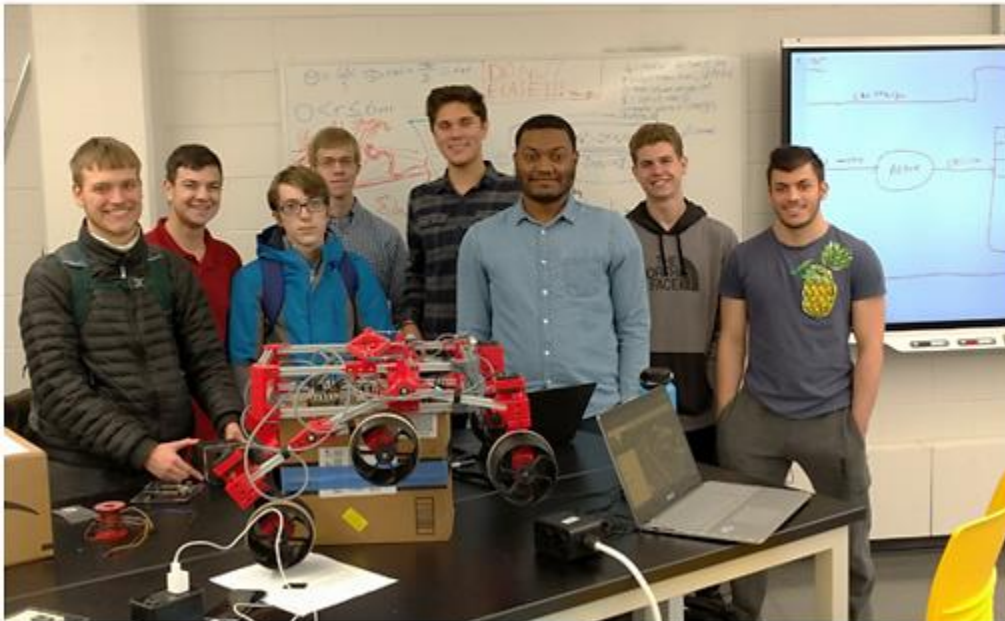


*An autonomous small scale of the NASA JPL Mars Curiosity Rover for object detection, collision avoidance and navigation on unknown terrains*

---



*Fall 2019, Spring 2020, Summer Workshop 2020  
Independent Studies in Engineering Research*

## Table of Contents

Introduction .....	4
Overview .....	5
Block diagram.....	5
Major Components .....	5
Mechanical Design Team 3 .....	7
Rocker-bogie design.....	7
Parts list.....	8
3D printed parts .....	13
Filament selection.....	13
Part modifications.....	15
Failures.....	18
Aluminum extrusions .....	20
Aluminum rods.....	20
Laser-cut acrylic .....	22
IMU shock mounting.....	24
Wheel traction .....	25
Electrical Design Team 4 .....	27
Electrical overview .....	27
Microcontroller considerations .....	27
Power considerations .....	28
Battery selection .....	28
Fuse size .....	28
Wiring considerations .....	30
Components.....	33
Servo and motor control.....	33
Microcontroller .....	36
Object detection sensors .....	36
IMU and GPS .....	38
Battery.....	39
PCB - Servo motor control board .....	39

Power Distribution .....	41
Microcontroller shield.....	44
Printed Circuit Board Layout, Microcontroller Shield Board .....	47
Software Design Team 1 .....	48
Architecture .....	48
Main .....	48
Libraries.....	49
Examples .....	50
Microcontroller and serials.....	52
Motor and controls .....	53
Object detection .....	56
Ultrasonic Sensor .....	56
LIDAR Sensor .....	61
Collision avoidance .....	61
Navigation .....	62
Localization .....	62
Heading and bearing tracking .....	62
Calculating bearing and distance .....	64
Calculating directional movement.....	65
Calibration.....	68
Addendum.....	68
Troubleshoot diagnostic .....	69
What points to test with? .....	69
How to generally troubleshoot rover .....	69
Wheel test incorrect wrong direction and or turn.....	70
Rover turns in place indefinitely .....	71
Rover heads towards wrong direction.....	71
Rover's bearing and distance match calculator, but still goes wrong direction .....	71
Object avoidance not turning in place and or going forwards .....	71
Testing code from home .....	72
Addendum.....	72
GitHub public repositories .....	72

FPGA Design for Object Detection Team 2 ..... 73

    Introduction ..... 73

    Chosing FPGA technology ..... 73

    Description of the image processing ..... 74

    Image processing diagrams..... 75

    References ..... 76

The NASA Mars Curiosity Rover Teams ..... 77

    Team 3 Mechanical Design ..... 77

    Team 4 Electrical Design ..... 78

    Team 1 Software Design ..... 78

    Team 2 FPGA Design ..... 79

## Introduction

*Fall 2019 – Spring 2020 Independent Studies in Engineering Research*

*“An autonomous small scale of the NASA JPL Mars Curiosity Rover using ordinary sensors and FPGA programmable logic blocks interconnected to microcontrollers for object detection, collision avoidance and navigation on unknown terrains. This is not an indoor project.”*

The purpose of the NASA JPL Mars rover is to demonstrate that a trivial amount of navigation information e.g. going from Point A (start) to Point B (end) can be achieved with ordinary sensors, FPGAs and microcontrollers. The design integrates object detection and collision avoidance during the autonomous journey. Other known designs incorporate microcomputers to process sizable amounts of data. The novelty of our project is the use of off-the-shelf components and embedded microcontrollers for autonomous operation.

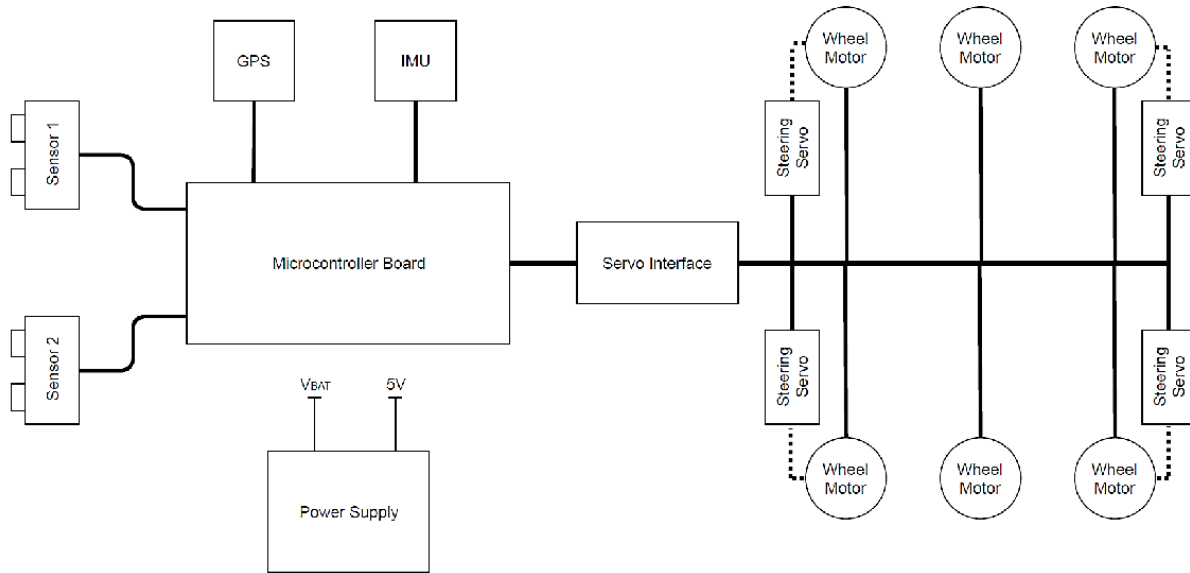
*The mechanical foundation is entirely 3D-printed at the college and takes about 300 hours of machine time per rover. It was designed by Roger Cheng and nicknamed “Sawppy the Rover.” The design was inspired by NASA’s Jet Propulsion Laboratory (JPL) Open Source Rover project. Most of the differences between Sawppy and its JPL inspiration were motivated by a desire to reduce cost and complexity. Sawppy, whose layout and proportions, mimics that of Mars Curiosity and Mars Perseverance. It faithfully reproduces the suspension kinematics of real rovers and is intended to be a hardware platform for future software projects in autonomous operation.*

The mechanical foundation is something our project didn’t design, but we built it at the college from Open Source libraries. Our project is totally Open Source and is funded by NASA through the Pennsylvania Space Grant Consortium (PSGC).



## Overview

### Block diagram



## Block Diagram

### Major Components

Components were added to the existing mechanical (chassis) design so the SAWPPY Rover could perform the following functions:

- Obstacle Detection
- Path Calculation
- Movement

### Obstacle Detection

One or more sensors (*Sensor 1* and *Sensor 2*) allowed the detection of obstacles which would impede the rover and allow the detection of a gap wide enough to let the rover pass through.

### Path Calculation

A *GPS* (global positioning system) board is used to determine the rover's starting position and aid in its determination of arriving at a destination. Because of the limited accuracy of the GPS (>10m) an *IMU* (internal measurement unit) was also used to supplement the path calculation



## Movement

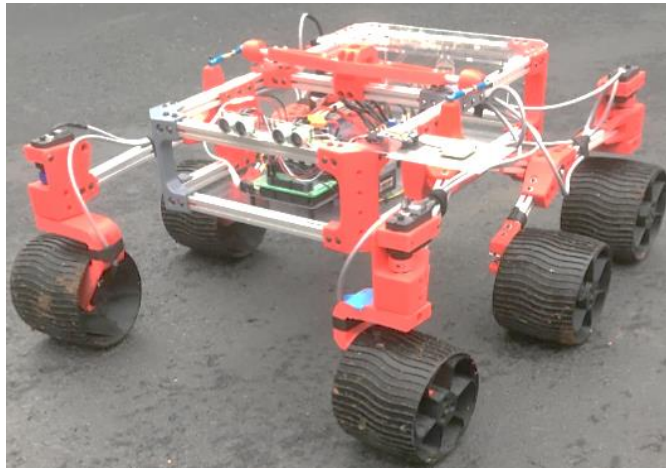
In general, the rover must be able to go forward, go backward and turn. These movements allow it to arrive at its final destination and maneuver around obstacles encountered on the way.

Movement is accomplished by 6 *Wheel Motors* and *Steering Servos* on the four corner wheels. These 10 devices allow the rover to perform a wide range of maneuvers. A *Servo Interface* allows control of each motor and servo.

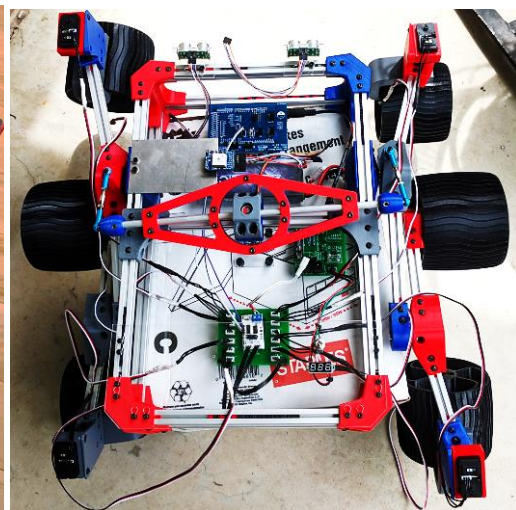
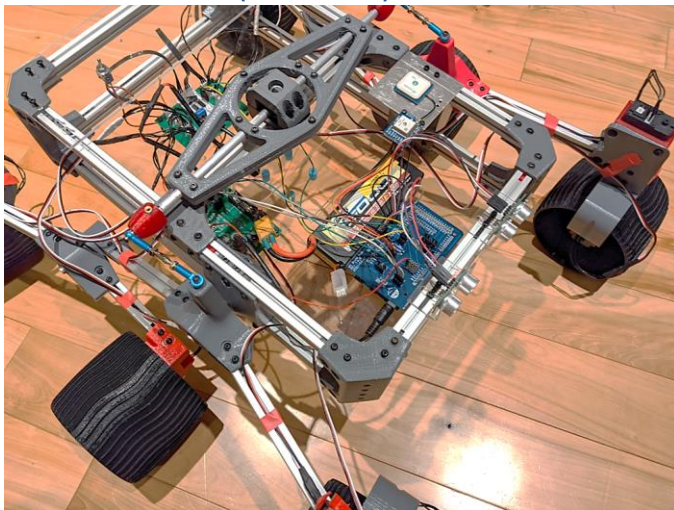
There were also some general-purpose components to support these functions.

- *Microcontroller Board*- executes the code and provides the required interface to the other devices
- *Power Supply*- provides the necessary voltages, limits the current draw of a fault and allows power to be switched off

## Rover 1 (Instructor)



## Rovers 2 and 3 (Students)



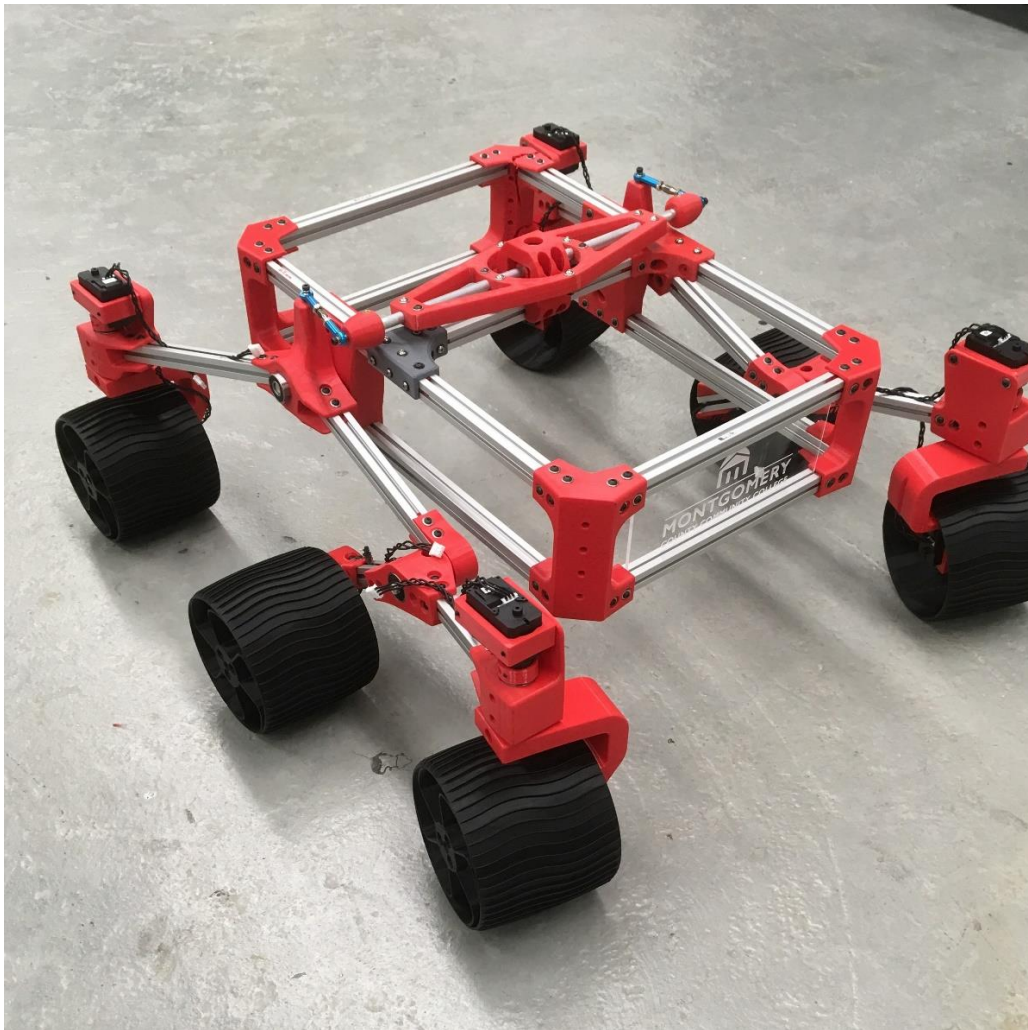
## Mechanical Design Team 3

*Ezra Galapo and Jaden Weed*

### Rocker-bogie design

NASA invented the rocker-bogie in 1988 for the use of the Mars Rover SOJOURNER. It has since been used in 5 other rovers sent to Mars. The interesting part about the rocker-bogie is that there are no springs or stub axles. This helps it go over different obstacles and allows it to tilt up to 45 degrees without tipping over.<sup>1</sup> Each of 6 wheels has its own motor which puts less strain on each motor. The front and back wheels each have a servo so they can be steered independently. This allows the rover to turn in place.

Figure 1



<sup>1</sup> <https://en.wikipedia.org/wiki/Rocker-bogie>



## Parts list

The list of mechanical parts to be purchased and manufactured for this rover to be built are below.

### Parts to be Purchased

QTY	DESCRIPTION	PART #	MANUFACTURER
10	Servo motors	LX-16A	LewanSoul
30	608 Bearings	608 2RS PRX	Bearings Limited
300	M3x8mm socket head screws	ZSS616M3X8	AMPG
28	M3x16mm socket head screws	ZSS616M3X16	AMPG
300	M3 washers		Amazon
300	M3 nuts		Amazon
100	M3 threaded inserts		Amazon
100	5/16" external retaining clips (E-Clips)	97431A310	McMaster-Carr
100	M3x8mm set screws	M51240.030.0008	Fabory
2	Turnbuckles	166617	Hobbypark
10	M3x16mm bolts		Amazon
4	1/4" long Philips #2 brass screws	98685A220	McMaster-Carr
1	1/8" acrylic sheet		McMaster-Carr
1	3/32" acrylic sheet		McMaster-Carr
4	Dampeners for IMU		Amazon
1	GPS External Antenna aluminum plate 7in <sup>2</sup>		

### Parts to be Machined

Order at least 845mm of 8mm Multipurpose 6061 aluminum shaft (rods) from McMaster-Carr

Manufacturer # 4634T34

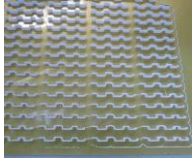
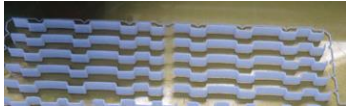


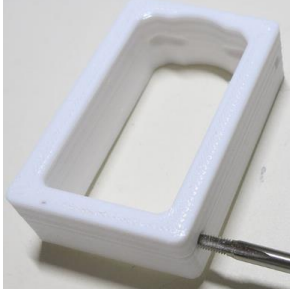
Order 300 mm of 8mm steel for differential shaft (cut to length?)



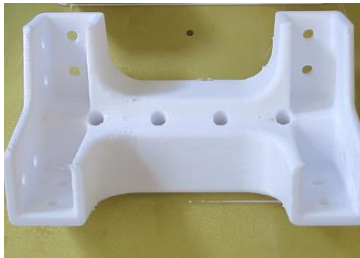
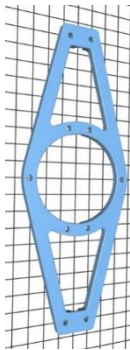
#### Cut list for 8mm aluminum rods




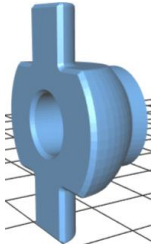
QTY	DESCRIPTION	LENGTH TO BE MACHINED (mm)
1	Differential Shaft* steel	300
6	Wheel Axle Shaft	50
4	Steering Shaft	61
2	Suspension Bogie Pivot Point	66.5
2	Suspension Rocker Pivot Point	84

#### 15x15mm aluminum extrusions to be ordered cut to length

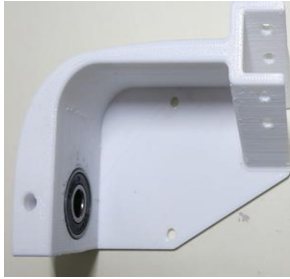




QTY	DESCRIPTION	LENGTH (mm)
4	Main body box, lengthwise	385
4	Main body box, widthwise	245
1	Main body differential fixed beam	238
2	Suspension member connecting rocker joint to front wheel	182
2	Suspension member connecting rocker joint to bogie joint	161
2	Suspension member connecting bogie joint to mid wheel	122

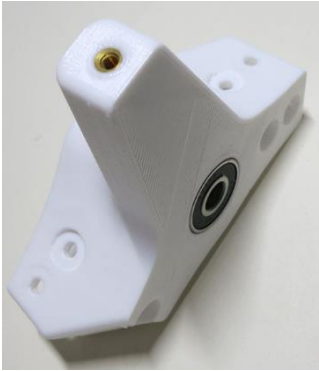

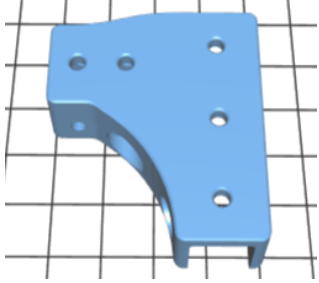
Qty	Part	Picture	Description
88	2 Nut Holder		<p>These hold two M3 nuts at the correct spacing for rover assembly.</p>
12	3 Nut Holder		<p>These hold three M3 nuts at the correct spacing for rover assembly.</p>
6	Wheel hub		<p>The wheels attach to this hub. A set screw engages a flat on the 8mm drive shaft.</p>
10	Servo-Coupler		<p>This connects the servo to an 8mm shaft.</p>
10	Servo-Bracket		<p>Mounts the servo is to the rest of the chassis.</p>

6	Wheel		
4	Steering Knuckle		<p>The steering knuckle connects the steering shaft to the driving shaft. It is a key part of rover suspension geometry.</p>
4	Body corner		<p>These make up the four corners of the main equipment bay.</p>
2	Differential Brace		<p>The differential on the rover is different from the differential in an automotive drivetrain. Instead of distributing different forces across different drive wheels, this differential distributes suspension forces across the two sides. It is a key part of how the rocker-bogie suspension keeps all six wheels on the ground.</p>

2	Differential Link		
1	Differential Lower		
1	Differential Upper		
2	Rod Support		Supports for the rod and the upper and lower differential.



