagonal cutters.
   a. The most important thing is to **take your time** and not rush—you don't want to accidentally burn through a channel on the PCB board or leave a glob of solder somewhere that might cause a short-circuit down the road.

Step 3 – Capacitors

What You'll Need

- 12x 0.1µF capacitors
- 2x 1µF capacitors
- 1x 1000µF capacitors
- 1x 470 µF capacitors
- 7x 10µF capacitors

Assembly

1. Gather the capacitors together and **check which capacitors are electrolytic/tantalum or ceramic/film.** The polarity matters in electrolytic/tantalum capacitors but does not matter for ceramic/film capacitors. This means that you need to match the orientation indicated on the board for the electrolytic/tantalum capacitors whereas the ceramic/film capacitors can go in either way.
2. Begin to solder the capacitors onto the board, **keeping them as tight to the board as possible** (remember you can secure them to the board before soldering by replicating the bending technique shown in Figure 3.2). Keeping the capacitors tight to the board will make the board less tall in the end and reduce the possibility of short-circuits.

Step 4 – Molex Headers, Male Breakaway Pins, and Op-Amp

What You'll Need

- 4x 2-pin Molex Headers
- 2x 2-pin Male Breakaway Pins
- 1x 3-pin Male Breakaway Pins
- 1x Op-Amp (ICL7611)

Assembly

1. Break off two 2-pin male breakaway pins and solder one of them into the "Raspberry Pi Power" slot and the other into the "11.1V" slot.
2. Break off a 3-pin male breakaway pin and solder it into the "Camera Servo Control" slot.
3. Gather the four 2-pin Molex Headers and solder one into the "LED" slot, one into the "Pi_PowerSwitch" slot, one into the "MainPowerSwitch" slot, and one into the "12V Power" slot.
4. Finally, solder the Op-Amp into the "ICL7611" slot, making sure to match the orientation indicated by the outline.

Step 5 – Teensy3.2, Adafruit Motor Drivers, and Pololu MiniMU

What You'll Need

- 2x 36-pin long Female breakaway sockets
- 1x Row of SMC Header Pins (for the underside of the Teensy3.2 board)
- 1x Teensy3.2 board
- 2x Adafruit TB6612 Motor Drivers
- 1x Pololu MiniMU
- Male breakaway pins (should come with the above boards)

Assembly

1. This is probably the trickiest part of the PCB assembly process and requires the most precision and care. To start off, let's get our rows of female sockets ready. We're going to break off 2x 6-socket rows, 2x 10-socket rows, 2x 7-socket rows, 2x 14-socket rows, and 1x 5-socket rows. To break off these chunks of sockets, start by removing the pin *after* the number of sockets needed using pliers, like I'm doing here for a 14-socket row (I'm removing the 15th pin):

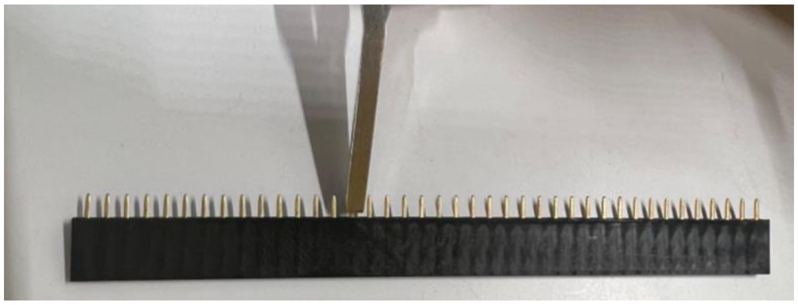


Figure 4.1: Pulling out the pin after the number of sockets needed in a row of female sockets.

2. Next, use your diagonal cutters to cut along the socket of the pin you just removed **without cutting into the adjacent sockets**, like this:

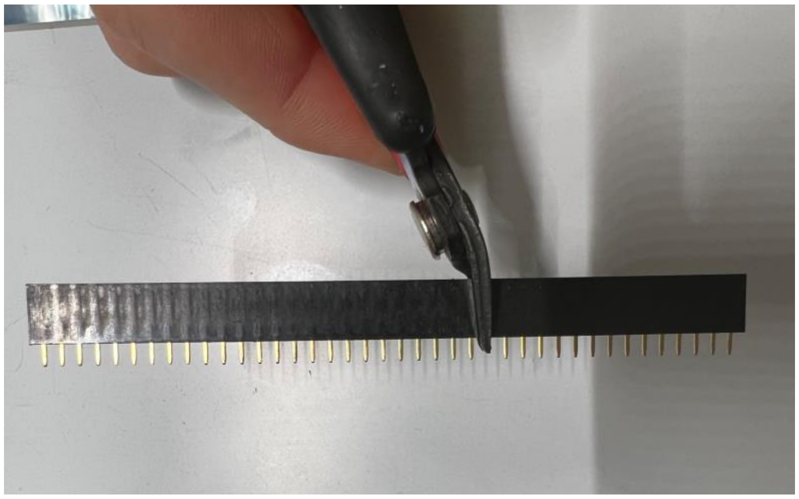


Figure 4.2: Breaking off a 14-socket row using diagonal cutters.

3. Do this for all of the required sockets listed in Step 1.
4. Now **carefully and precisely** solder these socket rows into the slots for the Teensy3.2 board and the Adafruit Motor Drivers. **Keep them as straight as possible.** Here was my first attempt and you can see how the middle socket rows for the Teensy are crooked:

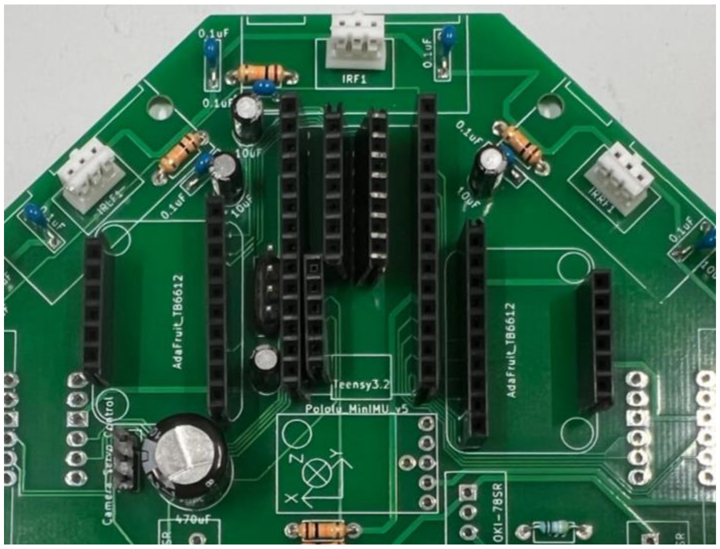


Figure 4.3: Showing a crooked soldering job of inserting the female socket rows into the PCB.

Here is what it looked like after I straightened them out a bit:

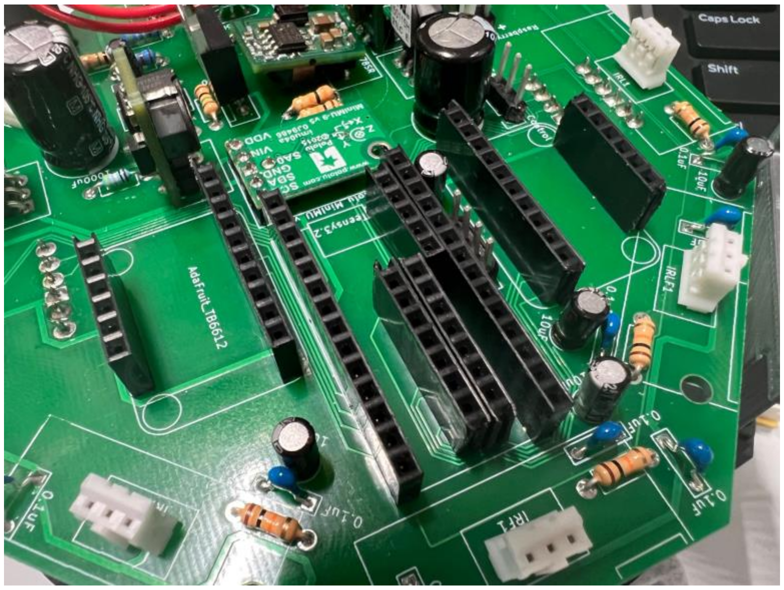


Figure 4.4: Showing a straighter soldering of the female socket rows into the PCB.

5. Now let's prepare the Teensy3.2 board. Use the rows of male breakaway pins that come in the bag with the Teensy and break off 2x 14-pin rows and 1x 5-pin row. Stick the short ends of the pins up the bottom of the Teensy and solder them on the top. If you're having trouble stabilizing the Teensy, try using some styrofoam to stick the long ends of the pins into to hold them in place. **Make sure the pins are as straight as possible**.
6. Now solder the bent side of the row of SMC header pins to the exposed contacts on the bottom of the Teensy. **Solder them as aligned and straight as possible**, otherwise you'll have a lot of trouble inserting them into the female sockets. The end result should look like this:

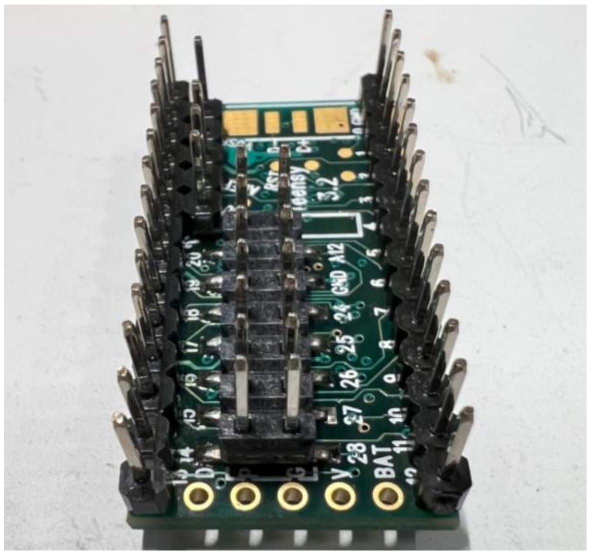


Figure 4.5: Bottom side of Teensy3.2 board with soldered pins.

7. Now break off 2x 6-pin rows and 2x 10-pin rows and solder them to the Adafruit Motor Drivers (each gets one 6-pin row and one 10-pin row). The result should look like this:

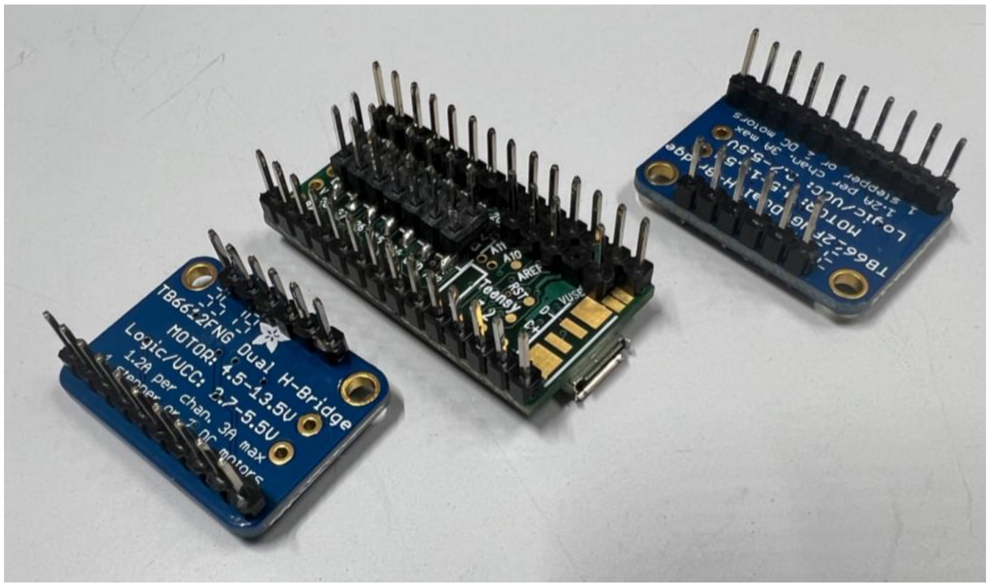


Figure 4.6: Bottom sides of the Teensy board and the two Adafruit boards with soldered pins.

8. Next, break off a 5-pin row and a single pin and solder them to the MiniMU board.
   a. **Note:** I didn't use female sockets for the MiniMU because I discovered that it raised the MiniMU off the PCB such that it blocks the microUSB port on the Teensy.
9. Go ahead and solder the MiniMU to the PCB board in its allocated space.
10. **Wait to insert the Teensy and Adafruit boards into their female sockets until the rest of the PCB is finished** (in case of soldering issues or other things that might risk damaging these boards).

