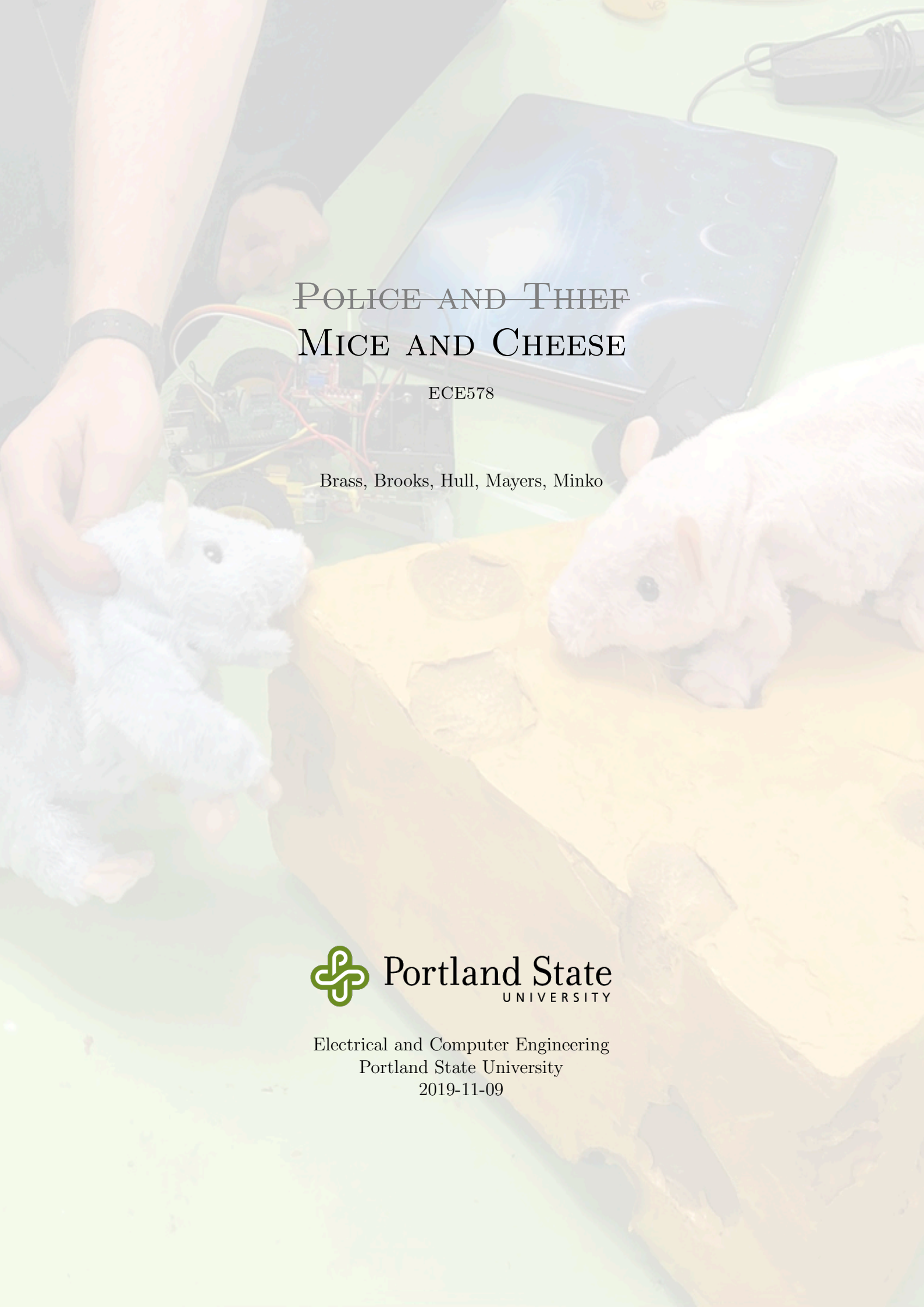

~~Police and Thief~~
Mice and Cheese

ECE578

Brass, Brooks, Hull, Mayers, Minko

Electrical and Computer Engineering
Portland State University
2019-11-09

Contents

1 Overview

As a new design scheme, we changed the *Policemen and Thief* to the less violent, *Mice and Cheese*.

Design a game with 3 "Viking Bots": two dressed as mice and the third as a wedge of cheese. The goal of the game is to keep the cheese away from the hungry maws of the two little mice. The game uses object recognition to find the location of each game piece on the board. The program builds a game board and determines the proper strategy so that the agents can move effectively in the right direction. The user goes first and can control the course of the cheese, and the mice will continuously move towards the cheese after each turn. Will the cheese escape the mice? Or will it get eaten? Tune in and find out next time on Perkowskis comedy cartoon hour.

2 Hardware

The hardware for our project consists of 3 robots and all their associated parts. In the past, this project utilized two Viking Bots and a single, more massive hexapod robot. Our requirements for the project were to replace the hexapod robot with another Viking Bot and to improve the robustness of the hardware. This would ensure that our hardware can run more consistently, at a faster pace, and in more diverse conditions than were possible in previous implementations of the project.

2.1 Goals

1. Purchase and assemble a new Viking Bot
2. Assess what hardware was left over from previous implementations of project
3. Upgrade and standardize battery packs for all robots
4. Improve wiring reliability and cable management
5. Ensure that the robots can run consistently at top speed for fast gameplay

2.2 Design

We assembled our projects robot cars from an inexpensive kit, which was quick to assemble. The Viking Bot robot kit consists of the following parts:

- An acrylic sheet with mounting holes and cutouts for wires
- A set of two DC motors and wheels
- A swivel caster for a back wheel
- A battery pack (AA battery size)
- An L298N H-Bridge Module
- A Raspberry Pi microprocessor board

Due to the nature of using a "kit" robot, we didnt have a lot of latitude to design the hardware we were using. However, we undertook the development of a better battery and power management system for the robots after discovering the following problems:

- Batteries we inherited had differing voltages, causing robots to function differently from one another
- Some robots used multiple battery packs to achieve uniform voltages, adding extra weight to the robot
- The V_{IN} port powered the Raspberry Pi's, causing damage to the boards.
- H-Bridge modules were providing inconsistent outputs and current limiting the Raspberry Pi's

To solve these problems, we designed a rechargeable battery and power system using three 18650 batteries that provided a stable voltage for the motors and Raspberry Pi. This design increased our run time, avoided damage to the Raspberry Pi's, and was easier to implement across all three robots.

2.3 Implementation

Initially, we were told by a member of the previous team that their principal concern was using non-identical battery packs. This irregularity caused one of the robots to turn more quickly than the other. Also, we knew that we were replacing the hexapod robot with a Viking Bot, which required ordering a new robot kit and assembling it.

2.3.1 Assess Project Hardware

Our first task was to assess what hardware the previous team left. Unfortunately, the robots we found in the old project locker were in a sad state. The locker included all sorts of batteries, servos, screws, electronics, and other parts in a large pile. The two Viking Bot robots we inherited had loose jumper wires and tape all over them. There were three types of battery packs in the locker for the robots: a USB pack (too low voltage for constant use), AA battery packs (not rechargeable), and a sizable Li-Po battery pack (no charger available). We were not able to test any of the hardware since we did not have a battery of the correct voltage that we could recharge. These irregularities left us in the position of needing to correct the power issues immediately so we could test the robots. In order to correct the mess in the locker and to avoid losing vital parts, we cleaned out all the components. We organized them by type into labeled boxes. This organization has significantly improved the storage locker and made the project run more smoothly.

Figure 1: Organized locker

Figure 2: Battery and servo parts

2.3.2 Battery and Power System

After learning that the Viking Bots use the L298N H-Bridge module for powering the motors and the Raspberry Pi, we were able to determine that we needed to be able to provide close to 12V for the system. Additionally, the robots were set up to use the 5V output from the H-Bridge module to power the Raspberry Pi. However, after inspection of the L298N schematic,

we discovered two problems with this setup:

- The L298N requires a separate 5V power supply if supplying the module with more than 12V in order to protect the H-Bridge chip.
- The 5V onboard voltage regulator for 5V output (L78M05) can only supply up to 0.5A of current on the output.

Therefore, using a battery pack that is greater than 12V requires a different power supply for obtaining 5V. Initially, this was our only consideration, and we purchased rechargeable batteries in the 18650 size for the system. We selected these batteries their high mAh ratings and high current output ability. Then, we purchased battery holders, batteries for all the robots, and a battery charger. Using three batteries in series produces $3.7V \cdot 3 = 11.1V$ nominal voltage for the system. However, when fully charged, these batteries can reach 4.2V and provide 12.6V to the system, potentially damaging the onboard voltage regulator and the Raspberry Pi.

In addition, the Raspberry Pi requires up to 2.5A for peak power. This requirement is 2A over the maximum available from the onboard voltage regulator supply. Due to this rating, we needed to find a different solution for powering the Raspberry Pi in order to provide a stable voltage with enough amperage to function safely.

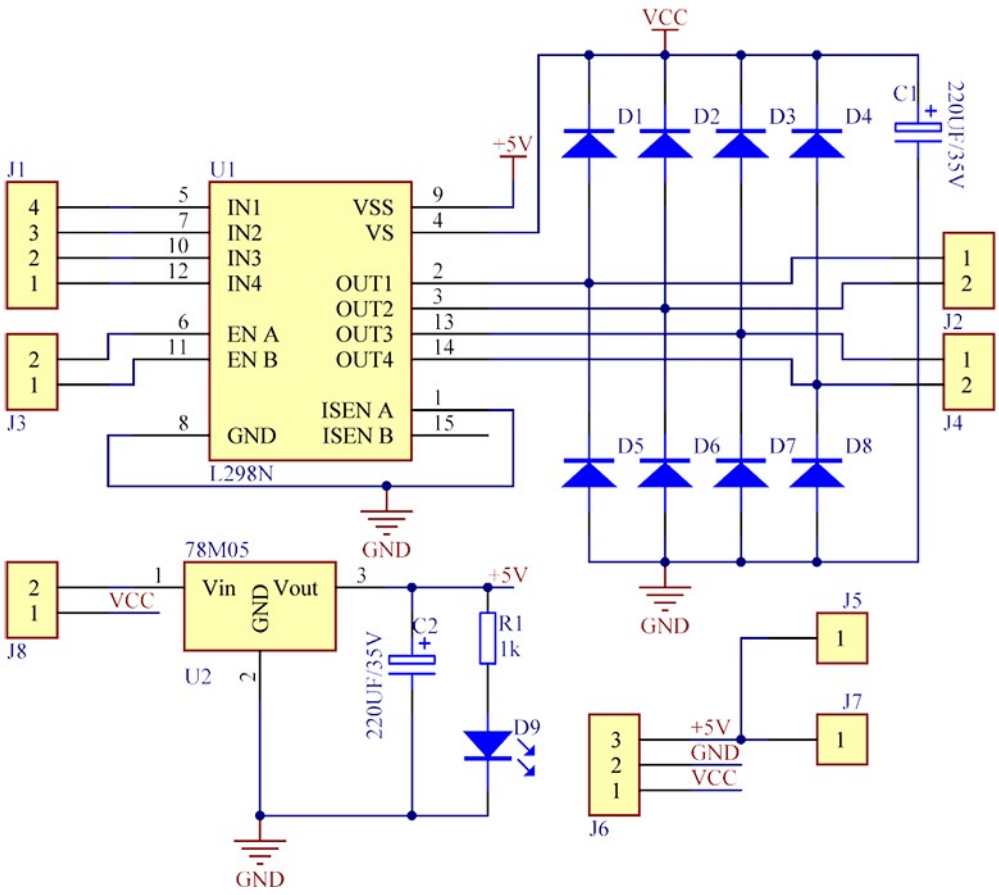

Figure 3: Schematic

Our solution was to add a separate voltage regulator that adjusts to the correct voltage for the Raspberry Pi and provides enough amperage for it to run without brownouts or damage to the board. We chose the DROK adjustable voltage regulator for this task as we could tune them to precisely 5V with the potentiometer on the underside, and they can handle up to 3A (0.5A over the Raspberry Pis max requirements).