2	Fixed Knuckle		<p>The two fixed knuckles connect the middle non-steerable wheels to the aluminum extrusions making up the suspension structure.</p>
2	Front corner		<p>These hold the steering actuators along with the steering shaft.</p>
2	Rear Corner		<p>These hold the steering actuators along with the steering shaft.</p>
2	Bogie Pivot		<p>Houses two 608 bearings and two aluminum extrusions, one connecting to mid wheel assembly and the other connecting to rear corner wheel assembly</p>
2	Bogie Body		<p>This will become part of the rocker assembly</p>

2	Rocker/suspension rocker joint		Houses the two bearings for the rocker assembly to pivot on, and connects to two aluminum extrusion beams. One leading to the bogie pivot point.
2	Rocker Body Mount		Connects the rocker-bogie suspension of one side to the main chassis equipment box.
2	Differential End		

### 3D printed parts

Most of the rover parts are 3D printed. There are 67 major parts. These took over 300 hours to print.

### Filament selection

One of the first decisions was which filament to use. We looked at PLA and PETG.

#### PLA characteristics

- Vegetable-based plastic material
- Very brittle
- Easy to take off of bed
- Good support structure
- Bio compostable

### PETG characteristics

- PETG is a modified version of PET (polyethylene terephthalate) mixed with glycol which makes it less brittle, clearer, more durable and impact resistant. PETG is extremely durable and prints without odor. Little to no warping, ideal for printing big stuff.
- Very strong, but can be scratched up very easily
- Very bad support structure, however very good layer adhesion, so the prints come out strong
- Great chemical resistance, along with alkali, acid and water resistance.
- Odorless.

### 3D Printing filaments comparison



### 3D PRINTING FILAMENTS COMPARISON

	PRINTING TEMPERATURE	BED TEMPERATURE	PRINT DIFFICULTY	STRENGTH	FLEXIBILITY	DURABILITY	RESISTANCE TO TEMPERATURE	SOLUBLE	FOOD SAFE
PLA	200 220 240	0 60	● ● ●	● ● ●	● ● ●	● ● ●	60	✗	✓
PLA 3D850	200 210 220	0 60	● ● ●	● ● ●	● ● ●	● ● ●	75	✗	✓
PLA 3D870	200 210 220	0 60	● ● ●	● ● ●	● ● ●	● ● ●	75	✗	✗
EASY PRINT	190 200 210	0 60	● ● ●	● ● ●	● ● ●	● ● ●	55	✗	✓
WOOD	200 220 240	0 60	● ● ●	● ● ●	● ● ●	● ● ●	60	✗	✗
BOUN	210 220 230	0 60	● ● ●	● ● ●	● ● ●	● ● ●	65	ACETONA	✗
ABS	230 240 250	80 100	● ● ●	● ● ●	● ● ●	● ● ●	100	ACETONA	✓
ABS HIGH IMPACT	230 240 250	80 100	● ● ●	● ● ●	● ● ●	● ● ●	100	ACETONA	✗
ABS FIREPROOF	210 220 230	80 100	● ● ●	● ● ●	● ● ●	● ● ●	95	ACETONA	✗
ABS MEDICAL	230 240 250	80 100	● ● ●	● ● ●	● ● ●	● ● ●	100	ACETONA	✓
FLEX	215 225 235	0 100	● ● ●	● ● ●	● ● ●	● ● ●	105	✗	✗
HIPS	225 235 245	80 100	● ● ●	● ● ●	● ● ●	● ● ●	100	D-LIMONENO	✓
PETG	215 235 255	60 90	● ● ●	● ● ●	● ● ●	● ● ●	85	✗	✓
PP	210 220 230	60 100	● ● ●	● ● ●	● ● ●	● ● ●	60	✗	✓
NYLSTRONG	245 255 265	95 110	● ● ●	● ● ●	● ● ●	● ● ●	210	✗	✗
GLACE	205 220 235	70	● ● ●	● ● ●	● ● ●	● ● ●	75	ALCOHOL	✗
CLEAN	190 220 250	—	—	—	—	—	—	—	—
SUPPORT	210 240 270	70 100	● ● ●	● ● ●	● ● ●	● ● ●	—	D-LIMONENO	✗

### PLA (polylactide) vs. PETG (polyethylene terephthalate)

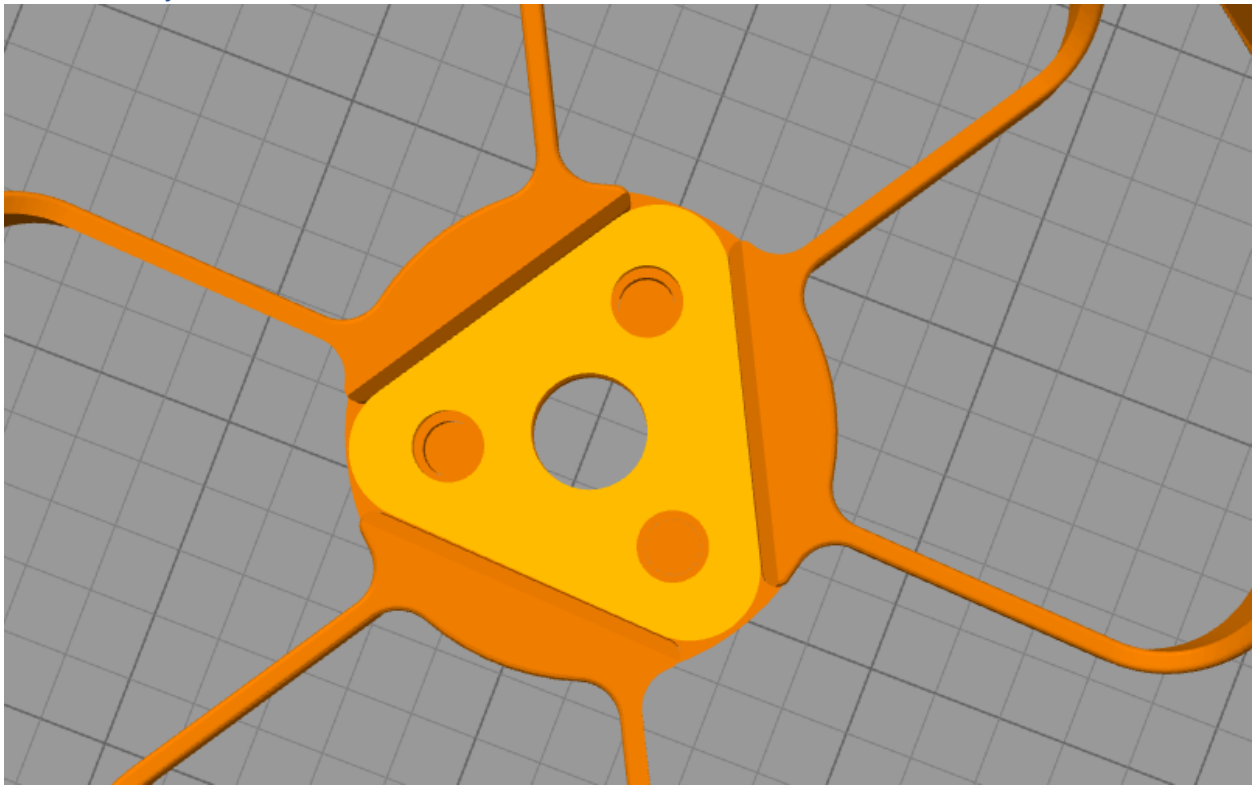
- PLA is more brittle than PETG, unless you strengthen it.
- PLA and PETG have very similar densities.
- PETG needs a heated bed, whereas PLA can be printed cold.
- Layer adhesion with PETG is typically unmatched, leaving very strong and durable prints.
- PLA prints supports that are easy to remove, whereas these are harder (but not impossible) to remove with PETG.

Overall PETG was used for the durability that it has over PLA.

### Part modifications

There was a problem with the wheels and the wheel hubs. The wheel hub was too big for the wheel. There were two solutions: make the wheel hub smaller or make the wheel bigger. Since the wheel did not contain any other part, it was made smaller. A 3% larger wheel worked.

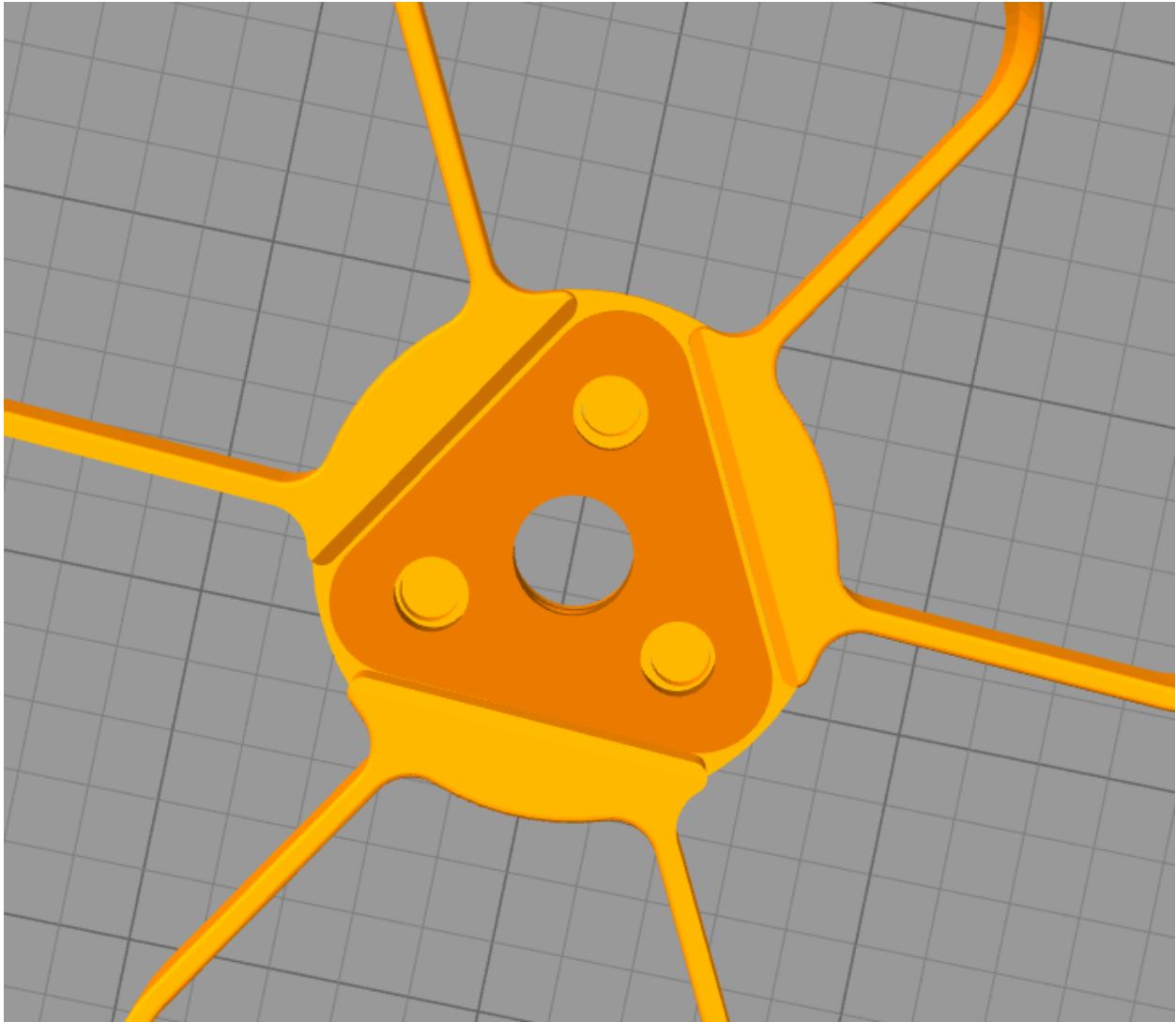
#### Before the adjustments



Originally it was a tight fit and very hard to take off.



After the adjustments



With a 3% larger wheel it can be more easily removed.

PETG wheels needed multiple trials to get the printer setting right. Slowing the print speed was ultimately used to produce quality wheels.

### First Attempt



The quality was terrible, but once the printing speed was slowed, the quality improved immensely.

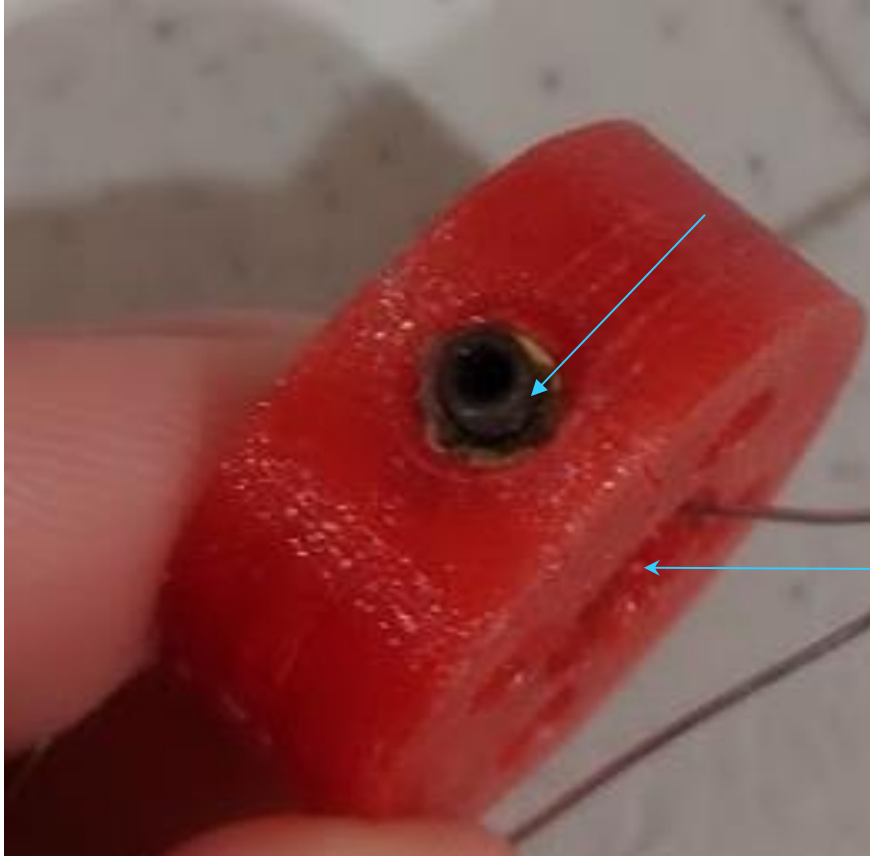
### Final Product



The biggest problem was all of the failures that occurred while printing as well as the stresses on different parts.

## Failures

Most of the failures were produced during 3D printing. The biggest problem once the pieces were assembled were the servo couplers. There was too much stress and they cracked by the brass heat set.



There is a document with all of the failures and what may have been the issue for some of them. Other ones are hard to tell.



This is an example of one where something happened during the print at the end, and there were too many variables to figure out what exactly was wrong with it.



## Aluminum extrusions

The aluminum extrusions are the key components that hold everything together. They are what make up the chassis as well as connecting the wheels and legs into one unit.



There are 6 different sizes to be made for the rover to make sure the chassis is leveled.

## Aluminum rods

The rover chassis uses 8mm diameter rods as one of the components. 30cm long aluminum rods were purchased as a raw material. The design calls for the following final components:

Qty	Description	Grooves	Detents	Overall Length (mm)
6	Wheel axle drive shaft	2	2	50
4	Steering shaft	2	2	61
2	Bogie pivot shaft	3	0	64.5

2	Rocker pivot shaft	3	0	82
1	Differential shaft	0	0	300

A hacksaw, vice and file were used to cut the rods to the correct length and smooth the ends.



The grooves were used to hold the E clips. These were made by fabricating a hacksaw blade clamping fixture and turning the cut rod in a drill press. The drill press table was adjusted so the hacksaw blade fixture contacted the rod at the correct location. The fixture was pulled against the rotating rod. The hacksaw blade produced a 1mm wide groove which matched the E clip thickness.



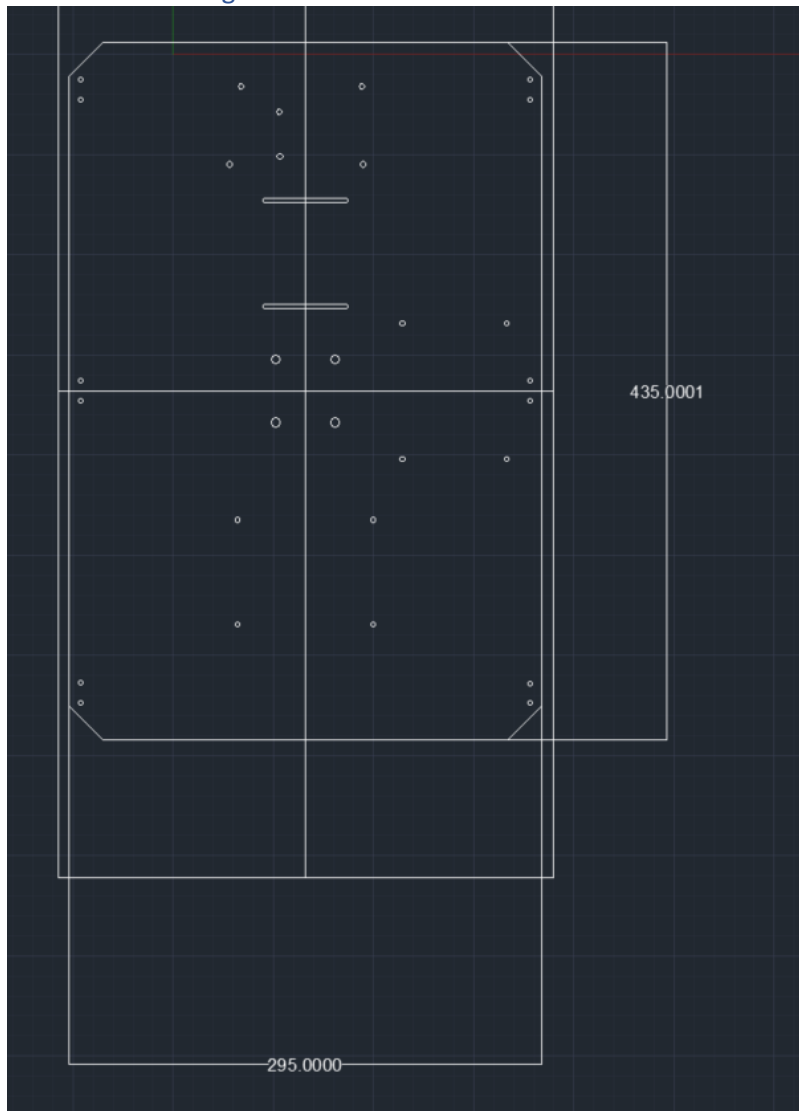
The detents were made by filing the rods as they were clamped in a vice.

## Laser-cut acrylic

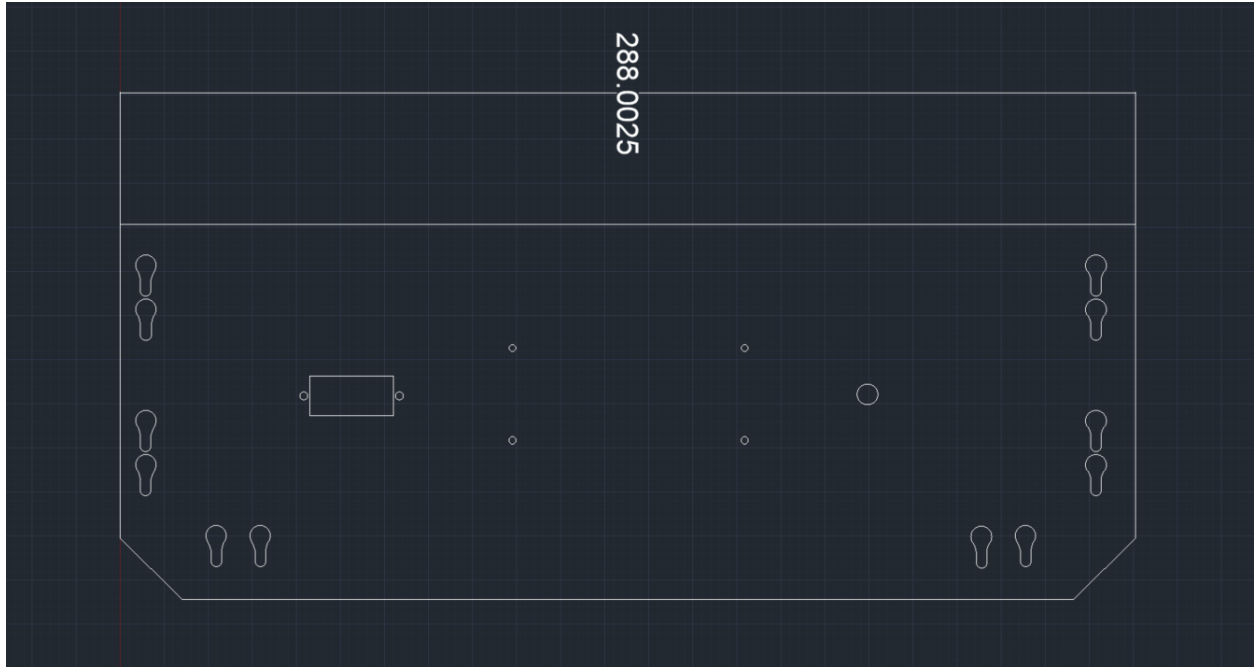
Once the construction of the rover chassis was finished, it needed a place to mount the electronics. A bottom plate, top plate and sensor brackets were designed for this purpose.

There had to be multiple designs for the bottom and top plates. AutoCAD was used to make design. Acrylic was used for laser cut parts and polycarbonate for handmade parts. The laser cutter was used for most of the pieces. Because AutoCAD is theoretical, different plates were made before it was correct. In addition to the bottom and top plates, the brackets for the ultrasonic sensors were cut from acrylic.

