Miniature Mars Rover

FINAL REPORT

Team Number: sdmay19-09

Client: Iowa State University
Adviser: Dr. Zambreno

Team Members/Roles:

Calvin McBride:	Hardware/Software Developer, Communication, Status Reporter, Meeting Note Taker
Sam Westerlund:	Lead Software Developer, Hardware Developer, Communication
Mitchell Freshour:	Lead Hardware Developer
Sam Oswalt:	Hardware Developer

Team Email: sdmay19-09@iastate.edu
Team Website: sdmay19-09.sd.ece.iastate.edu

Revised: 2019-4-29 Version 1

Table of Contents

To help spur interest in Iowa State University's engineering programs and catch the interest of prospective students, ISU's ECpE department has devised several display pieces that demonstrate the engineering prowess their senior students wield. One of these projects is the Miniature Mars Rover project, a miniature 6-wheeled mars rover model that requires knowledge of software, mechanical, and electrical engineering to construct.

Our team's primary goal is to get the rover (based on JPL's Open-Source Rover project) up and running, with secondary but preferred goals being the use of computer-visual and machine-learning to navigate, classify its environment, and possibly handle objects. It is our hope that these features, along with the ability to remote-control the rover, will help catch the interest of visiting prospective students, as well as give future senior design teams a well-built platform to extend for their own projects.

The information provided in this document includes instructions on how to operate the rover, design choices, and explanation of the components that make up the rover.

1. Requirements

The client is expected to connect to the wi-fi (the same one used by the Mars Rover software). Once on the wi-fi, the client enters the IP address of the rover in their browser of choice. The client is then presented with the client facing UI to interact with the backend component to control the Mars Rover.

1a. Use Cases

Our end-product has two different groups of users: student/faculty drivers and senior design teams.

The first group consists of potential students (mostly young tech savvy adults), interest groups, and faculty who will demonstrate the rover's features. This group will utilize the rover to see first-hand what can be built and what has been built by Iowa State University engineering students. Code will not be shown to this group, only the product of the code. They will also have access to view the image classification and manual control of the rover. The control and data view is a simple intuitive web interface, with instruction provided to not confuse or hamper the experience of the user. Overall, we want to impress this group with a well rounded finished product.

The second group are senior engineering students taking Senior Design who are tasked with extending the rover's functionality for their own senior design project. This second group is expected to have technical knowledge, contrary to the first group who are expected to have minimal technical experience. This group will start where we left off, and to ensure a fast transition, well written documentation will be provided.

1b. Functional Requirements

- Raspberry Pi reads input from an external web interface, controls hardware, and processes data.
- Camera streaming and labeling of objects in view to the user.
- Working camera for computer-vision.
- Working sensors LIDAR and camera to detect and map it's surroundings.
- Implementation of a neural network to detect common indoor objects and people.
- The end will manually control the Mars Rover through the browser at the same page of the camera input. This page will have simple buttons that will move the rover in that direction.

1c. Non-functional Requirements

- Battery life must have a minimum of 2 hours.
- Rover should take no more than 1 second to respond to manual user input.
- Remote-control signals should reach several meters, ideally throughout most of the a given building.
- Python will be used as the programming language to control the hardware. React will be used as the web interface.
- The rover will be able to handle inclines at 15 degrees, and uneven terrain where the disparity from the left to right wheels is less than 1 inches.

2. System Design & Development

2a. Design Plan

We decided to use the JPL Open-Source Rover as the starting point for our miniature rover. Other robot chassis were considered, but JPL's rover was well-documented, open-source, and thoroughly tested, making it ideal for our team. Our rover will most-likely use the Intel Movidius Compute Stick as a small but functional VPU, upon which many computer-vision and machine-learning tasks will be executed. Our team is currently looking into camera systems that will satisfy our computer-vision needs, as well as other sensors such as LIDAR being considered.

2b. Design Objectives, System Constraints, Design Trade-offs

The JPL Open-Source Rover project has been tested and verified by a much more technically knowledgeable and experienced team of engineers, and we feel an in-depth analysis of the base rover design is not necessary beyond the testing laid out by the JPL in their testing and calibration documents included in their documentation. This includes testing for the printed circuit boards, calibration of the motor controller units, as well as testing for connecting and operating the rover using the provided software. We have compiled what we believe are accurate strengths and weaknesses in our proposed design:

Strengths:

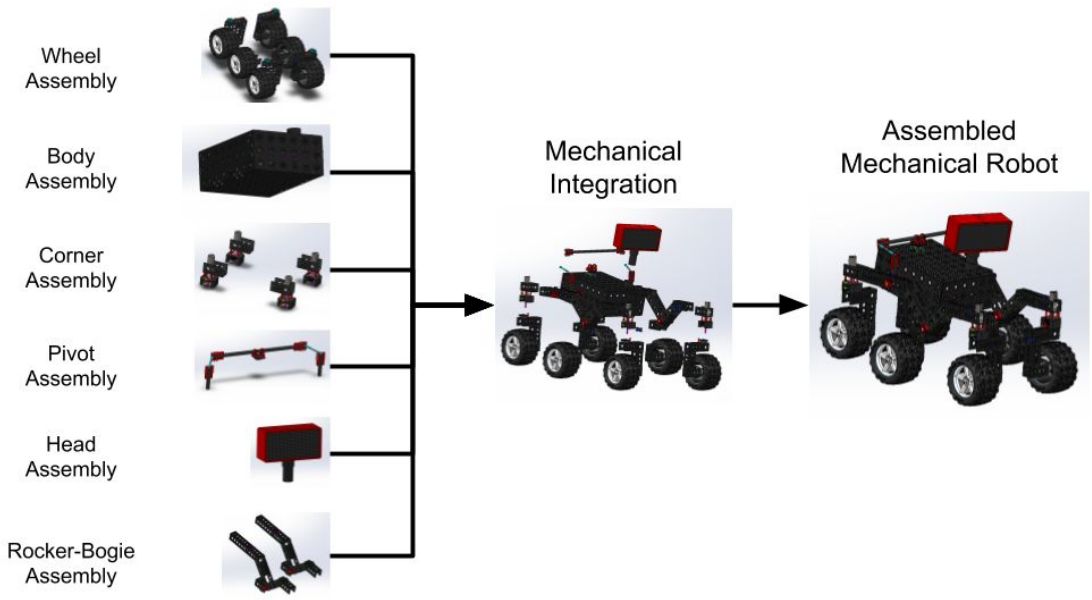
- Well-tested and thoroughly vetted rover base/starting point.
- Relatively cheap and easy to understand.
- Several sensors for accurate and thorough environmental analysis by rover.
- Raspberry Pi is well-documented, making tasks like computer-vision and machine-learning easy to start and develop with.

Weaknesses:

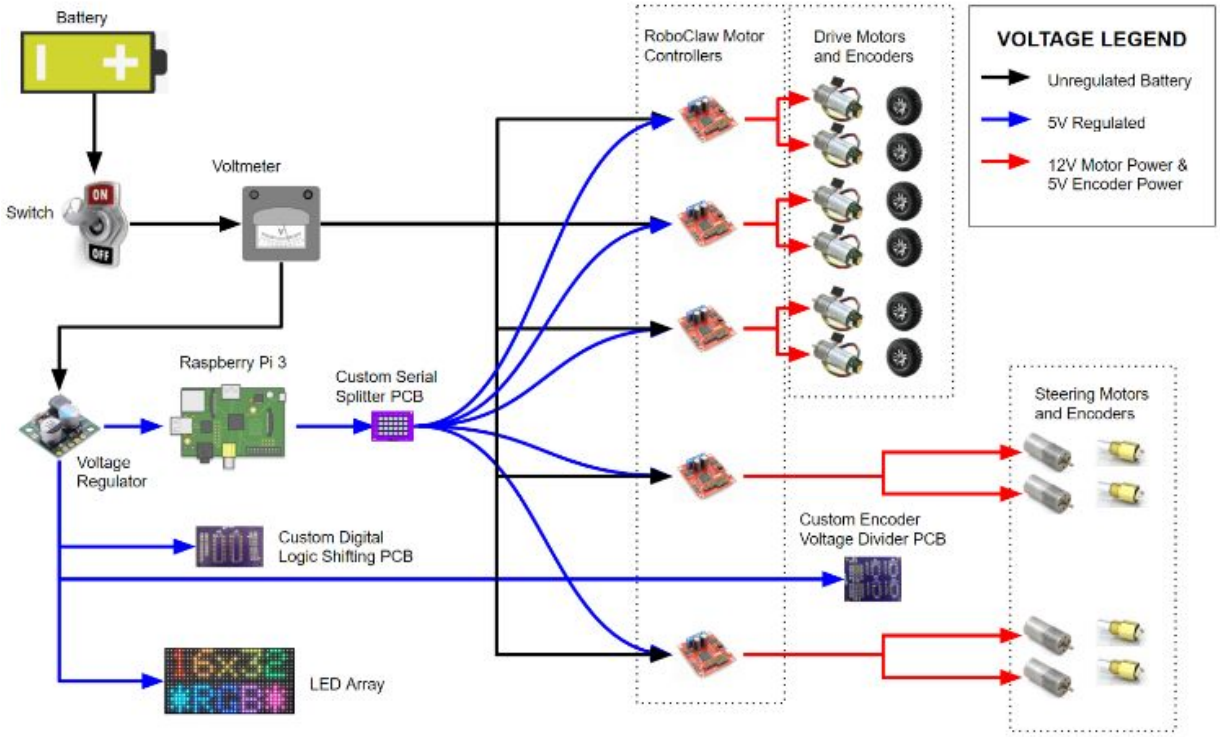
- Many electrical components are not vacuum-sealed, making them susceptible to bad weather and water.
- Rover construction involved many construction and fabrication techniques that our group is unfamiliar with, and also states a time of 200 experienced man hours to construct.
- Computer-vision is a relatively young field in computer science, making it a somewhat difficult task to implement for our junior team.
- Computer-vision and machine-learning is a relatively high cost in terms of processing power, making the rover's 3-4-hour battery life even shorter. Further issues arose with power delivery and a powered USB port will have to be implemented by a future team.
- RPLidar draws too much current for the Raspberry Pi's limits.

