Fundamentals of Robotics

ENGR3390: Final Project

Your Name

Team Charlie: Isabella Abilheira, Chris Allum, Olivia Jo Bradley, Oscar De La Garza, Nabih Estefan, Maya Sivanandan

May 11, 2021

Olin College

Table of Contents:

Introduction

For the Fundamentals of Robotics final project, our team was tasked with creating a rover that could complete several challenges in ‘The O”, the courtyard in the middle of Olin campus. The baseline challenges included navigating through unobstructed terrain, reaching a supply station, and delivering a payload to a drop box. Beyond that, as a team we aimed to have our rover navigate through stationary obstacles, park within the medium dock box, and deliver the payload to our specific dropbox, which would be designated by an April tag.

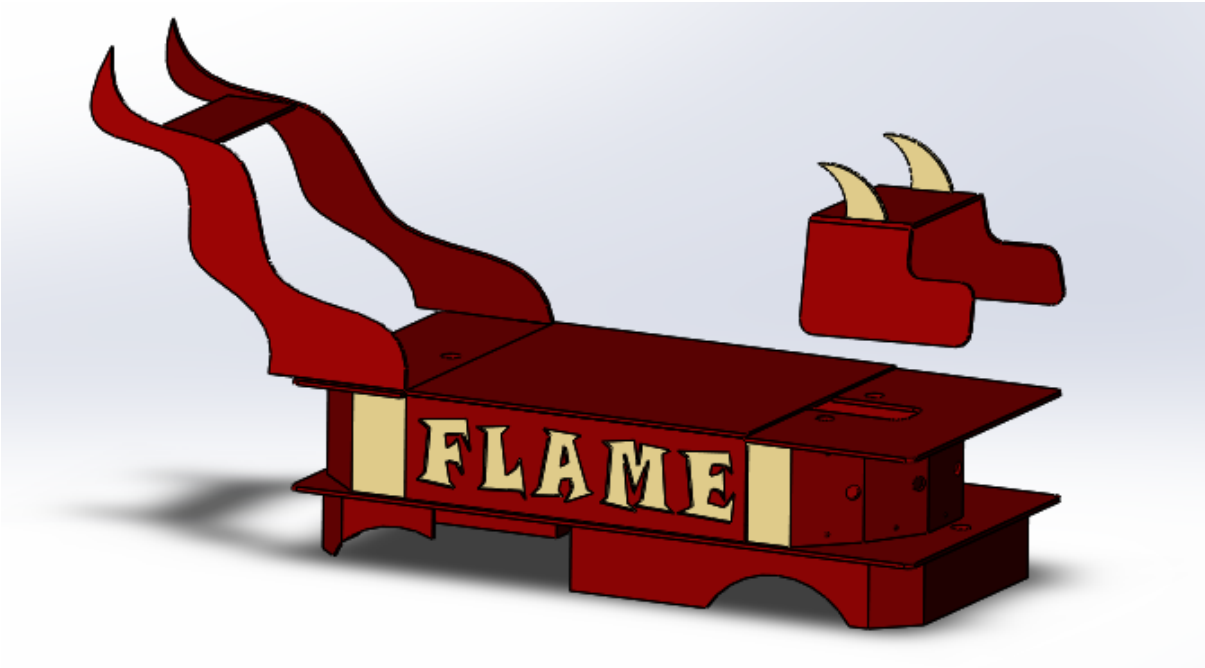
Our team was initially divided into 3 subteams: mechanical design, electrical design, and control software. The mechanical design team was in charge of designing and fabricating the body of the rover and payload mechanism, while the electrical team decided what sensors and motors would be used on the robot and how best to wire them to avoid damage and make them easy to access and troubleshoot. Then the software team was in charge of writing the Sense, Think, and Act functions that would allow the rover to operate autonomously. As the project progressed mechanical and electrical teams finished their tasks sooner, eventually the whole team shifted to the software team and worked on the code for the robot.

Mechanical Design

For the mechanical design, there were two main components to design, fabricate, and assemble: the body of the car and the grabber.