### Bottom Plate Design



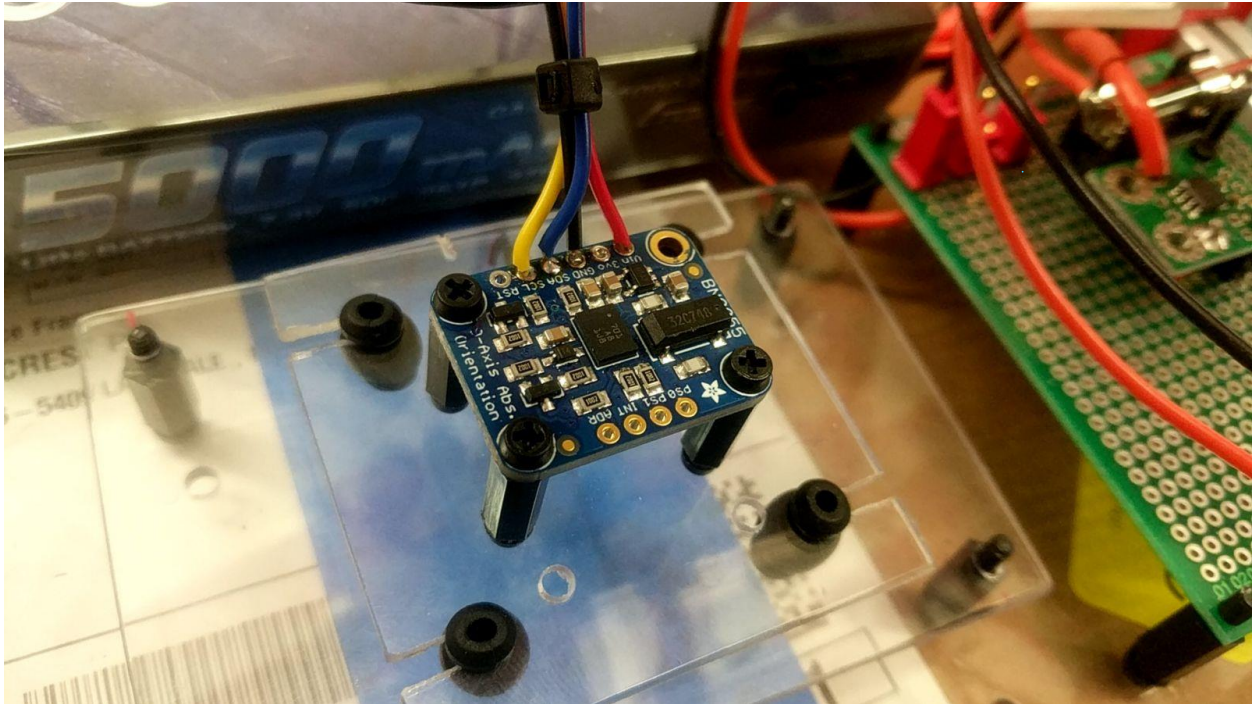
Top Plate Design



## IMU shock mounting

The rover used an IMU (inertial measurement unit) to measure small distances moved. The IMU in this rover consisted of a three-axis accelerometer, a three-axis gyroscope, and a magnetometer. To accurately measure the distance the IMU had to be mounted in the center of the rover and use dampeners to filter noise from traversing rough surfaces.

IMU





## Wheel traction

The rover wheels are patterned after the wheels used on the Mars rover. While acceptable on grassy surfaces, the 3D printed version for this rover provided little traction on a hard surface<sup>1</sup>. Three possible solutions were considered.

1. Adding large rubber bands on the outside of each wheel. Two suitable products were found.



Ranger Bands by Scol Survival Supply (6.5-inch flattened length, 1.5-inch width)



Size 107 rubber bands (180mm flattened length, 15mm width)

2. Spraying the outside of the wheels with Plasti Dip.

This idea came from the following entry on New Screwdriver:

<https://newscrewdriver.com/2018/06/09/plasti-dipping-sawppy-the-rovers-wheels/>

Attaching the wheel to a rod inserted into a hand drill and turning slowly helped to produce an even coating. It took 5 coatings before the surface had significant grip.



3. Using a flexible filament in the 3D printing process. Multiple choices exist. Due to time constraints, this idea was not tried.

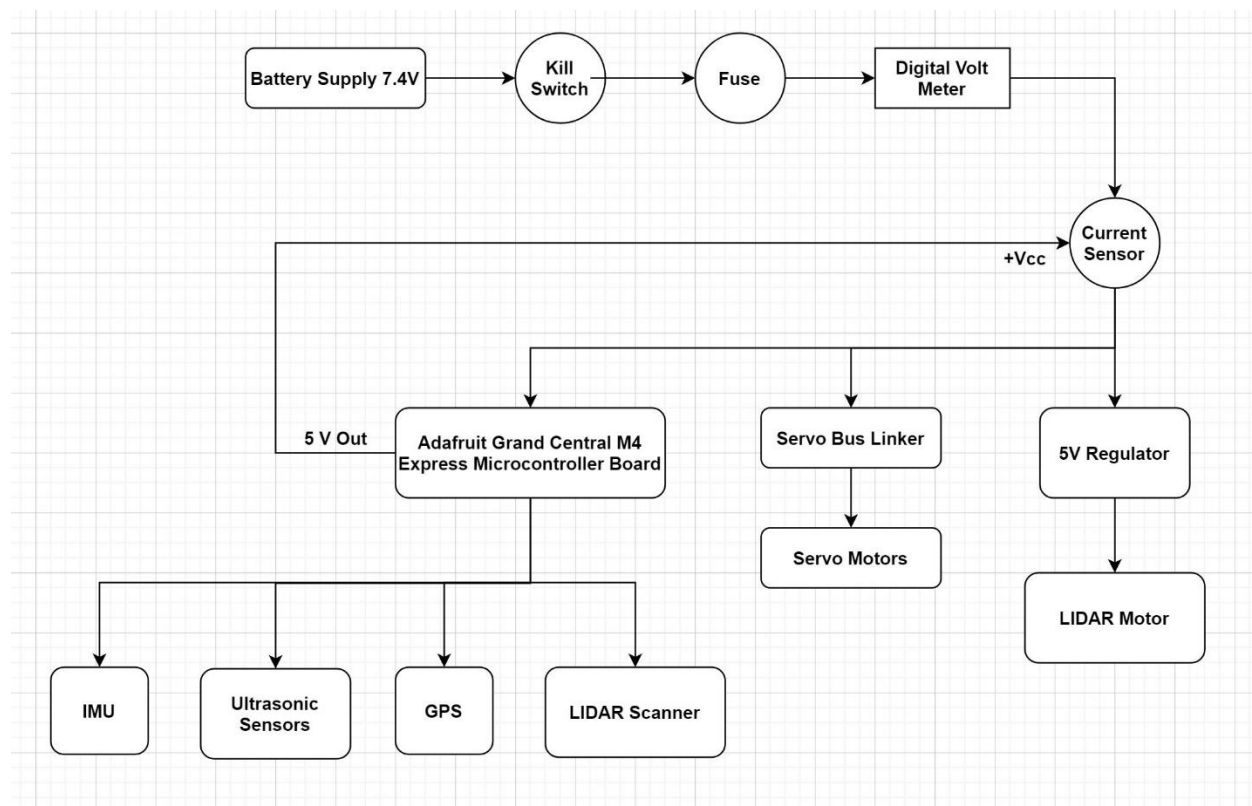
<sup>1</sup> This was initially an advantage before the wheel motors and turning servos were adjusted properly for turns. When improperly adjusted there was undue stress on the mechanical components. Slipping wheels minimized this stress.

## Electrical Design Team 4

*Paul John Balderston*

### Electrical overview

Diving directly into analyzing the rover's electrical schematics may be a cumbersome approach for an individual attempting to acquaint oneself with the system that has been implemented. The block diagram given below provides an elegant and simplified introduction to the rovers' onboard electronics. The figure primarily shows how power is distributed to all major components and, subsequently, which devices send and receive information from each other. To develop a full understanding of the electrical design of the rovers one should continue their reading of the electrical design portion of the project notebook.



Block Diagram- Electronic Components

### Microcontroller considerations

Perhaps the most significant constraint of the project was that all computation was to be microcontroller based. What microcontrollers may lack in memory space and processing power when compared to their microcomputer cousins is made up for in cost effectiveness and ease of use. Microcontrollers are not capable of running an operating system. While this in and of itself is a limitation, it greatly influenced and led our team to use efficient mathematical algorithms to process information and optimize the rover's navigation. This lack of an operating system does allow for quick and easy software development in the Arduino IDE. Procedural and object-oriented programming are used to create the scripts responsible for the rover's operation.

Despite the inherent limitations of a microcontroller, our project has demonstrated how one is able to acquire adequate memory space and computational power for autonomous robotics applications from these inexpensive, easy to use devices.

When selecting the microcontroller to be used onboard the rover we aimed for memory space and processing power while putting a special emphasis on having a high number of pinouts in order to allow for the implementation of many peripheral devices. The following microcontrollers were evaluated on the associated parameters listed.

Parameter	Adafruit METRO 328	Adafruit METRO M0 Express	Adafruit Grand Central M4 Express
Microchip	ATmega328P	ATSAMD21G18	ATSAMD51P20
Word size (8, 16 or 32 bits)	8	32	32
Clock speed	16MHz	48MHz	120MHz
RAM (aka SRAM)	2KB	32KB	256 KB
Flash	32KB	256KB	1MB
A/D convertor (number of bits, number of channels)	6 channels x 10 bits	20 channels x 12 bits	20 channels x 16 bits
Interface voltage (5V or 3.3V)	5V (can use 3.3V-5.5V)	3.3V	3.3V
Serial (UART) channels	1	6	8
I2C interface	Yes	Yes	Yes

## Power considerations

### Battery selection

The supply voltage requirements of the motors limited the maximum battery voltage. While the LX-16A datasheet does not explicitly specify a supply voltage range it does specify other parameters at 2 supply voltage values, 6V and 7.4V. Consequently, a 2S LiPo battery with a nominal 7.4V value was chosen. The rover has plenty of space a battery with a capacity of 5000mA-hr was specified. With 6 wheels turning at a maximum speed 1630mA is used. Using 80% of the battery capacity allows 2.5 hours of driving.

### Fuse size

Implementing a fuse into the rover's power distribution system provides a necessary failsafe. In the event that an onboard component was to short-circuit or spontaneously draw excess current, the fuse would prevent all other onboard electronics from being damaged.

To properly size the fuse, one must develop an awareness for the maximum average current draw of the system. The servo motors are capable of the largest amount of power consumption of any onboard component. Recognizing this fact allows for the simplification in determining the aforementioned

parameter. One may utilize the maximum current draw of all the servos running at maximum speed to obtain a close approximation of the parameter for the system as a whole.

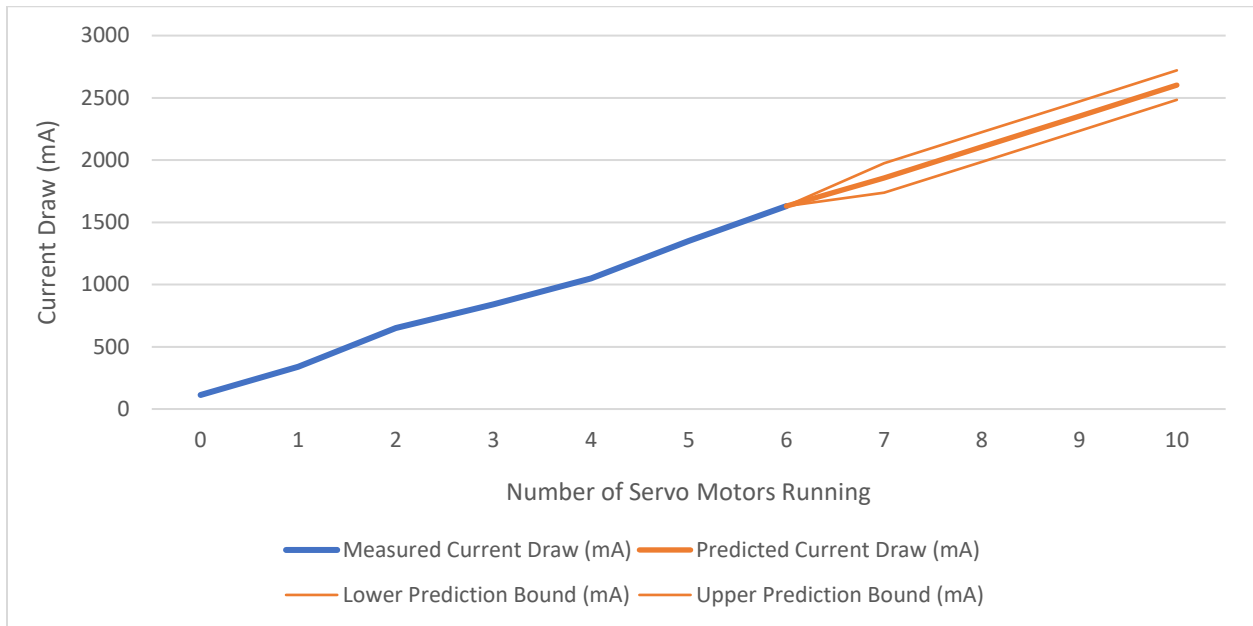
Proceed by elevating the rover so none of the wheels make contact with the floor or ground. Connect a PC to the servo bus linker board via USB. Make sure the LewanSoul Bus Servo Terminal software is installed. Connect a digital multimeter between the servo bus's associated connector terminal on the power distribution board and the corresponding wire which plugs into said terminal and connects to the servo bus linker board. Set the multimeter to measure current. Open the Bus Servo Terminal application. Begin by running one servo at maximum speed. Measure and record the current draw of this servo motor. Incrementally activate the remaining servos excluding those used for steering. Measure and record the current read by the multimeter each time an additional servo begins running. Use excel to organize and plot the acquired data once the current draw from all six rolling servos has been measured and recorded. To determine maximum current draw, extrapolate the current trend line out to include 10 servos. The multimeter readings and predicted current draw of 10 servos is shown in the tables below.

Note: For this test the battery voltage was 7.8V.

Motors Running	Measured Current Draw (mA)
0	113
1	340
2	650
3	840
4	1050
5	1350
6	1630

Motors Running	Predicted Current Draw (mA)	Lower Bound of Prediction (mA)	Upper Bound of Prediction (mA)
7	1855	1737	1973
8	2104	1985	2222
9	2353	2234	2471
10	2602	2483	2720





With 10 servos running a current of approximately 2.6A is predicted. A fuse should be rated for 135% of the typical maximum steady state current. The following calculation was performed to determine the proper amperage fuse for the rover.


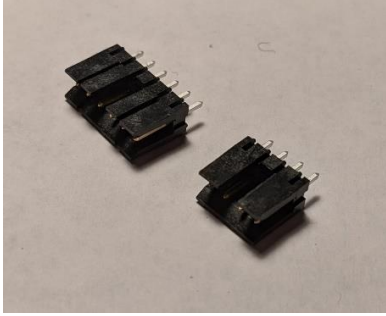
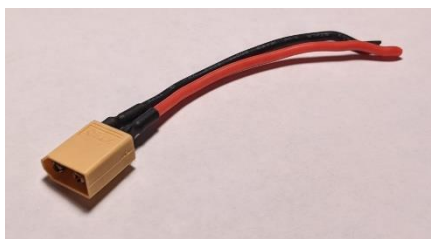
$$2.6Amps * 1.35 = 3.51Amps$$

After rounding this value up to the next standard amperage fuse, a 5 Amp fuse was selected.

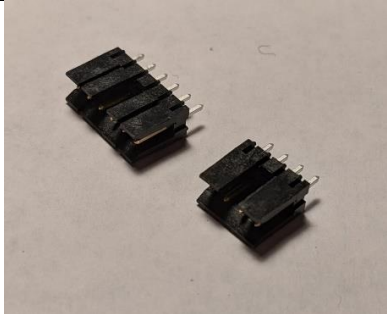
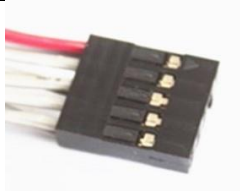
### Wiring considerations

Flimsy wiring and faulty connectors often prove to be large sources of error in any electrical system. All connectors were selected to enhance reliability and ease of repair. The following tables show which connectors are implemented on each printed circuit board.


## Power distribution board

<u>Qty</u>	<u>Description</u>	<u>Picture</u>
4	Barrel Connectors (5.5mm Outer Diameter, 2.1mm Inner Diameter)	
1	4-Pin Dupont Male Connector	
1	XT60 Connector	

## Microcontroller shield board

<u>Qty</u>	<u>Description</u>	<u>Picture</u>
2	3-Pin Dupont Male Connectors	
8	4-Pin Dupont Male Connectors	
1	6-Pin Dupont Male Connectors	
1	9-Pin Dupont Male Connectors	
4	5-Pin Dupont Female Connectors	

## Servo motor control board

<u>Qty</u>	<u>Description</u>	<u>Picture</u>
12	Molex 0022035035 3-Position Connectors	

All peripheral devices utilize the male Dupont connectors with the appropriate number of pins to facilitate the delivery of all signals from the peripheral component to the device it is transmitted information to and drawing power from.

The connectors shown on the previous pages aim to provide extreme reliability by interlocking both ends of the connection. All connectors are low cost and if damaged can easily be replaced. The Dupont female to female crimp housing proved to be exceptionally easy repair. The crimp locked in the housing may simply be removed and replaced with another wire if the connection becomes damaged.

The barrel connectors, used primarily for power distribution, have a hot and ground wire splitting from the male end and may be spliced and soldered onto another wire in order to extend their length.

Heat shrink tubing is used wherever wires are spliced together in order to ensure a secure connection and reliable connection. More information on the Molex connectors and their associated wires is in Components, Servo and Motor Control Section.

## Components

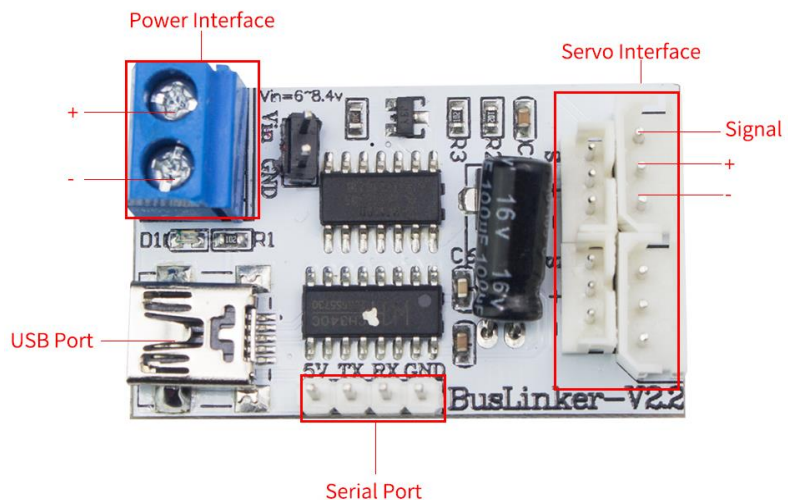
Developing a cost-effective opensource rover provides the opportunity for a larger number of individuals to implement their own version of the SAWPPY project. Prioritizing overall affordability as an essential benchmark for the system incentives one to seek out off-the-shelf electronics with high levels of technical capability. The hardware used to facilitate and enable autonomous navigation, localization, object detection, and obstacle avoidance is discussed in the following section. A list of these components and their electrical characteristics is provided below.

<u>Component</u>	<u>Manufacturer</u>	<u>Part Number</u>	<u>Supply Voltage (V)</u>	<u>Maximum Current (A)</u>	<u>Logic Level</u>
Grand Central M4 Express Microcontroller Board	Adafruit	4064	7.4		3.3V
Servo Driver Board (Bus Linker)	LewanSoul	B073WRLJB2	7.4		5V
Servo Motors	LewanSoul	B073WR3SK9	7.4	2.5	5V
Ultrasonic Sensors	RCWL	1601	3.3		3.3V
IMU	Adafruit	2472	3.3	0.012	3.3V
GPS Receiver	Adafruit	746	3.3	0.025	3.3V
GPS Antenna	Cirocomm	580	N/A	N/A	3.3V
2S, 5000mA-hr, LiPo Battery	Duratrax	ONYX 5000	7.4V	N/A	N/A

## Servo and motor control



LX-16A servo motor



Bus servo interface board

The LewanSoul LX-16A servo motors are utilized to both drive and steer the rover. These servos uniquely possess two operating modes. The servo mode allows the motors to turn to a specific angle between 0 to 240 degrees. Meanwhile, the servos are also capable of operating as a typical DC motor, continually spinning at a given speed. The servo mode is utilized for steering and the motor mode is used for rolling.

LewanSoul provides a communication bus that distributes power and transmits operating instructions to each individual motor. The serial communication interface allows for the servos to all be connected to one of two ports on the bus linker board. Each individual servo possesses an assigned serial address. The LewanSoul Bus Servo Terminal allows the assignment of these addresses and testing in both servo and motor modes. This is shown in figure #.



LewanSoul Bus Servo Terminal Software Suite

The servo bus linker board will provide the proper power and operational data to the appropriately assigned servo. This allows one to either “daisy chain” or connect the power, data, and ground pins of the servos in series with their respective counterparts and still receive the appropriate signals transmitted from the bus linker board.



## Servo motors

The LX-16A Bus Servo has a 3-position connector that is wired to a Bus Servo Interface Board. There are cables with the appropriate connectors available, but they are too short for the ten LX-16A Bus Servos used on the rover.



Two methods of making a longer cable were used.