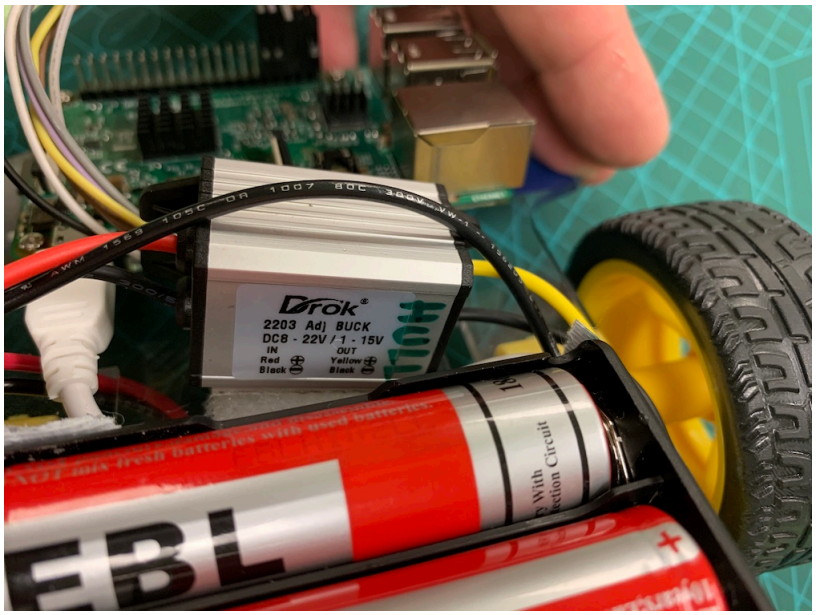

Figure 4: Voltage regulator

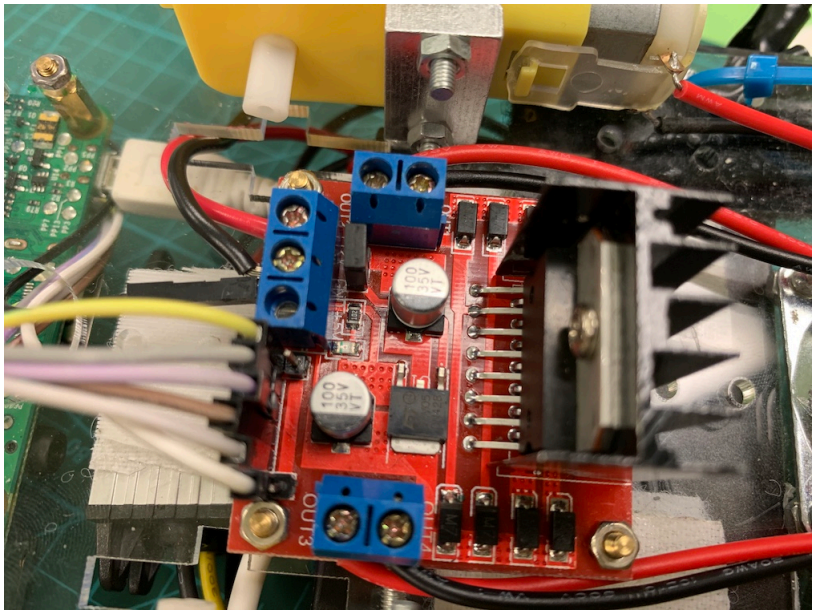

Figure 5: H-Bridge module

2.3.3 Building the Third Viking Bot

Assembling the robot kit was very straight forward. The kit contained instructions and required only a screwdriver to assemble. Building a new Viking Bot from scratch allowed us to understand better how to make improvements to the wiring and set up to make the robot more robust.

Figure 6: Robot kit fully assembled with H-Bridge module

We standardized the setup of the robots, putting the battery pack over the back swivel wheel for better weight distribution. We mounted the H-Bridge between the motors underneath the deck, the Raspberry Pi at the front, and the voltage regulator between it and the battery pack. We also attached a power switch for each robot to save on battery life. Finally, we used shrink wrap for solder connections to avoid short circuits. Then we zip-tied wires to the chassis paying close attention to keeping them out of the way. We mounted major components using velcro for ease of access. Later, we added each robots IP address to the bottom for easy identification.

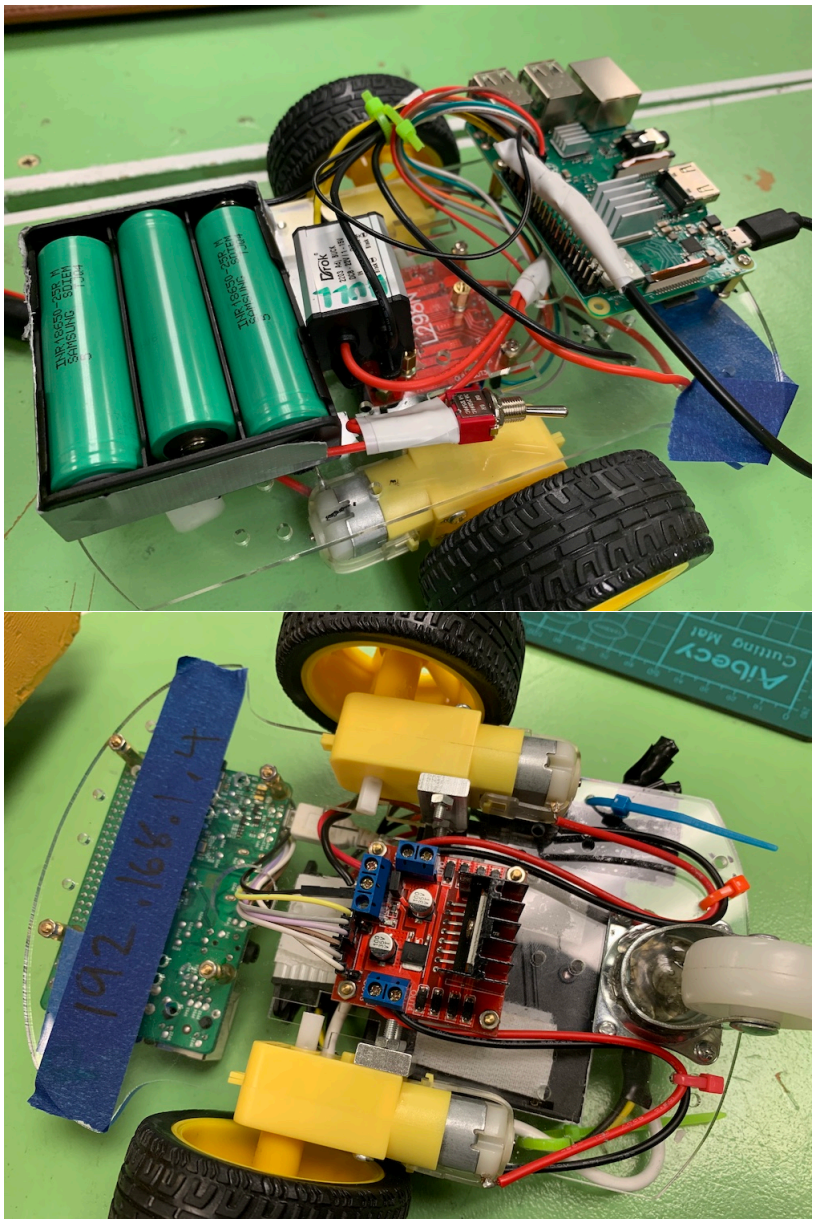

Figure 7: Finished robot with robust wiring and updated power system

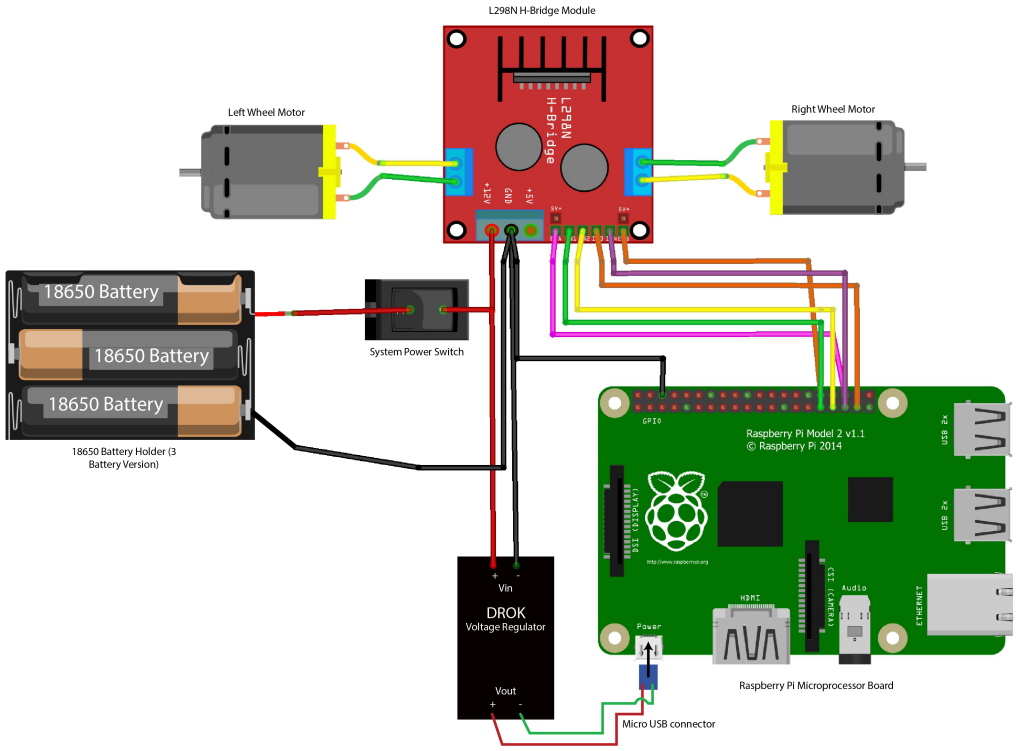

Figure 8: Final wiring diagram for Viking Bots

2.3.4 Fixing damaged Raspberry Pi's

When our battery packs first arrived, we were still attempting to utilize the 5V regulator on the H-Bridge modules. However, when we attempted to power on the robots, we discovered that one of the Raspberry Pis we inherited was not powering on. We then tried using our own Raspberry Pi in the system, and it immediately started smoking and was damaged.