Step 6 – Transistor and Regulators

What You'll Need

- 1x Transistor (IRF9530)
- 1x 5V Lower Current Regulator (LP2950)
- 5x 5V Regulator (OKI-78SR)

Assembly

1. Begin by using your pliers to bend the pins on the five 5V OKI-78SR regulators like this:

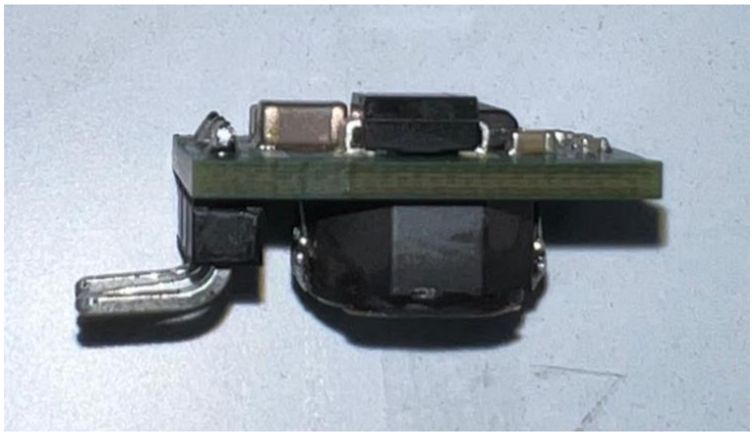


Figure 5.1: Bending the pins on the OKI-78SR regulator.

This allows you to solder them onto the PCB vertically. I discovered that if you don't bend the pins and solder these regulators onto the board horizontally, there just isn't enough room for them amongst all of the other parts.

2. Now solder these OKI-78SR regulators into their designated slots on the PCB:

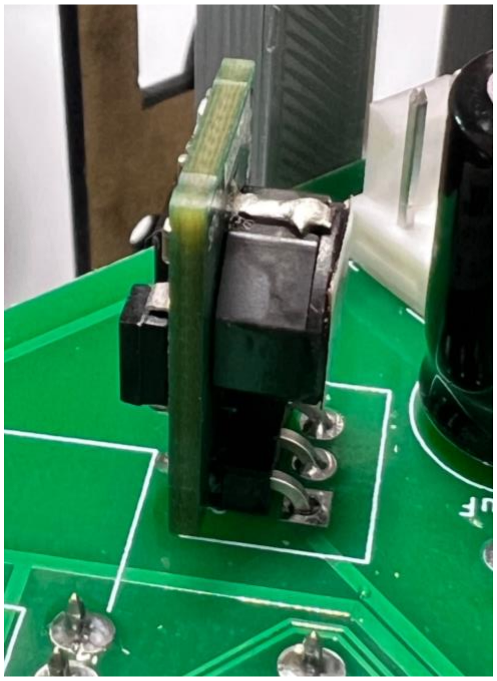


Figure 5.2: OKI-78SR regulator on the PCB.

3. Now solder the 5V LP2950 lower current regulator into its spot on the PCB. Bend the pins to try to get the regulator close to the PCB, but don't force it too much or the pins could snap:

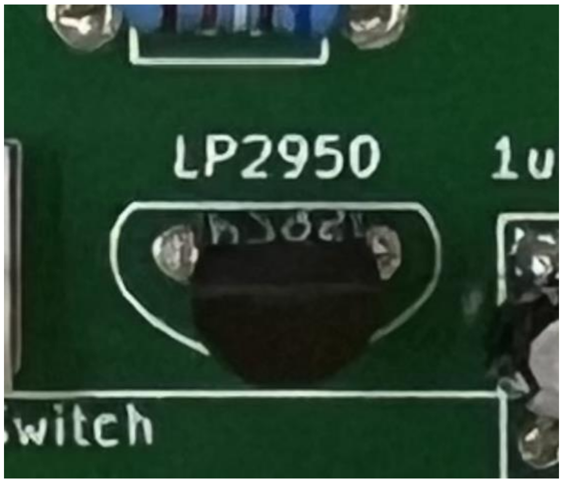


Figure 5.3: LP2950 regulator on the PCB.

Make sure the orientation of the regulator is correct!

4. Finally, solder the IRF9530 transistor into its spot on the PCB, matching the orientation indicated by the outline:

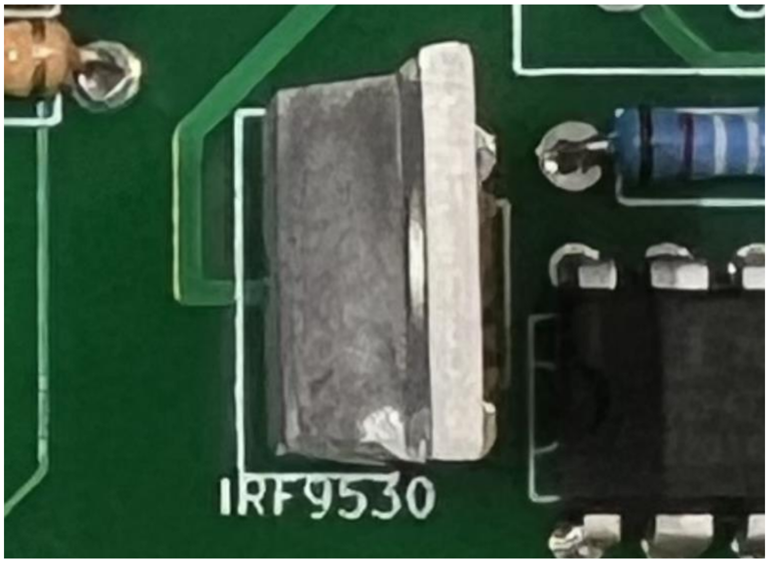


Figure 5.4: IRF9530 transistor on the PCB.

Step 7 – Bottom Molex Housing

What You'll Need

- 4x 6-Pin Molex Housings

Assembly

1. The last things you'll solder to the PCB are the 6-pin Molex Housings. These will be for the motors! Simply flip the PCB over and solder them into the designated slots like this:

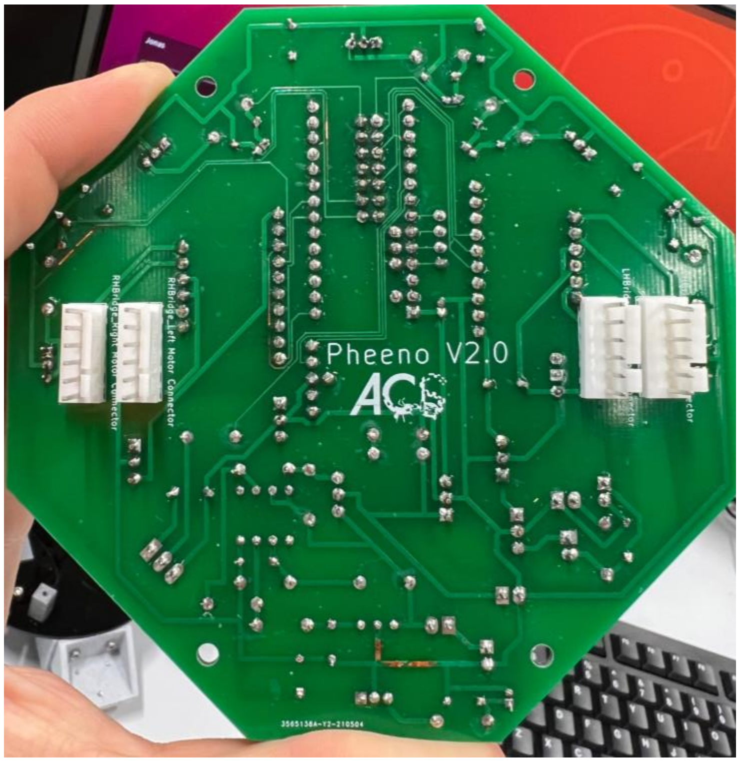


Figure 6.1: 6-pin Molex Housing on the bottom of the PCB.

Note: You are actually only going to use two of these housings (one for the left motor, one for the right), but soldering two on each side provides you with some options for how you want to connect the motors later (i.e. if the wires are closer to one side versus the other, etc.).

Step 8 – Inserting Boards, IR Transceivers, and Inspection

What You'll Need

- Your completed Teensy3.2 board
- Your completed Adafruit Motor Driver boards
- 6x Sharp IR Transceivers

Assembly

1. Insert the Teensy and the two Adafruit boards into the female socket slots you made for them in Step 5.
2. Insert the Sharp IR Transceivers into the 3-pin mounts you made for them in Step 1.
3. Congratulations! You've finished the PCB! Inspect all your soldering work and connections and be sure there are no potential short-circuits or damaged components.

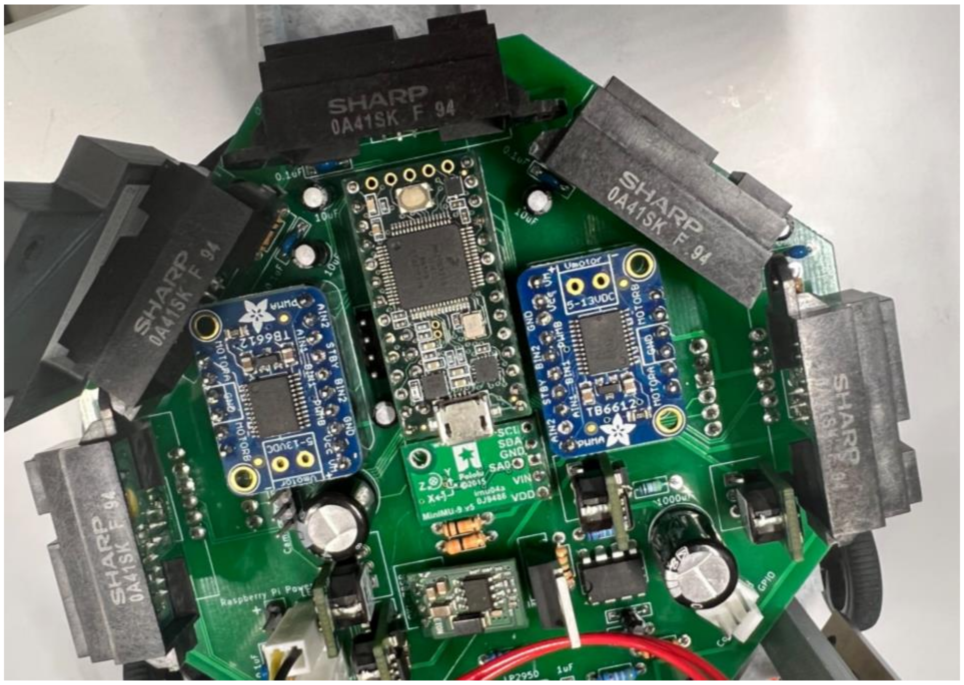
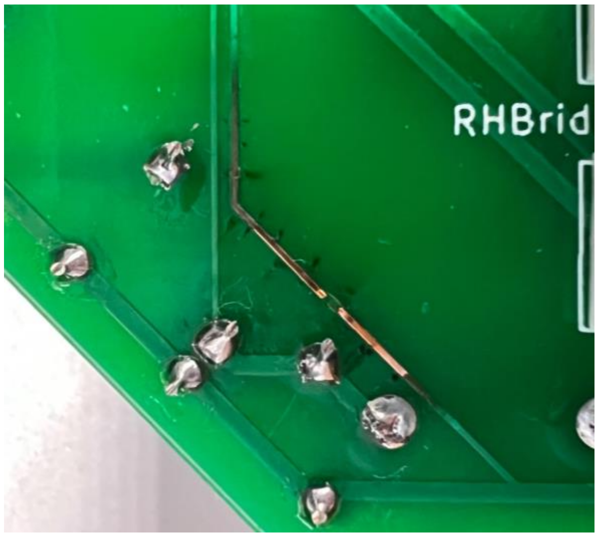


Figure 7.1: Final PCB assembly, with inserted boards and IR transceivers.

Chapter 1 Reflection

Assembling the PCB was definitely tedious, but discovering various tricks to make the process more streamlined was very rewarding. Bending pins in certain ways or securing components for easier soldering are tips I hope future readers of this manual find useful and effective. Before I had put together this PCB, there was a prior PCB that had been assembled yet was not operational. The first week of my research was spent trying to figure out what was wrong with this existing PCB and whether it could be salvaged. In the end, I found that several channels had been burnt through on the backside of the PCB, which might have accounted for the overheating and instant smell of smoke that arose whenever the board was hooked up to power. Here you can see a few of the damaged channels:



What’s more is that the major components such as the Teensy3.2 board and the Adafruit Motor Driver boards were soldered *directly* to the PCB and would therefore be extremely difficult to remove for debugging purposes. I decided to build a new PCB from scratch, making adjustments along the way such as having female socket connections for the Teensy and Adafruit boards so that they could be easily removed if needed. This also enabled me to go through the entire assembly process myself and uncover a lot of the tricks highlighted in this chapter.

Chapter 2: PheenoV2 Parts Assembly

Introduction

This phase of building the Pheeno robot was probably the most straightforward. This is because the construction of the parts (chassis, base, switches, etc.) is essentially identical to that of the PheenoV1 robot, for which there exists documentation on the ACS Laboratory website. The GitHub repository provides all the needed STL files for the PheenoV2 robot as well, which are slightly different from the PheenoV1 STL files but the build is almost exactly the same. As such, I am not providing in-depth assembly instructions in this chapter, especially since my focus was more on the PCB design, software setup, and actual deployment and testing.