2c. Architectural Diagram, Design Block Diagram -- Modules

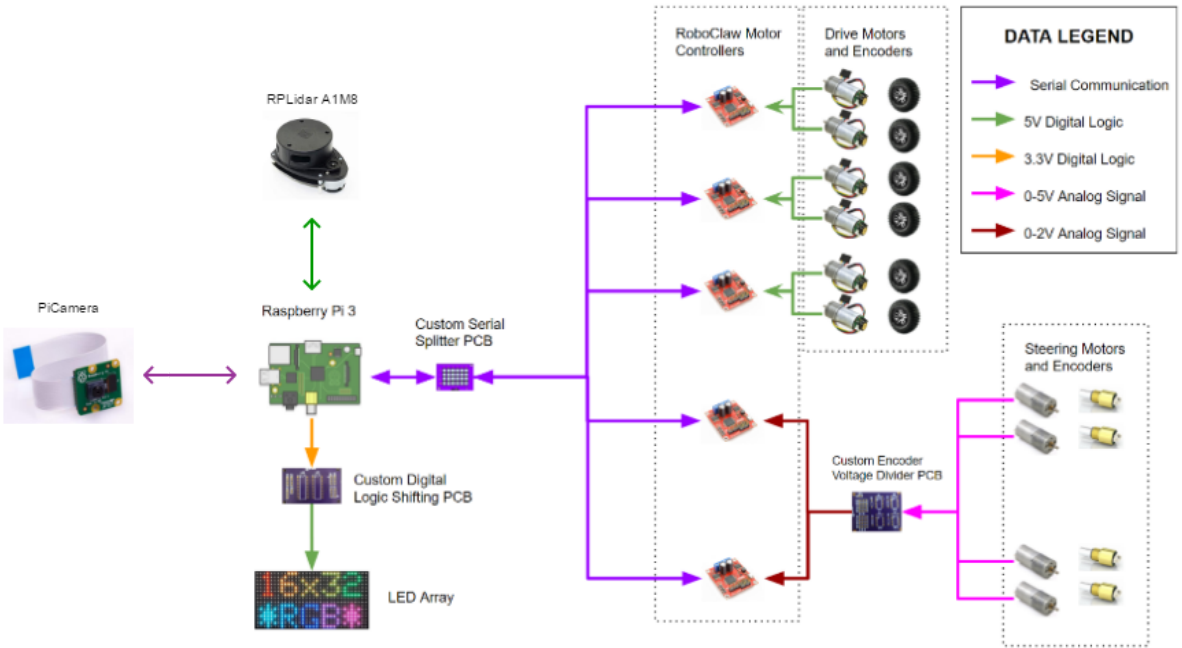
Mechanical System Diagram[6]:



Power System Diagram:[5]



Data Transfer Diagram:[5]



Software System Architecture

- The Raspberry Pi hosts both the UI and the Rover Control service. The services are set to start automatically when the Raspberry Pi starts.
- The Lidar device is connected to a USB port.
- The motor controllers are connected to the GPIO ports on the Raspberry Pi.
- The Intel Movidius Compute Stick is connected via a USB port (not currently in use).
- The LED Matrix is currently not connected but the software is already implemented with the NASA JPL iteration.