Body

As a group, we decided on creating a dragon themed rover. Our household group name is related to fire, and we are all fire performers, so we wanted to continue that theme in our rover. To start designing our rover, we looked at the general basic design, and modified it to match our aesthetic choices. We used GrabCAD to manage our files and keep them organized. At the end of this stage, our CAD looked like this:



After we designed all of our pieces, we chose to laser cut a prototype out of cardboard. This allowed us to learn if our dimensions were correct and fit with the RC car frame, which portions are easier to assemble, which portions we should consider 3d printing, and what holes we should add for electrical wiring. Below is a photo of this prototype.

After laser cutting, we chose to 3d print many of our mounts, specifically the lidar array mount, pan-tilt servo mount, pi mount, battery mount, and pi cam mount. We chose to 3d print these as they were either difficult to create with sintra (such as the lidar array or the pi cam mount) or we felt more comfortable mounting them with bolts rather than glue (such as the battery mount).

We also chose to make several ease-of-use and accessibility changes. One major change was the back cover. While the decorative sides were fun, we found that sliding it on top was a bit of a hassle, and the letters were easily bumped and bent. Instead we chose a hinged back cover that we can flip open and close easily. This allowed us to protect the electronics while it was drizzling. We also chose to mount many of our components with bolts. This allows us to remove panels easily to help organize wiring, swap out components, and mount things securely.

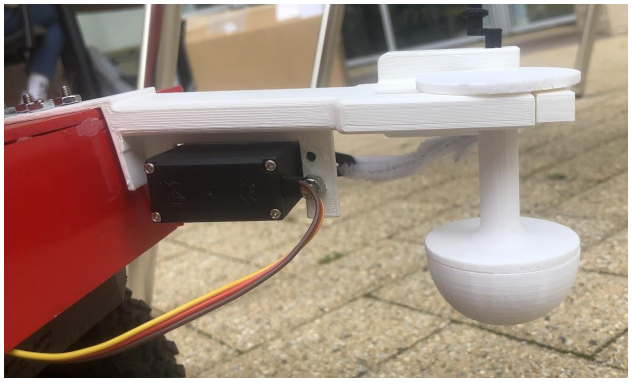
For the rest of the body panels, we chose to shopbot them for accuracy and speed. This not only saved us time, but allowed our parts to have straight and accurate dimensions. Getting to use the shopbot was a personal goal of mine. I had not been able to use the machine since getting trained last semester, and I wanted to use this project to expand my skills.

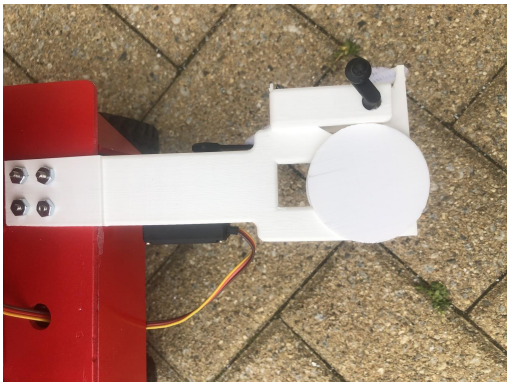
We assembled our rover with PVC glue and Cyanoacrylate glue in addition to bolts. We painted our rover red and gold for the colorful dragon look. Overall, we were able to

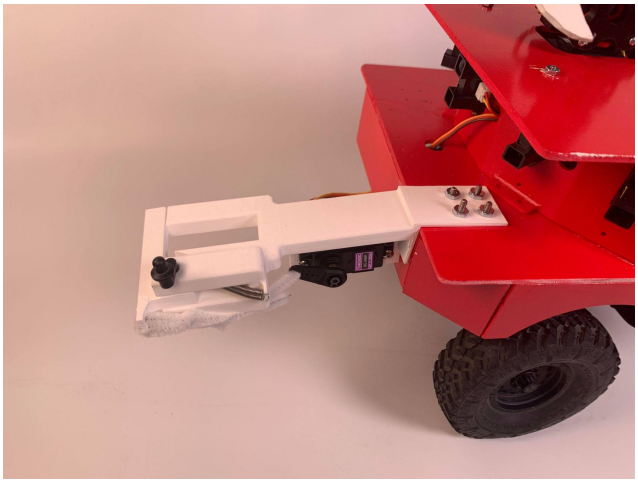
complete this portion early which meant that the rover was semi-functional early in our timeline and both the software and electrical team were able to use it.