PheenoV2 STL Files

The STL files can be found here. This includes the STL files for the gripper module, however my focus for this project report was more concerning the chassis and using it to test obstacle avoidance software and swarm algorithms.

Parts Assembly Instructions

The instructions for building the chassis, base, mounting parts, wires, and switches can be found starting with the power cable in the “Underside of Board” section on the ACS Laboratory page here. Note that the drivetrain might look a little different in these instructions compared to the one you 3D print from the PheenoV2 STL files. This is only because these instructions are intended for the PheenoV1 robot, however the process is exactly the same.

Chapter 3: Raspberry Pi Software Setup Guide

What You'll Need

For **Raspberry Pi Model 3B+**

Desktop Monitor with HDMI port
USB-A Keyboard
USB-A Mouse
Clean MicroSD card (preferably >=8GB) + MicroSD to SD adapter
SD card reader
HDMI Male to HDMI Male cable
MicroUSB power supply (~5V, 2.5A DC output)

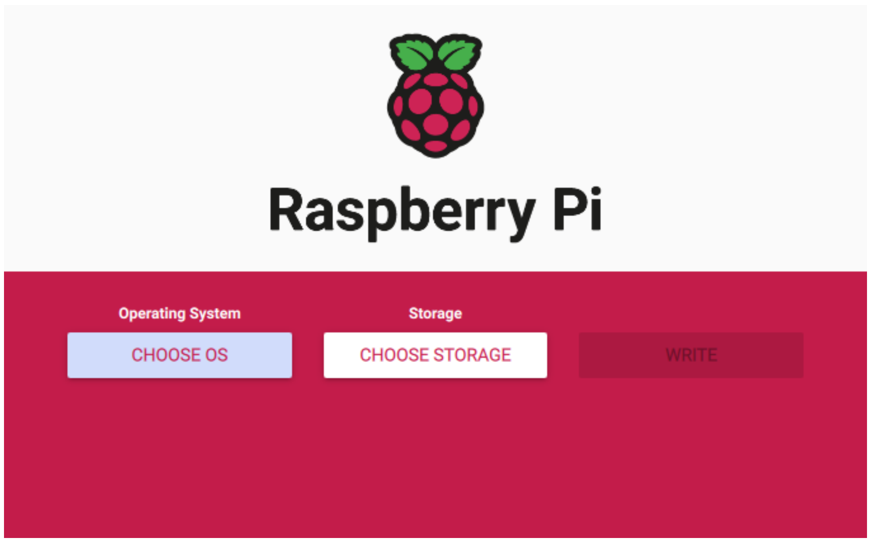
For **Raspberry Pi Model 4**

Desktop Monitor with HDMI port
USB Keyboard
USB Mouse
Clean MicroSD card (preferably >=8GB) + MicroSD to SD adapter
SD card reader
HDMI Male to **Micro HDMI** Male cable
USB-C power supply (~5.1V, 3A DC output)

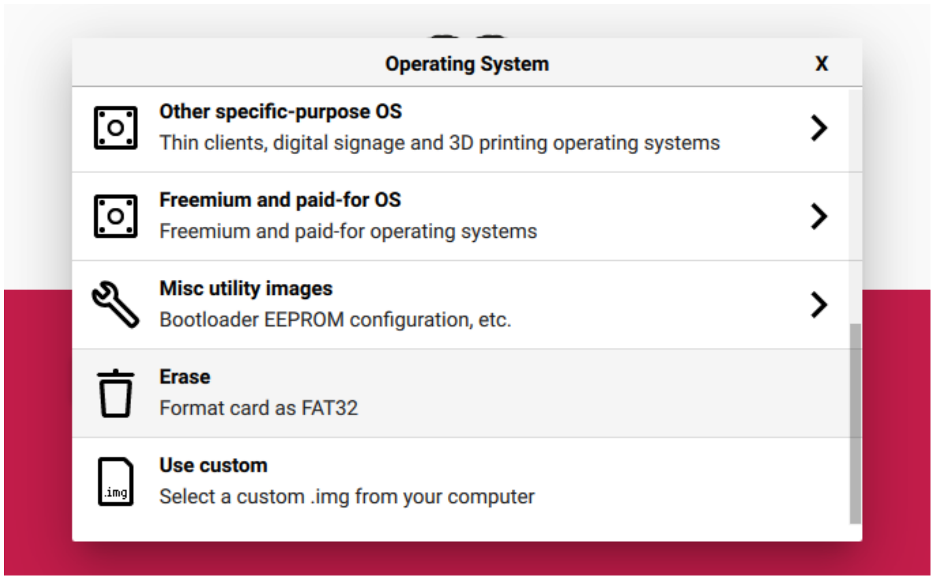
Software Setup

MicroSD Card

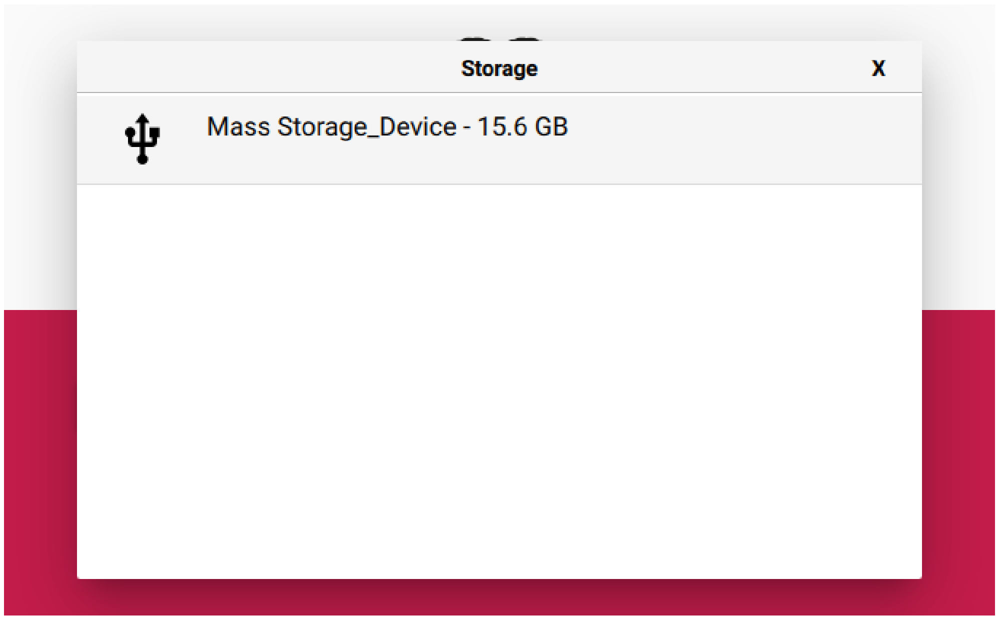
1. Our first step will be to install the **Ubuntu18.04** operating system onto the MicroSD card. Download the ubuntu-18.04.5-preinstalled-server-arm64+raspi3.img.xz image from Ubuntu site.
2. Now insert the MicroSD card into the MicroSD to SD adapter and insert that into a computer (if your computer does not have an SD card port, you can use an SD card reader, which is just an adapter from SD card to USB-A, USB-C, etc.).
3. Next, install the **Raspberry Pi Imager** here and open it once it has installed. You should see a window like this:



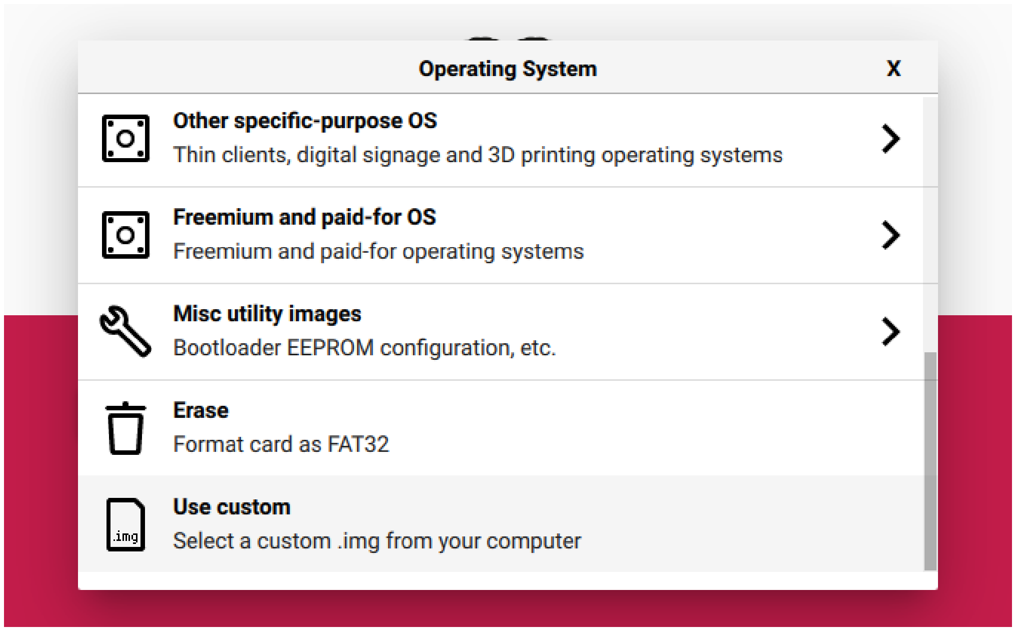
4. Now **you need to clean your MicroSD card** before installing Raspberry Pi OS (if you already cleaned your card, you can skip to Step 6). Luckily, if you haven't already done so, you can do it through the Raspberry Pi Imager. Simply click on "CHOOSE OS," scroll down and select the "Erase" option:


Operating System
X
Other specific-purpose OS
Thin clients, digital signage and 3D printing operating systems
Freemium and paid-for OS
Freemium and paid-for operating systems
Misc utility images
Bootloader EEPROM configuration, etc.
Erase
Format card as FAT32
Use custom
Select a custom .img from your computer
.img

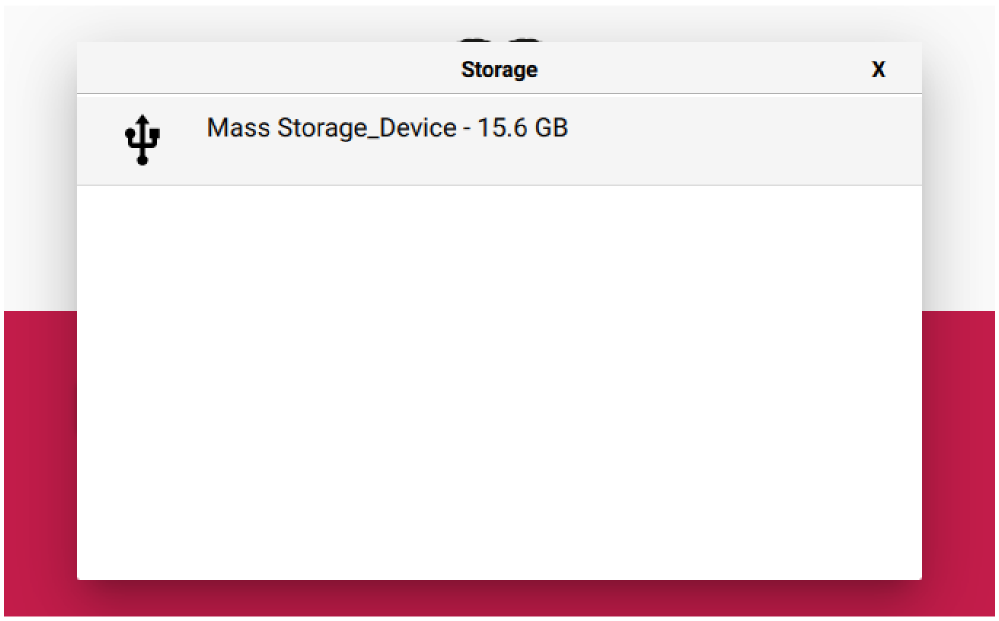
5. Now **select your MicroSD card for the storage option** by clicking on "CHOOSE STORAGE" and selecting your card:



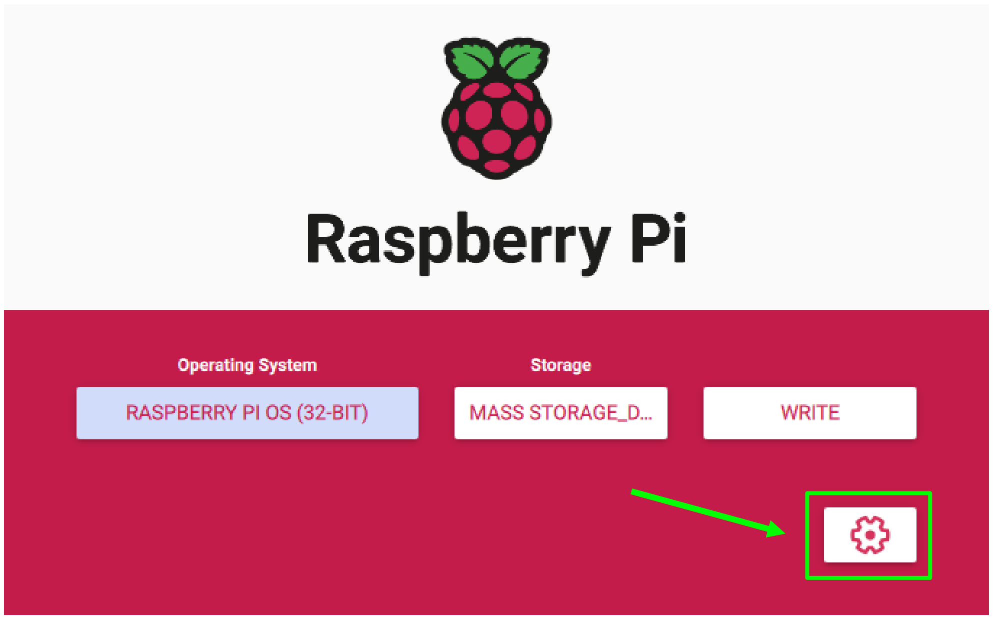
6. Finally, click “WRITE” and wait for the card to be wiped (shouldn’t take too long).
7. Now we are finally ready to install Ubuntu. Under the “Operating System” option in Raspberry Pi Imager, select “Use custom” and select the “ubuntu-18.04.5-preinstalled-server-arm64+raspi3.img.xz” image:


Operating System
X
Other specific-purpose OS
Thin clients, digital signage and 3D printing operating systems
Freemium and paid-for OS
Freemium and paid-for operating systems
Misc utility images
Bootloader EEPROM configuration, etc.
Erase
Format card as FAT32
Use custom
Select a custom .img from your computer
.img

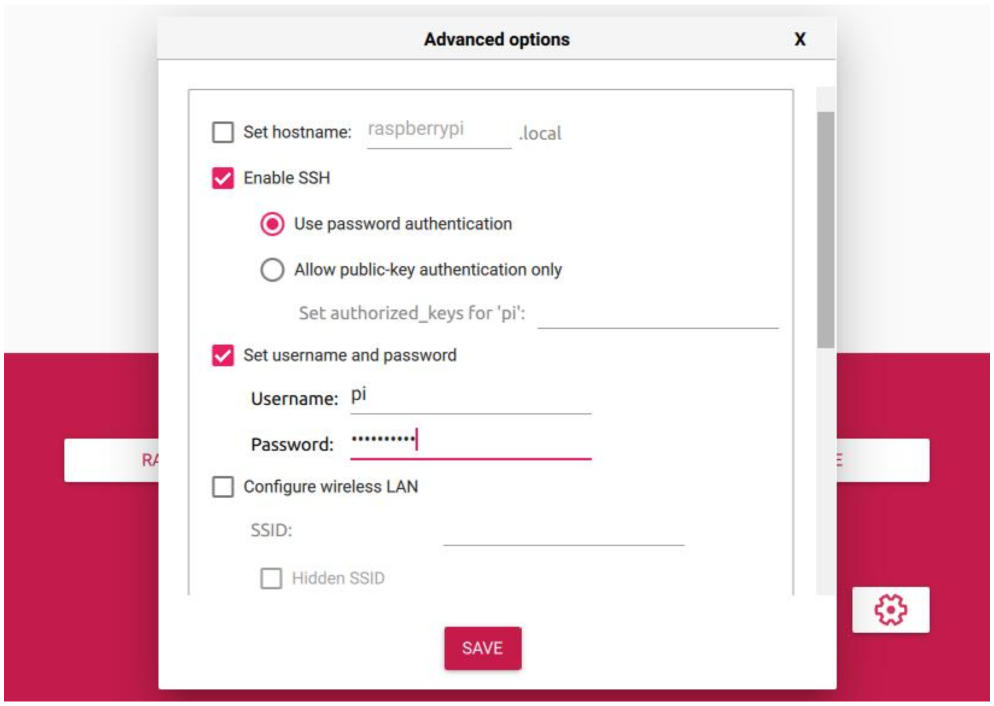
8. Then under the “Storage” option, select your MicroSD card:



9. Before clicking “WRITE,” you can set up some pre-installation settings by clicking the gear symbol in the bottom right:



10. I would recommend **enabling SSH**, as this will allow you to access your Raspberry Pi from your computer/laptop without the need for a separate desktop monitor, keyboard, and mouse (be sure to set a good password!):


Advanced options

- [ ] Set hostname: raspberrypi .local
- [x] Enable SSH
  - [✓ Use password authentication]
  - Allow public-key authentication only
  - Set authorized_keys for 'pi':
- [x] Set username and password
  - Username: pi
  - Password: ••••••••••
- [ ] Configure wireless LAN
  - SSID:
  - [ ] Hidden SSID

11. Save any changes you made in "Advanced Options" and click "WRITE." This process may take a good 30 minutes, so feel free to go grab a snack or play some table tennis.

Booting Up the Raspberry Pi

1. Once the Ubuntu image has finished installing onto your MicroSD card, go ahead and eject it from your computer in your computer's "Files" application (or equivalent) and then pull it out of the port. Insert the MicroSD card into the designated port on the Raspberry Pi.
2. Now to turn on the Pi, first plug in your monitor via the HDMI cable (into the HDMI port on the Pi) and then plug in your keyboard and mouse into any of the USB-A ports on the Pi. Finally, plug your power supply into a wall outlet and then connect it to the power port on the Pi (micro-USB for Raspberry Pi 3B+, USB-C for Raspberry Pi 4).
3. The bootup process should automatically begin.

Installing Necessary Software and Packages

Catkin Installation and Setup

1. **Catkin** is ROS' official build system, responsible for handling package distributions and cross-compiling. It comes installed with ROS, but if you need to get it on Ubuntu you can run this:

```
pi@raspberrypi:~ $ sudo apt-get install catkin
```

2. Next we need to **source our ROS environment.** This can be done with this:

```
pi@raspberrypi:~ $ source
/opt/ros/<YOUR_ROS_VERSION>/setup.bash
```

*Note:** be sure to replace `<YOUR_ROS_VERSION>` with the name of the ROS version you have installed (e.g. `melodic` for Melodic, `kinetic` for Kinetic, `noetic` for Noetic, etc.).

3. Now let's create and build a **catkin workspace** where we can run all our ROS processes down the road. First, create and enter a catkin directory like this:

```
pi@raspberrypi:~ $ cd
pi@raspberrypi:~ $ mkdir -p ~/catkin_ws/src
pi@raspberrypi:~ $ cd ~/catkin_ws/
```

4. Now **build** the workspace like this:

```
pi@raspberrypi:~/catkin_ws$ catkin_make
```

***Note:** if you get an error message after running this saying catkin isn't installed, it probably means you haven't sourced the environment properly, so check out *Step 2* above.

5. **** VERY IMPORTANT STEP ****
To avoid having to source environments every time you startup your computer, we're going to **permanently source all the necessary `setup.bash` files** (for ROS and catkin)

by adding the source commands to the **~/.profile** and **~/.bashrc** scripts, which you can do by running the following:

```
pi@raspberrypi:~ $ echo "source
/opt/ros/<YOUR_ROS_VERSION>/setup.bash" >> ~/.profile
pi@raspberrypi:~ $ echo "source
/opt/ros/<YOUR_ROS_VERSION>/setup.bash" >> ~/.bashrc
pi@raspberrypi:~ $ echo "source
~/catkin_ws/devel/setup.bash" >> ~/.profile
pi@raspberrypi:~ $ echo "source
~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

***Note:** be sure to replace `<YOUR_ROS_VERSION>` with the name of the ROS version you have installed (e.g. `melodic` for Melodic, `kinetic` for Kinetic, `noetic` for Noetic, etc.).

Installing ROS Onto the Raspberry Pi

1. We can install ROS straight from the terminal! Open up the **Terminal** application on your Raspberry Pi and run the following command to setup the ROS repository:

```
pi@raspberrypi:~ $ sudo sh -c ‘echo “deb
http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main”
> /etc/apt/sources.list.d/ros-latest.list’
```

2. Next we’ll need to add the official **ROS key** to authenticate the ROS installation process (and avoid hackers wanting to intercept your Raspberry Pi internet traffic). This key is universal for all ROS versions. To do this, run the following command:

```
pi@raspberrypi:~ $ sudo apt-key adv --keyserver
'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

3. Now we’ll want to **update** and **upgrade** the Raspberry Pi’s package information, which we can do with the following:

```
pi@raspberrypi:~ $ sudo apt-get update && sudo apt-get
upgrade
```

4. Finally, we can **install the full Desktop suite for ROS Melodic**:

```
pi@raspberrypi:~ $ sudo apt install ros-melodic-desktop-
full
```

Setting Up a ROS Environment

1. If you didn't already do so in Step 5 of the Catkin Installation and Setup section, **setup the "`.bashrc`" file** so that the ROS environment variables are automatically added to your bash session every time you boot up your Raspberry Pi:

```
pi@raspberrypi:~ $ echo "source
/opt/ros/melodic/setup.bash" >> ~/.bashrc
pi@raspberrypi:~ $ source ~/.bashrc
```

2. Now let's **install some of the package dependencies** needed for building ROS packages:

   a. For Python 3:

```
pi@raspberrypi:~ $ sudo apt-get install -y python3-rosdep
python3-rosinstall-generator python3-wstool python3-
rosinstall build-essential cmake
```

   b. For Python 2:

```
pi@raspberrypi:~ $ sudo apt-get install -y python-rosdep
python-rosinstall-generator python-wstool python-rosinstall
build-essential cmake
```

3. Finally, we're gonna **initialize `rosdep`**, which enables you to easily install a lot of system dependencies for ROS development:

```
pi@raspberrypi:~ $ sudo rosdep init
pi@raspberrypi:~ $ rosdep update
```

Installing Additional Necessary Packages

1. Now we'll install some **additional packages** that will be needed to run the robot successfully! First, let's install `pip`:

```
pi@raspberrypi:~ $ sudo apt-get install python-pip python3-
pip python-dev
```

2. Now using `pip`:

```
pi@raspberrypi:~ $ sudo -H pip install setuptools wheel
cmake picamera numpy
```

   a. **Tip**: if installing via `pip` like this gives an error, try the following method:

```
pi@raspberrypi:~ $ python3 -m pip install
setuptools wheel cmake picamera numpy
```

3. Next we'll install **OpenCV**, an awesome library for dealing with anything related to computer vision and image manipulation:

```
pi@raspberrypi:~ $ sudo apt install python-opencv python3-
opencv
```

   b. **Tip**: if installing via `apt` like this gives an error, try the following method:

```
pi@raspberrypi:~ $ python3 -m pip install python-
opencv python3-opencv
```

4. Finally, we'll install the ROS distribution for **CV Bridge**, a handy package for converting certain messages into variables that OpenCV knows how to read:

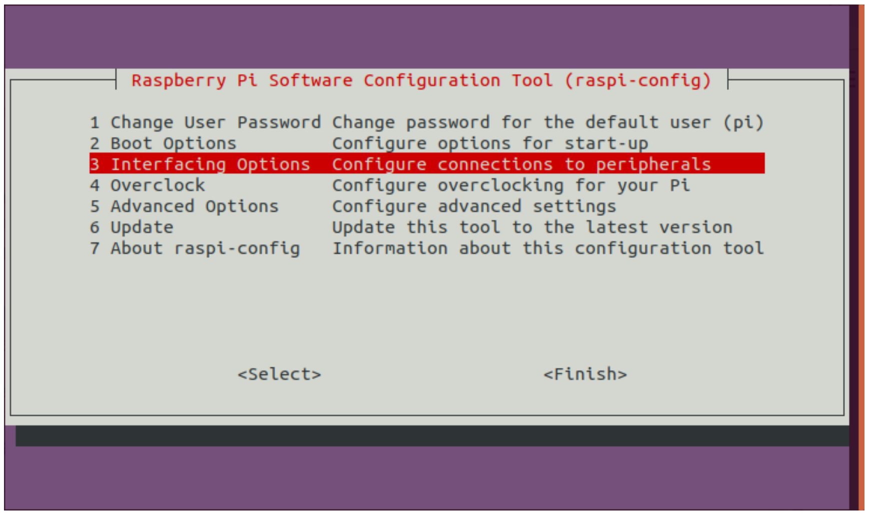
```
pi@raspberrypi:~ $ sudo apt install ros-melodic-cv-bridge
```

Testing the Raspberry Pi Camera

1. A crucial component of the Pheeno robot is its camera, as it allows it to analyze and pick out certain objects in its environment. Before doing this, however, let's test it out to make sure that it works! First, make sure your Raspberry Pi is ***turned off*** and plug in your camera into the Raspberry Pi via the ribbon cable. Once secured, boot up your Raspberry Pi as you normally would.
2. Next, we'll need to first **enable the camera on the Raspberry Pi.** Launch the `raspi-config` tool by doing:

```
pi@raspberrypi:~ $ sudo raspi-config
```