2d. Description of Modules, Constraints, and Interfaces

All the software runs on the Raspberry Pi. There are two major components to the software, the UI service and the Rover control service (diagrammed in section 3). The client only has capability of connecting to the rover through the UI service on their web browser. We choose to add the client facing UI because it presented a good way to stream the camera images as well as add additional aspects of controlling the rover and visualizing the data from the rover.

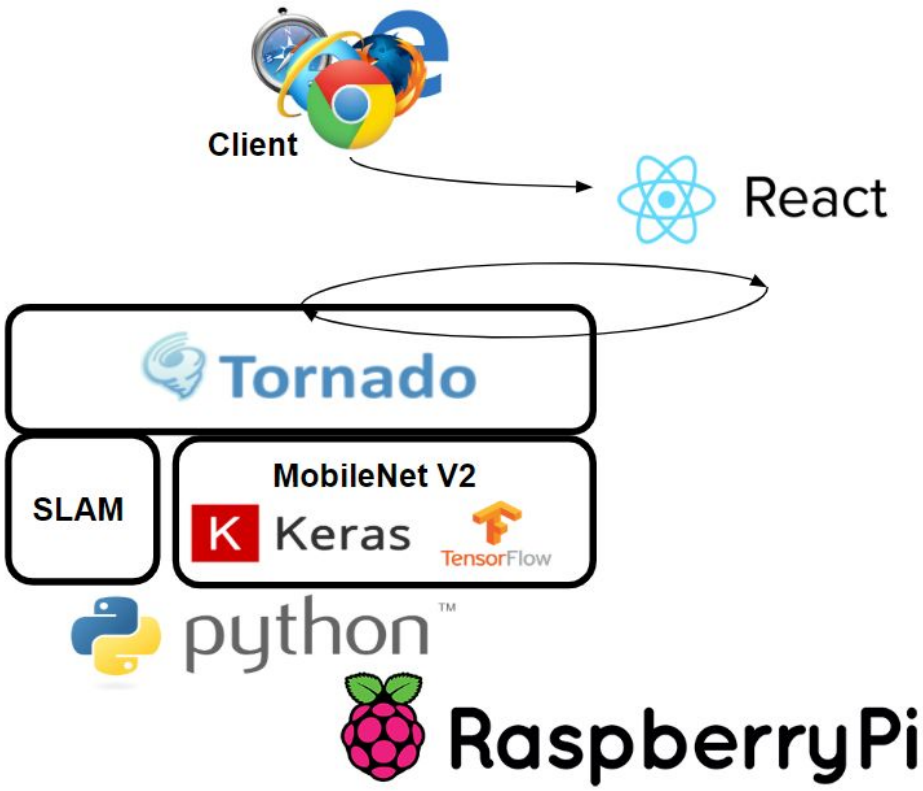
The hardware in the rover handles power delivery and data transmission to all the components of the rover. Power goes through a voltmeter so that the user can observe the battery's output. A terminal block is then used to split power from the battery to all of the motor controllers, as well as to a voltage regulator. This regulator provides a 5V output to the Raspberry Pi (see power system diagram, 2c). The Raspberry Pi sends commands out to a custom serial splitter PCB. This then splits commands to the 5 motor controllers. These motor controllers give the command out to their motors to either drive or turn our steering arms to corner. For the corner motors, we also have an absolute encoder which passes it's position back to another customer voltage divider PCB. This PCB translates the 5V signal from the encoder down to the 2V that the motor controller expects. The LIDAR and the PiCamera both plug directly into the Raspberry Pi(See Data Transfer Diagram, 2c). A constraint that was not addressed was the inability to provide the LIDAR with enough current from the Raspberry Pi to produce an accurate map of our surroundings.

3. Implementation

3a. Implementation, Technologies, Software Used

Software

- Client:
  - Any browser that supports HTML5 and has JavaScript enabled
- UI Service:
  - React.js
- Rover Control Service:
  - Python
    - Keras
      - MobileNet V2
    - Tensorflow
    - Tornado (websockets)
    - SLAM
  - SSH



Hardware

- Mechanical:
    - Wheels
    - Body
    - Corner Steering
    - Rocker-Bogie
    - Pivot
- Power: (see power system diagram, 2c)
    - 77Wh lithium ion battery
    - Voltmeter to display information to operator
    - Terminal block to split power to motor controllers
    - Voltage regulator to send 5V to Raspberry Pi for power
- Motors:
    - 10 Brushed 12V DC motors
        - 6 drive motors, 4 corner motors
    - 5 RoboClaw Motor Controllers
        - 3 drive boards, 2 cornering boards
- Data Transmission: (see data transfer diagram, 2c)
    - Serial Splitter PCB
        - Splits commands sent by raspberry pi amongst motor controllers
    - Voltage Divider PCB
        - Translates 5V signal from absolute encoder to 2V signal to motor controller

Electrical wiring and integration was conducted in accordance with provided JPL electrical build guide[5].