After investigating the cause of the issue, we determined that the 5V regulator on the H-Bridge module was damaged. This regulator was causing some increased current draw or voltage spike that fed into the Raspberry Pi. Since the system was set up to feed power to the Pi through the VIN pin, there was no protection on the circuit, and it fried the diode that typically protects the Raspberry Pi. We assume that the H-Bridges voltage regulator became damaged from being supplied with more than 12V as the system under test had been using the higher-rated Li-Po battery during the previous iteration.

This situation was what prompted us to switch to a dedicated voltage regulator for the Pi, but it left us without a control board for the robots.

After inspecting the other non-working pi we inherited, we saw that it too had damage to the diode. Thankfully we were able to order more diodes and replace them. This diode reinstallation fixed the boards and allowed us to use them for the robots. However, we still had an issue with using the VIN pin as it was not protected.

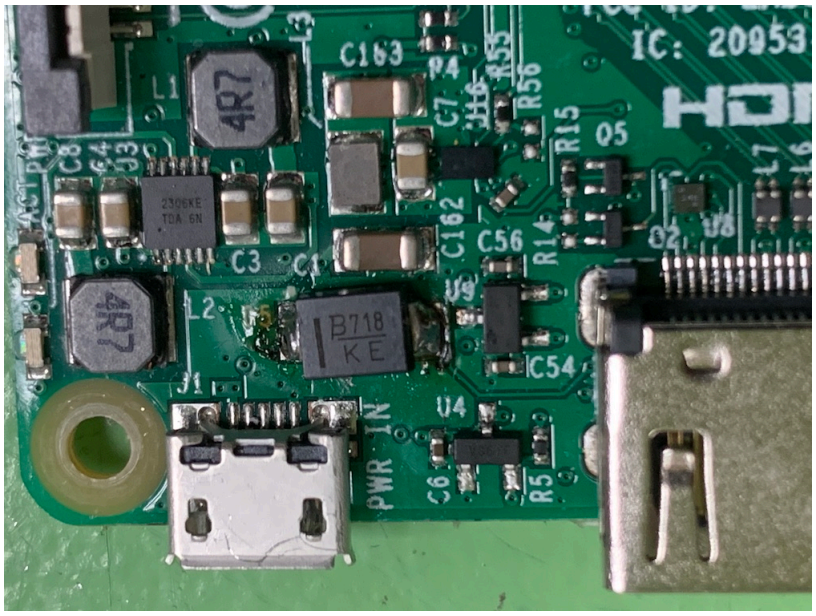

Figure 9: Diode replaced just above the Micro USB port

We cannibalized three micro USB cables to get power from our new voltage regulators and send it into the power port of the Raspberry Pi. To do this, we cut and stripped the USB cable and found the V+ and V- wires with a multimeter. After finding these, we wired them to the correct lines on the voltage regulator output, then plugged the micro-USB into the Raspberry Pi power port. This solution gave us multiple layers of protection against more voltage issues.

2.3.5 Building an Enclosure for the Robots

During the project, we realized the need for an additional task in the hardware category. One of our robots needed an enclosure in order for our cameras to differentiate it from the others during training. We decided that the project would transition from police chasing a thief to rats chasing cheese. So we needed to develop and cheese-like enclosure that was light enough for the Viking Bot to be still able to move quickly.

Our initial thought was to purchase a foam Green Bay Packers Cheesehead hat and modify it for our project. Unfortunately, due to time constraints and availability, we were not able to get one in time for this to work. We instead decided to try to create a custom enclosure ourselves.

A trip to the hardware store yielded some purple insulation sheets, gorilla glue, and yellow spray paint. The next step was to trace out a cheese wedge on the sheets, then cut out four of them, like a layer cake. After making a rough cut of the shape, a smaller template was made using a sheet of bamboo. This bamboo template was then taped to each rough cut piece of foam and used as a cut guide through a hot wire foam slicer.

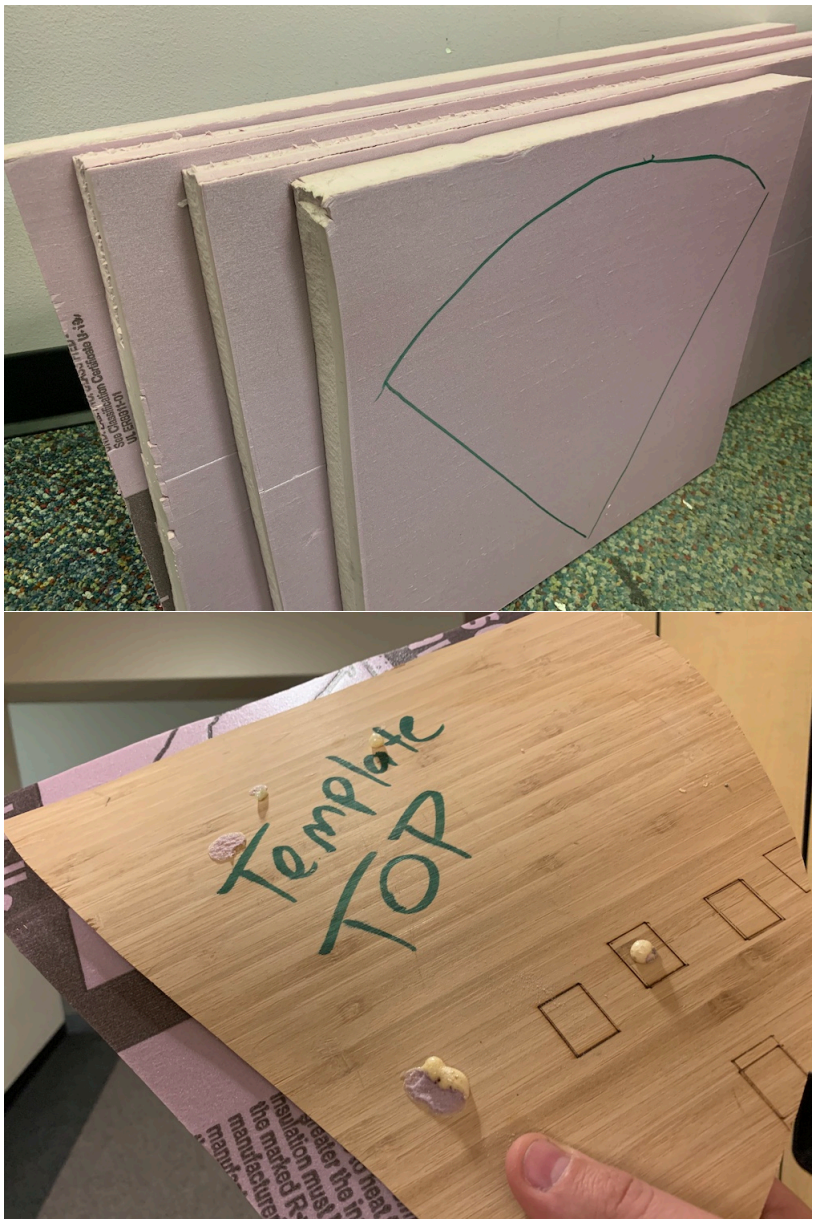

Figure 10: Foam sheets and bamboo template

Next, the bottom two wedges had the center removed to make space for the robot electronics and wheels. We cut slots between these two sheets to make space for bamboo mounting brackets for attaching to the robot. We glued the sheets together, clamped the lot of them, and left it to dry. After the glue dried, we used a heat gun to melt classic holes into the cheese for a more recognizable look. Then, we used plastic wood putty to smooth out

any differences between the sheets and to add some texture to the enclosure. We repeated this process three times to ensure reliable putty coverage, and then spray painted the entire thing yellow. Lastly, we mounted the robot inside.