1. Cut the available short cables in half and splice an appropriate length of 3 conductor cable in the middle. The 3-conductor cable was 22AWG with red-white-black colored insulation (OliYin 50 feet 22AWG Servo Cable, available from Amazon).
2. Cut a 3-conductor cable to length, strip the insulation on the wire ends, crimp a contact on each wire and insert into a housing. It took some research to identify the actual connector on the LX-16A Bus Servo and the Bus Servo Interface Board. The DigiKey web site has an excellent part search tool. Once the connector was identified, the DigiKey web site listed compatible contacts and crimping tools. The following parts were used:

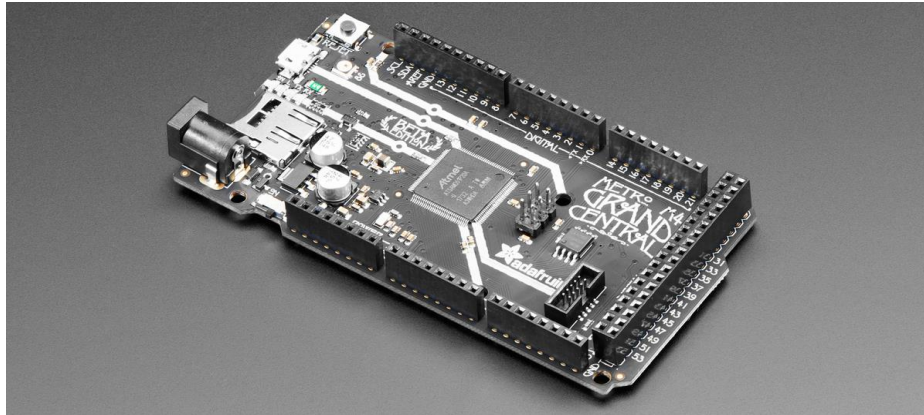
Housing	Molex 0050375033 3 position contact housing
Contact	Molex 0008701038 socket contact
Crimping Tool	Molex 64016-0201, service grade tool

Note: The crimping tool is about \$140. The contacts are small and it took some practice to make a reliable crimp.

The LX-16A Bus Servo is controlled with a serial communication protocol. Each servo is assigned a unique address so multiple servos can connect to the same serial bus. The Bus Servo Interface Board has 2 servo connectors. A new printed circuit board was developed to replicate the servo connectors so the 10 servos could connect to one Bus Servo Interface Board.

## Microcontroller

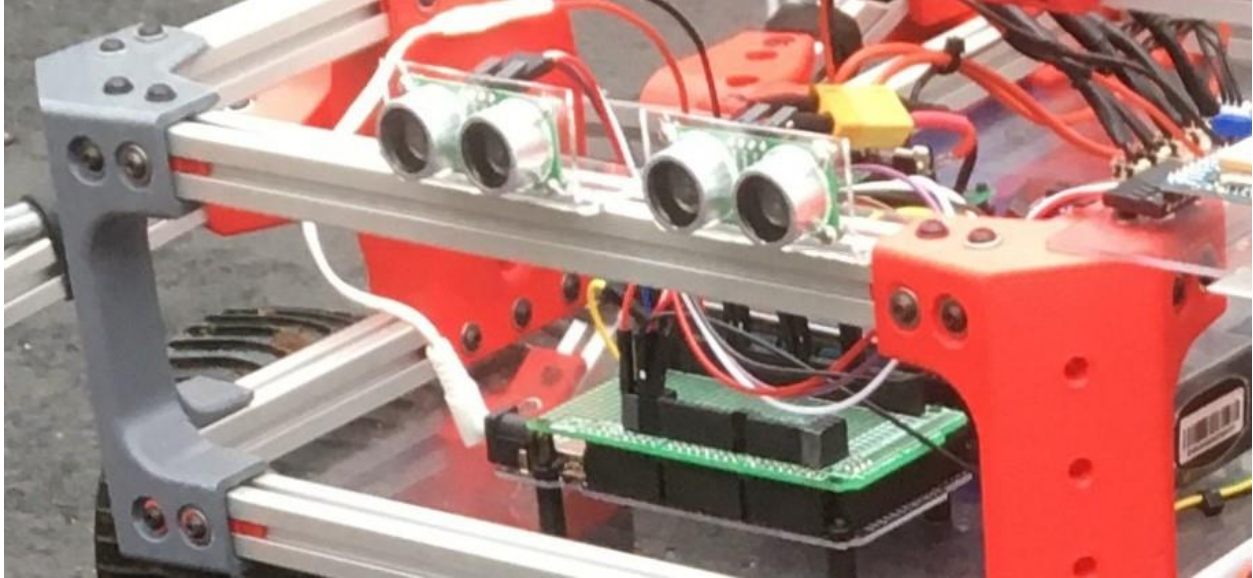
The Adafruit Grand Central M4 Express was selected as the microcontroller. The M4 provided plenty of memory space and processing power. However, its true advantage was the plentiful number of pinouts. The M4 is equipped with 70 GPIO pins, 16 of which are analog in and two are analog out. There are 5 serial communication channels present and may be used for additional I2C/SPI/UART connections. The M4 is assembled using the same footprint as the popular Arduino Mega 2560. This elevates the boards ease of use as perforated shield boards may be used to prototype PCB designs to connect to the M4. For reference, a picture of the Adafruit board is given below.



## Object detection sensors

Robust object detection is crucial for the facilitation of autonomy and obstacle avoidance. Primarily ultrasonic and optical sensors were tested. Multiple time of flight sensors and laser distance sensors were compared to the ultrasonic that are used in the final implementation of the system. The optical sensors we tested proved to be unreliable when attempting to measure the distance of a dark surface, such as asphalt, or detect an object that caused the light emitted from the sensor to scatter, bushes were shown to have this affect. The optical sensors not only lacked reliability but also lacked the range and field of view provided by the ultrasonic sensors. Time of flight sensors emit a light beam the travels in a straight line forward and is reflected back to the sensor. This provides an extremely narrow field of few. The ultrasonic sensors allow sound waves to propagate out from their source in a cone like manner. The datasheets of the implemented ultrasonic sensors suggest a field of view of approximately 30 degrees. Once the sound waves reflect off of an object and travel back to the sensor the of the object from the sensor is determined.

Two ultrasonic sensors are mounted on the front of the rover. This maximizes the accuracy and field of view of distance sensing along with narrowing a blind spot that would otherwise be present near the front of the rover.



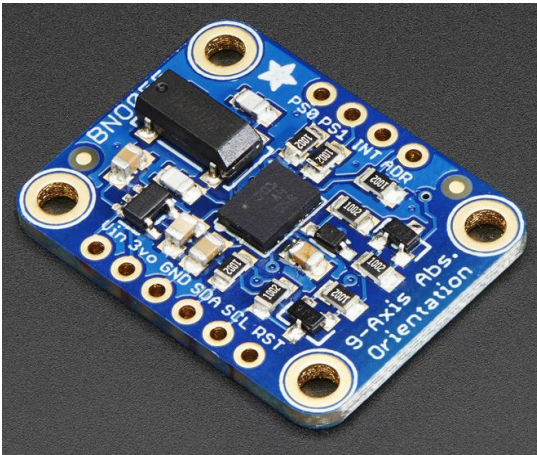
Initially, HC-SR04 ultrasonic sensors were used. These sensors operated with a 5V power supply and logic level. This required a resistor divider to lower the voltage of echo signal to be compatible with the 3.3V logic level of the microcontroller. Changing to ultrasonic sensors that operated with a 3.3V power supply and logic levels allowed us to eliminate the resistor divider.

## IMU and GPS

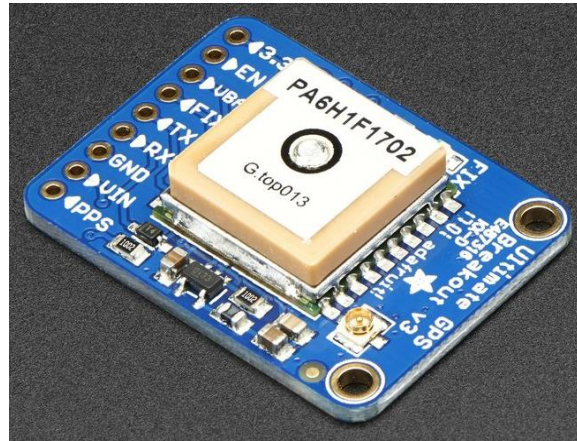
The IMU (Inertial Measurement Unit) and GPS (Global Positioning System) enable the localization and tracking of the rovers' position. The GPS is used to give the general position of the rover. Civilian GPS is accurate within radius of approximately 15 meters. This inaccuracy is the reason to add an IMU and use sensor fusion.

The IMU collectively houses a gyroscope, accelerometer, and magnetometer. By combining readings, orientation in three-dimensional space can be precisely measured. The IMU readings are combined with the measurements of the GPS. A weighted average of all localization parameters is computed to achieve a position reading with the highest level of accuracy modern technology is capable of. These two components are essential for facilitating accurate localization and position.

A 7 square inch aluminum ground plate was added under an external GPS antenna. This increases the antenna's gain. This was added to allow rover testing indoors. For outdoor use even the increased gain is probably not necessary.



IMU board



GPS board



## Battery

The rovers each use a 7.4V 5000mAh 50C 2S LiPo battery. To prevent permanent damage a LiPo battery should never be discharged below 3V per cell. A 3.5V per cell limit is a more conservative value to improve the battery lifespan.<sup>1</sup> The 2S term in the battery description indicates that it has 2 cells in series, so 7.0V is the appropriate cutoff voltage.

To verify this value on our rover, data was taken on the battery at various states of discharge. While the battery was powering the wheel motors, a digital voltmeter measured the battery voltage and the battery capacity was measured with a Hyperion EOS Sentry battery tester.

DVM Voltage (V)	Capacity (%)
7.93	75
7.80	68
7.74	69
7.69	62
7.65	61
7.63	60
7.60	54
7.57	54

Excel produced the following trendline:

$$\% = 59.6 * V - 396$$

Extrapolating the data shows at 7.0V, the capacity is 21% which is an appropriate place to stop using the battery and recharge it.

The Operating Instructions that came with the battery included: "Never discharge battery to a level below 3V per cell under load."

<sup>1</sup> *A Guide to Understanding LiPo Batteries*, [www.Hyperion-world.com](http://www.Hyperion-world.com)

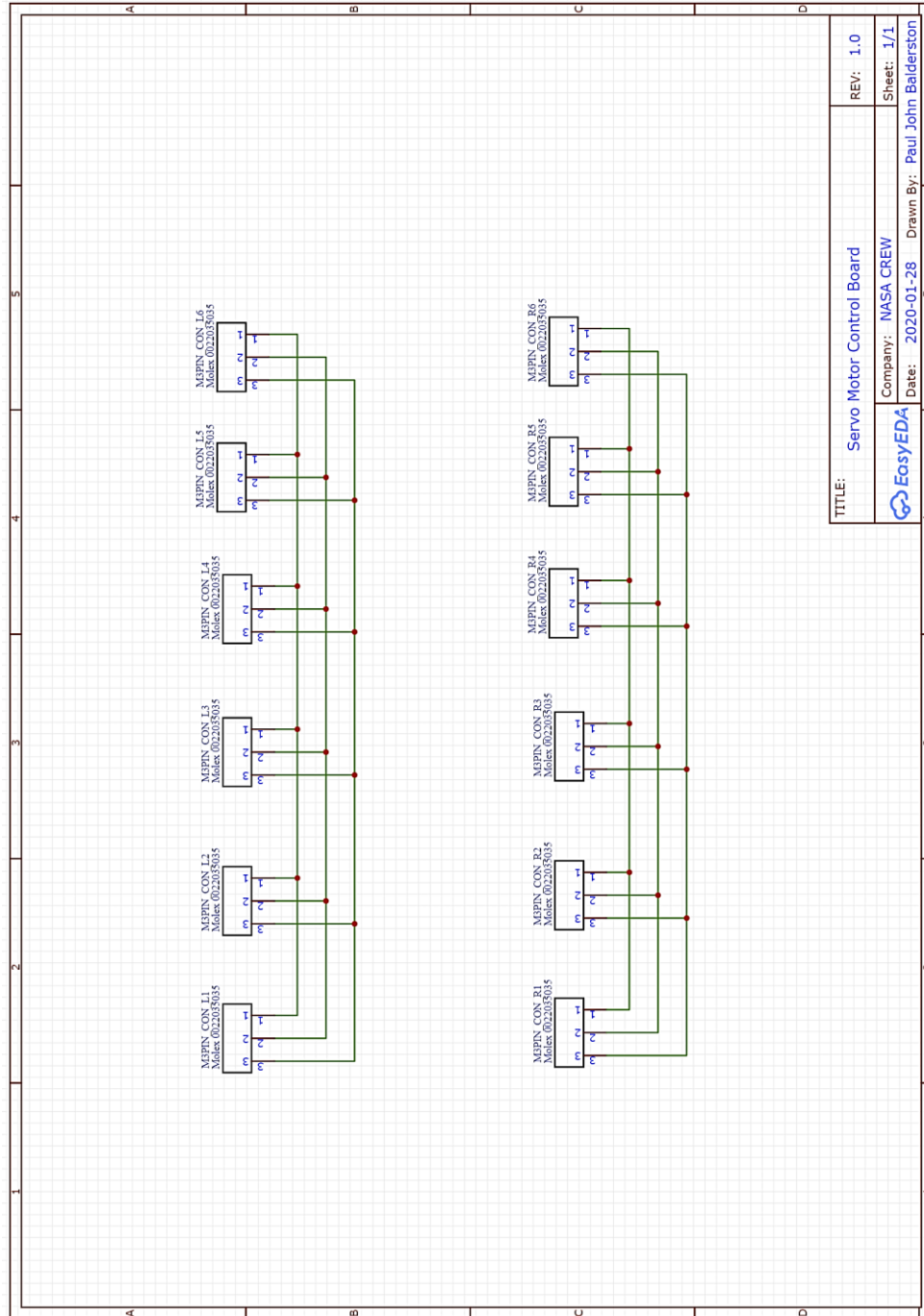
## PCB - Servo motor control board

The servo motor control board provides power and operating instructions to the servo motors. The LewanSoul bus linker is attached to the middle of the board through the use of a hook and loop adhesive patch. The bus linker uses two ports to send signals to the servos on the right and left-hand sides of the rover independently. Two cables run from the servo bus linker to the Molex connectors on their respective sides of the board. These connections occupy two out of the six connectors wired in parallel on each side of the board. The servo motors are connected to the five remaining Molex connectors on each side. Any servo may be plugged into any Molex connector and still receive the proper signals thanks to the servo bus linkers serial communication protocol.

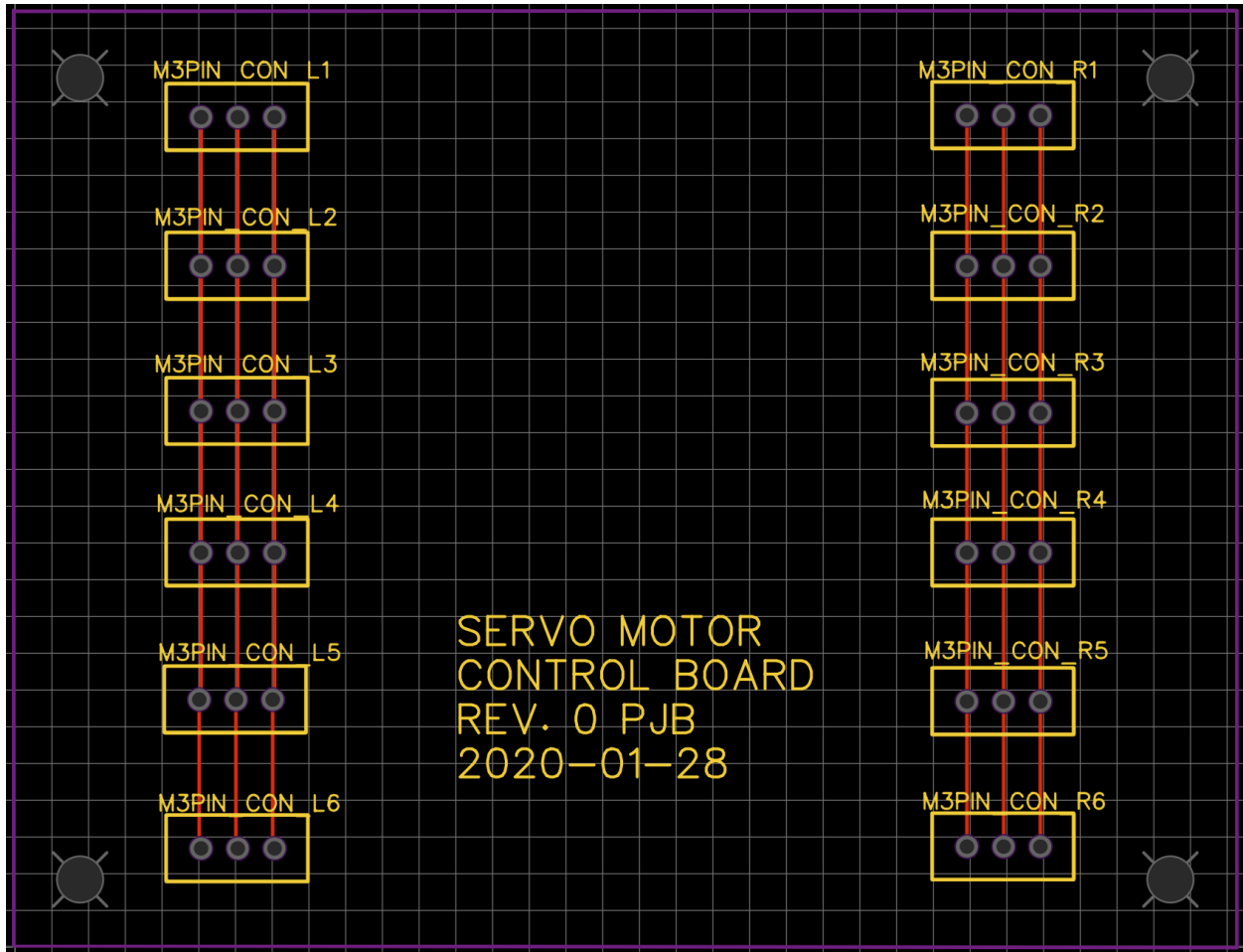


<u>Reference Designator</u>	<u>Component</u>	<u>Manufacturer</u>	<u>Part Number</u>	<u>Quantity</u>
M3PIN_CONN	Servo Connectors	Molex	22035035	12
N/A	Bus Linker Board	LewanSoul	N/A	1

Parts List- Servo Motor Control Board



Schematic, Servo Motor Control Board



Printed Circuit Board Layout- Servo Motor Control Board

## Power Distribution

The power distribution board is designed to protect the battery distribute the battery voltage, provide a high current 5V, and provide signals to record battery voltage and current.

Perhaps the most important failsafe of the entire system is the 5 Amp fuse. Current flows directly into the fuse from the battery connector, preventing the battery from being damaged in the event of an excessive current draw. The fuse holder allows both 5mm x 20mm and 3AG size fuses.

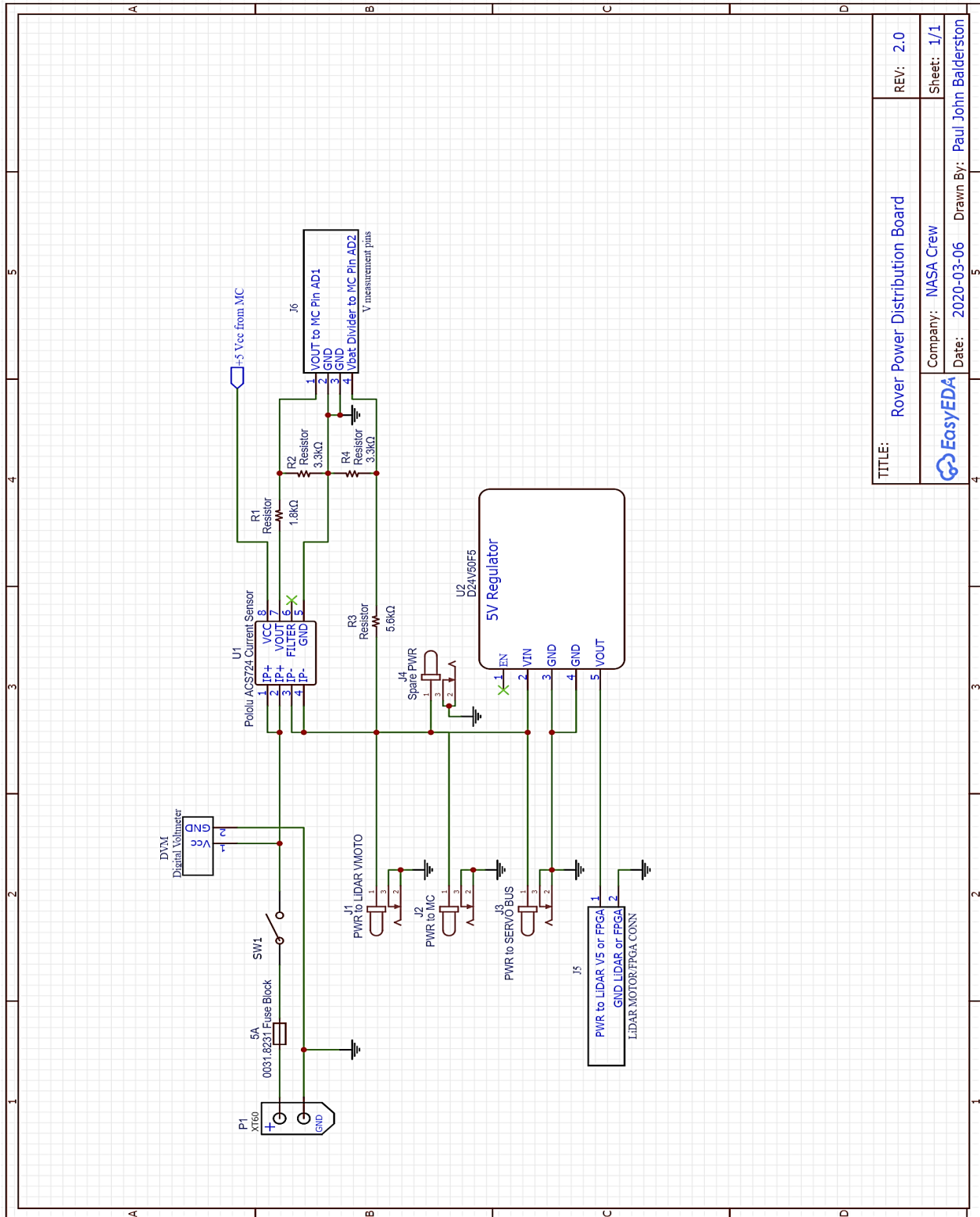
Barrel connectors are used to establish secure connections between major electrical components and the power distribution board. Resistor dividers lower the range of analog signal to the microcontroller analog to digital converter. A 5V regulator provides a higher current supply than is available on the microcontroller board. A current sensor provides an analog signal for use by the microcontroller.