3b. Rationale for Technology/Software Choices

Since we used the platform created by NASA's JPL we didn't have to make many design choices. Software wise we chose to go with React as the client facing UI because it is fast and easy to set up. Python was already chosen as the language for controlling the rover so we built on top of that using Tornado for websockets and Keras since they are free libraries available in Python. MotorClaw controllers provided us with two input controllers that could utilize both serial communication, and allow for the use of absolute encoders.OshPark PCBs were used at the recommendation of ETG with designs provided by the JPL Documentation[2]. Our RPLidar system was chosen as it provided good range and rate at a reasonable price, and also advertised Raspberry Pi connectivity.

3c. Applicable Standards and Best Practices

Information Technology-ACSE and Presentation Layer Services-Application Program Interface: This standard defines a primitive interface for connecting to other layers of software. The standard divides the software into several sublayers that include a presentation layer and ASCE layer. It requires the functionality of accessing from a nonosi application and an osi application. It includes layers for application specific services, upper layers, and data transport. The purpose of the standard is to provide an organizational structure for creating an API. This standard was used in the creation of the interaction between the client and the rover using websockets.

IEEE Standard for Software Reviews and Audits: This standard defines the procedures and the process of conducting software reviews and audits. There are five different types described in the

document these are management reviews, technical reviews, inspections, walk-throughs, and audits. This standard is to be used throughout the life cycle of the software, but preference to early adoption is preferred. Reviews can be conducted by internal people or outside people. The general requirements for the review include team participation, documentation of the results, and documented procedures for conduction of the review. Steps for the 5 subtypes include introduction, responsibilities, input, entry criteria, procedures, exit criteria, and output. This standard was used to conduct reviews on the software that was created during this cycle.

Systems and software engineering -- System life cycle processes: This standard creates a framework for software cycles. The standard can be implemented at any part of the project at any time. But preference towards early adoption and planning. The stakeholders/customers are also considered. The goal of the software cycle is to achieve customer satisfaction. This method seems very similar to Agile. The whole purpose of the standard is to ensure that everyone is communicating and documenting, this will allow the company to grow with its own established environment and processes. The standard can be used by a project, organization, supplier, or process assessors. This standard was used and we commonly received feedback from the client and demoed the software often when we made changes.

4. Testing

4a. Test Plan

Testing our design utilized several hardware and software testing suites. Hardware testing was done with equipment located in Coover Hall's Senior Design lab, and software testing was done on university workstations and personal computers using software that is free/open-source. Hardware tools included multimeters calibration software, while software tools will include IDEs.

Testing the accuracy of the neural network model was done via real world testing. Since we are not training the model, it was not given validation data to tests against, only a test scenario was used.

Testing of electrical hardware was done in accordance with the provided documents by the JPL. One document covers testing for the printed circuit boards required for the rover's operation, to ensure that signals are split and sent to the motor controllers and that voltage is properly translated between absolute encoders and motor controllers[4]. The other document covers calibration and testing of the motor controller units to ensure they are providing the motors with proper voltage in order to move, as well as transmitting data about the drive and corner motor's positions to be used by the software[3]. Both were used to ensure the functionality of the motors, and the proper delivery of power and data between the motors, the boards, and the Raspberry Pi.

As we dove further into the project, the testing needed was fairly straightforward, as much of it was documented within the Github files for the rover. We needed to test each PCB board individually before mounting them to our board. Then, after also mounting the roboclaws, we needed to hook them up to their respective motors and test each of those individually as well. Efforts needed to be made to keep track of motor/roboclaw pairings, so as to avoid potential unwanted complications later on.

4b. Unit Testing

The rover was built with expandability in mind. At the current moment most of the software components remain partially implemented yet functional. Because of the limitations on the amount of time available to work on software, unit testing was not done and some of the UI components are not as visually pleasing. System tests and code checks were the methods performed to check the software. This was acceptable because not very many individual methods were created and the end result of virtually all these functions working properly were demonstrated by the system tests passing.