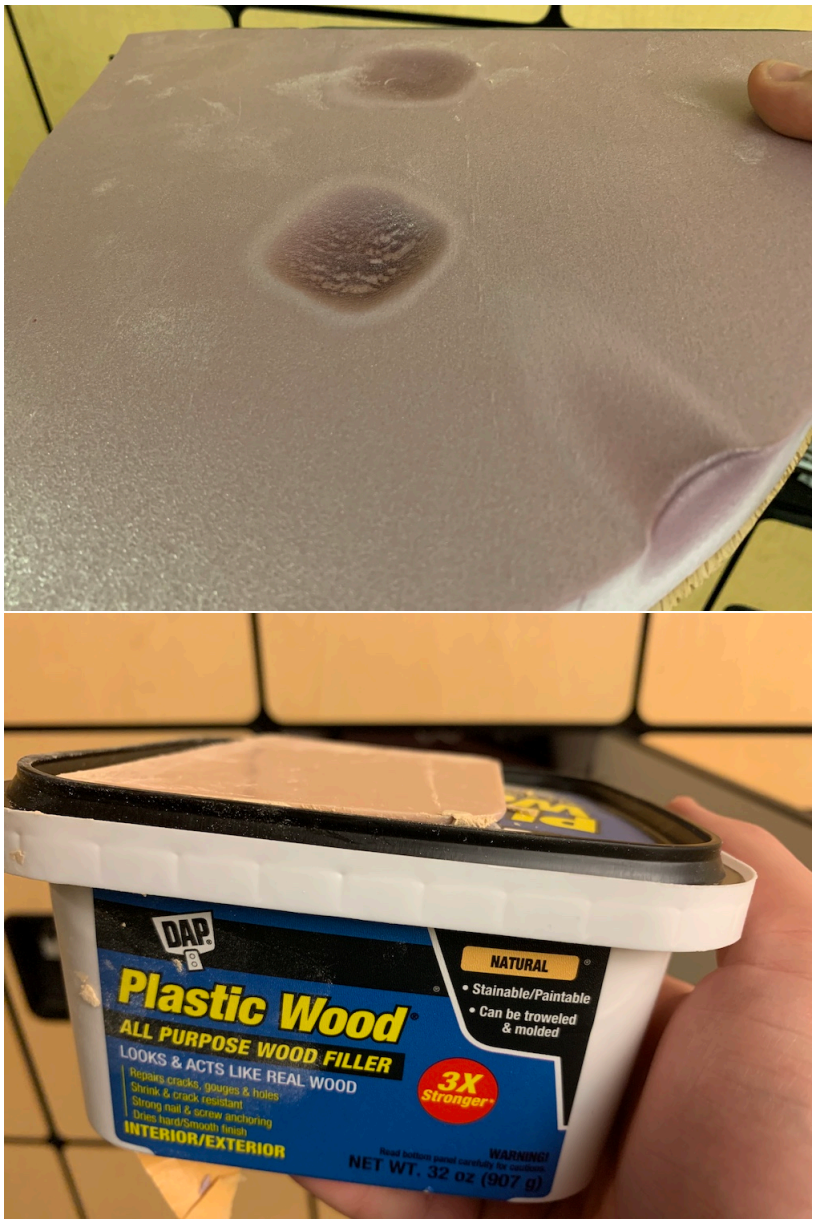

Figure 11: Foam wedge with melt spots and wood putty

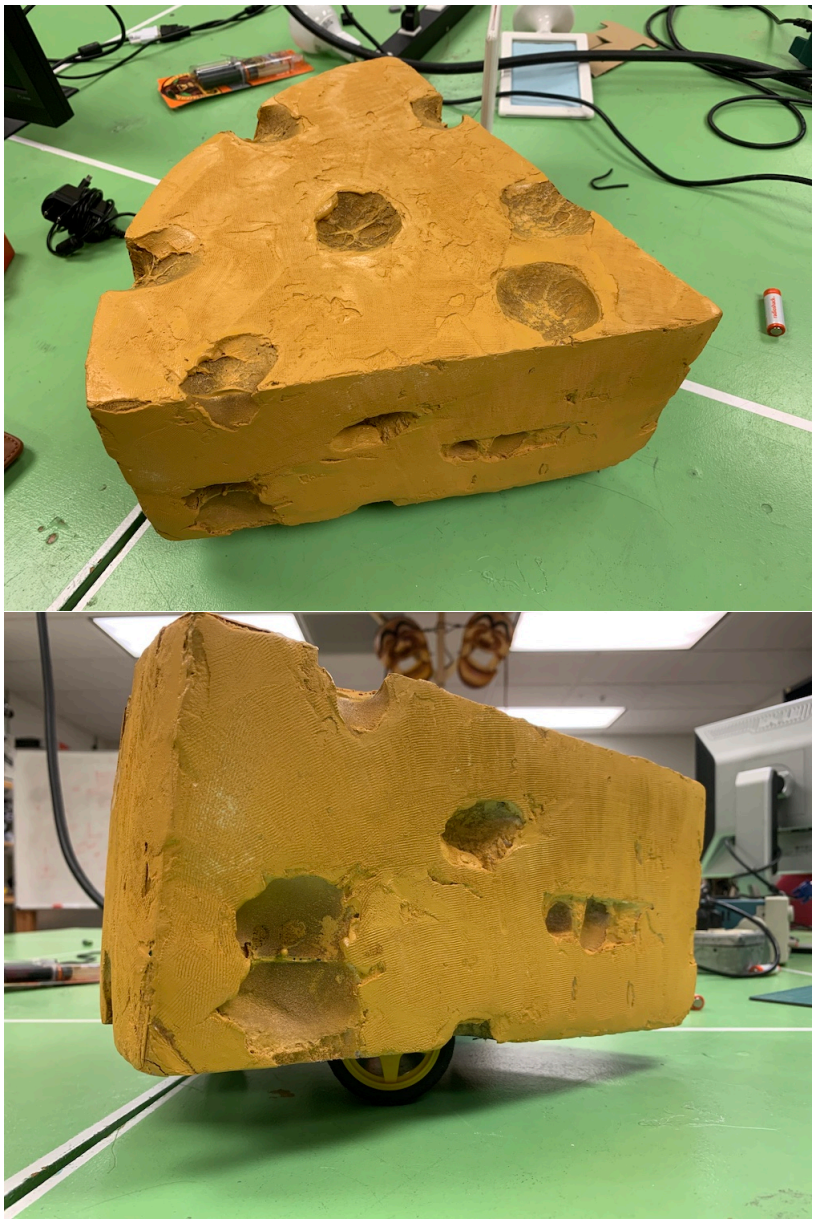

Figure 12: Finished cheese wedge from top and side view

We ordered rat stuffed animals for the enclosures of our other Viking Bots. The original plan was to order rats that were large enough to unsew and wrap around our robots. With this in mind, we ordered the most enormous stuffed rats we could find within our price range, which ended up being a length of 19 inches. However, upon their arrival, we learned that they included the tail in the length measurement, and the tails were quite

long. That meant that we did not quite have enough space to wrap them around the robots, so we decided to mount them to the top of the robots instead, like rats piloting the Viking Bots.

We ordered two white rats, but we soon realized that we should make them different colors so the vision model could recognize them more easily. To accomplish this, we attempted to use grey and brown fabric dye to give the rats a darker color. However, the dye had an unusual reaction to the fabric of our rats. After washing and drying, the "grey" rat turned blue, and the "brown" rat turned pink. Still, this provided enough difference for us to feel comfortable training the vision model with them.

Figure 13: White rat getting ready for dyeing

Figure 14: Hand-dying to avoid dunking

Figure 15: Rats on Roman and rats on cheese

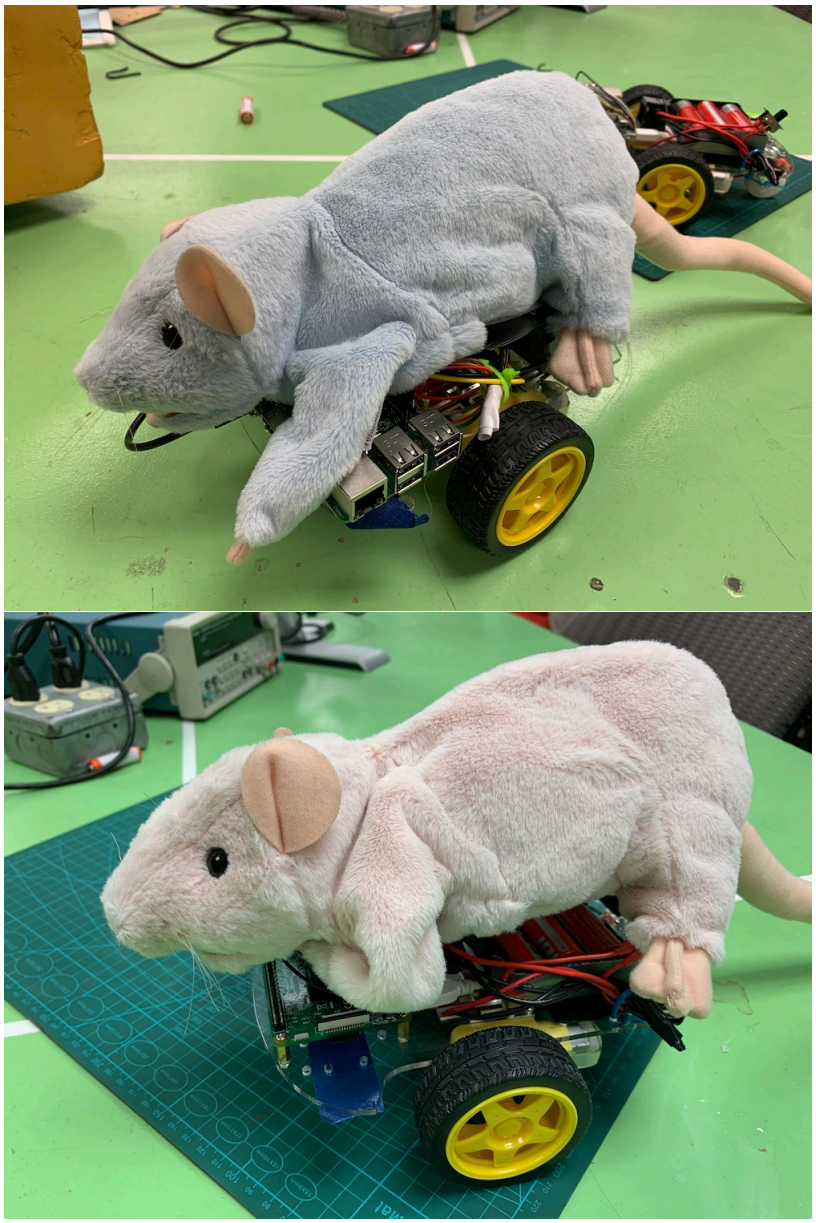

Figure 16: Blue and pink rats mounted on robots

2.4 Challenges

- The complete disarray of the project equipment in the locker introduced much confusion as we tried to determine what we needed to fix and what we needed to order. This chaos resulted in us spending more money to order parts and way more time than we anticipated just assessing what

was available to us and what was working.

- A member from the previous group initially told us that everything was up and running and just needed battery replacement and assorted electrical fixes. This miscommunication further added to the confusion while getting started and resulted in us not understanding the actual state of the project right away. Overall, it wasted a large chunk of our time.

- Our battery holders came with wire rated for fewer amps than we needed. During our initial test, we used the H-Bridge that we did not know was faulty. This problem caused an increased current draw and ended up melting a battery pack wire and almost blowing up the battery. This situation caused us to think very carefully about how we should improve the power system on the robots.

- Frying the diodes on the Raspberry Pi was a significant setback for us. We were not able to test multiple robots until we had more controllers set up. We were able to order new parts and replace the diodes quickly, so we did not waste too much time on this.

- Figuring out what the previous group built with the wiring was a big challenge, as well. There were no diagrams or schematics for the wiring left over from the last team. We had to deal with different types of batteries, taped connections, strange switch setups, and daisy-chained jumper wires that were all contributing to inconsistent results.