A kill switch and digital voltmeter are wired to the power distribution board and mounted on the rear top plate of the rover. Allowing the operating voltage of the battery to be monitored and the rover to be easily switched on and off.

The plated through holes for the barrel connectors were incorrectly designed to be circular when in fact the pins of the barrel connectors are oblong. Excess solder was used to create solder joint to attach the barrel connectors to the printed circuit board. This adequately secured the connector to the pad using the solder as an anchor.

<u>Reference Designator</u>	<u>Component</u>	<u>Manufacturer</u>	<u>Part Number</u>	<u>Quantity</u>
P1	XT60 Battery Connector	Sparkfun	PRT-10474	1
5A	Fuse Block Cartridge	Schurter Inc.	0031.8231	1
5A	5Amp Fuse	Littelfuse Inc.	0218005.MXP	1
SW1	Kill Switch	HiLetgo	MTS102	1
DVM	Digital Volt Meter	N/A	N/A	1
U1	Current Sensor	Pololu	ASC724	1
U2	5V Regulator	Pololu	D24V50F5	1
R1	1.8k $\Omega$ Resistor	N/A	N/A	1
R2, R4	3.3k $\Omega$ Resistor	N/A	N/A	2
R3	5.6k $\Omega$ Resistor	N/A	N/A	1
J1, J2, J3, J4	Female Barrel Connector	CUI Devices	PJ-060A	4
J5	Male Header Pins	N/A	N/A	2
J6	4 Position Dupont Connector	Amphenol	78211-004LF	1

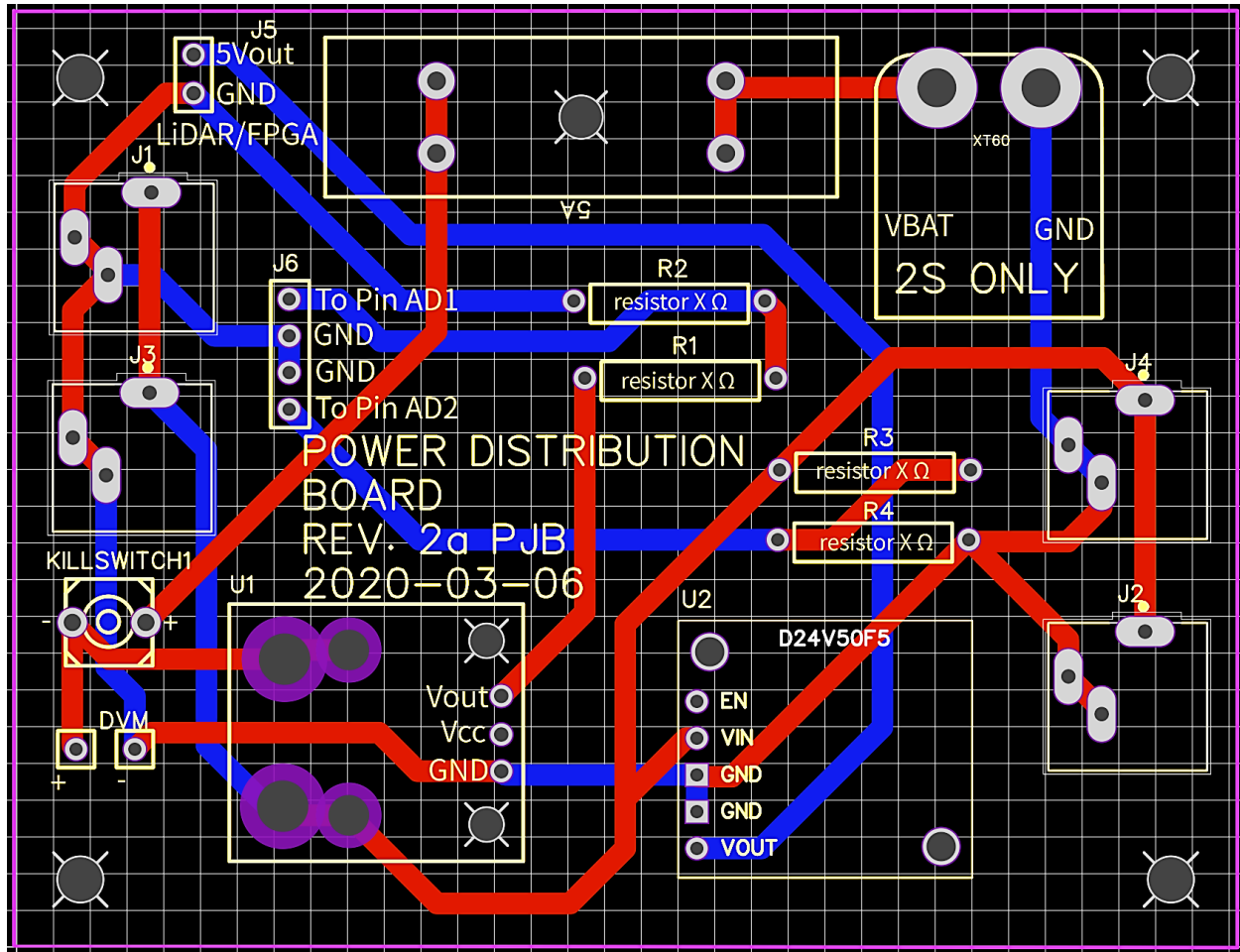
Parts List- Power Distribution Board



Schematic, Power Distribution Board

TITLE:	Rover Power Distribution Board	REV:	2.0
Company:	NASA Crew	Sheet:	1/1
Date:	2020-03-06	Drawn By:	Paul John Balderston





Printed Circuit Board Layout, Power Distribution Board

### Microcontroller shield

The M4 microcontroller connects to any peripheral device used to obtain information regarding the rover’s location and environment and send instructions to operate the servos. The shield board sits atop the M4 microcontroller board, providing a means of easy connectivity and additional pinouts for those components which may require 3.3V or 5V power to be supplied from the microcontroller.

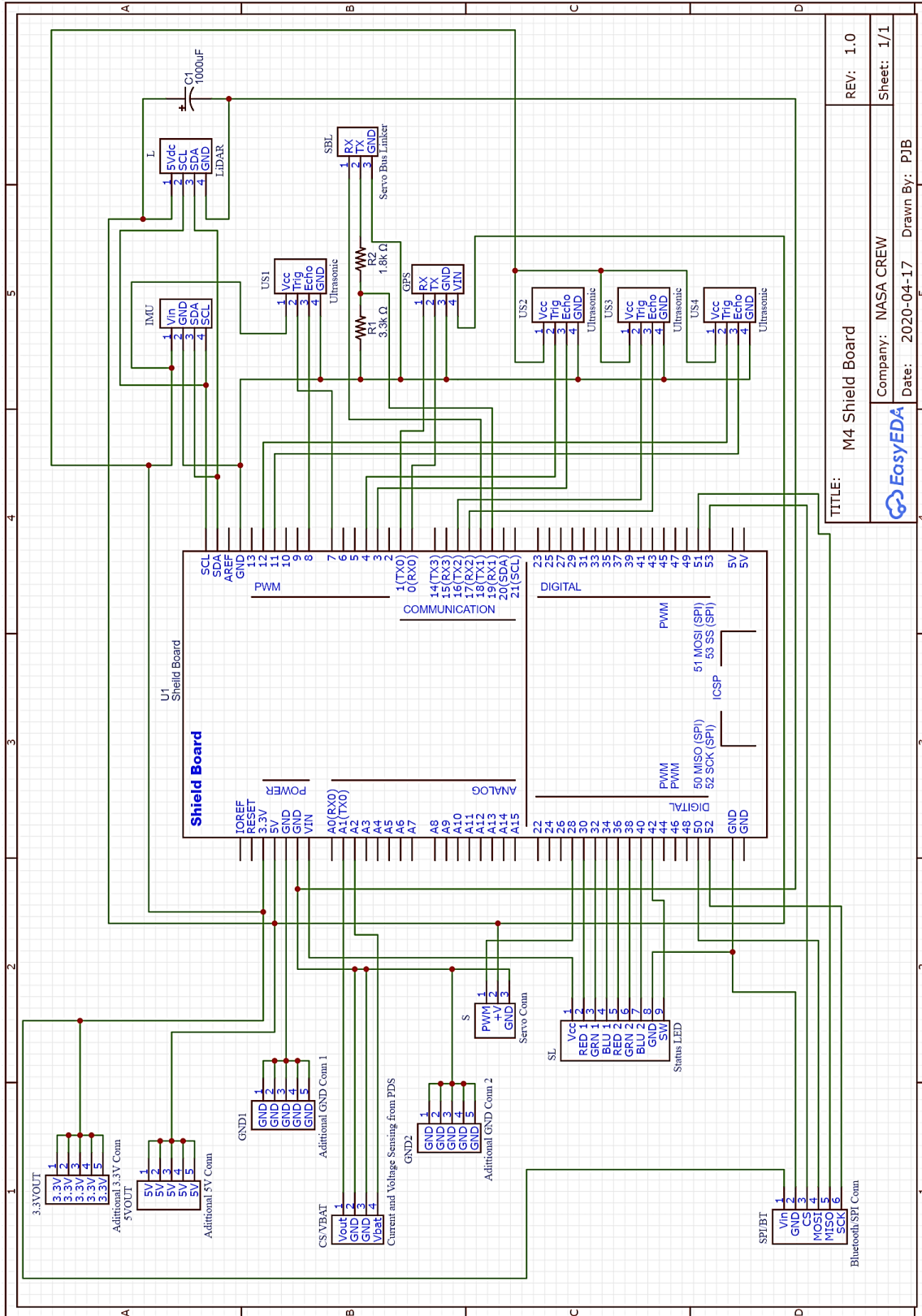
Dupont connectors are used to attached peripheral devices to the shield board. Theses secure connectors are strategically placed to allow for ease of use and enables multiply components to be connected to the 3.3V, 5V, and GND pins at once.

5-position female connectors are attached to the 3.3V, 5V, and GND pins. Enabling additional sensors and devices to be mounted onboard the rover and utilize the microcontroller to supply voltage to and ground the newly installed component. This allows for rapid onboard testing of new components.



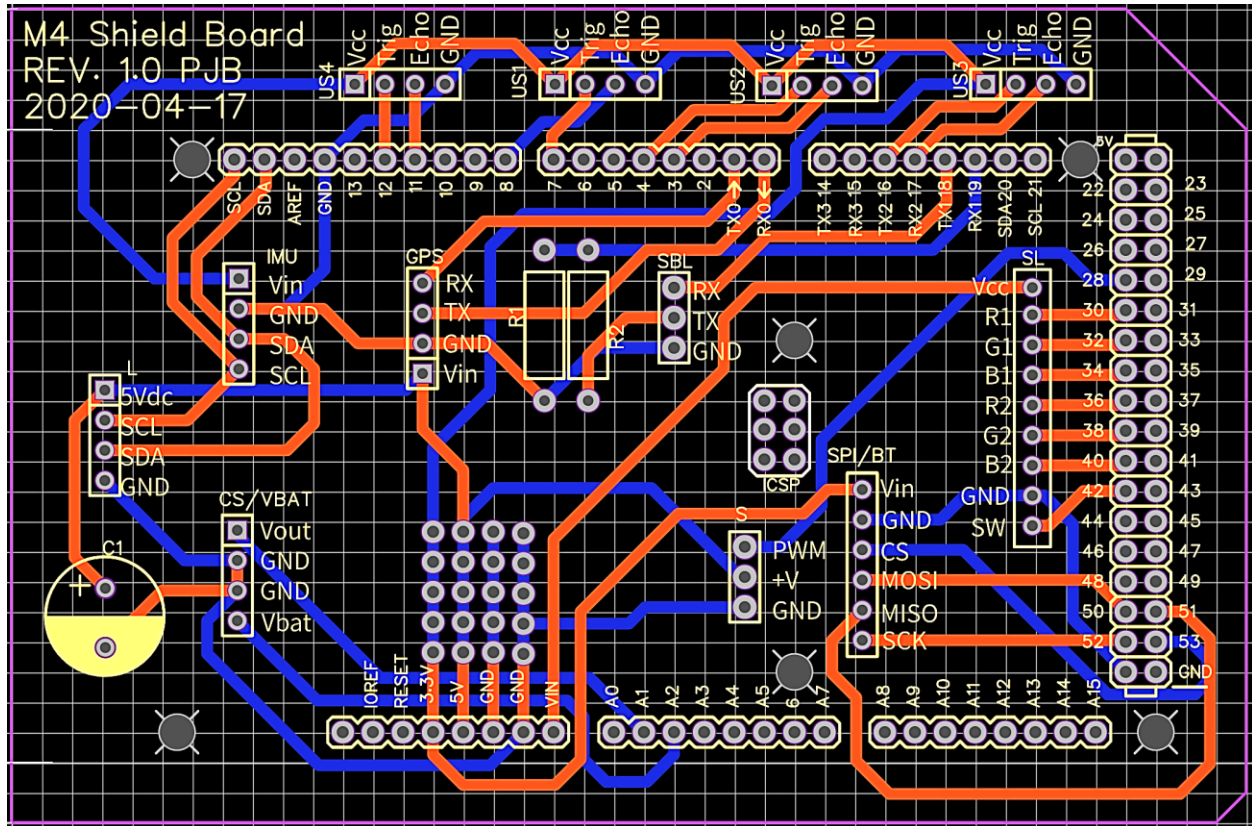
<u>Reference Designator</u>	<u>Component</u>	<u>Quantity</u>
US1, US2, US3, US4, IMU, GPS, L, CS/Vbat	4 Pin Dupont Connectors	7
S, SBL	3 Pin Dupont Connectors	3
SPI/BT	6 Pin Dupont Connectors	1
SL	9 Pin Dupont Connectors	1
N/A	Male Header Pins	106
C1	1k $\mu$ F Capacitor	1
R1	3.3k $\Omega$ Resistor	1
R2	1.8k $\Omega$ Resistor	1

Parts List- Microcontroller Shield Board



TITLE: M4 Shield Board	REV: 1.0
Company: NASA CREW	Sheet: 1/1
Date: 2020-04-17	Drawn By: PJB

Schematic, Microcontroller Shield Board



Printed Circuit Board Layout, Microcontroller Shield Board

# Software Design Team 1

*Noah Williams and Salvatore Sparacio*

## Architecture

The purpose of this document is to describe each and every folder within the project. It is different from the Design Document which describes the why and how. This describes the what, what each individual folder does.

There are two important folders. Libraries, which contains all of the essential code needed. And Sketches, which is a specific implementation of a library. Inside sketches are two folders Main and Examples. Main contains the finished sketches that combine multiple libraries. Examples are a record of all tried methods, and test individual libraries.

The exact architecture used is called Component Based Programming. Each part is an individual component that attaches to the sketch. Most components work as individual units, with a setup method and an update method. The setup method is called during the setup in Arduino, the update is code that updates continuously. Doing so allows us to make changes without it affecting all of the codebase. Understanding this design allows for future programmers to add to the code without breaking or affecting it much. The sketches are intentionally small, and if one wanted to add their own code, they simply make a component, with a setup and update, then call them inside a sketch.

Beginner programmers may notice something odd. That is, there are many deprecated folders located within the project. This is intentional, it is a common practice (there exists many other practices) in Software Engineering to never delete (if you can) and instead create a new library to replace. Keeping old code helps to not only prevent breaking the system, but fixing future bugs. In addition, there were many past and failed attempts. The old code keeps a record of all methods used for future programmers who wish to add future revisions (such as LIDAR).

The additional code does not affect the size of the sketch. When uploaded to M4, only the referenced libraries are loaded.

## Main

Main is the finished product. The final sketches for quickly running the rover.

<b><u>Manual Destinations</u></b>	Performs Rover going to A to B while avoiding obstacles. Set points manually.
<b><u>VirtualRover</u></b>	Performs Rover going to A to B while avoiding obstacles. Set points automatically.
<b><u>WHEEL TEST</u></b>	Tests the wheels by driving in a particular order.

## Libraries

Libraries is the actual code. It performs all of the necessary calculations to ensure a sketch works.

<b><u>Adafruit_BNO055</u></b>	Adafruit's BNO055 Library.
<b><u>Adafruit_GPS</u></b>	Adafruit's GPS Library
<b><u>Adafruit_LSM303_U</u></b>	Adafruit's LSM303 IMU (deprecated)
<b><u>Adafruit_Sensor</u></b>	Adafruit's General Sensor Library
<b><u>AutonomousDrive</u></b>	Combines several libraries to Drive the Rover from A to B While Avoiding obstacles.
<b><u>Button</u></b>	Code to add mechanical buttons.
<b><u>Distance</u></b>	Allows Rover to turn in place (deprecated)
<b><u>DriveTrain</u></b>	Handles turning the wheels. This includes turning in place and speed of motors.
<b><u>GPS</u></b>	Controls the GPS.
<b><u>Map</u></b>	Loads internal maps from the SD Card. (Requires Virtual Rover)
<b><u>MultiSerial</u></b>	Handles creating multiple Serials on the Grand Centro M4
<b><u>SerialCommand</u></b>	Creates Software Serials
<b><u>SparkFun_VL6180X</u></b>	Used for Time Of Flight Sensors.
<b><u>coroutine</u></b>	Creates coroutines for single threaded multitasking. This is to be used instead of Arduino's Delay function.
<b><u>imumaths</u></b>	Handles Matrix maths and Quaternions.
<b><u>internalMapping</u></b>	Stores points from LIDAR spin (deprecated)
<b><u>kalmanFilter</u></b>	Combining IMU and GPS data, note the actual Kalman Filter section of the code is commented out, but the library is still heavily used. Due to this, the name is a misnomer.
<b><u>node</u></b>	Node class for creating a graph for LIDAR (deprecated)
<b><u>orientation</u></b>	Computes all of the IMU calculations.
<b><u>save</u></b>	Saves to an SD Card.



<b>sphereNode</b>	3D representation of Node class (deprecated)
<b>ultrasonic</b>	Handles setting up an ultrasound.
<b>vector3D</b>	Handles vector operations. Note: IMU maths also handles similar tasks, but they differ in memory usage. Vector3D is specific with 3 floats, IMU maths is general and can be nxn.

## Examples

Examples are examples of a library. It shows exactly how to run a library with a sketch. There are many deprecated examples that exist solely to show how to run a function. This is important when adding in new code, or understanding how the library works. In practice, you can copy and paste functionality of the examples and create a unique new sketch.

<b>AutonomousDrive</b>	Drives rover from point A to B. Similar to Manual Destination, but shows the entire code within the Arduino IDE. (Deprecated)
<b>BearingCalculations</b>	Shows bearing and heading calculations.
<b>Button</b>	Mechanical button test.
<b>DistanceMovementTest</b>	Test the rover using the IMU to move a set distance.
<b>DistanceReading</b>	Records Ultrasound distance to SD card.
<b>GPS</b>	Shows GPS data.
<b>HeadingTest</b>	Prints heading.
<b>LoadMap</b>	Loads map from SD card (from Virtual Rover).
<b>LoadWayPoints</b>	Tests communication with Virtual Rover app.
<b>RPLIDAR_map</b>	Tests LIDAR scans.
<b>RoverTest</b>	Drives the Rover while recording voltage. (Deprecated).
<b>Save</b>	Tests saving any string to SD card.
<b>SaveMap</b>	Test saving GPS points to SD card.
<b>UltrasoundTest</b>	Prints ultrasound data.

<b>WaypointsTest</b>	Test the linked list data structure used for waypoints.
<b>WheelDirection</b>	Tests the wheel direction math when tracking a path.
<b>localization</b>	Prints IMU data for Unity Virtual Rover. Allows to see IMU rotations on unity 3D model.
<b>matrix</b>	Performs examples of using the matrix math.
<b>orientation</b>	Tests IMU calculations.
<b>rawDataIMU</b>	Test the BNO055 updates. This is Adafruit's test.

## Microcontroller and serials

A Grand Central M4 is the microcontroller used to process the code. It is compatible with Arduino and Adafruit libraries giving easy access without the need for an operating system. It has around 256 KB of RAM, and processes at 120MHz. The M4 is very capable of handling nearly all aspects of navigation. It can easily perform complex equations thousands of times per second. This includes complex equations like matrix multiplication, solving inverses of small matrices, sensor fusion, integrals, derivatives, trig functions, and quaternions. All of these are used in this project. The M4 struggles to handle SLAM based techniques such as mapping, localization, and camera.

Our implementation of the M4 uses C++ programming language, as well as additional serial communications.

The following table shows the addresses created for the serials.

```
/*
  | Serial |   | TX arduino pin |   | RX arduino pin |   | SERCOM |
  =====
  | Serial2   | 18 |   19           |   | 4             |
  | Serial3   | 16 |   17           |   | 1             |
  | Serial4   | 14 |   15           |   | 5             |
  =====
*/
```

All we care about are pins (14, 15, 16, 17, 18, 19) since these pins connect to SERCOM ports used to communicate with the board through a UART.