Grabber

Outside of the main body of the rover, we also needed to make a grabber that could hold the payload as the rover traversed the course and release the payload once the rover reached a drop box. To simplify this mechanism as much as possible, we made the grabber with a spring. This meant the servo motor would only have to function as the opening mechanism, and the grabber would not accidentally drop the payload in the wrong location due to a loss of power to the servo. We also had to make sure the grabber arm did not obscure any of the sensors the rover was using for guidance. In order to avoid this issue, we made the grabber arm quite short, so it could be statically mounted. A longer arm would have blocked the view from the PiCam, and would have needed a separate servo motor to rotate it out of the camera's view when not depositing a payload.

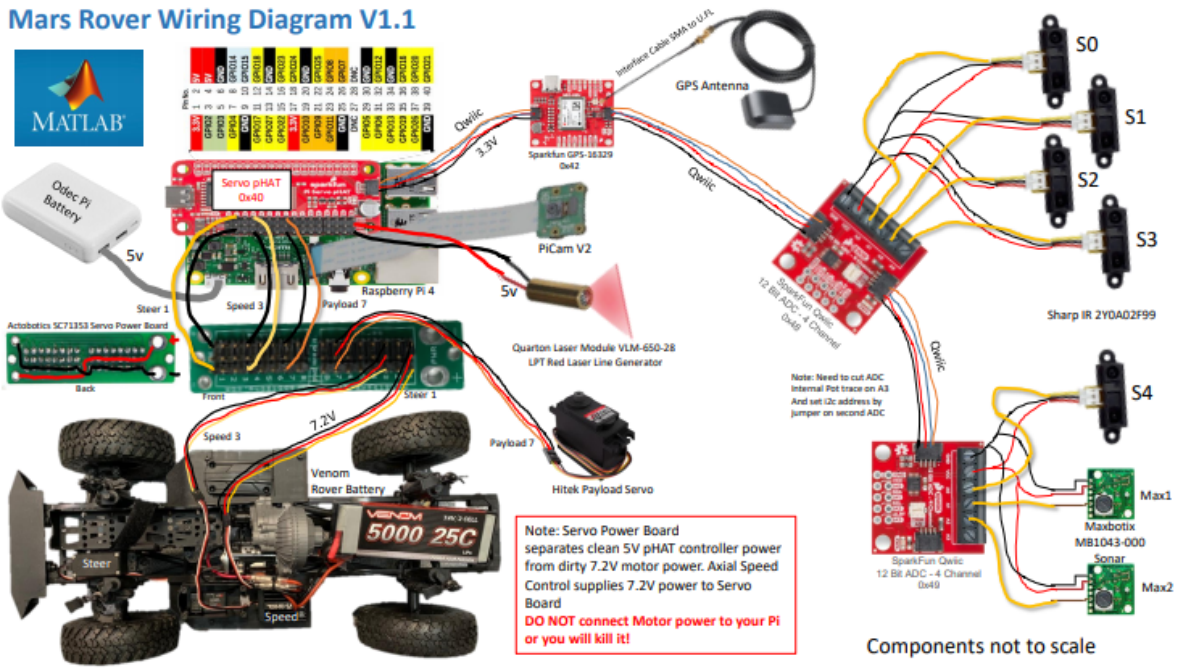




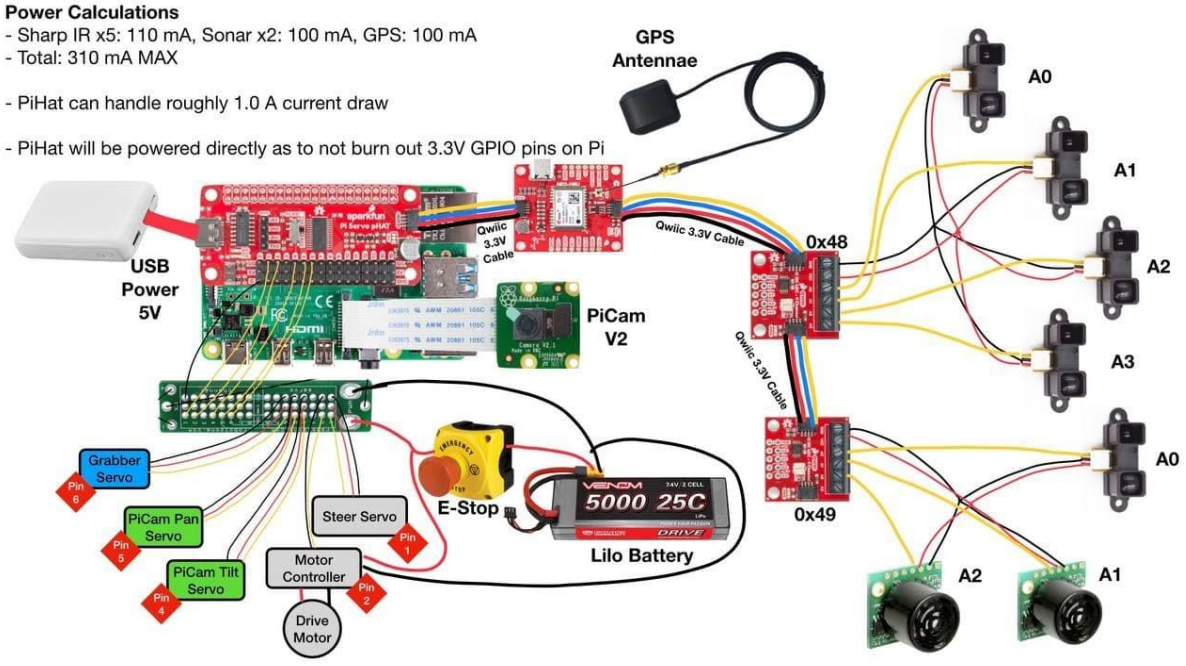


Electronics Design

For the Electronics Design, we started off with the Wiring Diagram given to us at the start of the project, which can be seen below.



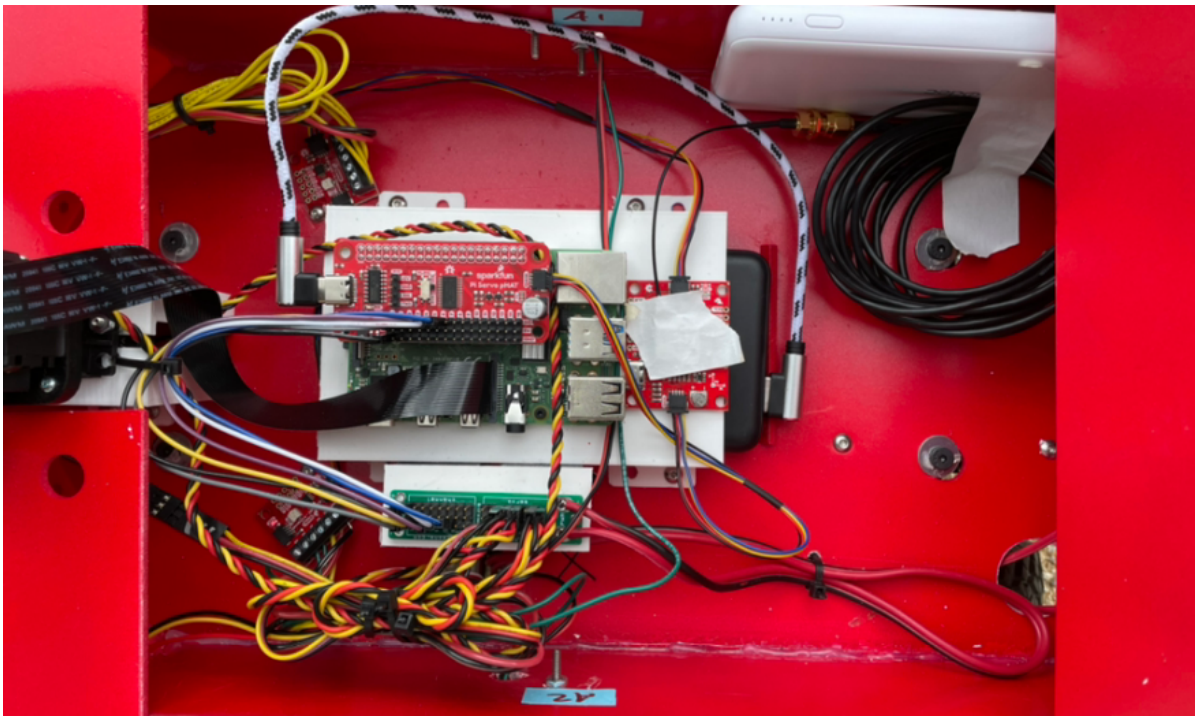
From this, we kept a lot of the things we can see, with big changes being some pins being moved around, not including the laser, and the inclusion of the E-Stop. You can see our final wiring diagram below.