- Correcting the hardware issues and making the system more robust took far more time than we had allocated for it. We were under the impression that the hardware was all working and needed minor adjustments. Instead, it required us to rebuild it completely. This issue meant we had a lot less time than we thought to work on computer vision and game concepts than we originally planned.

2.5 Future improvements

- A shield board for the Raspberry Pi, so wires use either screw terminals or solder connections instead of jumper wires.

- A better H-Bridge module that allowed for powering the Raspberry Pi without the need for an additional voltage regulator.

- Larger motors could replace the current versions to get increased torque and speed.
- Different wheels with softer tires would make an excellent addition to getting better traction on slick floors.
- Laser cutting a smaller deck for the robots would allow us to fit the rats to over the top for better aesthetics.
- If we use a standalone voltage regulator, we can upgrade the 18650 battery pack size to a four-battery holder version for even longer run-time. This change would also provide extra power for adding sensors or cameras to the robots.

3 Movement

Each Viking Bot has a Raspberry Pi that sends signals to an L298N H-Bridge. We created the controls using simple python functions that send commands which control the direction, power, and speed of the motors. It is possible to sign into the board remotely and send instructions using the wifi capabilities of the Raspberry Pi.

3.1 Goals

- Setup our wifi network for full control and plenty of bandwidth
- Control the movements of each robot remotely via an SSH connection
- Control speed, timing and direction of the robot through Python commands
- Streamline hardware so that all robots are controlled the same and yield the same results after receiving commands

3.2 Design

The Viking Bots are functionally Braitenberg vehicles. Instead of receiving inputs directly from a light source to the motor, here we provide stimulus through an SSH connection to the robots Raspberry Pi controller. When the Raspberry Pi receives the command, it then provides power to the motors for a specified amount of time. The previous team determined the setup of our robots, so our design choices were limited. Therefore, we describe the system and some problems associated with it below.

- The central computer and the robots Raspberry Pi controller must be on the same wifi network to be able to connect.
- The robot's Raspberry Pi controller runs a python script that creates a server-client. This client accepts commands from the central computer.
- When a command is received, the robots' Raspberry Pi runs a python command that will provide control signals to the H-Bridge.
- The H-Bridge module receives the control signals from the Raspberry Pi and provides power to the motors on the specified motor output lines.
- Using time delays and PWM (pulse width modulation), the robots can be made to turn right or left and go forward or backward.

While experimenting with this system, and taking into account the feedback from the last team in their report, we discovered a few issues with the previous setup of the system:

- The previous system used the wifi connection from the robotics lab, but the demo was in the circuits lounge, which is 100ft away. This distance left the wifi link spotty and caused frequent drops.
- The system jogs (slightly moves forward and backward) the robots to obtain their direction, then sends commands to advance them to the next space, slowing down the gameplay.
- Each robot's hardware was previously different, causing robots to need different parameters to get to the next game space properly.

We solved these problems by setting up the robots on a different wifi network, attempting to track the movement of the robots and avoid the need for jogging, and streamlining the hardware, as detailed in the hardware section above.

3.3 Implementation

Since the movement system was mostly complete and more straightforward to understand than the vision training and hardware setup, we didnt need to spend as much time on this section. However, we were anxious to test the robots movement as soon as possible to verify this. Also, one of our main project goals was to increase the speed of the game. Therefore, we were interested in determining if we could gain any speedup from making changes to this system.

3.3.1 Assessing the current state of movement

Our first goal was to determine what the status of the movement controls was. However, our hardware was without batteries and had damaged Raspberry Pis when we started the project. We were able to use the micro SD card from one of the team Raspberry Pis and test it in a personal Raspberry Pi reasonably quickly using a power supply for the system instead of batteries.

After some rewiring of the jumper wires and swapping out a broken H-Bridge, we were able to get the motors spinning on our test rig. Shortly after that, we were able to SSH into the Raspberry Pi and send commands to the robot, causing its motors to spin.

3.3.2 Finding the correct programs to use

Finding the right programs to use for the robot motor control was a challenge. We struggled with determining which files from the previous groups source folder we were supposed to run on the robot and which we were supposed to run on the main computer. Part of this initial confusion was because we didnt understand that the robots were not controlling themselves. We expected more robust code for their control and looked over the simple commands that are used to drive the motors forwards and backward.

Below you can see a code snippet that shows the setup of the Raspberry Pis pins for use with the H-Bridge. Finding this section of code was vital in making sure that the Raspberry Pis were connected to the H-Bridge modules correctly. Also, we can see at the bottom that 'enable' pins on the H-Bridge module are set up for PWM so we can control its speed.

3.3.3 Testing and making speed improvements

Unfortunately, we needed to wait for the hardware to be completed to test the entire system; this delay took longer than expected. We were finally able to check all the present work while we set up to take images for vision training. At this time, we learned that the wifi connection from the Robotics Lab was not sufficient to have a reliable and repeatable system.

Fortunately, we were able to determine that we could get a significant speed improvement by changing the parameters for the PWM of the 'enable' pins on the robots. These enable pins were previously set to a lower level and resulted in the robots moving more slowly. Our biggest concern for speed was our cheese robot since it was carrying the heaviest load. After increasing the PWM settings for this robot, it was able to move very quickly, and we had no problem achieving the speeds we required. These speed improvements allow the game to operate as fast as the player's input.

```
1  # switches on the H-bridge
2      L298N_IN1 = 26
3      L298N_IN2 = 19
4      L298N_IN3 = 13
5      L298N_IN4 = 6
6      L298N_ENA = 16
7      L298N_ENB = 12
8      GPIO.setmode(GPIO.BCM)
9
10     # initializing GPIO pins to low outputs
11     GPIO.setup(L298N_IN1, GPIO.OUT)
12     GPIO.setup(L298N_IN2, GPIO.OUT)
13     GPIO.setup(L298N_IN3, GPIO.OUT)
14     GPIO.setup(L298N_IN4, GPIO.OUT)
15     GPIO.setup(L298N_ENA, GPIO.OUT)
16     GPIO.setup(L298N_ENB, GPIO.OUT)
17     GPIO.output(L298N_IN1, GPIO.LOW)
18     GPIO.output(L298N_IN2, GPIO.LOW)
19     GPIO.output(L298N_IN3, GPIO.LOW)
20     GPIO.output(L298N_IN4, GPIO.LOW)
21
22     # Set duty cycle to 100%
23     pwm_a = GPIO.PWM(L298N_ENA, 500)
24     pwm_b = GPIO.PWM(L298N_ENB, 500)
```

Figure 17: Python code snippet showing setup for H-Bridge

3.3.4 Setting up our wifi network

To solve the wifi issues, we decided to attempt to use one of the PSU wifi networks. However, we learned that they are locked down for security and do not allow you to SSH into other devices. We discussed this with some other teams to find out that it is difficult working with the CAT (Computer Action Team) to overcome this setup as well.

Therefore we were left with the choice of using a phone mobile hotspot or setting up our wireless network. Since cellular service is spotty in the basement, we decided to purchase a wireless router that we could control fully. Since we were not plugging our router into the school's network for an internet connection, we did not have any issues with security or configuration requiring working with the CAT.

We wanted an inexpensive router that could provide excellent speed and a consistent connection for our system. This purchasing decision took some time while we evaluated our options. We looked at used wireless routers from Goodwill, but these had no documentation, and we were concerned with reliability. We looked at new routers handling lots of data bandwidth and supporting the next generation of wifi technology. However, on closer inspection, these routers provided a slow speed for 2.4gHz bandwidths. Since we have some devices that use the 2.4 gHz bandwidth, we decided to go with the inexpensive NETGEAR AC750 for just under $40. After using this router in the circuit lounge to connect our system, we had a great connection and did not drop the signal during testing.

Figure 18: NETGEAR AC750 wireless router

3.3.5 Complete robot server system

Below is a block diagram of the entire system. This diagram shows the connections between computers, robot hardware, and wireless network. It gives a visual representation of the robot system and how it relates to the motors turning to move the robot to the next game space.

The code for running this server is listed in the appendix under *Python code for robot server*.

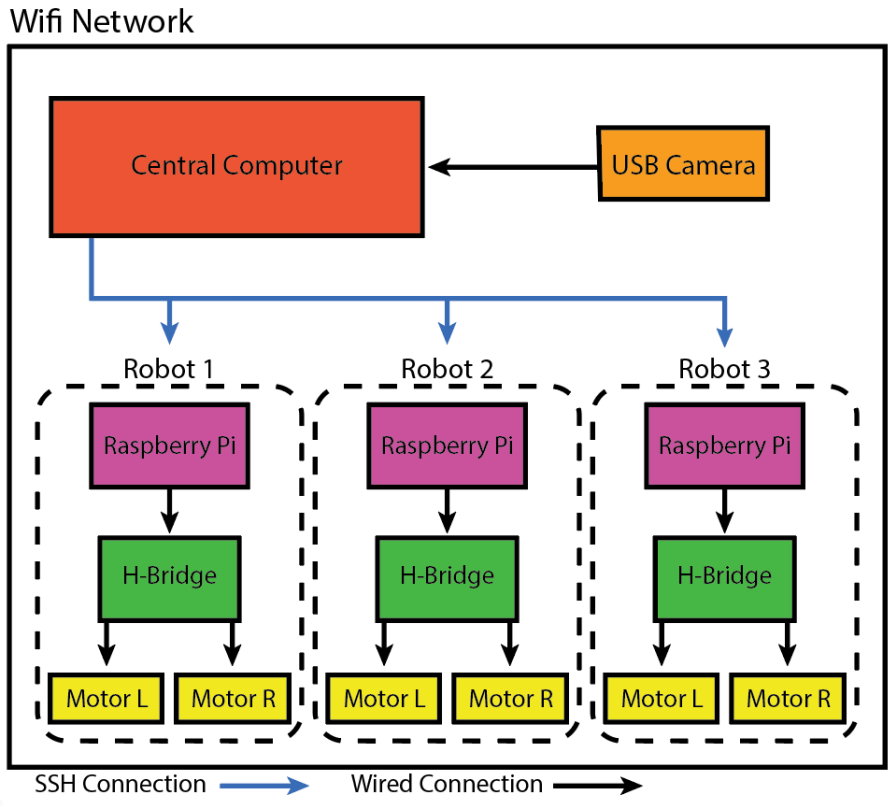

Figure 19: System block diagram

3.4 Challenges

Like most of the challenges in this project, the movement has been a problem with documentation. The main control program on each robot is simple and easy to follow; however, the mechanics of the game and functionality was the most significant challenge.