Each UART (inherited from HardwareSerial) specifies what pins and what SERCOM they communicate with. Then each SERCOM needs a handler. These handlers are called during an interrupt.

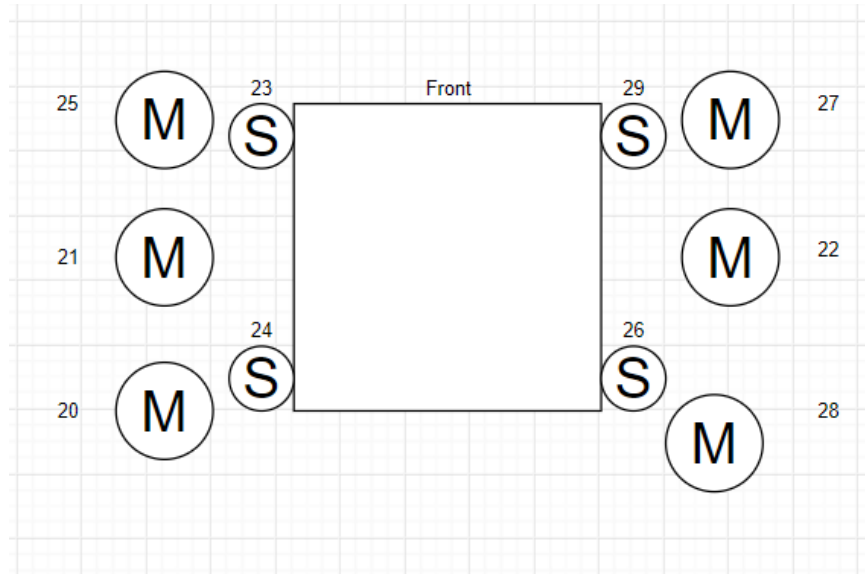
Once handlers are made, the TX and RX pins are specified and there is a working Serial port. There are limitations however.

### There can only be:

- up to 4 UART devices
- up to 2 SPI devices
- up to 2 i2C connections

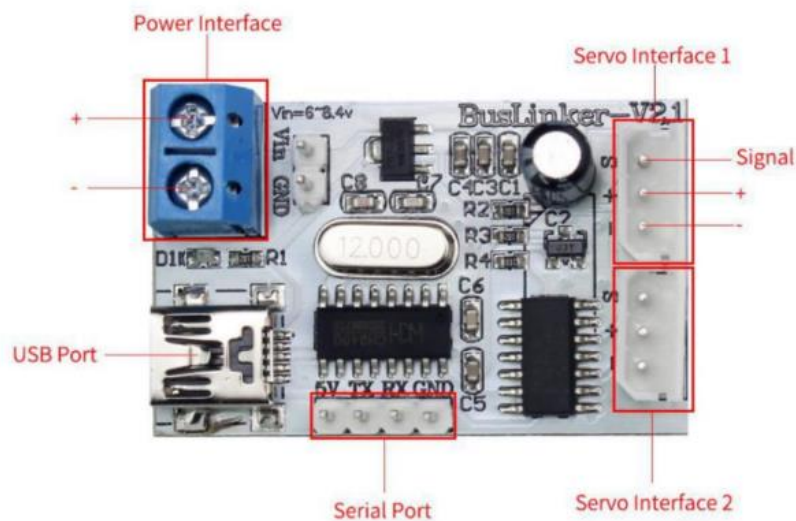
## Motor and controls

The Rover consists of 6 driving motors (**LX-16A Bus Servo**) and 4 steering servos.



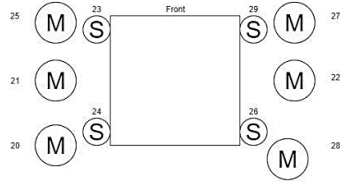
All servos/motors communicate over UART with a baud rate 115200bps and provides:

1. Angle feedback
2. temperature feedback
3. voltage feedback
4. 2 working modes (Continuous running 360 degrees, variable adjust 240 degrees)



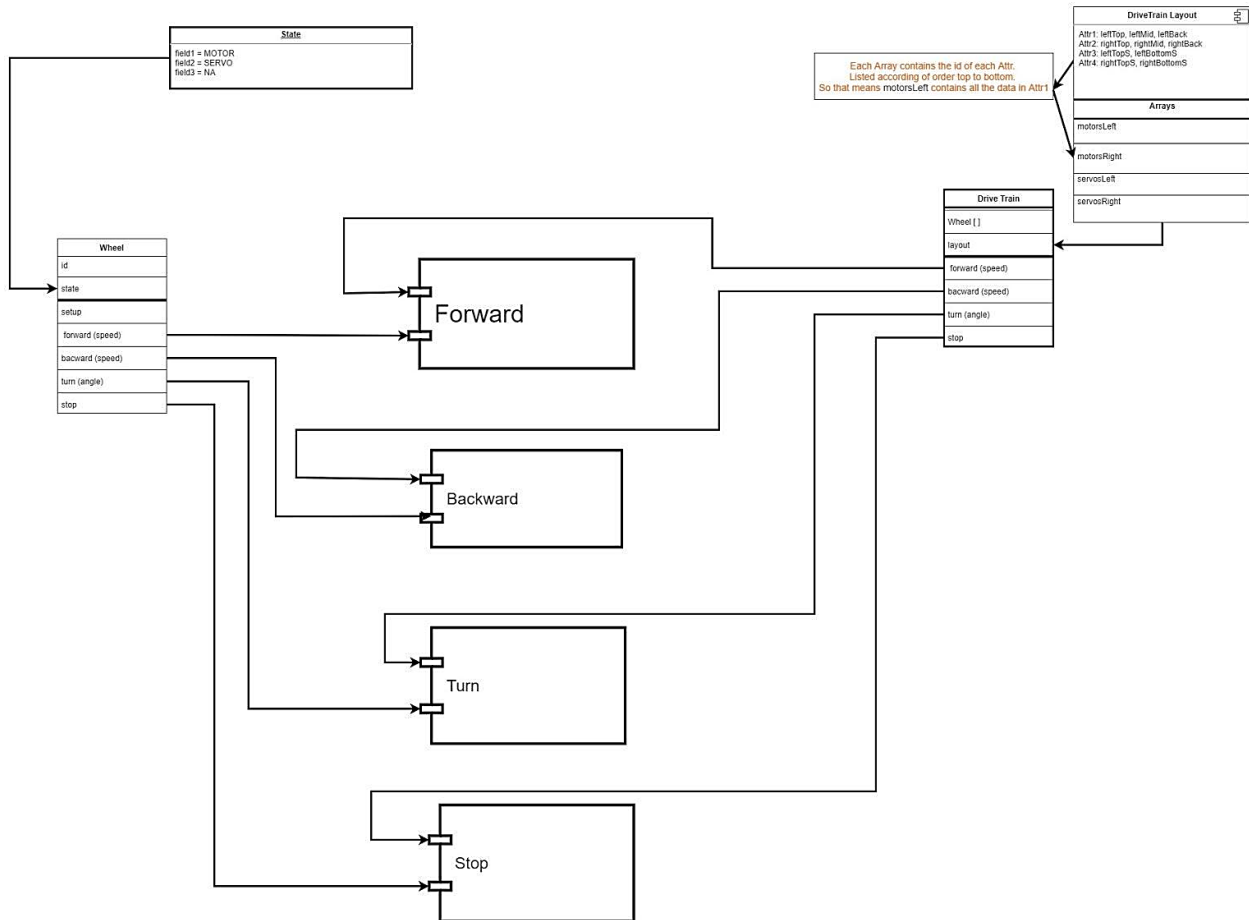
Each servo/motor has a particular ID for which to be commanded by. The chassis and code are defined below:

Drive Train Programmed layout



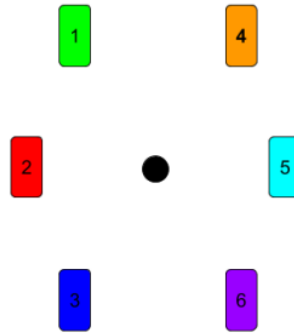
Motor Programmers Layout			
#	LEFT	RIGHT	Motor(M) / Servo(S)
1	25	27	M
2	21	22	M
3	20	28	M
4	23	29	S
5	24	26	S

Wheel Control Logic Construction





When turning the Rover different wheels must run at different speeds. This speed of the wheels is determined by the point of turning and its distance from the wheel. The calculations are as follows:



$$V_1 = X \frac{R_1}{R_2} = X \frac{\sqrt{d_3^2 + (d_1 + r)^2}}{r + d_4} \quad (7)$$

$$V_2 = X \quad (8)$$

$$V_3 = X \frac{R_3}{R_2} = X \frac{\sqrt{d_2^2 + (d_1 + r)^2}}{r + d_4} \quad (9)$$

$$V_4 = X \frac{R_4}{R_2} = X \frac{\sqrt{d_3^2 + (r - d_1)^2}}{r + d_4} \quad (10)$$

$$V_5 = X \frac{R_5}{R_2} = X \frac{r - d_4}{r + d_4} \quad (11)$$

$$V_6 = X \frac{R_6}{R_2} = X \frac{\sqrt{d_2^2 + (r - d_1)^2}}{r + d_4} \quad (12)$$

And -100, 100 is the given turn of the servos, that represents a percentage of the tightest left or right turn in inches.

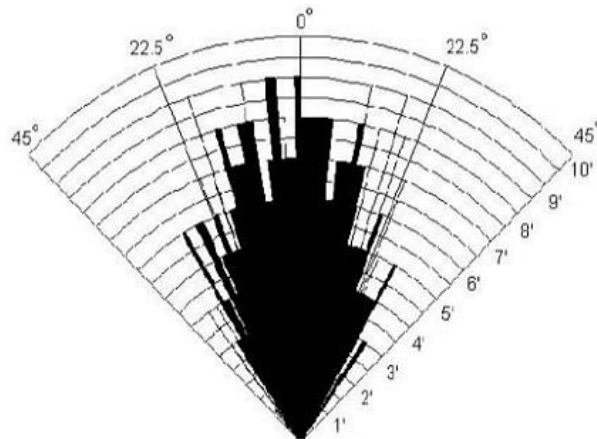
## Object detection

Object detection can use ultrasonic and light-based (LIDAR) sensors. Due to limitations of the M4's memory, the objects are avoided in an ad hoc fashion.

## Ultrasonic Sensor

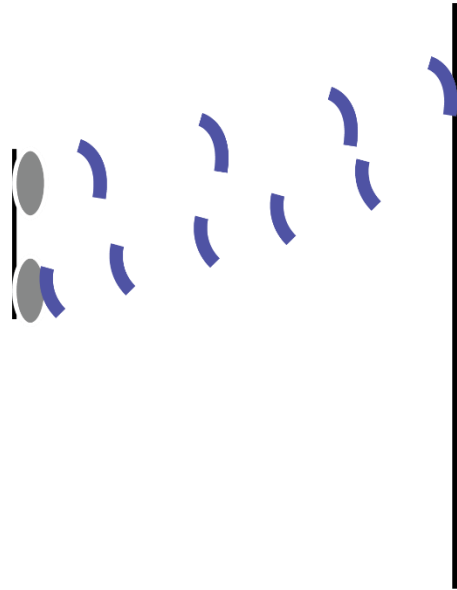


An ultrasonic sensor uses sound to detect the distance of an object. The sound is propagated as a wave, and thus expands omnidirectional as it travels. Each ultrasonic sensor has a particular FoV (Field of View). The exact field of view is as follows:

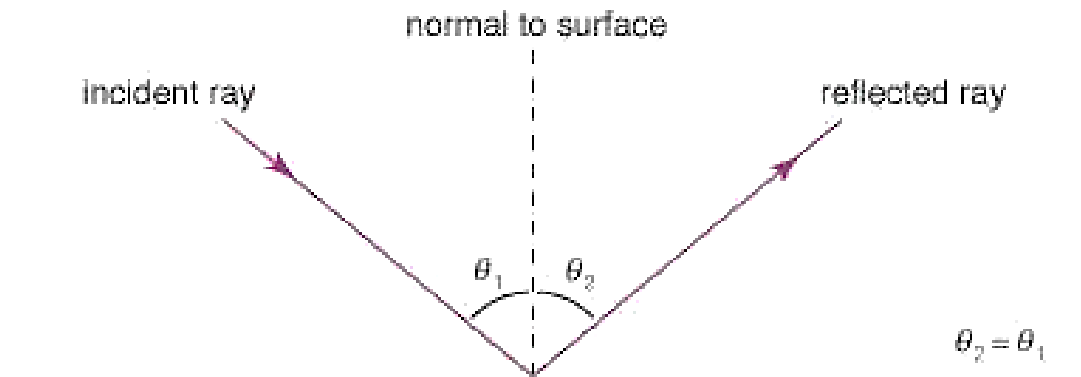


*Practical test of performance,  
Best in 30 degree angle*

The sound reflects off a surface into the receiver. The time difference between when the sound is emitted and then received gives the distance. This speed is dependent on the material it propagates through. Different densities cause sound to travel slower or faster. Therefore, it works differently (but possible) at detecting water, soft bodies, and humid air.



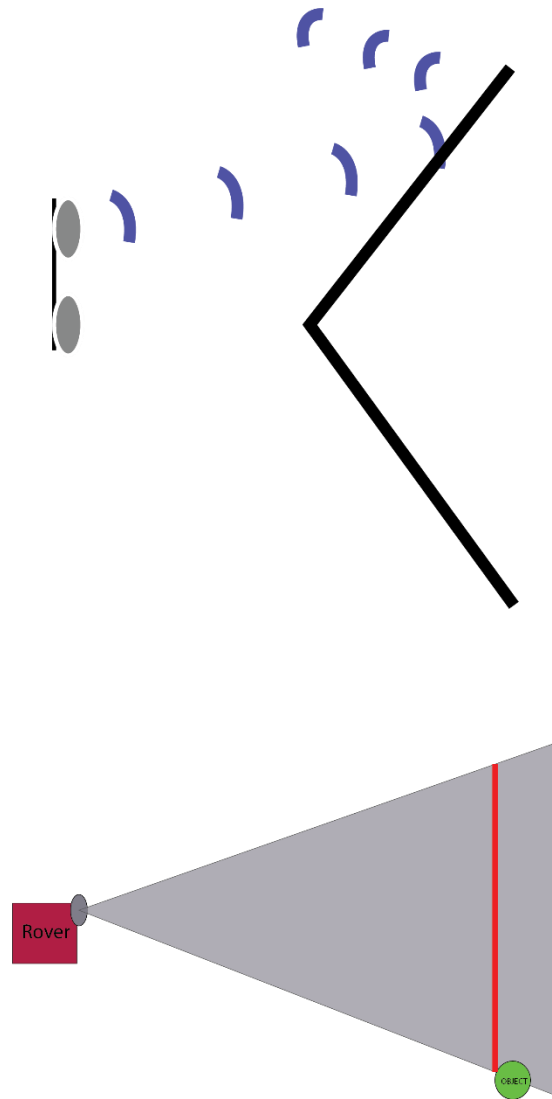
This can be accurately calculated by the following:



© 2006 Encyclopædia Britannica, Inc.

Both light and sound are emitted as waves, which are reflected by constructing a vector with the angle of the dot product of the incident ray with the normal vector

For sharp angles and corners, it does not detect the returning sound wave as illustrated above. Moreover, sound speeds differ within water, and against soft materials.

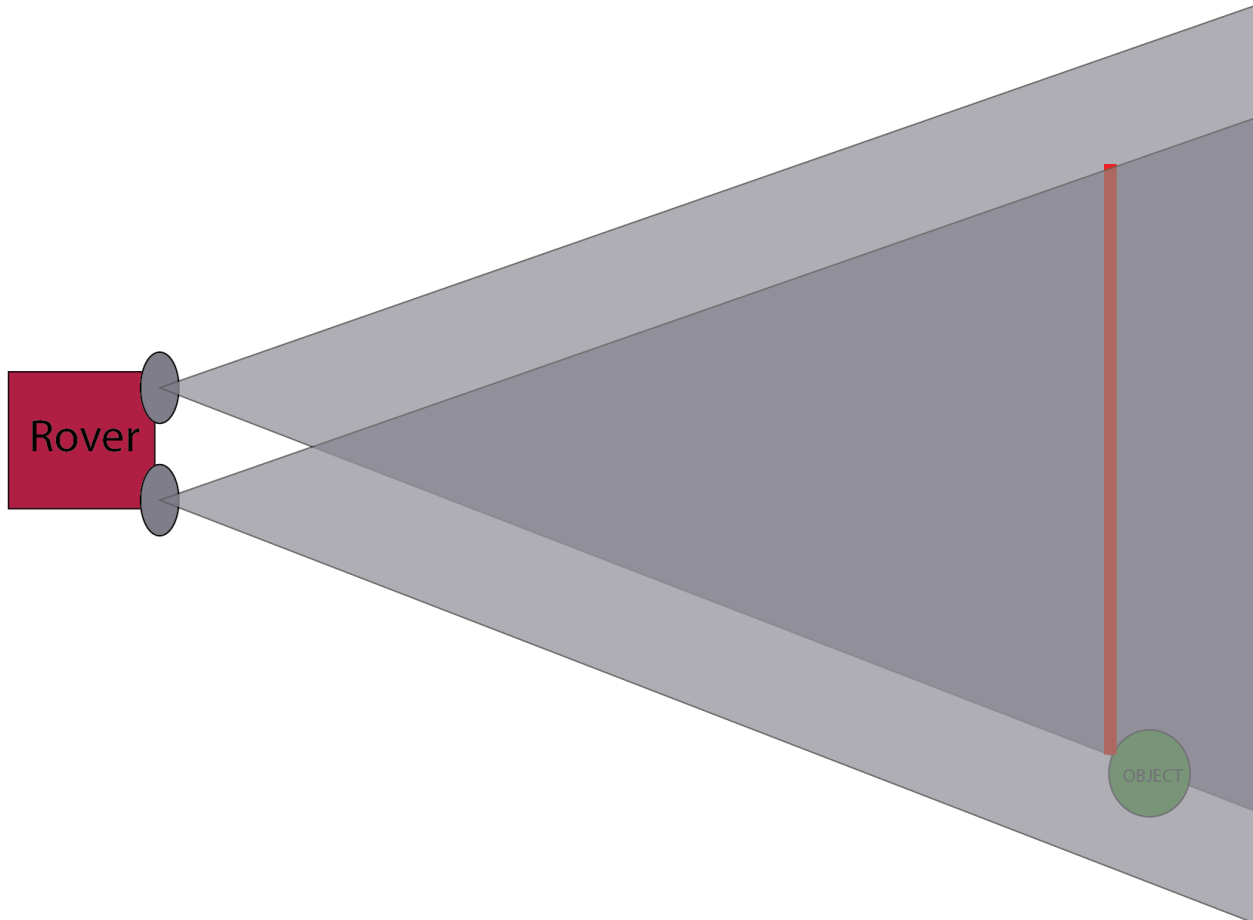


The gray circle is the ultrasonic sensor. The gray area is its field of view. Here you have an object with an assumed width of 20 centimeters. Yet, the red line displays what the rover believes is the width.

This is because ultrasonic sensors can only give distance and not angle. It has little information for where an object exists within its field of view, and thus has a large room for error. This error can be calculated as the following:

$$\varepsilon = 2 \tan\left(\frac{\theta}{2}\right) * distance$$

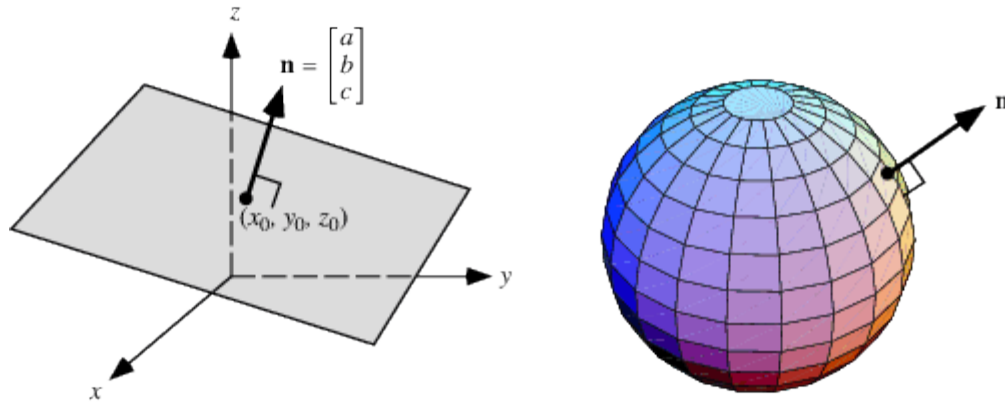
Using two ultrasonic sensors provides a full field of view that covers the entire rover.



Clearly, there is an evident blind spot. Moreover, this does not solve the estimated error. An angle can be estimated using various different mathematical formulas. By calculating the difference each sensor detects the object, an approximate angle can be given. This technique is highly prone to error and is often complex.

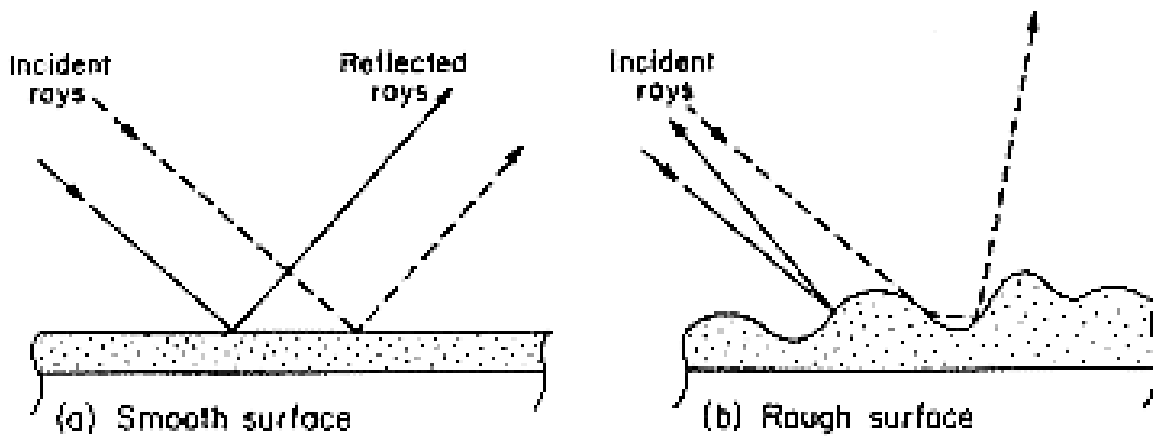
This reflection is extremely important as it gives the difference between sound and light. When sound propagates it propagates omnidirectionally in 3D space. This means that, sound has a cone FOV, and will detect irregular surfaces such as the grass.