After having done the diagram and power calculations the path to finish wiring was pretty straightforward. While the mechanical team prototyped the body of the rover, the

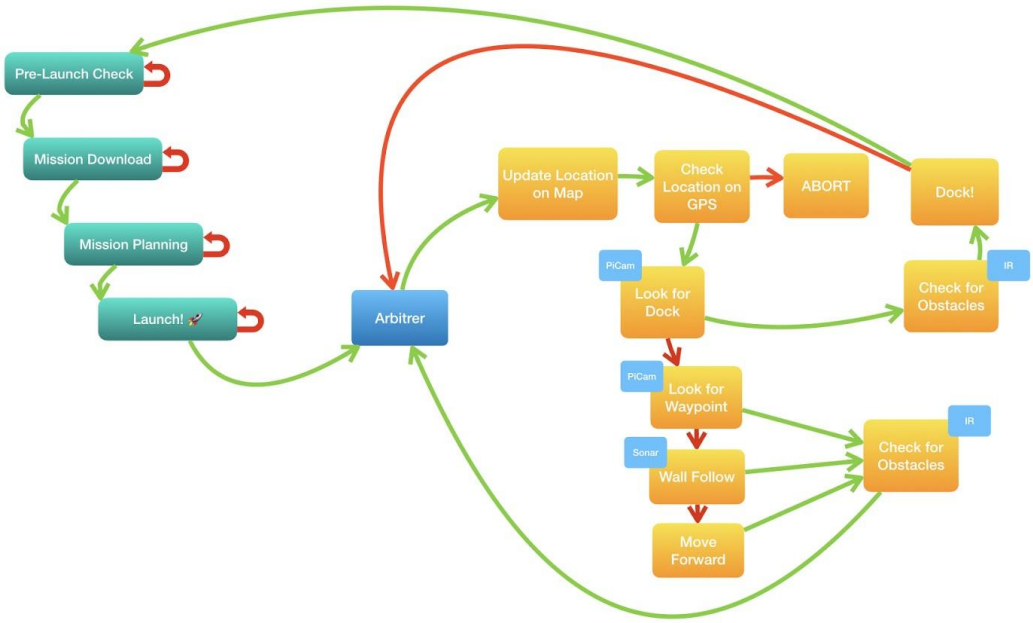
electrical team worked on the Roadkill harness, a board that connects all the sensors and allowed us to troubleshoot the sensors before they were mounted to the rover.

Once the body of the rover had been finished, we were able to bolt down our pi onto its placeholder and mount all sensors on the rover as we had designed. After mounting them we wired them all and zip-tied and cleaned up wiring as much as possible so it didn't look like a mess. Below is a photo of the final wiring inside the rover body



Rover Control Software

For the start of our Control Software development, we began by creating our Finite-State Machine (FSM) that defines the way the rover is thinking and acting. Below you can see our FSM.



As you can see in the FSM, our code is broken into multiple different parts. In our actual development, we separated our files into 4 distinct sections: SETUP, SENSE, THINK, and ACT.

SETUP

Our setup functions are divided into two parts. First, we have the functions we were given by default, those two are:

- *ads1015.m* to setup the sensors through the Qwiic
- *I2C_Servo_pHat.m* used to setup the servos we are using through the pHat and return the roverServos object which we use to control them.

On the other hand, we have the setup functions we created, of which there are 4:

- *SETUP_adc.m* which uses *ads1015.m* to create our adcDevice object for the different connections we are using to the sensors.
- *SETUP_gps.m* which we use to setup and calibrate the GPS in the cases where we use it.
- *SETUP_pi.m* which connects to our Raspberry Pi on the rover and sets up the blink LED on it and returning our robotPi object.
- *SETUP_piCam.m* which sets up the piCam when its connected to our Pi, giving it our preferred settings and creating the robotCam object.

SENSE

On the side of our SENSE functions, we have created a good amount of them for different sensors and cases of use. For starters we once again have functions that were given to us, which in the case of SENSE are all GPS-related functions. These are: *MEO_M8U.m, UBX.m,* and *UBX_CLASS_ID.m*. On the other hand, we have a bunch of SENSE functions we created ourselves which can be separated into 4 categories, piCam, IR, Sonar, and General Use.