3. Use the arrow keys to go down to "Interfacing Options" and hit enter:

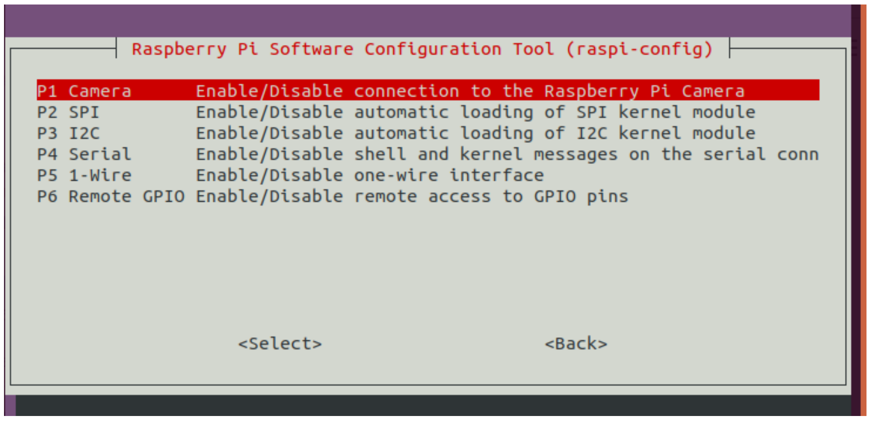


```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Change User Password Change password for the default user (pi)
2 Boot Options         Configure options for start-up
3 Interfacing Options  Configure connections to peripherals
4 Overclock            Configure overclocking for your Pi
5 Advanced Options     Configure advanced settings
6 Update               Update this tool to the latest version
7 About raspi-config   Information about this configuration tool

              <Select>                    <Finish>
```

4. On the next screen, hit enter on the "Camera" option:

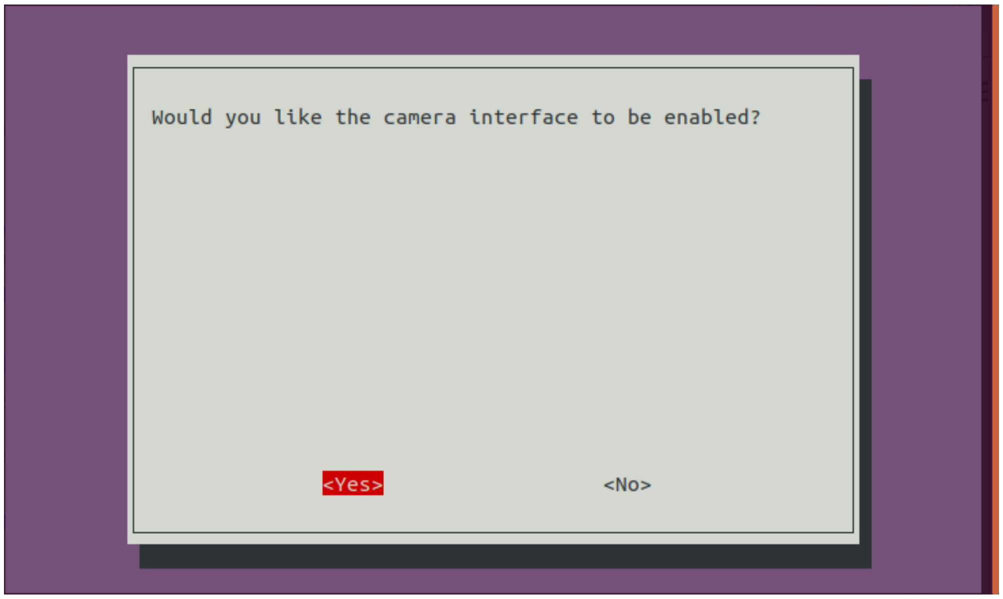


```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera      Enable/Disable connection to the Raspberry Pi Camera
P2 SPI         Enable/Disable automatic loading of SPI kernel module
P3 I2C         Enable/Disable automatic loading of I2C kernel module
P4 Serial      Enable/Disable shell and kernel messages on the serial conn
P5 1-Wire      Enable/Disable one-wire interface
P6 Remote GPIO Enable/Disable remote access to GPIO pins

              <Select>                    <Back>
```

5. On the next screen, when prompted "Would you like the camera interface to be enabled?," make sure "Yes" is highlighted and hit enter:



Would you like the camera interface to be enabled?

6. On the next screen, hit enter when "Ok" is highlighted and then press the right arrow key twice (to highlight "Finish" on the `raspi-config` home screen) and hit enter to exit out of `raspi-config`.
7. Another thing we'll need to add is a specific module to allow OpenCV to interface with the Raspberry Pi camera. I had this bug come up several times, and this fix worked wonders. Open the `modules.conf` file as root like so:

```
pi@raspberrypi:~ $ sudo nano /etc/modules-
load.d/modules.conf
```

8. Add the following as a new line in the file:

```
bcm2835-v4l2
```

9. Save the file and **reboot your Raspberry Pi.**
10. Now your camera is enabled! To test it, simply run the following:

```
pi@raspberrypi:~ $ raspivid -t 10000
```

This should display a live feed of what the camera sees on your monitor for 10 seconds. If you see this, then your camera is all set up and ready to go!

Arduino and Teensy Setup

1. Now let's get started with setting up the software needed to upload code to the Pheeno robot itself. First, you'll need to install Arduino onto the Raspberry Pi, which can be done here.

2. Now clone the `pheeno_arduino` repository in your home directory by doing:

```
pi@raspberrypi:~ $ cd ~
pi@raspberrypi:~ $ git clone
https://github.com/ACSLaboratory/pheeno_arduino.git
```

3. Next you'll need to install the pre-made Arduino libraries for the PheenoV2. To do this, copy the contents of the "`lib`" folder in the `pheeno_arduino` repo to the "`[PATH_TO_ARDUINO]/libraries/`" folder, where "`[PATH_TO_ARDUINO]`" is the path to the Arduino installation:

```
pi@raspberrypi:~ $ cd ~/pheeno_arduino/lib
pi@raspberrypi:~/pheeno_arduino/lib$ cp *
/[PATH_TO_Arduino]/libraries/
```

4. Now in order for ROS to work with Arduino, we'll need to install some ROS-related libraries so that the Raspberry Pi can interface with the Arduino through serial connection. For ROS Melodic, do the following:

```
pi@raspberrypi:~ $ sudo apt-get install \
  ros-melodic-rosserial ros-melodic-rosserial-client \
  ros-melodic-rosserial-python ros-melodic-rosserial-
arduino
```

5. Once installed, source the ROS setup.bash file in order to access these libraries:

```
pi@raspberrypi:~ $ source /opt/ros/melodic/setup.bash
```

6. To build these files, run the following:

```
pi@raspberrypi:~ $ cd ~/pheeno_arduino/lib/
```

```
pi@raspberrypi:~/pheeno_arduino/lib$ rosrun
rosserial_arduino make_libraries.py
```

7. Now to set up the Teensy Board! To do so, follow the instructions here. This will show you how to install and use a handy Arduino add-on for the Teensy called Teensyduino.
    a. **Tip:** Be sure to download the Linux udev rules file for the Teensy and copy it to your `/etc/udev/rules.d/` directory like so:

```
pi@raspberrypi:~ $ sudo cp 00-teensy.rules
/etc/udev/rules.d/
```

    I ran into the problem of the Teensy not running any of the Arduino code I was sending to it until I realized I had not downloaded this file!

8. Finally, let's try uploading a program to the robot! At this phase, we will just upload directly from Arduino (we will get to using ROS later!). To do so, first plug in the Teensy to your Raspberry Pi via a USB A to microUSB cable. A green light should begin to blink on the Teensy.
9. Launch the Arduino application and in the "Tools" menu at the top, navigate to the "Board" option and select "Teensy3.2".
10. Also in "Tools," go to the "Port" option and select the port that the Teensy is connected to.
11. Go to "File" and open the "`RandomWalkExample.ino`" file located in the "`~/pheeno_arduino/lib/PheenoV2Basic/examples/RandomWalkExample/`" folder.
12. Now press the button on the Teensy to launch the Teensyduino loader program.
13. Back in the Arduino IDE, verify the "`RandomWalkExample.ino`" code by clicking the checkmark button in the top left and once verified, upload it to the Teensy by clicking the arrow button to the right of the checkmark.
14. Once uploaded, press the button on the Teensy one more time.
15. **Make sure the robot is on a flat surface and won't fall off the edge of a table.**
16. Plug in the battery and flip the switches on.
17. Within a few seconds, your Pheeno should start driving around in random directions!
18. Try uploading the "`RandomWalkObstacleAvoidExample.ino`" example and watch as the Pheeno dodges obstacles in its path!

Pheeno ROS Setup

After installing ROS and catkin, let's set up the Pheeno ROS software.

Pheeno ROS Installation

1. First, **enter the catkin workspace** you created (outlined in the Catkin Installation and Setup section):

```
pi@raspberrypi:~ $ cd ~/catkin_ws/src
```

2. Next **clone the ACS Laboratory pheeno_ros repository:**

```
// For ROS Indigo
pi@raspberrypi:~/catkin_ws/src$ git clone -b indigo-devel
https://github.com/acslaboratory/pheeno_ros.git

// For ROS Kinetic
pi@raspberrypi:~/catkin_ws/src$ git clone -b kinetic-devel
https://github.com/acslaboratory/pheeno_ros.git

// For ROS Melodic
pi@raspberrypi:~/catkin_ws/src$ git clone -b melodic-devel
https://github.com/acslaboratory/pheeno_ros.git
```

3. Now let's get the `rosserial` package, which will be required for communication between the Raspberry Pi and the Teensy board:

```
// For ROS Indigo
pi@raspberrypi:~/catkin_ws/src$ git clone -b indigo-devel
https://github.com/ros-drivers/rosserial.git

// For ROS Kinetic
pi@raspberrypi:~/catkin_ws/src$ git clone -b jade-devel
https://github.com/ros-drivers/rosserial.git
```

```
// For ROS Melodic
pi@raspberrypi:~/catkin_ws/src$ git clone -b melodic-devel
https://github.com/ros-drivers/rosserial.git
```

4. We’ll also want the **`common_msgs`** package to get additional message types like Twist and action messages:

```
// For ROS Indigo
pi@raspberrypi:~/catkin_ws/src$ git clone -b indigo-devel
https://github.com/ros/common_msgs.git

// For ROS Kinetic
pi@raspberrypi:~/catkin_ws/src$ git clone -b jade-devel
https://github.com/ros/common_msgs.git

// For ROS Melodic –> We’ll use Noetic for common_msgs
pi@raspberrypi:~/catkin_ws/src$ git clone -b noetic-devel
https://github.com/ros/common_msgs.git
```

5. Next we’ll get the **`actionlib`** package:

```
// For ROS Indigo
pi@raspberrypi:~/catkin_ws/src$ git clone -b indigo-devel
https://github.com/ros/actionlib.git

// For ROS Kinetic
// NO Kinetic branch, however actionlib should come
// pre-installed so it shouldn’t be an issue

// For ROS Melodic
pi@raspberrypi:~/catkin_ws/src$ git clone -b melodic-devel
https://github.com/ros/actionlib.git
```

6. Now we’ll need the **`cv_bridge`** package to have some helpful functions for converting **`image`** messages in ROS to OpenCV **`Mat`** variables:

```
// For ROS Indigo
pi@raspberrypi:~/catkin_ws/src$ git clone -b indigo
https://github.com/ros-perception/vision_opencv.git

// For ROS Kinetic
pi@raspberrypi:~/catkin_ws/src$ git clone -b kinetic
https://github.com/ros-perception/vision_opencv.git

// For ROS Melodic
pi@raspberrypi:~/catkin_ws/src$ git clone -b melodic
https://github.com/ros-perception/vision_opencv.git
```

7. And last but not least, we'll need the all important **`pheeno_ros`** package:

```
// For ROS Indigo
pi@raspberrypi:~/catkin_ws/src$ git clone -b indigo-devel
https://github.com/acslaboratory/pheeno_ros.git

// For ROS Kinetic
pi@raspberrypi:~/catkin_ws/src$ git clone -b kinetic-devel
https://github.com/acslaboratory/pheeno_ros.git

// For ROS Melodic
pi@raspberrypi:~/catkin_ws/src$ git clone -b melodic-devel
https://github.com/acslaboratory/pheeno_ros.git
```

8. Now let's finally **install the packages** by building your catkin workspace and installing in your `~/catkin_ws/` directory:

```
pi@raspberrypi:~/catkin_ws/src$ cd ~/catkin_ws/
pi@raspberrypi:~/catkin_ws$ catkin_make -j2
pi@raspberrypi:~/catkin_ws$ catkin_make install -j2
```

Preparing the Raspberry Pi for Remote Connection

1. Now before we get to setting up the software on your master computer, we need to prepare the Raspberry Pi to be able to **accept remote ROS commands** from this master computer. The first step is something that will save you a lot of headaches down the road: setting up a terminal network and static IP address. There are a lot of steps for this, but trust me, it's worth it. The ACS Laboratory website has excellent instructions on how to do this in the "Setting Up the WiFi Connection" section here.
2. Next we need to **allow for remote unknown IP address connections**. To do this, run the following:

```
pi@raspberrypi:~ $ echo "export ROSLAUNCH_SSH_UNKNOWN=1" >>
~/.bashrc
pi@raspberrypi:~ $ source ~/.bashrc
```

3. We'll also want to **enable ssh** on the Raspberry Pi:

```
pi@raspberrypi:~ $ sudo systemctl enable ssh
pi@raspberrypi:~ $ sudo systemctl start ssh
```

4. Lastly, we'll need to **create an environment setup** file in the Catkin workspace on the Raspberry Pi so the master computer knows how to set up the Pi once it establishes a remote connection with it. To do this, go into your "`catkin_ws`" directory, create a file named "`env.sh,`" and open it like this:

```
pi@raspberrypi:~ $ cd ~/catkin_ws
pi@raspberrypi:~/catkin_ws$ touch env.sh
pi@raspberrypi:~/catkin_ws$ nano env.sh
```

5. Paste the following into the open "`env.sh`" file:

```
#!/bin/bash
export ROS_MASTER_URI=http://[YOUR_COMPUTER'S_IP]:11311
export ROS_IP=[YOUR_RASPBERRY_PI'S_IP]
export ROS_HOSTNAME=[YOUR_RASPBERRY_PI'S_IP]
source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
exec "$@"
```

Replacing `[YOUR_COMPUTER’S_IP]` and `[YOUR_RASPBERRY_PI’S_IP]` with the static IP addresses you created in Step 1.