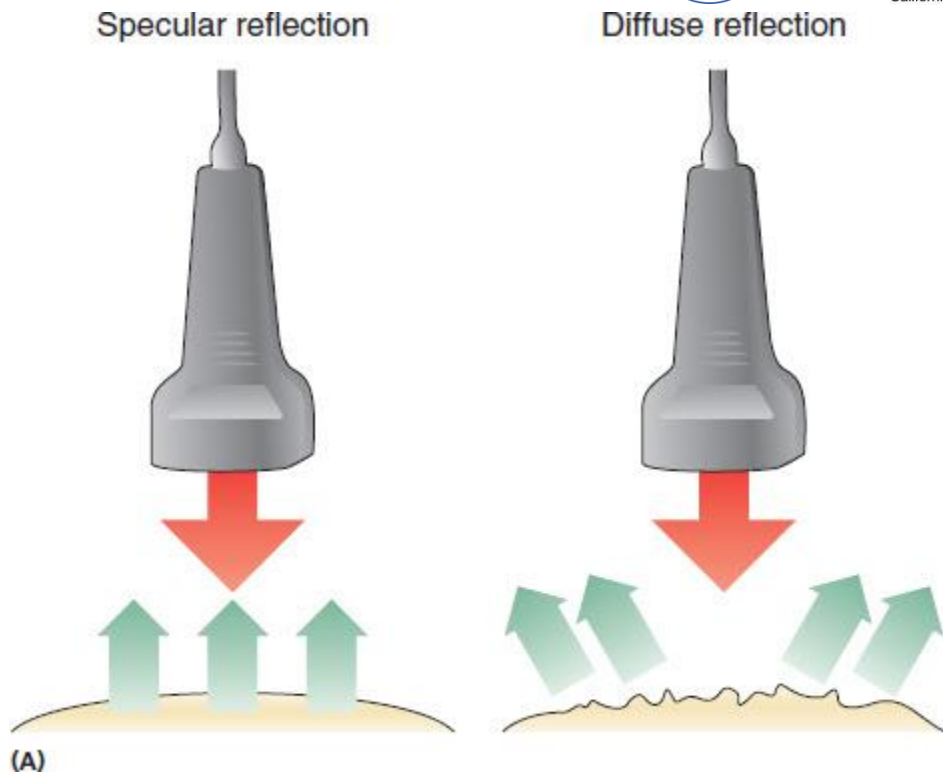


On a curved surface, the reflection cannot be predicted. It can shoot a number of angles, and thus it may reflect back into the receiver.

Additionally, on irregular bumpy planes it is possible for the sound to direct itself back into the sensor, thus causing it to falsely detect the ground.



Because sound propagates omnidirectional, the reflected rays will inevitably be received by the receiver. It is clear here what ultrasound may have trouble detecting. Understanding the law of reflection, and propagation of sound is enough to know what ultrasound can, and cannot detect. A chart or table can never enumerate all objects. In particular, for this project's use case, ultrasound will have trouble on outdoor terrain. This is because the ground is a bumpy surface, and the FOV will cause diffusion with the ground.



Diffuse reflection causes rays to reflect in all directions. Because ultrasound grows omnidirectionally as it travels, it is highly likely it will detect these diffused rays. Thus, inconsistently detecting false objects (such as the ground).

### LIDAR Sensor

A LIDAR is a rotating laser that uses light to detect the distance of an obstacle. Because light has a tiny FOV it can accurately detect an angle, and moreover, diffusion does not affect its receiver. However, due to the small FOV it is not sufficient at covering a large portion of the rover's surface.

LIDAR fixes the tiny FOV issue by rotating and calculating distance in angles at 360 degrees. Both Ultrasound and LIDAR are similar when it comes to the law of reflection, of which is the most important aspect of object detection, but light has a much narrower FOV and travels much faster. Light will not detect an object that does not reflect light such as black surfaces.

Moreover, LIDAR is capable of storing the exact positionings of objects in 3D space, due to its extreme precision. This is good for calculating the rover's position, and essential for indoor vehicles. Due to the memory limitations, and goal of this project, such an implementation is not feasible.

### Collision avoidance

Collision Avoidance uses sensor-based devices to detect range and angle. The obstacle avoidance method assumes a range in meters, and an angle in degrees. It performs flat horizontal planes regardless of sensor used. Due to the limitations of the M4, and the inaccuracies of the GPS, the world position of the obstacles are not stored.

When an obstacle is detected on the desired path at a given range and angle the rover enters a specified state given by the state machine diagram. Once the rover has entered an avoidance state it performs the following sequence of events:

DETECTED X RANGE, Y ANGLE.	BACKUP X METERS	TURN LEFT/RIGHT 90 DEGREES	FORWARDS X METERS.
----------------------------	-----------------	----------------------------	--------------------

The implementation is within the AutonomousDrive.cpp file.

In general, obstacle avoidance is not sufficient for getting to many destinations. It fails many practical scenarios (buildings, pathways, roads, gates, circular arrangements, right corners, unsolvable paths etc.) In most cases the best way to avoid an obstacle is to simply never aim at it. Never direct a path towards an obstacle.

It should be noted that the failures produced by obstacle avoidance are related to the algorithm, not any particular sensor. The sensor is irrelevant here, and rather the methodology is flawed. Storing mini maps, or using visual cameras can both aid obstacle avoidance, but are memory intensive. Yet, even the very best obstacle avoidance system will not match a simple pathfinder. Obstacle avoidance is necessary, but acts as an aid to pathfinder. As no pathfinder can account for all objects, and no detection system can generate a path. As such, obstacle avoidance is a last resort for when pathfinding fails, but it should never be the main tool for navigation.

## Navigation

Navigation is the most important and complex aspect of this software. This document will brief over the algorithms, equations, and devices used. It is to show the math for those wishing to understand the calculations. There are many ways to navigate, and the most obvious being Breadth First Search. As such, it is best to read the Software Design Document to know why each particular algorithm is used, and exactly what it does in the overall system. There may also be a fair bit of terminology here that is explained in more depth within the Software Design Document.

## Localization

Localization is tracking the rover's orientation and global positioning in 3D space. There are two spaces which we will refer to. World Space represents the Global Positioning of the rover relative to the Earth. In actuality, World Space is the GPS device, which gives Global Positioning. Local Space represents positioning relative to the rover. Where the starting position of the IMU represents the vector [0,0,0]. Both the World Space and Local Space are in  $\mathbb{R}^3$ . Due to the scope of our design, there are only a few devices used for the entirety of localization. The following devices are GPS and IMU. These two devices complement each other. A brief summary would be that GPS updates slow, consistent, low resolution, and prevents drift. Meanwhile, IMU updates fast, inconsistent, high resolution, highly prone to drift. GPS gives world positions, while IMU gives local.

## Heading and bearing tracking

Heading is the current direction the rover is pointing. Bearing is the desired heading to reach a point. Track or Course refers to the path calculated and travelled. Here we show a possible method to match the rover's Heading with the Bearing and proceed on the course.

Given a destination within World Space, that is a Latitude and Longitude, the rover must find the bearing. Once the bearing is found, the heading of the rover must then match the bearing, and the rover linearly follows the line. The Heading and Bearing Tracking algorithm repeats this process until the destination is reached. Tolerance is needed to account for errors produced by the heading, and the errors produced by mechanical processes such as misalignments of the wheels.

---

### Bearing Tracking Algorithm

---

- 1: While ( $|\text{Heading} - \text{Bearing}| < \text{tolerance}$ )
  - 2: Forwards
  - 3: Turn
- 

There are two slight adjustments needed to be made for a practical implementation. Firstly, the turning sequence must be minimal. The rover must turn the direction closest to the desired angle. Secondly, the turning must be continuous and not discrete. The wheels must slightly adjust, and not abruptly to avoid erroneous movement.

```
void followBearing(){
float angle1 = ((bearing - heading) / 180) * 100;
float angle2 = (((bearing+360) - heading) / 180) * 100;
float angle3 = (((bearing - 360) - heading) / 180) * 100;
    if (abs(angle1) < abs(angle2) && abs(angle1) < abs(angle3))
    {
        wheelDirection = angle1;
    }
    else if (abs(angle2) < abs(angle3) && abs(angle2) < abs(angle3))
    {
        wheelDirection = angle2;
    }
    else
    {
        wheelDirection = angle3
    }
}
```

The smallest angle must be found. To find the smallest possible angle, each possible angle reference must be calculated.

For the calculation of heading the IMU BNO055 is used. The BNO055 fuses both the gyroscope, accelerometer, and magnetometer in order to provide a quaternion that represents the exact rotation

of the device. For our usage, we convert the quaternion provided in order to produce a scalar value representing the angle of the forward vector. For our purposes, we use the (NASA) JPL Quaternion Convention.

```
void orientation::computeCompass(sensors_event_t * event) {
    heading = fmod(360 + toDegrees(quaternion.toEuler().x()), 360);
    incline = -toDegrees(quaternion.toEuler().z());
}
```

Where the function, “toEuler”, represents the conversion between quaternions and Euler angles. The math is standard Quaternion Conversion defined as follows:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)) \\ \text{asin}(2(q_0q_2 - q_3q_1)) \\ \text{atan2}(2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

The variables  $q_{\{n\}}$  represent the elements of the quaternion vector. The vector is 4D. This conversion properly calculates the YAW angle of the rover to around 0.01 degree of precision. It should perhaps be noted that a simple Magnetometer Compass is not capable of such precision.

## Calculating bearing and distance

The Bearing is the directional point for which the rover must travel to get to a particular segment of the line. The bearing is constant, calculated using the latitudinal and longitudinal points of the waypoint. This calculation produces a curve on a sphere, that is then approximated as a line with the heading calculations. These equations update multiple times per second.

```
double RoverGPS::calculateBearing(double latStart, double lonStart, double latDest,
                                  double lonDest) {
    latStart = toRadians(latStart);
    latDest = toRadians(latDest);
    lonStart = toRadians(lonStart);
    lonDest = toRadians(lonDest);
    double y = sin(lonDest - lonStart)*cos(latDest);
    double x = cos(latStart)*sin(latDest) - sin(latStart)*cos(latDest)*cos(lonDest-
lonStart);
    double bear = toDegrees(atan2(y, x));
    bearing = fmod((360 + bear), 360.00); //ensures that degree is between 0 and 360.
    return bearing;
}
```

```
double RoverGPS::calculateDistance(double latStart, double lonStart, double latDest,
double lonDest) {

    latStart = toRadians(latStart);
    lonStart = toRadians(lonStart);

    latDest = toRadians(latDest);
    lonDest = toRadians(lonDest);

    double phiStart = latStart;
    double phiDest = latDest;

    double deltaLat = (latDest - latStart);
    double deltaLon = (lonDest - lonStart);
    double alpha = sin(deltaLat / 2)*sin(deltaLat / 2)+

        cos(phiStart)*cos(phiDest)*

        sin(deltaLon / 2)*sin(deltaLon / 2);

    double c = 2 * atan2(sqrt(alpha), sqrt(1-alpha));
    distance = (EarthRadius * c)*1000;
    return distance;
}
```

The haversine formula is used to calculate the distance between the current global positioning of the rover, and the way point. The haversine formula is the spherical laws of cosines but adapted for numerical computation which adjusts for floater imprecision. The following shows the calculations performed in the code:

$$d = 2r \arcsin \left( \sqrt{\text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1)} \right)$$

$$d = 2r \arcsin \left( \sqrt{\sin^2 \left( \frac{\varphi_2 - \varphi_1}{2} \right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2 \left( \frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

When the distance is below the given accuracy radius, then the destination is reached.

### Calculating directional movement

When the rover moves, it moves along three axes, the X, Y, Z axis. Calculating the proper direction, the rover moves in helps at determining small distances which the GPS cannot perceive. This is particularly important when avoiding an obstacle.

Two things are needed to calculate this movement. Firstly, the velocity can be calculated by taking an integral of the acceleration values, a second integral gives the position.



$$\iint_0^t (a + \varepsilon + \lambda) dt$$

$$\int_0^t (at + \varepsilon t + \lambda t + v_0) dt$$

$$\frac{1}{2}at^2 + \frac{1}{2}\varepsilon t^2 + \frac{1}{2}\lambda t^2 + v_0 t + x_0$$

There is an amount of drift that naturally occurs due to this calculation.

$\varepsilon$  is floater imprecision, and  $\lambda$  is physical mechanical drifting. Both grow quadratically. Additionally, the integral has a mathematical error produced by the additional variables.

$$Error = \frac{1}{2}\varepsilon t^2 + \frac{1}{2}\lambda t^2 + v_0 t + x_0$$

It should also be noted that the particular case of integration performed has different values of error. In our usage case we used a trapezoidal integration, that was integrated every 100 milliseconds:

```
//Trapezoidal double integration to compute position and velocity from
accelerometer.
//Delta time is 0.1 since we take a sample every 100 milliseconds.
void orientation::trapezoidalIntegration() {

    float basevelocityx = abs(velocity.x());
    float basevelocityy = abs(velocity.y());
    float basevelocityz = abs(velocity.z());

    float baseaccelerationx = ax;
    float baseaccelerationy = ay;
    float baseaccelerationz = az;

    float accelerationx = abs(acceleration.x());
    float accelerationy = abs(acceleration.y());
    float accelerationz = abs(acceleration.z());

    velocity.x() = basevelocityx + baseaccelerationx + ((accelerationx -
```

```
baseaccelerationx)/2);
    velocity.y() = basevelocityy + baseaccelerationy + ((accelerationy -
baseaccelerationy)/2);
    velocity.z() = basevelocityz + baseaccelerationz + ((accelerationz -
baseaccelerationz)/2);

    velocity = velocity* 0.1;//Delta time.
    if (abs(acceleration.x()) < tolerance) {
        velocity.x() = 0;
        basevelocityx = 0;
    }
    if (abs(acceleration.y()) < tolerance) {
        velocity.y() = 0;
        basevelocityy = 0;
    }
    if (abs(acceleration.z()) < tolerance) {
        velocity.z() = 0;
        basevelocityz = 0;
    }
    float forceX = basevelocityx + ((abs(velocity.x()) - basevelocityx) / 2);
    float forceY = basevelocityy + ((abs(velocity.y()) - basevelocityy) / 2);
    position.z() = position.z() + basevelocityz + ((abs(velocity.z()) -
basevelocityz) / 2);
    if(abs(acceleration.x()) > tolerance || abs(acceleration.y()) > tolerance){
        applyForwardsForce(forceX+forceY);
    }
}
```

The final result only takes into consideration the magnitude of the value. This magnitude is then converted into an X, Y, Z coordinate system using the forward vector.

```
//Applys linear acceleration according to the orientation of the rover using the
Hamilton Product.
void orientation::applyForwardsForce(double f){
    imu::Vector<3> forwards = imu::Vector<3>(0, 1, 0); //The forwards vector,
representing the Y plane going forward.
    forwardsVector = quaternion.rotateVector(forwards);//Performs the hamilton
product using cross products.

    position = position+(forwardsVector*f);
    distanceTraveled = sqrt(position.x()*position.x() + position.y()*position.y());
}
```

This gives the total direction traveled in the 360-degree plane. Since the Rover only has 2 degrees of freedom, the Z axis is ignored.

## Calibration

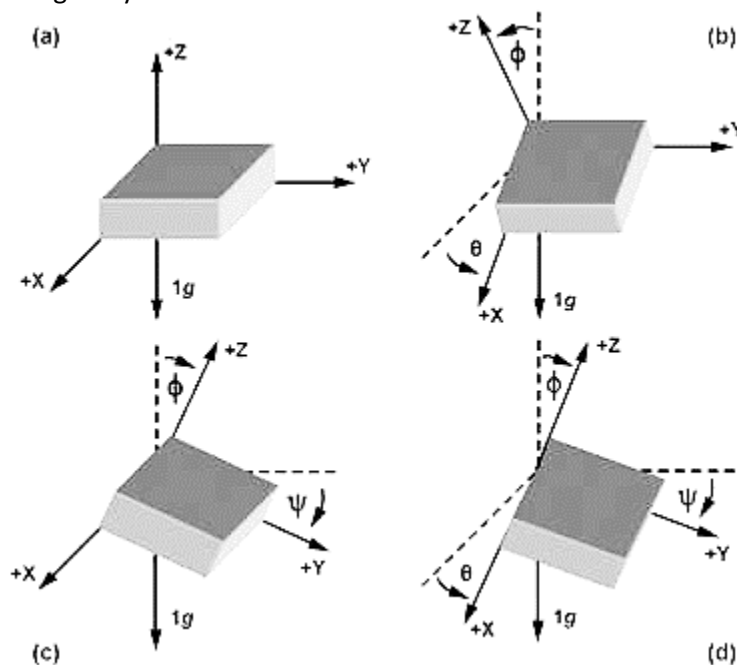
The BNO055 needs to calibrate in order to successfully calculate orientation. Additionally, there is a calibration step performed to get the average values of the system movement.

```
//Infinite Impulse Response
void orientation::IIRFilter(double x, double y, double z) {
    double k = 0.9;
    xf = k * xf + (1.0 - k) * x;
    yf = k * yf + (1.0 - k) * y;
    zf = k * zf + (1.0 - k) * z;
}
```

This stabilizes the movement values, which gives more precision to the meters travelled.

## Addendum

One question that was asked during many presentations by fellow college students was the following, “Why do we need to calculate roll angle?” The answer, is to eliminate the force of gravity. Here is a brief calculation to eliminate gravity.



This image represents the states it experiences due to the forces of gravity. As seen, as the system rotates it experiences this force in all axes. This can cause the rover to miscalculate its orientation and velocity. This can be calculated precisely with the following:

$$\begin{bmatrix} \cos\theta_y \cos\theta_z & -\sin\theta_z \cos\theta_x + \sin\theta_x \sin\theta_y \cos\theta_z & \sin\theta_x \sin\theta_z + \cos\theta_x \sin\theta_y \cos\theta_z \\ \cos\theta_y \sin\theta_z & \cos\theta_z \cos\theta_x + \sin\theta_x \sin\theta_y \sin\theta_z & -\sin\theta_x \cos\theta_z + \cos\theta_x \sin\theta_y \sin\theta_z \\ -\sin\theta_y & \cos\theta_y \sin\theta_x & \cos\theta_x \cos\theta_y \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

This can be solved as a system of equations. Let  $J$  be the rotation matrix. Let  $g$  be the gravity matrix. Let  $A$  be the linear acceleration. Let  $B$  be the total acceleration gathered from accelerometer.

$$(A + g)J = B$$

$$AJ + gJ = B$$

$$AJ = B - gJ$$

$$AJJ^{-1} = B - gJJ^{-1}$$

$$AI = B - gI$$

$$A = B - g$$

Hence, it is necessary to have all three axes of rotation in order to solve acceleration. This is performed on the microprocessor on the BNO055. Additionally, the BNO055 performs sensor fusion akin to the Kalman filter in order to retrieve more accurate results.

## Troubleshoot diagnostic

Designing the rover means encountering many bugs. This guide gives techniques to debug specific problems. These diagnostics are concerned with the wheel test sketch, and the manual destination. The virtual rover has its own separate FAQ document attached with the application. The very best benchmark to test the rover is manual destination. It combines all of the code, sensors, wheels, into a single sketch to perform a single task. It is best to use manual destination and not virtual sketch. Virtual sketch will set points automatically, but the driving code is identical. The issue here is that, virtual sketch chooses the shortest path with the least amount of turns (thus reducing mechanical stress).

## What points to test with?

In general, the best test will stress all parts in order to create a failed run. By finding when it fails it is easy to see where to improve.

Firstly, test it outdoors and preferably on grass. There should be multiple points that cover the North, South, East, and West. By doing so it stresses the turning code, and more so the bearing and tracking math. Additionally, points that cause the rover to turn 180 degrees, and points where the bearing is 180 degrees or 0 are also important to test. If you review the navigation document it is clear why, finding the smallest angle means those angles can cause the rover to turn in place.

All points should be at least 10 meters apart, since the GPS can only give 3 meters of accuracy consistently. The longer the distance the more room for error, and thus the better the test.

## How to generally troubleshoot rover

There are a few general techniques to keep in mind that work for nearly all sketches.

Firstly, be sure to connect the computer to the rover in order to see the serial monitor. This may be

difficult while it is driving, so here is a list of possible ways to do so:

- 1) Buy an SD card, every minute the rover records the serial monitor and saves to the SD card in a text called "RTD."
- 2) Buy a long USB connector, and walk with the rover.
- 3) Turn off the wheels, run the code and check the monitor.
- 4) Place the laptop (if it is lightweight) on top of the rover with the Serial Monitor opened. After finishing, unplug the laptop and read the serial monitor (it will record it).

Option one is good if the issue is simple, oftentimes option two is best if real-time knowledge is needed. The exact meanings of each print in the Serial Monitor is covered in the Quick Start Document. The remainder of this document covers what to do if an error is identified.

### Wheel test incorrect wrong direction and or turn

Ensure that each servo ID matches the Motors and Controls document. Ensure that the wheels are connected to the proper pins according to the Motor and Controls document (Serial 2). Ensure that all libraries are properly setup according to the Quick Start document.