PiCam

The PiCam is the sensor with the most SENSE functions. For starters it has a base april-tag detector function (*detectAprilTags.m*) which senses where the april tags for docs are. Besides that, we also have a general *SENSE_piCam.m* function that is used when we pause and look around. Finally, we have specific *SENSE_piCam_Dock.m*, *SENSE_piCam_PayloadDock.m*, and *SENSE_piCam_Waypoint.m* which are used to create movement curves for when were looking for the specific things that the titles of the functions say.

IR

On the other hand, the IR functions are very simple. We only have one function for sensing IR, called *SENSE_IR.m* which senses distances using our five IR sensors and creates a movement curve to evade objects close to the rover.

Sonar

Sonar is very similar to IR. We only have one function for Sonar sensing (*SENSE_Sonar.m*) which senses to both sides of the rover and creates movement curves to avoid walls/objects if they're any walls to the sides of the rover.

General Use

Finally, we have one big general use function which is used by pretty much all of the SENSE functions: *createBellCurve.m*. This is a special function we created that receives a point from 1-60 and creates a bell curve with that point being the maximum. We use it to create movement curves that define where our rover is going to move.

THINK

The biggest section of functions we have, and the one we had the hardest time developing, is the THINK functions. We have one general $THINK.m$ function which is the one we call on every Robot Loop. This function uses the current roverBehaviour defined by the MDF to choose which specific THINK function is actually going to be run.

Before we go into the specific THINK function it is worth explaining what an MDF is and how ours is used. MDF stands for Mission-Definition File and what we have done is create a table that defines the different behaviours our robot has to do during our demo. Then, we have a *WPn* variable that defines what the current roverBehaviour is. This way, we can have the rover do multiple things with the same code, we just need to change what *WPn* is.

You can see an image of our MDF table below.

1 WayPoint	2 xLongitude	3 yLatitude	4 Behavior
1	42.2763	-71.2279	PauseDock_____
2	42.2763	-71.2279	NavigationDock_
3	42.2763	-71.2279	FindDock_____
4	42.2763	-71.2279	Dock_____
5	42.2763	-71.2279	PauseNav_____
6	42.2763	-71.2279	NavPayload_____
7	42.2763	-71.2279	PayloadFindDock
8	42.2763	-71.2279	PayloadGoDock__
9	42.2763	-71.2279	PayloadDrop____
10	42.2763	-71.2279	PayloadFind____
11	42.2763	-71.2279	PayloadPickup__

Back to the THINK functions, there are two sections of THINK functions that can be chosen by *THINK.m* which are the same two sections our MDF is divided into: Navigation, and Payload.

Navigate

There are four navigation THINK functions in our MDF:

- **PauseDock_____**, which calls *THINK_pause.m*, which stops the rover and has the piCam look around and turn to whichever way we should be looking.
- **NavigationDock_**, which calls *THINK_navigate.m*, which is the main navigation function, using piCam, Sonar, and/or IR sensing to move around the Olin O.
- **FindDock_____**}, which calls *THINK_findDock.m*, which looks around sensing aprilTags to point our rover towards the dock once its been found.
- **Dock_____**}, which calls *THINK_go2Dock.m*, called only once *THINK_findDock.m* has found a dock, it is used to actually get our rover into said dock.

There are many important things about these previous functions that are worth mentioning. First of all, besides deciding where and how fast the robot should move, they also define what the roverBehaviour should be in the next loop. They do this by changing WPn and they do this in respect to the current *WPn* which will come in handy in a second. The second important thing about these functions are that they aren't specific to the navigation MDF section. Both *THINK_pause.m* and *THINK_navigate.m* are used in Payload missions, they just are in a different position in the MDF so they go to different roverBehaviours afterwards.

Payload

Talking about the Payload THINK function, we have many of them, which can be seen listed below:

- **PauseNav_____**, which calls the same *THINK_pause.m* function as above.
- **NavPayload____**, which calls the same *THINK_navigate.m* function as above.
- **PayloadFindDock**, which calls *THINK_findPayloadDock.m* a function very similar to *THINK_findDock.m* which looks for the smaller Payload Docks instead of the big rover ones.