- **No documentation:** This lack of literature lead to a lot of guesswork and reading through many code files to differentiate what was old, unimportant code, versus newer, used code. It made it difficult and lengthy to get the robots up and running in their previous (inherited) state.
- **No instructions for game use:** There was no explanation as to how to execute the game program or interact with it. There were some scattered pictures from machine vision training, but no media on the user directly playing the game.

- **No instructions on setup:** The robots all have numerous vestigial code files on them, dating back to the first incarnation of this project, which was drastically different. Vague or similarly named files created even more troubleshooting for setting up the correct data for running the game. The only initially known file was a basic script for manual robot movement testing.
- **No comments in code:** Much of the code was either uncommented or commented in simplistic ways, without explaining its fundamental purpose. This confusion was more of a problem in the custom movement files designed to run the robots. Much of this, we re-wrote ourselves, due to not understanding the minds of the previous group.

3.5 Planned Improvements

- **Better documentation:** Since this is a project that an entirely new team of students will pick up at some point, having excessive documentation to explain even seemingly trivial things would go a long way to ensure clarity and understanding.
- We can't assume anyone coming after us will know and be very familiar with any of the tools used here. Documentation more in line with Adafruits very thoroughly documented parts and schematics would help.
- An explanation, with pictures to match, detailing what each function did and why we configured it that way. Additionally, links to the less-common functions we used and how they work.
- In code comments explaining how each function operates.
- **Example of how to set up a game:** This was something we had to contact a previous team member to get help with as there was no information detailing which files to run on the robots. We aim to have step-by-step instructions describing how to set up the system the same way we did when we demonstrated the project.

4 Vision

4.1 Introduction

The vision module allows the Viking Bots to take whatever information they have around them in their environment and implement it into the software.

The software used for the vision portion and in preparation for training was OpenCV (an open-source computer vision and machine learning library), MATLAB programming language, Darknet (an open-source neural network framework), YOLO (a real-time object detection system), YOLO Mark (a GUI for marking bounded boxes of objects in images for training neural network YOLO). We use these frameworks for the preparation and integration of the training.

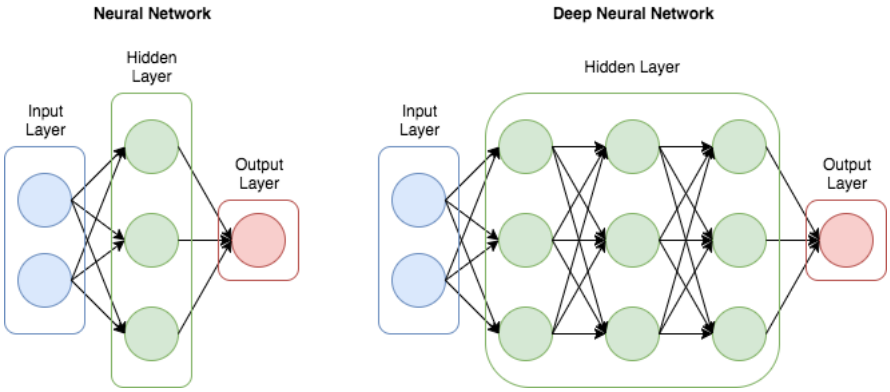

Figure 20: Neural network diagram

4.2 Goals

1. Determine how we want to set up the game board
2. Set up the camera at an angle where it will capture the whole game board
3. Take images of the robots on each one of the tiles that make up the game board
4. Generate text files for each image containing the coordinates of each robot
5. Write a program that generates permutations of each image and the text files along with them
6. Add the images and text files into Darknet for training

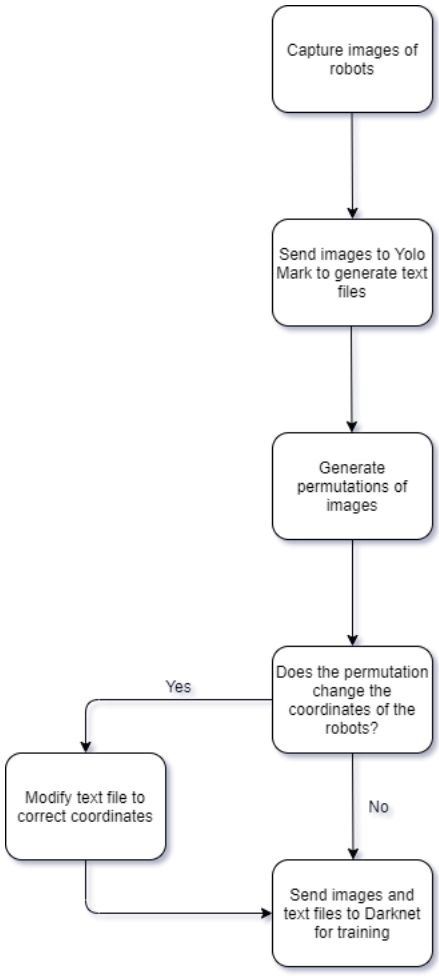

Figure 21: Flow chart for training machine vision

4.3 Design

We stuck with using the same design method as the previous group that worked on this project, with some modifications and the addition of the MATLAB program. We changed the Hexapod to another Viking bot for faster mobility. We also designed and created costumes for each robot to make it easier for object detection. The rats differentiate from each other by being different colors with the same silhouette, and the cheese differentiates from the rats by being an entirely different ensemble from the rats. These modifications caused us to retrain the robots, instead of using the same training data as the previous group.

4.4 Implementation

4.4.1 Setting up game board

To train the robots, we needed images of the robots on the game board, as well as text files containing coordinates of where each robot exists in each image. The process for this required us to set up the game board and camera, as it would be set up for the demonstration. Once everything was set up, we began placing the robots on the game board. One thing to note is that the game board is made up of 16 triangle tiles that are connected to make up one triangle tile.

4.4.2 Taking pictures

We then began taking images of the robots on the game board. It was essential that we made sure to have pictures of each robot being in each of the small triangle tiles. Another vital approach was to take pictures of the robots on each tile from different angles. We decided to rotate the robots 45 degrees in each small triangle tile until we captured all sides of the robots. We also took images of the robots being obstructed by other robots to train the neural network for those kinds of situations.

4.4.3 Marking images

Once we captured all of the images (201 images), we used YOLO Mark, a GUI to mark bounded boxes around each robot in each image. YOLO Mark then generates text files for the coordinates of each robot. As mentioned before, Darknet requires these text files for training the robots.

4.4.4 Permutation training

To improve accuracy for training, Kai wrote a MATLAB program that generated permutations of each image, such as color-shifting, adding noise, or rotating the image. This script reduces issues in detecting the robots as it forces the neural network to understand more fundamental features of the classes it's learning. The MATLAB program also copies and modifies the text files for each permutation, reducing the labor of re-boxing images with YOLO Mark.

4.4.5 Training the neural network

With the images we took, the permutations MATLAB generated, and the associated text files for each one, we began training Darknet. As this was a

complex process, we list the exact setup of Darknet in the appendix.

4.5 Challenges

Several of the challenges of this project stemmed from the team's lack of understanding about DNNs (Deep Neural Network) and ANNs (Artificial Neural Networks) and the previous team's use of Darknet. The last team's use of this was not well documented, and coupled with our teams lack of understanding of machine learning made it very difficult to get the software going in the beginning. Much time was spent trying to get library compatibilities and make sense of unclearly-named Github code files and repositories. Two team members were able to get Darknet and YOLO running on their computers (though one had to do it without the GPU because CUDA uses an NVIDIA specific library). This lack of hardware limited our ability to train and test out the software effectively.

4.6 Future improvements

- Better documentation for setup and use of Darknet/YOLO, especially creating and configuring new environments.
- A precise explanation of the previous team's code files and their purpose in the overall program.
- Offer better documentation for the next group as the majority of our time went to interpreting and reactivating previous work instead of making improvements to the overall game system.

4.7 MATLAB and permutation training

The associated MATLAB R2018a file generates permutations of all images in the training folder, such as color-shifting, adding grain, de-noising, and rotating. These permutations ensure the training algorithm learns to identify the robots in sub-optimal conditions, or learns to identify them more fundamentally. For example, adding noise to the image requires more advanced line-recognition. Rotating the images forces the algorithm to understand what the robots look like from different angles, for instance, if future groups set up the webcam differently. Color-shifting forces the algorithm to learn more about the shape and structure of the robots, instead of a simple color-matching.

The program also copies and outputs the associated text files, so it is imperative that the text files of the training (input) pictures already exist from YOLO Mark. User-adjustable settings, such as the number of permutations per image, have labels at the beginning of the program.

While none of these are technically necessary to train the algorithm, training using permutated images results in more consistent identification in suboptimal environments.

Note that we used MATLAB arbitrarily as a quick way to generate permutations without needing to set up various Python-related virtual environments and manage external dependencies. There is no practical reason why permutations require MATLAB specifically.

5 Useful Links

Link to GIT repository: https://github.com/mmayers88/Robotics

OpenCV installation: https://docs.opencv.org/3.4.7/d7/d9f/tutorial_linux_install.html

Darknet/YOLO: https://github.com/AlexeyAB/Darknet

YOLO training - https://medium.com/@manivannan_data/how-to-train-yolov3-to-detect-custom-objects-ccbcafeb13d2

Yolo_Mark: https://github.com/AlexeyAB/Yolo_mark

6 Major Purchases

Item	Cost	Date Received
Battery Charger (Universal Smart Battery Charger 4 Bay by Eastshine)	$27.95	2019-10-10
Battery Holders (6 pcs 3 x 3.7V 18650 by Sackorange)	$6.99	2019-10-10
Spare Robot Car (Perseids DIY Robot Smart Car)	$13.99	2019-10-16
Motor Driver (L298N)	$6.89	2019-10-16
18650 Batteries (Samsung 25R 18650 2500mAh 20A)	$28.05	2019-10-15
4x Voltage Dividers (12V to 6V Converter and Regulator by DROK)	$39.96	
NETGEAR AC750 Dual Band WiFi Router	$39.88	

Table 1: Itemized purchases

7 Team Roles

7.1 Hardware

- Assessed the state of the robots - Tyler, Mikhail and Bliss
- Determined the problems with the previous hardware setup, then re-built and rewired the robots - Tyler, Bliss, and Mikhail
- Aided in navigating the lab and locating parts/robots - Bliss
- Designed new battery solution - Tyler and Roman
- Installed new batteries, voltage regulators, power switches - Tyler, Mikhail and Bliss