6. Type Ctrl+X, then Y, and then Enter to save the file.
7. Finally, we want to make the file **executable**. To do this, run the following:

```
pi@raspberrypi:~/catkin_ws$ chmod +x env.sh
```

8. Congratulations! Now your Raspberry Pi is all good to go. Let’s head on over to your computer to set up the workstation software.

Chapter 3 Reflection and Next Steps

One area that I would like to delve deeper into is using a Raspberry Pi 3 versus a Raspberry Pi 4. The current Pheeno uses a Raspberry Pi 3, for which the standard Ubuntu 18.04 operating system and ROS Melodic software are supported. The installation of these for the Raspberry Pi 4 proved to be more challenging, and required me to test out different Ubuntu images just to find one compatible with the newer hardware of the Raspberry Pi 4. I was worried that using a different image might affect the performance of ROS on the Pi, however since I wouldn’t be running extremely memory intensive ROS software, this turned out not to be an issue.

Chapter 4: Workstation Software Installation and Setup Guide

Workstation ROS and Catkin Installation

Workstation ROS

1. The first and perhaps **most important** step in the initial installation process is to **check what version of ROS is already installed.** The ROS version will impact how we install and set up the rest of the software.
    a. **IF ROS IS NOT INSTALLED** you should refer back to previous Installing ROS Onto the Raspberry Pi section (the steps are exactly the same) or go to the ROS installation page. This project was run on **ROS Melodic Morenia for Ubuntu 18.04**. I would therefore recommend getting this version for your system (although ROS recommends Noetic Ninjemys for Ubuntu 20.04 and other newer operating systems, but I can't guarantee this project will work on newer versions).

Workstation Catkin

1. The installation and setup process for Catkin is **identical to the one we did earlier for the Raspberry Pi**. Follow the instructions in the Catkin Installation and Setup section in the previous chapter (except this time do them in the terminal on your master computer).

Pheeno ROS Setup + Pheeno Simulator Setup

The installation process for Pheeno ROS is the same as the one we did for the Raspberry Pi. However, this time we will also go through how to install and run the Pheeno ROS Simulator and test some programs virtually before deploying them to the robot.

Workstation Computer Pheeno ROS Installation

1. To install Pheeno ROS onto the workstation computer, following the same steps as in the Pheeno ROS Installation section in the previous chapter. Make sure to do it in your Catkin workspace!

Running the Pheeno ROS Simulator

1. The **Pheeno ROS Simulator** is a great way to test out code in a virtual environment before moving onto the real thing. This simulator relies on **Gazebo**, which comes fully integrated into the ROS Melodic, ROS Noetic, and ROS distributions, but we'll need some additional packages as well. Run the following to install them (replacing `melodic` with whatever your ROS version is):

```
$ sudo apt install ros-melodic-gazebo-ros-pkgs ros-melodic-
gazebo-ros-control
$ sudo apt install ros-melodic-urdf ros-melodic-xacro ros-
melodic-rviz
```

2. Next, in your "`~/catkin_ws/src`" directory, **clone the ACS Laboratory `pheeno_ros_sim` repository** using the following command:

```
// For ROS Indigo
~/catkin_ws/src$ git clone -b indigo-devel
https://github.com/acslaboratory/pheeno_ros_sim.git

// For ROS Kinetic
~/catkin_ws/src$ git clone -b kinetic-devel
https://github.com/acslaboratory/pheeno_ros_sim.git

// For ROS Melodic
~/catkin_ws/src$ git clone -b melodic-devel
https://github.com/acslaboratory/pheeno_ros_sim.git
```

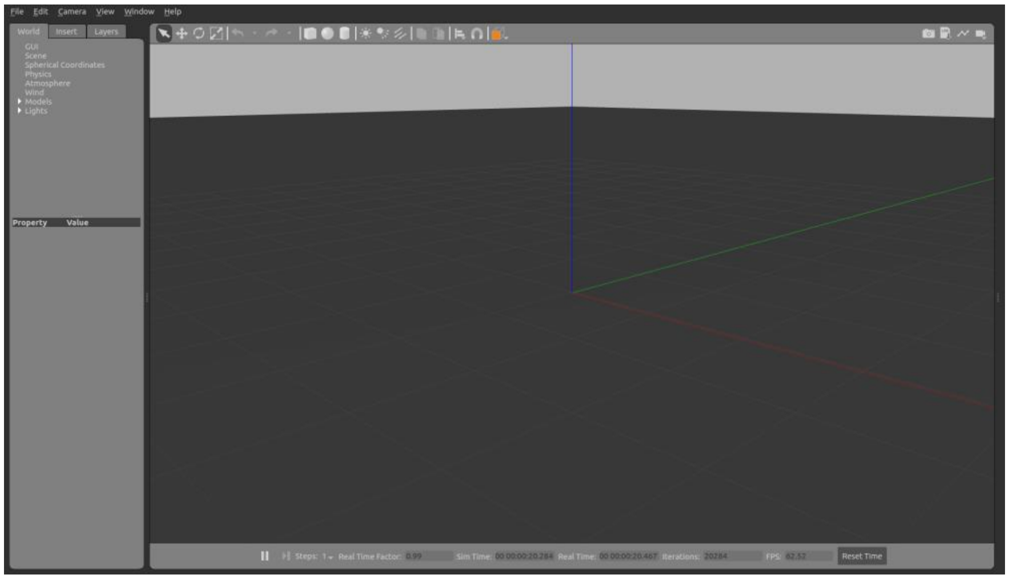
3. Now with **`pheeno_ros`**, **`pheeno_ros_sim`**, and all the relevant packages built and installed, let's try booting up the simulator! First, navigate to the **`~/catkin_ws/src/pheeno_ros_sim/launch`** directory:

```
$ cd ~/catkin_ws/src/pheeno_ros_sim/launch
```

4. Now we'll use **`roslaunch`** to launch the **`testbed.launch`** file:

```
~/catkin_ws/src/pheeno_ros_sim/launch$ roslaunch
testbed.launch
```

5. If all goes well, Gazebo should have been launched and you should see the following window open up:



IF YOU AN ERROR WHEN TRYING TO RUN STEP 2:

- This probably means the **`setup.bash`** file was not sourced. See **Step 5** in the Catkin Installation and Setup section.

6. If Step 5 above was successful, then you can try running more of the pre-installed **`.launch`** files in the **`~/catkin_ws/src/pheeno_ros_sim/launch`** directory! This can be done like this:

```
$ cd ~/catkin_ws/src/pheeno_ros_sim/launch
~/catkin_ws/src/pheeno_ros_sim/launch$ roslaunch
<NAME_OF_LAUNCH_FILE>
```

***Note:** the **`view_pheeno_rviz.launch`** and **`view_pheeno_with_camera_rviz.launch`** launch files might throw the following error when trying to launch them:

```
[ERROR] [1675883790.196822]: Could not find the GUI,
install the 'joint_state_publisher_gui' package
```

To fix this, simply install the **`joint-state-publisher-gui`** package:

```
$ sudo apt install ros–<YOUR_ROS_VERSION_NAME>-joint-state-publisher-gui
```

where `<YOUR_ROS_VERSION_NAME>` is replaced with `melodic`, `kinetic`, etc.

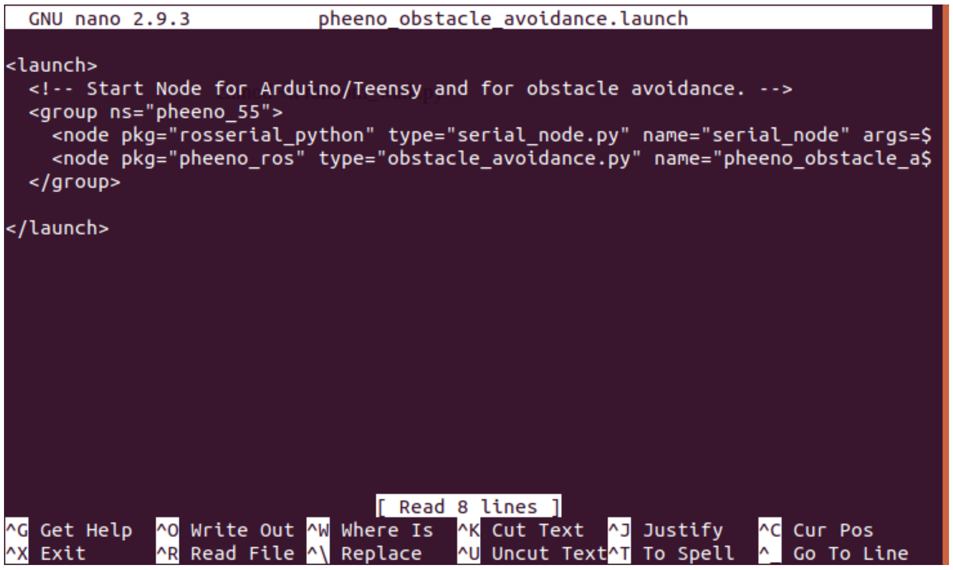
7. If you want to try making your own launch files, the ACS Laboratory website has a page on how to do that here.

Sending ROS Commands to the Robot

1. Now we're ready to start sending commands to the Pheeno robot remotely from your computer using ROS. Given that the Raspberry Pi and workstation software are set up correctly, this is actually a fairly simple process. First let's head into the "`~/catkin_ws/src/pheeno_ros/launch/`" directory.

```
$ cd ~/catkin_ws/src/pheeno_ros/launch/
```

2. In the directory you'll notice a few pre-existing launch files. Go ahead and open the "`pheeno_obstacle_avoidance.launch`" file using `nano` just to see what it looks like. You'll notice two nodes, one for running the serial connection between the Raspberry Pi and the Teensy3.2 board and the second for running the "`obstacle_avoidance.py`" program:



```
  GNU nano 2.9.3             pheeno_obstacle_avoidance.launch

<launch>
  <!-- Start Node for Arduino/Teensy and for obstacle avoidance. -->
  <group ns="pheeno_55">
    <node pkg="rosserial_python" type="serial_node.py" name="serial_node" args=$
    <node pkg="pheeno_ros" type="obstacle_avoidance.py" name="pheeno_obstacle_a$
  </group>

</launch>

                                [ Read 8 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text^T To Spell    ^_ Go To Line
```

3. We need a way to actually connect to the Pheeno's Raspberry Pi remotely, so we're going to create a new launch file in the same directory. Use `touch` to create a file called "`remote_pheeno_obstacle_avoidance.launch`" and use `nano` to fill it with the following:

```
<launch>
    <group ns="pheeno_41">
        <machine name="pheeno_41_remote"
             address="[YOUR_RASPBERRY_PI'S_IP_ADDRESS]"
             env-loader="~/catkin_ws/env.sh"
             user="[YOUR_RASPBERRY_PI'S_USERNAME]"
             password="[YOUR_RASPBERRY_PI'S_PASSWORD]"
             default="false">
        </machine>
        <node machine="pheeno_41_remote" name="serial_node"
              pkg="rosserial_python"
              type="serial_node.py" args="/dev/ttyACM0/">
        <node machine="pheeno_41_remote"
              name="pheeno_obstacle_avoidance"
              pkg="pheeno_ros" type="obstacle_avoidance.py"
              args="-n 41"/>
    </group>
</launch>
```

Be sure to replace the "`address`," "`user`," and "`password`" values with the ones specific to your Raspberry Pi. The "`pheeno_41`" name is also arbitrary; you can call it whatever you want! Just be sure to be consistent with the naming throughout the file.