- **PayloadGoDock__**, which calls *THINK_go2PayloadDock.m*, which has the same structure as *THINK_go2Dock.m* but instead of going to a bigger rover dock it goes to the smaller Payload docks.
- **PayloadDrop____**, which calls *THINK_dropPayload.m* which takes care of dropping the payload into the dock .
- **PayloadFind____**, which would call *THINK_findPayload.m*, this was one of our stretch goals to be able to find a pickup Payloads around the track.
- **PayloadPickup____**, which would call *THINK_payloadPickup.m*, this was one of our stretch goals to be able to find a pickup Payloads around the track.

ACT

Finally, we have our ACT functions. As mentioned above on the Wiring section there are 5 servos on our rover. We have the servos that move the rover itself (speed and steering), the servos that move our piCam (pan and tilt), and finally the servo that controls our grabber to drop payloads. This means that we have separated our ACT into three functions that each control one of these three types of servos: *ACT_moveRover.m*, *ACT_moveCam.m*, and *ACT_movePayload.m*.

Actual Development

The actual development of this code was interesting. For the first week we started by creating the paths and structures for all of the functions mentioned above, and spent a lot of time understanding the Demo code we were given, and adapting it to our needs. Afterwards, we started by developing our ACT functions. We decided to start here because once those were working we would be able to drive the rover around with the joystick, which was already a huge step. Once we had these functions working, we moved first into SENSE and then into THINK, but not completely linearly. We started with crucial functions and then separated our team to get these done at the same time.

Besides the function, our code also has a big folder called **test_code** where we have multiple tests for different function (both SENSE, THINK, and ACT) which we used constantly throughout the project to make sure our functions were working properly.

Final Demo

The night before our demo, we lost a lot of servos and sensors. Our steering servo burnt out the night before so we had to take the whole rover apart to fix the RC setup for steering. This also meant we had to re-calibrate most of our movement the night before. This did not change anything drastically, but it did affect some of our accuracy (because we just didn't have time to calibrate as thoroughly as we had before). One of our Qwiic connectors seemed to falter, and it only output a string of 0 volts read to our pi, which we struggled to debug. We tried re-wiring it, changing the sensors, everything, but even the

potentiometer was reading 0 volts instead of the normal 3. On the actual demo Day we were able to replace our Qwiic connector but did not have time to test it, so we decided to keep it as a backup but not rely on it working. During the demo, MATLAB crashed multiple times, but overall we were proud of our accomplishments.

On the first mission (Navigation around the O) our rover didn't do very well. It had trouble sensing the waypoints as we expected him to, and ran slower than we expected (due to MATLAB issues that eventually fixed themselves), so we didn't get very far. For Mission 2 (Docking) we had planned to do the whole navigation from Mission 1 to get to the dock we needed to reach for Mission 2. Thus, we were able to redo and showcase our waypoint following code, and our obstacle avoiding code which had failed the first time through, and it docked almost perfectly. Finally, on Mission 3 we had problems with our computer that was running our code, so we had to change laptops, but we were able to get the rover to run up to the dock and open the grabber we designed to drop the payload. For some reason it didn't actually drop, we believe this might be due to the servo needing re-calibrating after all the wiring we had to change last second, but you could see that the rover had recognized the dock and AprilTag properly.

Overall, I'm very happy with how the Demo went, and even though it wasn't perfect, our Rover was robust, the wiring was easy to access for all of our last minute troubleshooting, and the code worked close to what we expected, even through all the last minute changes/fixes we needed to do.

My Contributions to The Team

My main job on the team was leading the mechanical design and fabrication. I created and organized the GrabCAD that managed all of our 3D modeling files, Solidworks, .stls, and more. I helped make most of the body parts and the laser cut and shopbot sheets. I manufactured and assembled both the laser cut and final versions of the car, from cutting the sheets, organizing the parts and attaching them together and onto the car. On my own, I designed, printed, and iterated on all of the 3D printed mounts and parts on the car, including the lidar array mount, pan-tilt servo mount, pi mount, battery mount, pi cam mount, and the grabber. I also helped paint and decorate the rover.

Beyond mechanical work, I was present for most of the testing, and helped my teammates with troubleshooting and debugging, as well as disassembling and reassembling the rover to rewire components.