- Replacement of burned out diodes on Raspberry Pis - Mikhail
- Initial schematic for connection to Raspberry Pi - Bliss
- Second schematic showing connections to motors and H-bridge - Tyler
- Final schematic using Fritzing - Mikhail
- Researched and purchased components, Raspberry pis, batteries, wire, battery packs, wifi router, H-bridge modules, new robots, etc. - Mikhail, Roman, Kai, Bliss, Tyler

7.2 Design

- Selection and sourcing of the stuffed rats - Mikhail, Tyler, Roman
- Selection and sourcing of the dye used for the rats - Mikhail
- Painting/Dyeing of the rats from white to a recognizable color - Tyler
- Materials sourcing and design of the foam cheese - Tyler
- Cutting of the sheets of foam with template - Tyler and Roman
- Glueing, melting, painters putty application, sanding, painting, other fabrication of cheese - Tyler

7.3 Machine Vision

- Initial setup and testing of Camera and stand - Tyler and Roman
- Testing of camera for overhead view of gameboard - Roman
- Captured images of the robots for training by placing them in all possible game board locations and rotating them in each position - Tyler, Roman, Mikhail
- Marked bounded boxes for each image around the robots using YOLO Mark GUI and generated text files for each image - Roman
- Initial setup of Darknet and YOLOmark for training. Helped the rest of us get the software up and running - Kai
- Worked on getting Darknet setup for algorithm training - Bliss
- Algorithm training using Darknet - Mikhail

7.4 Software

- Provided team with initial software setup for direct robot control - Bliss
- Removed gesture control and set up new control system - Mikhail
- Changed IP addresses and setup bots with new wifi router - Mikhail
- Setup and reinstalled software on Raspberry Pi for third robot - Mikhail
- Developed software to make permutations of images and generate new text files describing image location - Kai
- Developed MATLAB program to calculate the state space for all possible game moves. - Kai
- Developed possible methods for increasing the state space beyond a solvable game. - Kai

7.5 Integration

- Attempted Darknet and YOLO integration successfully - Mikhail, Kai
- Attempted Darknet use with GPUs enabled - Mikhail
- Attempted Darknet and YOLO integration unsuccessfully - Bliss, Roman, Tyler
- Met Wubin, from previous team to go over system use and setup - Mikhail, Bliss
- Attempted weight training of new robot models/agents - Mikhail

7.6 Research

- Researched Darknet use and implementation - Mikhail, Kai, Bliss, Roman, Tyler
- Researched CUDA and NVIDIA driver usage and implementation for a Linux environment - Bliss
- Researched alternative H-Bridge options - Tyler
- Researched better battery packs - Tyler, Roman
- Researched alternate training options through AWS, GCP - Kai

7.7 Miscellaneous

- Report writing and documentation - Mikhail, Roman, Kai, Bliss, Tyler
- Transfer of report to Latex, final editing. - Kai
- Developed slide deck and presentation - Kai

8 Appendix

8.1 MATLAB code of image permutator

```
% Image Perm
% Kai Brooks
% github.com/kaibrooks
% 2019
% MATLAB R2018a
%
% takes an image and makes a bunch of permutations of it for training an image recognition algorithm
%
% folder structure must be:
% (base dir)/images/training    for the input (training data) files
% (base dir)/images/output      for the output (permutated) files
% images must be .jpg

clc; close all; clear all; rng('shuffle');

% user settings -----------------------------------------------------------


makeImages = 10;    % (10) permutations of each image to make
repeatProb = 0.3; % (0.3) probability an image goes back through filtering again
maxAngle = 30;      % (30) max angle rotations will make
fuzz = 0.01;        % (0.01) fuzz in noise

deleteExistingFiles = 1; % deletes previous output before saving new run

% other vars (no touch) ---------------------------------------------------
rotated = 0;
flipped = 0;
```

```
28 filts = 0;
29 cont = '';
30
31 % go
    ------------------------------------------------------------------------

32
33 % check for older data
34 oldFiles = dir(fullfile('images/output/', '*.*')); % existing
       output from previous runs
35 if deleteExistingFiles % delete previous files
36     for k = 1 : length(oldFiles)
37         baseFileName = oldFiles(k).name;
38         fullFileName = fullfile('images/output/',
               baseFileName);
39         fprintf(1, 'Deleting %s\n', fullFileName);
40         delete(fullFileName);
41     end
42     fprintf('Deleted %i files\n',length(oldFiles))
43     oldFiles = dir(fullfile('images/output/', '*.jpg'));
44 end
45
46 % check if data exists and ask to overwrite
47 if size(oldFiles) > 0;
48     cont = input('Files already exist and may be overwritten.
           Y to continue: ','s');
49     if upper(cont)  "Y"
50         fprintf('End\n')
51         return
52     end
53 end
54
55 % get contents of training folder
56 getImages = dir(fullfile('images/training/', '*.jpg'));
57 getTxts = dir(fullfile('images/training/', '*.txt'));
58
59 % end if training folder is empty or unreadable
60 if length(getImages) == 0
61     fprintf('No .jpg images in images/training/\nEnd\n')
62     return
63 end
64
65 % warn if there isn't a matching txt for each image
66 if length(getImages)  length(getTxts)
67     prompt = sprintf('Warning: Unequal images and texts in
           training (%i images, %i txts). Y to continue: ',length
           (getImages),length(getTxts));
68     cont = input(prompt,'s');
69     if upper(cont)  "Y"
```

```
70          fprintf('End\n')
71          return
72      end
73  end
74
75  % actual loop start
76  fprintf('Starting permutation generator...\n');
77  for j = 1:length(getImages)
78
79      % get image
80      im = imread(fullfile('images/training/', getImages(j).
            name));
81
82      % get filename of image to prefix output permutations
83      outputPrefix = erase(getImages(j).name,'.jpg');
84
85      while i < makeImages
86          adjFactor = rand();
87          alg = randi(2); % <= 5 for color only, <= 2 for no
                coordinate changes
88          filts = filts + 1;
89
90          switch alg
91
92              case 3 % flip
93                  if flipped
94                      continue
95                  end
96                  temp = flipdim(im, 2);          % horizontal
                        flip
97
98              case 4 % flip again for more probability
99                  if flipped
100                     continue
101                 end
102                 temp = flipdim(im, 2);          % horizontal
                        flip
103
104             case 5 % rotate
105                 if rotated
106                     continue
107                 end
108                 temp = imrotate(im,randi([1 maxAngle]),'crop'
                        );
109                 rotated = 1;
110
111             case 1 % make fuzzy
112                 temp = imnoise(im,'gaussian',0.0,adjFactor*
                        fuzz);
```

```
113
114             case 2 % change contrast
115                 temp = imadjust(im,[.1 .2 0; .8 .9 1],[]);
116
117             case 6 % make b&w and increase contrast
118                 temp = imadjust(rgb2gray(im),[0.1 0.9],[]);
119
120             case 7 % denoise (must be b&w)
121                 amount = randi(4)+2;
122                 temp = wiener2(rgb2gray(im),[amount amount]);
123
124             case 8 % adjust HSV
125                 adjFactor = adjFactor;
126                 temp = rgb2hsv(im);
127                 temp(:, :, 2) = temp(:, :, 2) * adjFactor;
128
129         end % switch
130
131         imshow(temp);
132         f=getframe;
133         imwrite(f.cdata,'images/temp.png');
134
135         % write image and move on to next
136         if rand() > repeatProb
137
138             % write file
139
140             padded = sprintf( '%s_%03d',outputPrefix,i); %
                    prefix and add trailing zeroes
141
142             filename = sprintf('images/output/%s.jpg', padded
                    );
143             imwrite(f.cdata, filename);
144
145             if filts > 1
146                 fprintf("Image %s created with %i filters\n",
                        padded, filts)
147             else
148                 fprintf("Image %s created with %i filter\n",
                        padded, filts)
149             end
150
151             % copy text file and rename it
152             textFile = sprintf('images/training/%s%s',
                    outputPrefix,'.txt'); % get text file name
153             copyfile(textFile, 'images/output');
154
155             % get old and new names and 'copy' file
156             textFileOld = sprintf('images/output/%s%s',
```

```
                outputPrefix,'.txt');
157             textFileNew = sprintf('images/output/%s%s',padded
                    ,'.txt');
158             movefile(textFileOld, textFileNew);
159
160             % reset for next run
161             rotated = 0;
162             flipped = 0;
163             filts = 0;
164             i = i + 1;
165         else
166             %fprintf("Looping after applying %i\n", alg)
167             im = imread('images/temp.png');
168         end % if termChance
169
170     end % 1:makeImages
171     fprintf('Image %s.jpg finished with %i permutations\n',
            outputPrefix, makeImages)
172     i = 0;
173 end % 1:length(getImages)
174
175 fprintf('Done\n');
```