4c. Interface & System Integration Testing

Power system testing was done during integration to ensure that proper voltages and currents were distributed throughout the system. This consisted of verifictions using multimeters as we integrated each part. This ensured that at least 12V was given to the terminal block, all motor controllers, and to the voltage regulator. The voltage regulator was also verified to provide 5V to the Raspberry Pi using a multimeter. Motor controller and motor testing is conducted during motor controller calibration in accordance to JPL documentation[3] using Basic Micro Motion Studio, as well as during software integration testing. Our custom PCB testing was conducted in accordance to JPL documentation[4]. During our testing we discovered that the gears on our right side corner motors had bent and snapped in places. This lead to the motor locking up and preventing us from cornering. This was not able to be remedied as shipping time of parts would push us past our delivery date.

Once the hardware was assembled the components were all tested with the software to ensure that they worked. The camera was also testing by streaming it to the client UI.

5. Project & Risk Management

5a. Tasks & Responsibilities

Our project was split into two primary areas of knowledge: hardware/electrical and software. Our group consists entirely of software and computer engineers, and at the start of the project we did not have any professional or academic experience with robotics, circuits, or electrical engineering work. Because of this, we felt it was best to split responsibilities so that most of the team focused on hardware and machining parts. Two members focused on wiring, testing, and calibrating components, another member focused on machining and assembling components, and the final focused on software development and testing.

- Mitchel Freshour: Hardware lead who focused on electrical circuitry, power delivery, and component calibration. His work focused on ensuring all wiring was correct and electrical components such as the motors, their controllers, and power systems were working correctly.
- Sam Westerlund: Software Lead who focused on extending the JPL Open-Source Rover code to support the project's modifications during the first half of the project. Primary tasks then were integrating the LIDAR and Raspberry Pi camera, as well as implementing the machine-learning algorithms to have the computer vision. He also worked on assembling the rover during the second half of the semester. Assembling tasks included soldering, some wiring, and physically putting the rover's parts together.

- Calvin McBride: Hardware and software developer who focused on machining, laser cutting, 3D printing, and assembly the rover itself. Assisted Mitchell with calibration and testing after the rover was assembled.
- Sam Oswalt: Hardware developer who focused on wiring and calibration alongside Mitchell. Sam joined the team halfway through the project and quickly got up to speed on the hardware aspects to ensure everything was wired correctly and working according to the JPL documentation.

5b. Project Schedule

Our project's schedule, though optimistic, has endured several setbacks related to team organizational changes and lack of hardware experience. Our initial deadline for the rover's final assembly was March of 2019; however, issues with calibration, needing to re-order parts, and busy schedules pushed this deadline back by approximately a month, resulting in the rover assembly only being completed at the end of April, leaving minimal time for testing or further software development.

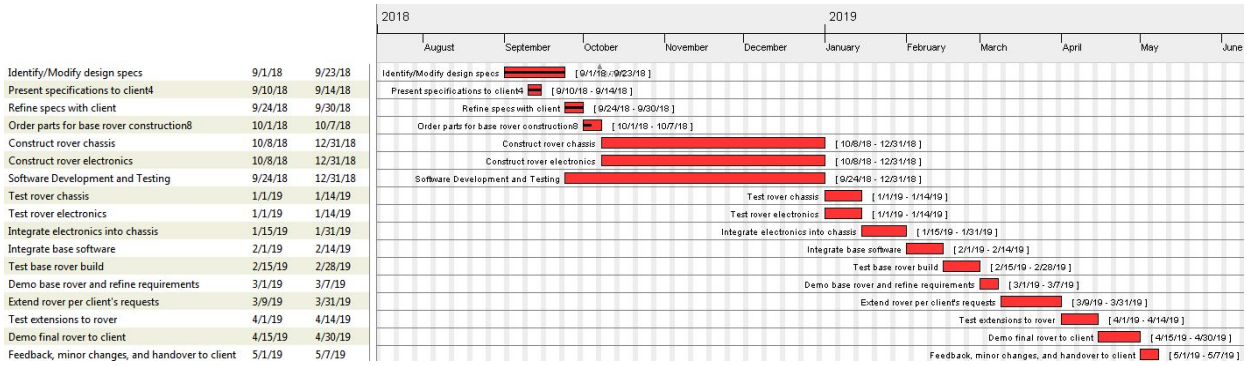


Figure: Initial Project Schedule

5c. Risks & Mitigation

At the beginning of the project, one of the primary risks we felt we were likely to encounter was actually related to our lack of knowledge related to electrical engineering and hardware machining/assembly. This has consistently been an issue we have had to deal with throughout the project, resulting in a slower working pace, dedicating time to learning machining and electrical assembly, and inaccurate project schedule estimates. Besides learning the subject matter as we progressed through the project, there was no way for us to pre-emptively mitigate this issue. Instead, we shifted our project end-goal to have a working or near-working miniature rover that would take minimal effort for future senior design teams to both finalize and expand upon.