4. All we've done here is add the "`machine`" argument to the launch file. What this does is it tells ROS to first connect to the Pheeno's Raspberry Pi and then launch the `serial` and `obstacle_avoidance` nodes on that Raspberry Pi.
5. Now before running the launch file, plug the battery into your Pheeno and flip the switches on. **Be sure the Pheeno won't drive off of an elevated surface or knock something over nearby.**
6. In the same launch directory, run the following:

```
~/catkin_ws/src/pheeno_ros/launch$ roslaunch
remote_pheeno_obstacle_avoidance.launch
```

7. If all goes well, your computer should connect with the Pheeno and the Pheeno should begin running its obstacle avoidance program! Play around with creating launch files like this and running different nodes/programs.

Chapter 4 Reflection and Next Steps

The success of the workstation software working to connect to the Pheeno remotely via ROS was heavily dependent on the Raspberry Pi software functioning properly. One inconvenience I encountered during this process was that I had not set up a static IP address for the Raspberry Pi, so every time the IP address changed, I had to go back into the launch files and manually edit the address whenever I wanted to run ROS remotely from my computer. This just goes to show the importance of Step 1 in Preparing the Raspberry Pi for Remote Connection, something I had totally neglected in my first run through of the setup process.

The latest ROS version that the workstation software currently supports is ROS Melodic, which was released in 2018 and has an end-of-life date of May 2023 (the time period this report was submitted). I had to go through the installation process of Melodic myself in order to know what worked and what didn't for the sake of preparing this guide.

Chapter 5: Pheeno Markov Chain Experiment

Introduction

I was able to implement this Pheeno Markov Chain Experiment from (Deshmukh et al., 2018), which is an approach for controlling a robot swarm using only the information that each individual robot can gather from its local environment. Although intended for a group of multiple robots, I successfully deployed this algorithm to the single robot I had developed as a proof of concept that this decentralized Markov chain approach was functional. I managed to continuously receive information from the Pheeno about its state and could read its relative position in a field of view from an overhead camera.

What You'll Need

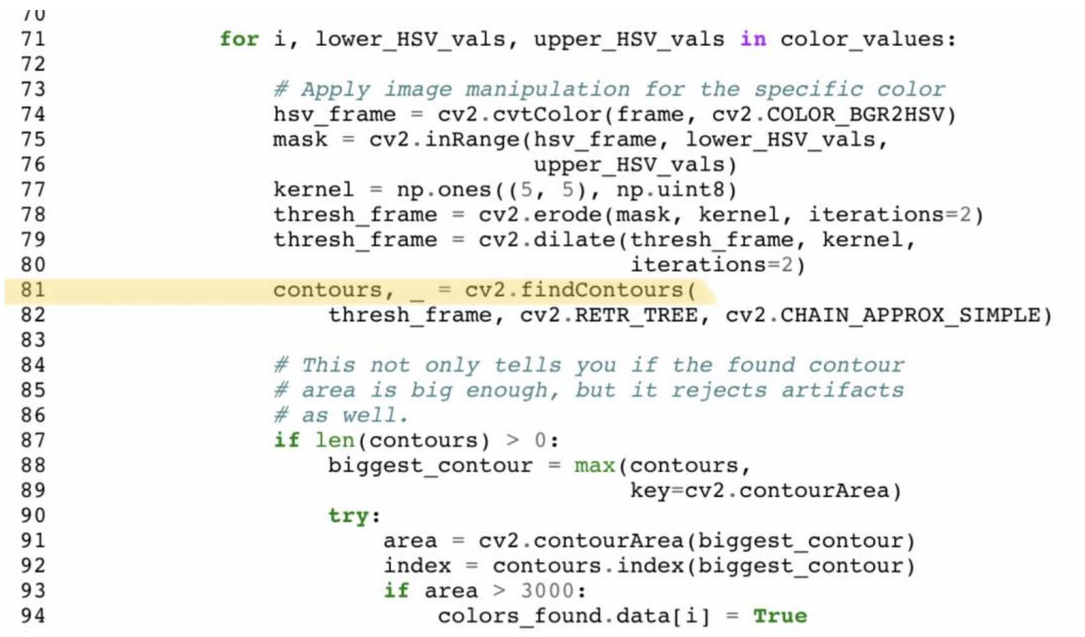
- A completed Pheeno robot with all the software outlined in the previous chapters.
- A computer with all the software described in Chapter 4: Workstation Software Installation and Setup Guide.
- A USB camera to plug into your workstation computer (this will be the overhead camera for the experiment), like this one.
- Some yellow paper or yellow tape.

Setup and Installation

1. The first step will be to **download the Pheeno Markov chain software**. On your computer, download the "pheeno_markov_chain-master" folder from here and extract it to your "`~/catkin_ws/src/`" directory. You should now have the folder "~/`catkin_ws/src/pheeno_markov_chain`" on your computer.
2. Now before we build the software, **there are a few errors that I encountered in this process** that took me some time to solve. To save you from having to debug these yourself, here is what you can do preemptively:
    a. Open the "pheeno_cam.py" file by doing:

```
$ nano
~/catkin_ws/src/pheeno_markov_chain/scripts/pheeno_cam.py
```

b. Scroll down to the line containing “**`contours, _ = cv2.findContours(`**“, as highlighted below (should be line 81):



```
70
71              for i, lower_HSV_vals, upper_HSV_vals in color_values:
72
73                  # Apply image manipulation for the specific color
74                  hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
75                  mask = cv2.inRange(hsv_frame, lower_HSV_vals,
76                                     upper_HSV_vals)
77                  kernel = np.ones((5, 5), np.uint8)
78                  thresh_frame = cv2.erode(mask, kernel, iterations=2)
79                  thresh_frame = cv2.dilate(thresh_frame, kernel,
80                                            iterations=2)
81                  contours, _ = cv2.findContours(
82                      thresh_frame, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
83
84                  # This not only tells you if the found contour
85                  # area is big enough, but it rejects artifacts
86                  # as well.
87                  if len(contours) > 0:
88                      biggest_contour = max(contours,
89                                            key=cv2.contourArea)
90                      try:
91                          area = cv2.contourArea(biggest_contour)
92                          index = contours.index(biggest_contour)
93                          if area > 3000:
94                              colors_found.data[i] = True
```

c. Add **“`_,`** “ to the beginning of this line so it becomes “**`_, contours, _ = cv2.findContours(`**“. In other words:

Change this:

```
contours, _ = cv2.findContours(
```

To this:

```
_, contours, _ = cv2.findContours(
```

d. Save the file.

e. The reason for this edit is due to an error I was getting that said “ValueError: too many values to unpack” when calling cv2.findContours(). Adding “_, “ to this line silences this error because it assigns the extra values needing to be unpacked to “_”.

3. Now it's time to build and install the project using Catkin. Navigate to the "`~/catkin_ws/`" directory and run a "`catkin_make`" and then a "`catkin_make install`," like so:

```
$ cd ~/catkin_ws/
~/catkin_ws$ catkin_make -j4
~/catkin_ws$ catkin_make install -j4
```

4. Now one error that I ran into and you likely will too has to do with the "`Controller.h`" file that is created after this installation. Here is the changes you'll need to make to fix this error:
    a. Open the "`Controller.h`" file like this:

```
$ cd ~/catkin_ws/devel/include/pheeno_markov_chain/
~/catkin_ws/devel/include/pheeno_markov_chain$ nano
Controller.h
```

    b. Change the <> to "" in the `#include <pheeno_markov_chain/ControllerRequest.h>` and `#include <pheeno_markov_chain/ControllerResponse.h>` lines, like this:

    Change this:

```
#include <pheeno_markov_chain/ControllerRequest.h>
#include <pheeno_markov_chain/ControllerResponse.h>
```

    To this:

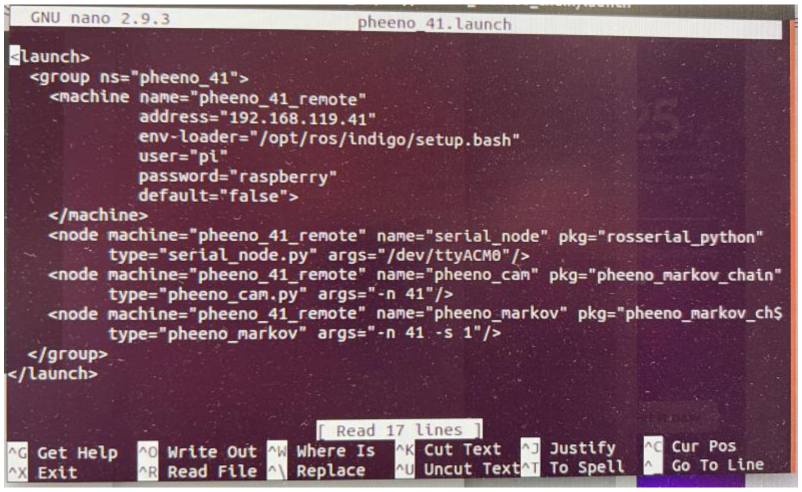
```
#include "pheeno_markov_chain/ControllerRequest.h"
#include "pheeno_markov_chain/ControllerResponse.h"
```

    c. Finally, build and install again by doing:

```
~/catkin_ws/devel/include/pheeno_markov_chain$ cd
~/catkin_ws/
~/catkin_ws$ catkin_make -j4
~/catkin_ws$ catkin_make install -j4
```

Deploying to the Pheeno Robot

1. Now let's go into how to go about actually running the Pheeno Markov chain with a robot. To do so, we are going to adjust the existing launch files in the pheeno_markov_chain repository. Navigate to the "`~/catkin_ws/src/pheeno_markov_chain/launch/`" directory and open the "`pheeno_41.launch`" file. You should see this:



```
  GNU nano 2.9.3                    pheeno_41.launch

<launch>
  <group ns="pheeno_41">
    <machine name="pheeno_41_remote"
             address="192.168.119.41"
             env-loader="/opt/ros/indigo/setup.bash"
             user="pi"
             password="raspberry"
             default="false">
    </machine>
    <node machine="pheeno_41_remote" name="serial_node" pkg="rosserial_python"
          type="serial_node.py" args="/dev/ttyACM0"/>
    <node machine="pheeno_41_remote" name="pheeno_cam" pkg="pheeno_markov_chain"
          type="pheeno_cam.py" args="-n 41"/>
    <node machine="pheeno_41_remote" name="pheeno_markov" pkg="pheeno_markov_ch$
          type="pheeno_markov" args="-n 41 -s 1"/>
  </group>
</launch>

                              [ Read 17 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

2. Now go ahead and edit the following in the file:
   a. Change the "`<group ns="pheeno_41">`" value to be whatever you would like to identify the Pheeno with (could be "`pheeno_1`", "`pheeno_67`", etc.).
      i. **Tip:** One thing you could do would be to identify the Pheeno by the last digits of its static IP address that you set up in Step 1 of the Preparing the Raspberry Pi for Remote Connection section. In the image above, you can see that the Pheeno was named "pheeno_41" since its IP address ends in "41."
   b. Change the "`<machine name="pheeno_41_remote">`" accordingly.
   c. **Important:** change the "`address="192.168.119.41"`" to the **static IP address** you set up for your Pheeno robot.
   d. Change the "`env-loader`" to be the "`env.sh`" file we created in the "`~/catkin_ws/`" directory (so it would be "`env-loader="~/catkin_ws/env.sh"`").
   e. Change "`user`" and "`password`" to the username and password of your Raspberry Pi.
   f. Save the file.

3. Next we need to **set up the overhead camera**. Plug in the camera to a USB port on your computer and secure it at least two meters above the surface upon which the Pheeno will be driving. Make sure the camera is pointing directly downwards.
4. Now **attach the yellow paper or yellow tape to the top of the Pheeno** so that it **covers most of the top** of the Pheeno. This is so that the overhead camera can recognize the Pheeno using OpenCV (yellow is the default color, but this can be easily changed in the `view_pheenos.py` code).
5. Now we're just about ready to deploy! Plug the battery into the Pheeno and flip the switches on.
6. In terminal on your computer, run the following to launch the launch file you just edited:

```
$ cd ~/catkin_ws/src/pheeno_markov_chain/launch/
~/catkin_ws/src/pheeno_markov_chain/launch$ roslaunch
pheeno_41.launch
```

7. Now with this running, run the following to initialize the Markov chain experiment:

```
~/catkin_ws/src/pheeno_markov_chain/launch$ roslaunch
markov_chain_experiment.launch
```

8. If all goes well, you should see a live feed of the overhead view from the camera on your computer, with a centroid tracking the Pheeno robot as it moves. Congratulations! The Pheeno Markov chain has been successfully deployed!

Chapter 5 Reflection and Next Steps

This setup and installation guide is a must for anyone trying to use this swarm algorithm, even if they are an expert on ROS. I discovered a handful of bugs and installation errors along the way, some of which took days to pinpoint and resolve. I therefore hope that these instructions spare future users from these impediments and allow them to spend more time on development and testing! Ultimately, I was able to successfully install this software, and as a proof of concept for implementing the Pheeno Markov chain experiment, this attempt was quite effective.

Chapter 6: Results and Final Reflection

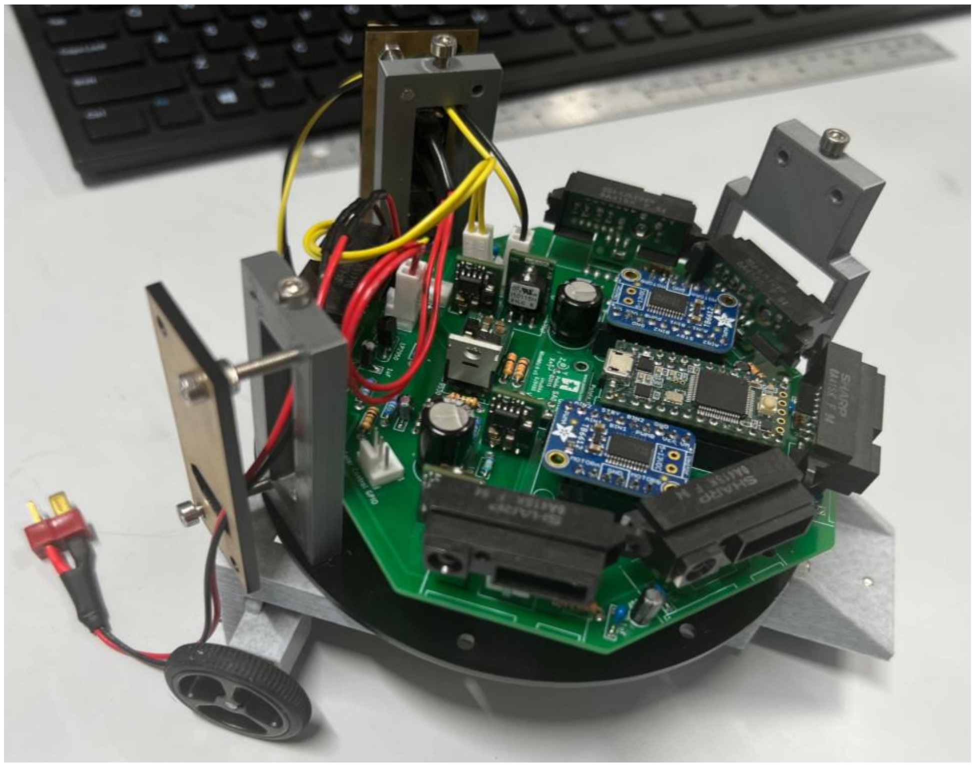
Results

After months of research and development, I ended up with a fully functioning Pheeno robot system, as well as additional ROS software that could be deployed to the robot to test programs such as obstacle avoidance and decentralized Markov chains. Additionally, I was also able to run simulations of the Pheeno programs using Gazebo, and could test with as many robots as needed.

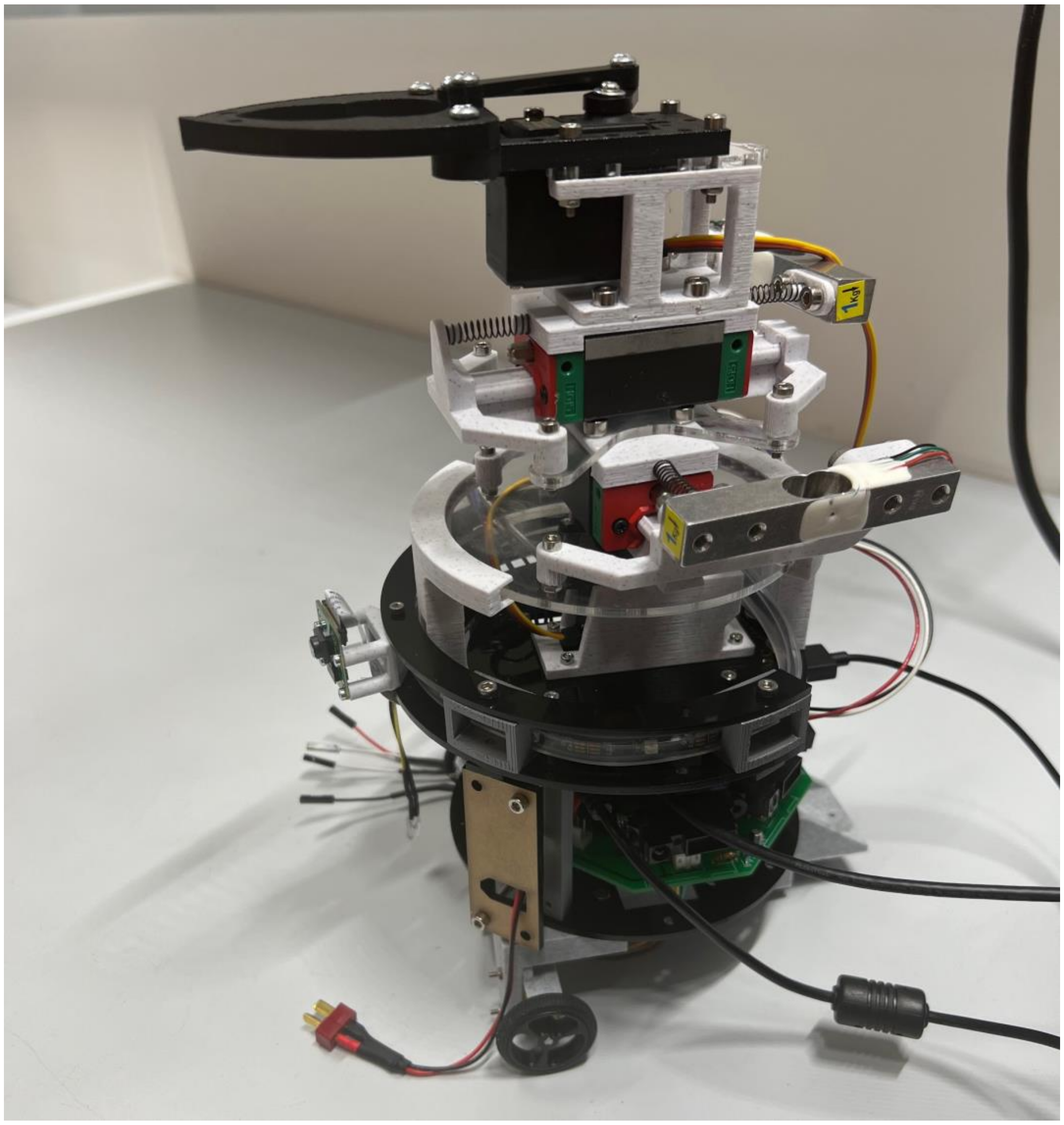
To document my results, I took several videos of both the Gazebo simulations and of course the robot in action:

- This video demonstrates the Single Pheeno robot simulation.
- Here is a Multiple Pheeno robot simulation.
- This video shows the Pheeno running a Random walk with obstacle avoidance program that was deployed using ROS from the workstation computer.

Here is a photo of the completed Pheeno without the gripper module:

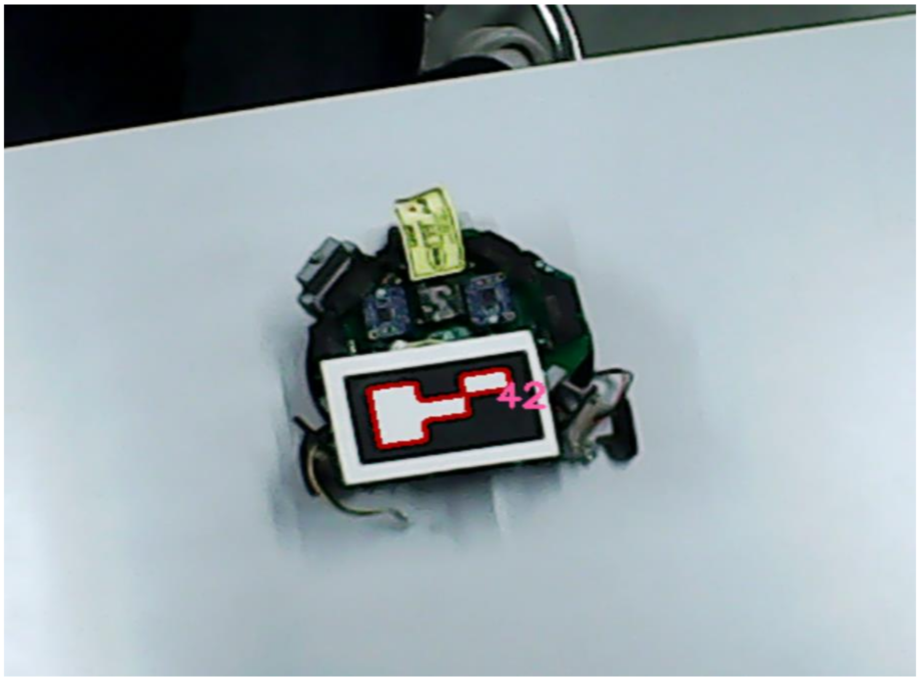


Here is a photo of the Pheeno with the gripper module mounted on top:



Future Work

Another experiment I started to run and would want to continue looking into is tracking the Pheeno robot's position, movement, and orientation using fiducial codes. A fiducial marking—similar to a QR code—is printed out onto a piece of paper and attached to the top of the Pheeno, and can be read using the overhead camera. The fiducial is generated using a Python program, and represents a unique two-digit number. Below is a screenshot of the view from the overhead camera as it reads the fiducial for the number "42":



This was my initial test of this program and I have yet to explore how to get position and orientation data based on the fiducial reading. In contrast with the Pheeno Markov chain experiment, which is decentralized since each Pheeno reports its own data, this would be a more centralized program, as the workstation computer is reading the data from all of the Pheenos at once using the overhead camera. This would be an excellent test to see which method is more effective in terms of swarm control and manipulation, but would of course require building some additional Pheeno robots. The code for this fiducial approach can be found here.

The last area for future development is the gripper module. For this report, I decided to focus more on the base and drivetrain assembly as these are most essential for swarm testing. Software for the gripper module does exist on the ACS Laboratory website, but this is for a different gripper design than the one on the current Pheeno robot. The current gripper was developed by a previous student, and consists of a claw controlled by a servo motor which rests on a platform that can rotate using a second servo motor. The claw also sits on a rail system attached to force sensors so that when the gripper is pushing or pulling on something in a given direction, the force sensors can read in these push-pull forces as data. With multiple robots, the idea is that this data is then communicated to the rest of the swarm to determine its overall movement. Because of the need for more than one Pheeno robot to perform this experiment, I chose to wait to report on the gripper in more detail until additional robots could be built.

Final Reflection

As I made my way through the development process of the Pheeno robot, I encountered challenges, setbacks, and unexpected errors, both on the hardware side and the software side. I had to find workarounds to some of these obstacles in order to continue onto the next step. One such obstacle was very early on when I was trying to set up a Raspberry Pi 4 Model B with the necessary software for the Pheeno system. The Pheeno software reliably runs only on Ubuntu 18.04, as this is the supported operating system for ROS Melodic (the latest ROS version that works for the Pheeno). The issue is that Ubuntu 18.04 is too old for the Raspberry Pi 4 models, so when trying to boot up the Pi with Ubuntu 18.04 installed, it gives an error saying that newer software is needed in order for the Pi to run. My “workaround” required a lot of trial and error of different Ubuntu images until I found one that worked for the Pi. I discovered that this specific image was not ideal for running ROS, however given that not too much processing was required for the Pheeno code, it ended up functioning just fine.

Although these challenges were time consuming, they led me to uncover many different tips and tricks that could be used to avoid them, or at least minimize their time consumption. This was especially true for the PCB assembly process, where coming up with clever methods of soldering and constructing components made the build procedure much smoother. These kinds of design tricks were part of the entire motivation for creating an instruction manual like this one, and actually proved to be rather fun to discover.

Bibliography

Vaibhav Deshmukh, Karthik Elamvazhuthi, Shiba Biswal, Zahi Kakish, and Spring Berman. “Mean-Field Stabilization of Markov Chain Models for Robotic Swarms: Computational Approaches and Experimental Results.” *IEEE Robotics and Automation Letters (RA-L)*, 3(3):1985-1992, 2018.

Sean Wilson, Ruben Gameros, Michael Sheely, Matthew Lin, Kathryn Dover, Robert Gevorkyan, Matt Haberland, Andrea Bertozzi, and Spring Berman. “Pheeno, A Versatile Swarm Robotic Research and Education Platform.” *IEEE Robotics and Automation Letters (RA-L)*, 1(2):884-891, 2016.