8.2 Python code for main function

```
1  from camera_system import Camera
2  from object_detector import Detector
3  from strategy import Strategy
4  from graph_builder import GraphBuilder
5  from control_system import Controller
6  import logging
7  import sys
8  import time
9  import json
10 import random
11
12 WEIGHT_PATH = '../model/custom_tiny_yolov3.weights'
13 NETWORK_CONFIG_PATH = '../cfg/custom-tiny.cfg'
14 OBJECT_CONFIG_PATH = '../cfg/custom.data'
15 ROBOTS_CONFIG_PATH = '../cfg/robots.json'
16
17 logger = logging.getLogger(__name__)
18
19
20 class FakeGame:
21     def __init__(self):
22         self.camera = Camera(None, draw=False)
23         self.display_camera = Camera(None, window_name='
               labeled')
```

```
24          centers = []
25          with open('centers.txt', encoding='utf-8', mode='r')
                as file:
26              for line in file:
27                  center = tuple(map(float, line.strip().split(
                        ' ')))
28                  centers.append(center)
29          self.centers = centers
30          self.graph_builder = GraphBuilder(self.centers)
31          self.orders = ['thief', 'policeman1', 'policeman2']
32          self.strategy = Strategy(self.orders)
33          self.object_list = {
34              "thief": {
35                  "confidence": 0.99,
36                  "center": self.centers[6],  # (width,height)
37                  "size": (0.15, 0.10),  # (width,height)
38              },
39              "policeman1": {
40                  "confidence": 0.99,
41                  "center": self.centers[1],  # (width,height)
42                  "size": (0.15, 0.05),  # (width,height)
43              },
44              "policeman2": {
45                  "confidence": 0.99,
46                  "center": self.centers[3],  # (width,height)
47                  "size": (0.15, 0.05),  # (width,height)
48              }
49          }
50          self.counter = 0
51          self.thief_movements = [13, 14, 15, 16]
52          self.escape_nodes = {10}
53          self.graph = None
54          self.objects_on_graph = None
55          self.instructions = None
56
57      def forward(self):
58
59          image = self.camera.get_fake_gaming_board()
60          self.display_camera.draw_boxes(image, self.
                object_list)
61          self.display_camera.display(image)
62
63          # build a graph based on object list
64          graph, objects_on_graph = self.graph_builder.build(
                self.object_list)
65
66          self.graph = graph
67          self.objects_on_graph = objects_on_graph
68
```

```
69          # generate instructions based on the graph
70          instructions = self.strategy.
                get_next_steps_shortest_path(graph,
                objects_on_graph)
71          logger.info('instructions:{}'.format(instructions))
72
73          # instructions['thief'] = [objects_on_graph['thief'],
                self.thief_movements[self.counter]]
74          self.instructions = instructions
75
76          self.counter += 1
77          for key, value in instructions.items():
78              self.object_list[key]['center'] = self.centers[
                    value[1] - 1]
79          time.sleep(1)
80
81          image = self.camera.get_fake_gaming_board()
82          self.display_camera.draw_boxes(image, self.
                object_list)
83          self.display_camera.display(image)
84
85      def is_over(self):
86          """
87          Check if the game is over.
88
89          Returns
90          -------
91          game_over: bool
92              True if the thief is at the escape point or the
                    policemen have caught the thief, otherwise
                    False.
93          """
94          game_over = False
95          if self.instructions is None or self.objects_on_graph
                is None or self.graph is None:
96              return game_over
97          if 'thief' in self.objects_on_graph:
98              if self.objects_on_graph['thief'] in self.
                    escape_nodes:
99                  game_over = True
100                 logger.info('The thief wins!')
101             else:
102                 for name, instruction in self.instructions.
                        items():
103                     if name != 'thief':
104                         if self.instructions['thief'][1] ==
                                instruction[1]:
105                             game_over = True
106                             logger.info('The policemen win!')
```

```
107         return game_over
108
109     def get_report(self):
110         """
111         Generate a game report(json, xml or plain text).
112
113         Returns
114         -------
115         game_report: object or str
116             a detailed record of the game
117         """
118         game_report = None
119         return game_report
120
121     def shuffle(self):
122         random.randint(5, 10)
123
124
125 class Game:
126     """
127     Each game is an instance of class Game.
128     """
129
130     def __init__(self, weight_path, network_config_path,
        object_config_path, robots_config_path):
131         """
132         Load necessary modules and files.
133
134         Parameters
135         ----------
136         weight_path: str
137             file path of YOLOv3 network weights
138         network_config_path: str
139             file path of YOLOv3 network configurations
140         object_config_path: str
141             file path of object information in YOLOv3 network
142         robots_config_path: str
143             file path of robots' remote server configuration
144         """
145
146         # fix robot movement order
147         self.orders = ['thief', 'policeman1']
148         # self.orders = ['policeman1', 'policeman2']
149         # self.orders = ['thief', 'policeman1', 'policeman2']
150
151         # initialize internal states
152         self.graph = None
153         self.objects_on_graph = None
154         self.instructions = None
```

```
155
156         # set up escape nodes
157         self.escape_nodes = set()
158
159         # construct the camera system
160         self.camera = Camera(1)
161
162         # construct the object detector
163         self.detector = Detector(weight_path,
                network_config_path, object_config_path)
164
165         # load gaming board image and get centers'
                coordinates of triangles
166         self.gaming_board_image = self.camera.get_image()
167         self.centers = self.detector.detect_gaming_board(self
                .gaming_board_image)
168
169         # construct the graph builder
170         self.graph_builder = GraphBuilder(self.centers)
171
172         # construct the strategy module
173         self.strategy = Strategy(self.orders)
174
175         # construct the control system
176         self.controller = Controller(self.detector, self.
                camera.get_image, robots_config_path)
177
178         # connect to each robot
179         self.controller.connect()
180
181     def is_over(self):
182         """
183         Check if the game is over.
184
185         Returns
186         -------
187         game_over: bool
188             True if the thief is at the escape point or the
                    policemen have caught the thief, otherwise
                    False.
189         """
190         game_over = False
191         if self.instructions is None or self.objects_on_graph
                is None or self.graph is None:
192             return game_over
193         if 'thief' in self.objects_on_graph:
194             if self.objects_on_graph['thief'] in self.
                    escape_nodes:
195                 game_over = True
```

```
196                 logger.info('The thief wins!')
197             else:
198                 for name, instruction in self.instructions.
                        items():
199                     if name != 'thief':
200                         if self.instructions['thief'][1] ==
                                instruction[1]:
201                             game_over = True
202                             logger.info('The policemen win!')
203         return game_over
204
205     def shuffle(self):
206         random.randint(5, 10)
207
208     def forward(self):
209         """
210         Push the game to the next step.
211         """
212         # get objects' coordinates and categories
213         image = self.camera.get_image()
214         object_list = self.detector.detect_objects(image)
215
216         # build a graph based on object list
217         graph, objects_on_graph = self.graph_builder.build(
                object_list)
218         self.graph = graph
219         self.objects_on_graph = objects_on_graph
220
221         # generate instructions based on the graph
222         instructions = self.strategy.
                get_next_steps_shortest_path(graph,
                objects_on_graph)
223         self.instructions = instructions
224         logger.info('instructions:{}'.format(instructions))
225
226         if self.is_over():
227             return
228         # move robots until they reach the right positions
229         while not self.controller.is_finished(self.centers,
                object_list, instructions):
230             # obtain feedback from camera
231             image = self.camera.get_image()
232             object_list = self.detector.detect_objects(image)
233
234             # calculate control signals
235             control_signals = self.controller.
                    calculate_control_signals(
236                 self.centers, object_list, instructions)
237
```

```
238             # cut extra signals
239             real_signals = []
240             for name in self.orders:
241                 for signal in control_signals:
242                     if signal['name'] == name:
243                         # if True:
244                         real_signals.append(signal)
245                 if len(real_signals) > 0:
246                     break
247 
248             # update internal states
249             self.controller.update_state(object_list)
250 
251             # move robots
252             self.controller.move_robots(real_signals)
253 
254             # obtain feedback from camera
255             image = self.camera.get_image()
256             object_list = self.detector.detect_objects(image)
257 
258             # update internal states
259             self.controller.update_state(object_list)
260 
261     def get_report(self):
262         """
263         Generate a game report(json, xml or plain text).
264 
265         Returns
266         -------
267         game_report: object or str
268             a detailed record of the game
269         """
270         game_report = None
271         return game_report
272 
273 
274 def main():
275     # set up logger level
276     logger.setLevel(logging.DEBUG)
277     handler = logging.StreamHandler(sys.stdout)
278     handler.setLevel(logging.DEBUG)
279     logger.addHandler(handler)
280 
281     # parse config file
282     if len(sys.argv) > 1:
283         config_path = sys.argv[1]
284     else:
285         config_path = '../cfg/game_config.json'
286     with open(config_path, encoding='utf-8', mode='r') as
```

```
        file:
287          config = json.load(file)
288
289      # load game parameters
290      weight_path = config['weight_path']
291      network_config_path = config['network_config_path']
292      object_config_path = config['object_config_path']
293      robots_config_path = config['robots_config_path']
294
295      # construct a game logic
296      game = Game(weight_path, network_config_path,
            object_config_path, robots_config_path)
297      # game = FakeGame()
298      # start the game logic
299      while True:
300          input('Press ENTER to the start a game:')
301
302          # keep running until game is over
303          while not game.is_over():
304              game.forward()
305
306          # get the game report
307          report = game.get_report()
308
309          # display the game report
310          print(report)
311
312          # shuffle the robots on the gaming board
313          # TODO: finish shuffle() function
314          game.shuffle()
315
316
317  if __name__ == '__main__':
318      main()
```

8.3 Python code for robot server

```
1   class Robot:
2       """
3       Robot class is a high-level abstraction of our real
            objects, and it has simple
4       interfaces which are easy to manipulate.
5       Examples:
6           robot = Robot(name, ip, port)
7           robot.connect()
8           robot.rotate(85)
9           robot.move_forward(2)
10          data = robot.get_sensor_data()
11          robot.disconnect()
```

```
12      """
13
14      def __init__(self, name, ip, port):
15          """
16          Construct a robot with corresponding name, ip and
                port.
17
18          Parameters
19          ----------
20          name: str
21              one of the names of our objects(thief, policeman1
                    and policeman2)
22          ip: str
23              a string which indicates the ip address of our
                    remote server,
24              for example, "192.168.1.1"
25          port: int
26              an integer which indicates the port number of our
                    remote server
27          """
28          self.name = name
29          self.ip = ip
30          self.port = port
31          self.client = zerorpc.Client(heartbeat=None)
32
33      def connect(self):
34          address = 'tcp://{ip}:{port}'.format(ip=self.ip, port
                =self.port)
35          self.client.connect(address)
36
37      def disconnect(self):
38          self.client.close()
39
40      def get_sensor_data(self):
41          try:
42              data = self.client.get_sensor_data()
43              result = {
44                  'flag': True,
45                  'data': data
46              }
47          except Exception as e:
48              result = {
49                  'flag': False,
50                  'message': repr(e)
51              }
52          return result
53
54      def rotate(self, alpha):
55          """
```

```
56          Rotate the robot with alpha degree clockwisely.
57
58          Parameters
59          ----------
60          alpha: int
61              the degree that the robot rotates(can be negative
                    for
62              counterclockwise rotation)
63
64          Returns
65          -------
66          result: dict
67              a dict consists of some feedback information
68          """
69          try:
70              result = {
71                  'flag': True
72              }
73              self.client.rotate(int(alpha))
74          except Exception as e:
75              result = {
76                  'flag': False,
77                  'message': repr(e)
78              }
79          return result
80
81      def move_forward(self, n):
82          """
83          Move forward n units.
84
85          Parameters
86          ----------
87          n: int
88              steps that a robot moves(can be negative for
                    backward)
89
90          Returns
91          -------
92          result: dict
93              a dict consists of some feedback information
94          """
95          try:
96              result = {
97                  'flag': True
98              }
99              self.client.move_forward(int(n))
100         except Exception as e:
101             result = {
102                 'flag': False,
```

```
103                 'message': repr(e)
104             }
105         return result
106
107
108 if __name__ == '__main__':
109     #robot_client = Robot('thief', '192.168.31.254', 4242)
110     robot_client = Robot('Policeman1', '192.168.1.102',
            4242)
111     robot_client.connect()
112     # robot_client.move_forward(1)
113     robot_client.rotate(60)
```