5d. Lessons Learned

Our team has learned many lessons over the course of this project, such as the importance of coordinating schedules, realistic deadline estimates, and technical knowledge related to electrical and mechanical engineering. Each team member had a varying and different schedule that made meeting together all at once difficult outside of one or two days a week, and we saw productivity increase when we worked out a concrete schedule for when we would meet and work on the project. This project has also taught us the importance of giving realistic but cautious schedule estimates to avoid deadline pushbacks and team frustration. Our team also realized late into this project that

communication between teams to fully understand what stage our project was in is a critical component of good planning and team work flow.

6. Conclusions

6a. Closing Remarks

Our team set out to build a miniature mars rover that utilized computer vision to help autonomously navigate its environment. Its primary long-term design goal was to be well documented and easily extensible for future senior-design teams. We feel that we did not adequately achieve the goal of autonomous navigation or accurate mapping utilizing the camera or LIDAR system, but we do feel we have left the project in such a way that it can be easily completed and extended by future teams. We feel this project has been a useful learning experience for each team member, having learned important lessons over the course of starting this project from scratch to bringing it to near-completion.

6b. Future Work

For future work on software, future teams should be able to build on top of the current design. As of right now both right corner motors will not turn due to jamming or broken gears. This could not be solved within time before our delivery date as replacement shipping would take too long. Future teams will need to mount and calibrate these in accordance to JPL Documentation[3][5][6].

Notes on Expandability: The Mars Rover will need a USB powered hub to correctly run the Lidar system. The Raspberry Pi has a limit for current that the USB ports can draw and the Lidar exceeds this amount.

6c. References

RPLIDAR[1]:

https://www.slamtec.com/en/Support#rplidar-a1

Original NASA JPL github page[2]:

https://github.com/nasa-jpl/open-source-rover

Roboclaw Calibration[3]:

https://github.com/nasa-jpl/open-source-rover/blob/master/Electrical/Calibration.pdf

PCB Testing[4]:

https://github.com/nasa-jpl/open-source-rover/blob/master/Electrical/PCB%20Testing.pdf

Electrical Build[5]:

https://github.com/nasa-jpl/open-source-rover/blob/master/Electrical/Electrical%20Build.pdf

Mechanical Build[6]:

https://github.com/nasa-jpl/open-source-rover/tree/master/Mechanical

6d. Team Information

Mitchel Freshour **(Computer Engineering) - Hardware**

Mitchell is a Computer Engineering Senior. He has previously interned with the Iowa DOT where he worked as an Information Technology Administrative Intern in a group of 4 engineers to develop software for various motor vehicle administrative agencies, as well as provided data to various government agencies to aid in their policy research and studies. He is currently at a co-op with AgLeader in Ames.

Sam Westerlund **(Software Engineering) - Software/Hardware**

Sam is a senior in Software Engineering who plans to graduate in May 2019. Previously interned at Solum Lab in Ames, Iowa. He has worked on web applications and various other projects of his own in his past.

Calvin McBride **(Software Engineering) - Hardware**

Calvin is a software engineering senior at Iowa State University and plans on graduating May 2019. He has had two co-ops and two internships at Rockwell Collins, Inc., where he worked with teams of engineers to develop and test software for flight-display systems in aircraft. He currently works at Iowa State University's Web Development department developing websites and applications for ISU and its clients.

Sam Oswalt **(Computer Engineering) - Hardware**

Sam is a senior studying computer engineering at Iowa State University and plans on graduating May 2019. His past projects have primarily focused on software development.

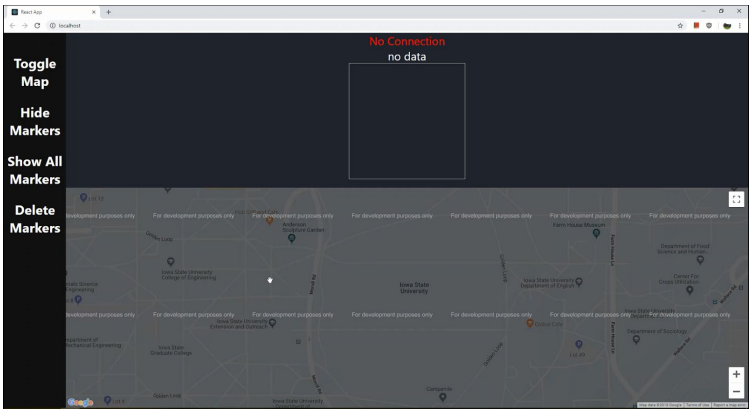
Appendix I

To power on the rover simply flip the switch located on the back right to the on position. Once the switch is flipped the voltage meter as well as various lights inside the rover will light up. Charging of the rover involves uncoupling the connectors outside of the battery, and connecting the charger to the battery.

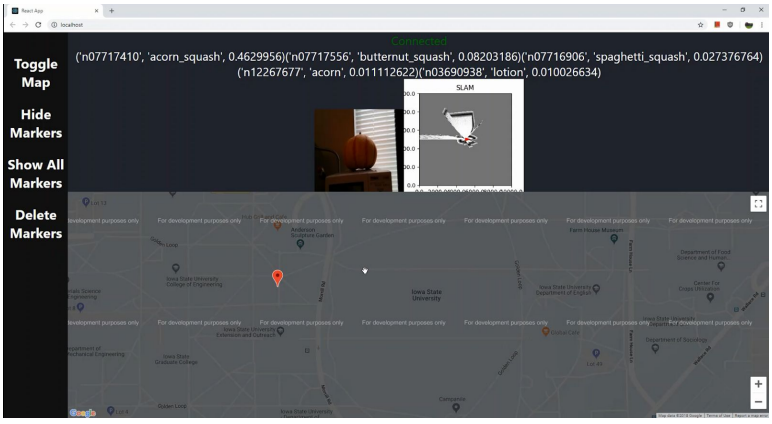
Software Setup

The software on the Raspberry Pi is set to run on startup. The user simply needs to power the rover on and wait a minute for the Pi to boot and initialize the software.

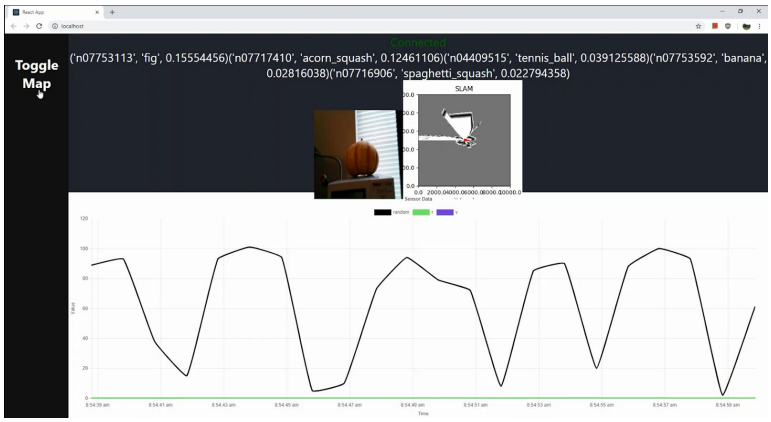
To connect to the software log in to the client computer. Once logged in connect to the WiFi called rover. Once connected enter the url address: “192.168.1.1:8080” into your browser of choice (chrome works best). The client is greeted with a web page where they can interact with the various components of the software. Initially it will say that it is not corrected but shortly afterwards the websocket will connect to the Python backend and it will begin loading data from the rover.



The client will be able to see the Lidar map, current image from the rover, and the classification from the image in plain text. The user also has the option to place markers on google maps. This is a feature for future teams to implement if they wish to use waypoints with GPS navigation.



To view the graph of current sensor/other data click on the navigation bar on the left. Pictured on the graph is random data from the rover as well as keyboard input data.



The user also has the ability to control the rover from the web interface. To control the rover the user simply needs to input keyboard controls. The arrow keys as well as the w, a, s, d keys control the forward, left, backward, right movements respectively.

If the rover fails to connect or the wifi disconnects (being out of range or other), the websocket will attempt to reconnect without the need of user interaction other than correcting the dropped connection.

Appendix II

Considerations to initial design was not made as the project provided a design for an already completed project. We assumed that the provided documentation would be sufficient for this project. Our alternate considerations mostly went into additions to be made to the existing design.

Appendix III

There were multiple errors in the NASA JPL build documentation. For example, the pivot motor shafts were the incorrect length. The pivot motors were in different positions at different image angles in the documentation which made assembly very confusing. There were also no pictures of the wiring and we had to guess on a lot of the specifics. Guessing on the electrical components was one of the hardest parts as we did not have any electrical engineers in the group.

Appendix IV

All code is available on the git repository provided by Iowa State University:
https://git.linux.iastate.edu/jtfree/com-s-362---chat-app