If none of the above works, then make sure to properly read Electrical and Mechanical Design documents to ensure that the wheels are connected and positioned properly.

## Rover turns in place indefinitely

Check the Serial Monitor, make sure it has properly calibrated. If it has not, it will clearly display that it is still calibrating. If it does this after calibration, then check the bearing and heading angles. One way to check such angles is using this website:

<https://www.movable-type.co.uk/scripts/latlong.html>

If you enter your location, and the destination, the distance and bearing should match with the serial monitor. If not, refer to the Navigation section, and check the math to see if an error exists.

If the bearing is 0, or 180, then check the smallest angle math manually, and see if the rover has chosen an angle, or continues to attempt to choose the smallest. At an angle of 180, and 0, there are multiple smallest angles, the code adjusts for this, but this is a critical area to check if this error occurs.

If nothing works, and the math works correctly when calculated through MatLab or Unity, then check that all servos are on the proper ID according to the Servo and Motor Controls document.

## Rover heads towards wrong direction

Firstly, make sure to have the serial monitor printed out to an SD card or a screen. Then check the destination and location points against this website:

<https://www.movable-type.co.uk/scripts/latlong.html>

The bearing and distance should match the calculator. If not, then read the Navigation document and ensure it matches the Movable's explanation. It should be noted that a thorough understanding of spherical trigonometry is needed. Do not simply copy the math to match Movable exactly, as the calculations presented on the website (the code posted), are inaccurate, and the actual website's calculator uses a different approach.

The best way to accurately understand the Navigation math is reading this document by Bob Chamberlain from NASA JPL: <https://cs.nyu.edu/visual/home/proj/tiger/gisfaq.html>. If the team has continuing issues and suspects the calculations are an issue, then it is favorable that at least a single member understands that entire document.

## Rover's bearing and distance match calculator, but still goes wrong direction

Then this is an issue with the IMU. Particularly the BNO055. Ensure that the IMU is properly updating, and calibrated, and wired. To do this, turn off the wheels, and turn the rover around and see if it properly updates the heading. Sometimes, the heading may not update at all, and this is due to wiring issues. Read the Electrical Design Documents for proper setup of the BNO055. To troubleshoot the BNO055, run the RawDataIMU sketch located within examples. Then turn the rover in place and see the updated values.

## Object avoidance not turning in place and or going forwards

The obstacle avoidance should match the obstacle avoidance diagram regardless of sensors used. If not then do the following.

Check the Serial Monitor's state, is it in the Avoidance state when detecting an obstacle? If not, make sure the sensors are working, and the code changes the state to avoidance. If it is unclear how the states work, then check the State Machine Diagram Document.



Ensure the IMU is properly connected. Check the mechanics, is the rover properly turning, or stuck? Check the Navigation Math, final section “Distance Movement.” and see if the values, and or code matches exactly. In general, the IMU is used to control obstacle avoidance. Turning in place requires the gyroscope to calculate how much to turn. Moving a particular distance requires the accelerometer to measure the distance.

If none of the above works, and it is suspected that perhaps the calculations are incorrect, then run the Distance Movement Sketch and ensure the expected values match this document:

<https://www.nxp.com/docs/en/application-note/AN3397.pdf> (it is again important to understand what the document describes, and not copy verbatim, as our implementation is not exactly the same). If it is not clear how to test this, then read the Software Design Document Obstacle Avoidance section, where the exact method is covered in depth. Additionally, read the Navigation document for more knowledge on drifting.

Lastly, the current code avoids obstacles in exact angles, and meters. If one wishes to change this to a time-based approach, then this may temporarily fix and simplify the code. How to implement such an approach, and the pitfalls are described in the Software Design Document.

## Testing code from home

One of the best ways to test BNO055 is using the Virtual Rover code located here: [https://github.com/U-K-L/virtual\\_rover](https://github.com/U-K-L/virtual_rover)

This requires Unity to be installed, it will give the ability to visually see the rover turning in 3D space. Simply run the localization sketch located in the examples folder, then connect it to the proper serial port (USB). Afterwards, turning the rover in place will update the 3D rover model. It is much easier to see if the IMU is properly updating, opposed to reading quaternion data.

The code comes, for free, with a queue based serial connection between Unity and Arduino. Usually, this costs money, but it comes for free here. If a team member has a fair understanding of Unity, this can drastically speed up programming time, especially if the team only has access (or no access) to the rover. Just to name a few things, it can be properly programmed to visualize Ultrasound, Law of Reflection, LIDAR scans, IMU rotations, Navigation math, Wheel speed, Ackerman Steering, GPS, Pathfinding, etc. Our Software Team has made heavy use of this program to debug many of these errors.

In addition, the program is free to be used for other projects as seen here:

<https://www.instructables.com/id/Ultrasonic-Joystick/> which uses our free solution to make an ultrasonic joystick.

## Addendum

Calculator for Bearing and Distance: <https://www.movable-type.co.uk/scripts/latlong.html>

In depth explanation of Haversine Formula: <https://cs.nyu.edu/visual/home/proj/tiger/gisfaq.html>

IMU calculations for avoiding obstacles: <https://www.nxp.com/docs/en/application-note/AN3397.pdf>

## GitHub public repositories

MC3 NASA Mars Rover Project repository <https://github.com/mc3-nasa>

## FPGA Design for Object Detection Team 2

*Will Pastor, Nicholas Gahman, Corey Shive*

### Introduction

The FPGA project was created as a sub-project of the Mars Rover Project, whose goal was to create an autonomous rover that could go to a given coordinate of longitude and latitude with zero human intervention using microcontrollers. One of the possible pathways to this goal was a dual camera system that would allow the rover to see just like a human would. Originally this idea was shut down because most microcontrollers cannot handle the computational requirements that a dual camera system would have. Eventually, a device that could handle said requirements was found. Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects <sup>(1)</sup>.

### Choosing FPGA technology

A FPGA is a hardware device which could be programmed and reprogrammed on the fly. This allowed the FPGA to be much more customizable than most microcontrollers, and more importantly could have all of its computational power directed towards a single goal. This single ability allowed it to have the computational capacity needed to make the dual camera system viable. Consequently, the FPGA/dual-cam project was born.

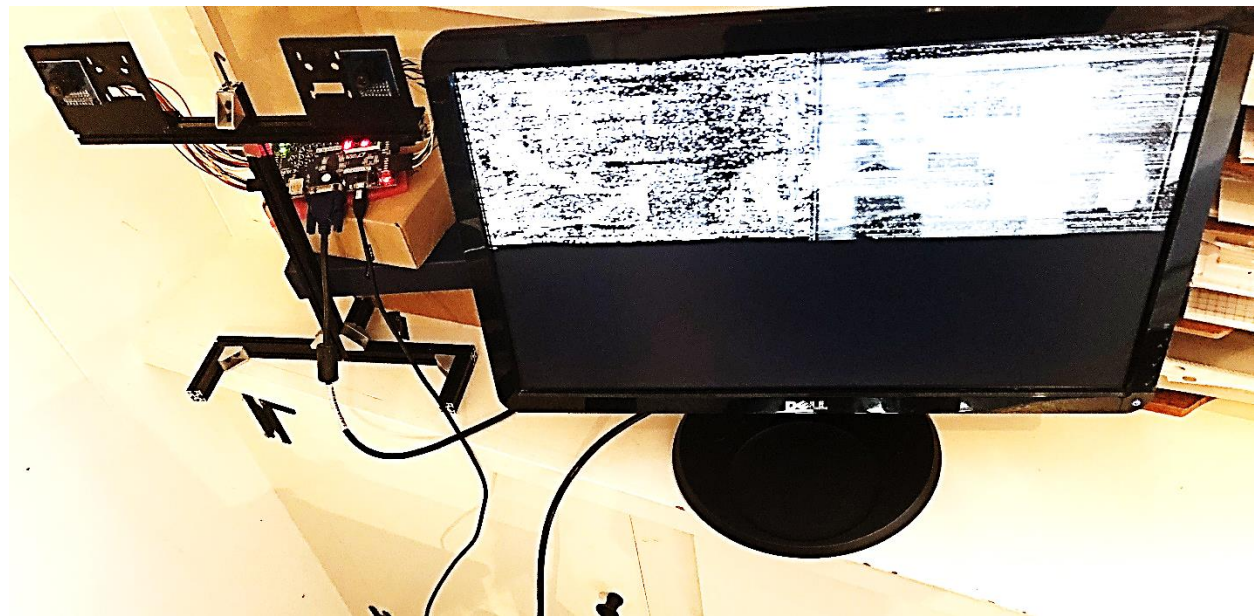
During the 2019 Fall semester we concentrated on the FPGA Alchitry boards, copper (CU) and gold (AU) <sup>(2)</sup>. The Alchitry boards were more suited for people with some FPGA expertise, and we needed an entry-level FPGA board suited for students getting started with FPGA technology. Early 2020 Spring semester, we selected the Basys 3 <sup>(3)</sup> as our development board.



*Figure 1: A picture of the dual cameras and FPGA wired together.*

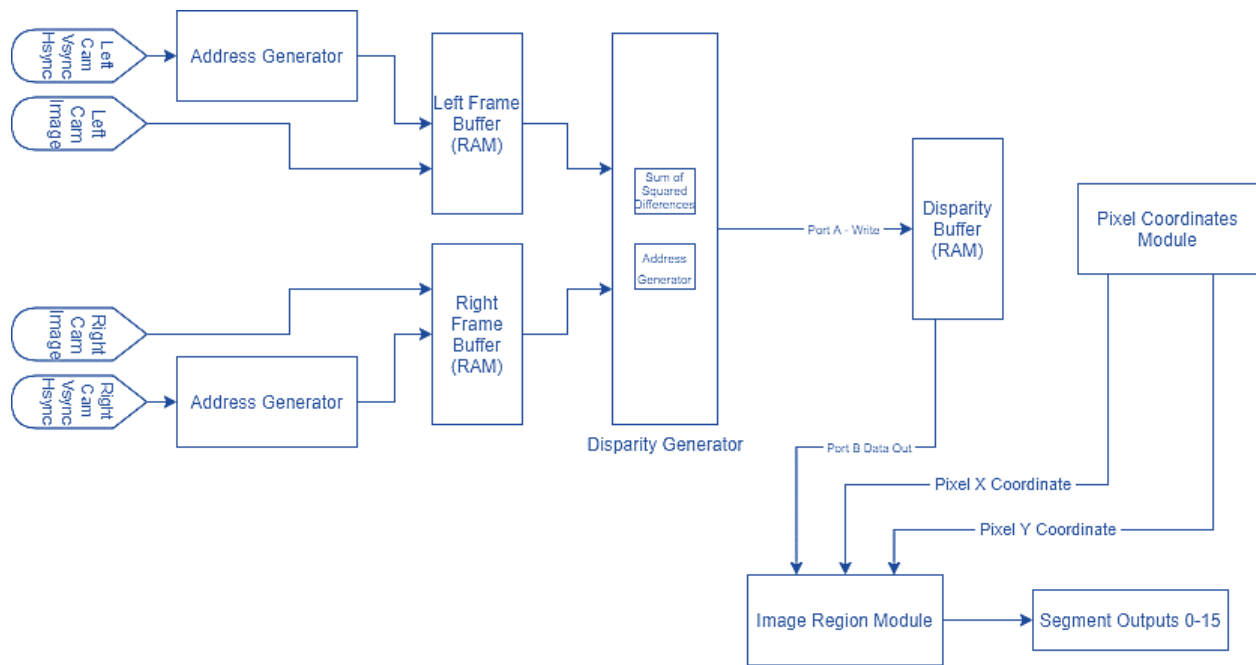
## Description of the image processing

The goal of this project is to use the dual camera system to find obstacles, and then broadcast the relative location of those obstacles to the main microcontroller on the rover. In order to do this, a five-step plan is initiated. The first step is to take the two images from the two cameras, and process them together to create a disparity map that estimates the depth at each point in the frame. Once this is done, the disparity map is divided into a 4x4 block grid. Each of the 16 blocks' average depth is found. If a given block's average depth is closer than a given threshold value, said block is dubbed an obstacle. Finally, the information of each block is sent back to the microcontroller in charge of the rover navigation. For now, 16 outputs are being used, but it may be simplified to four outputs or possibly even three, concerning the left, right and center respectively.



*Figure 2: A picture of the resulting display. The left image is the depth map, while the right image is what the dual cameras see.*

### Image processing diagrams



*Figure 3: A diagram of steps one and two of image processing.*

**Image Region Module:**

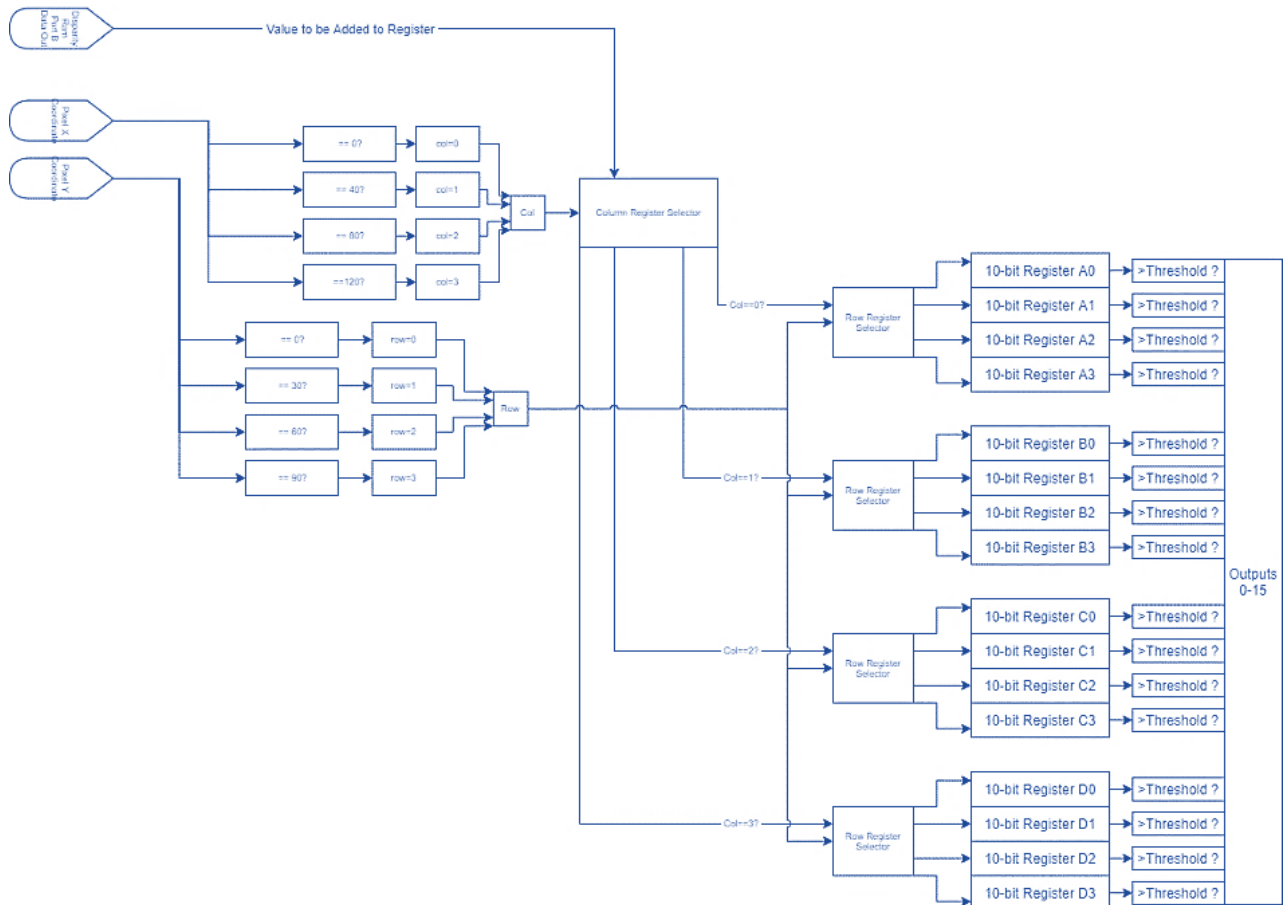


Figure 4: A diagram of steps three, four and five of image processing.

**References**

- (1) <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>
- (2) <https://alchitry.com/>
- (3) <https://projects.digilentinc.com/products/basys-3>



## The NASA Mars Curiosity Rover Teams

-In the order of the presentation

### Team 3 Mechanical Design

**Ezra Galapo**



Ezra is a student at Temple University but came as a guest student. He joined the NASA Mars Curiosity Rover team starting with the 2019 Fall semester and continuing with the 2020 Spring semester. His contribution to the project was the 3D-printing and laser cutting of the parts required for rovers 1 and 2.

### **Jaden Weed**



Jaden was a student at North Penn High School but came as a guest student under the “NASA Engage and Inspire” Program. Jaden joined the NASA Mars Curiosity Rover team starting with the 2019 Fall semester and continuing with the 2020 Spring semester. The knowledge of 3D-printing and laser cutting he acquired as a student in the NPHS Engineering Academy made him a valued member of the team. Jaden started his studies at Penn State University Park as a first-year student.



## Team 4 Electrical Design

**Paul John Balderston**



PJ graduated from Montgomery County Community College with an Associate Degree in Engineering Science, with an Electrical Engineering concentration at the end of the 2020 Spring Semester. He is currently studying Physics and Mathematics at Penn State University Park. Paul John aims to pursue a doctoral program in Physics after completing his undergraduate education. PJ joined the NASA Mars Curiosity Rover team starting with the 2019 Fall semester and continuing with the 2020 Spring semester. His contribution to the project was the electrical design of both rovers 1 and 2. This included the preparation of three PCBs which were manufactured, then implemented onboard the rovers.

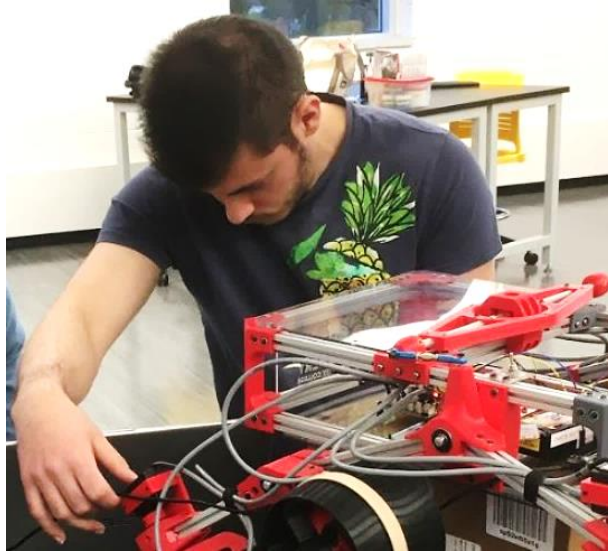
## Team 1 Software Design

**Noah Williams**



Noah earned his Associate of Science degree in 2019 Fall semester. He is currently enrolled at Temple University for Computer Science and joined the NASA Mars Curiosity Rover team starting with the 2019 Fall semester and continuing with the 2020 Spring semester. He worked on high-level software such as mathematical calculations and Artificial Intelligence. His contributions to the project were writing the software for navigation, collision detection, pathfinding, and obstacle avoidance for rovers 1 and 2.

## Salvatore Sparacio



Sal is working towards an Associate in Science degree in Computer Engineering. He joined the NASA Mars Curiosity Rover team starting with the 2019 Fall semester and continuing with the 2020 Spring semester. He worked on low-level features that connected the hardware with the software such as IO processing, motor controls, customized serials, and state machines for rovers 1 and 2.

## Team 2 FPGA Design

### William Pastor



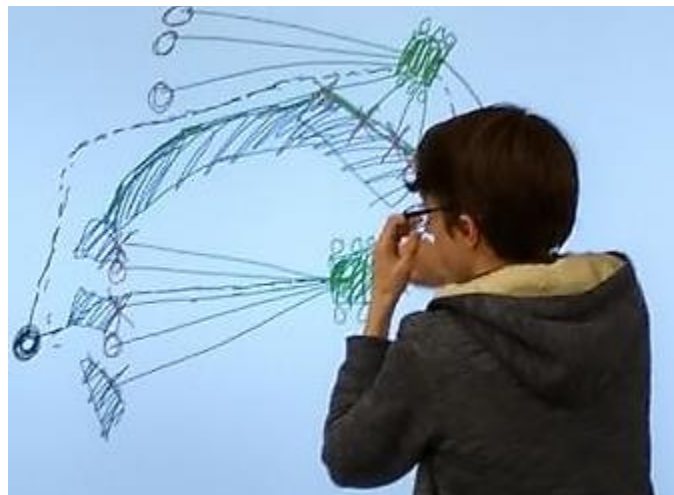
Will enrolled at the college during high school and continues coursework while conducting independent study projects. He's also considering transfer opportunities for undergraduate and graduate study in Artificial Intelligence and Machine Learning. Will joined the NASA Mars Curiosity Rover team starting with the 2020 Spring semester. Working alongside fellow students Nick Gahman and Corey Shive, Will proposed and led the research and development of an FPGA-based sensor for obstacle detection and collision avoidance.

### Nicholas Gahman



Nick is a dual enrollment student in Computer Science and Engineering. He joined the NASA Mars Curiosity Rover team starting in the 2019 Summer workshop, 2019 Fall semester and continuing into the 2020 Spring semester. In the first half of the Spring semester, Nick worked alongside fellow students Noah Williams and Salvatore Sparacio, contributing to the obstacle detection and relative movement portions of the rover. In the second half of the Spring semester, Nick worked alongside fellow students Will Pastor and Corey Shive researching FPGA devices to supplement Noah and Sal's work on object detection and collision avoidance. Nick continued work on the FPGA into Summer 2020 with Will Pastor. In Fall 2020, Nick began his studies at Penn State Abington as a first-year student with 47 credits. Presently, he is working on a machine learning research project concerning Text Summarization.

### Corey Shive



Corey is a dual enrollment student in Computer Science. He joined the NASA Mars Curiosity Rover team starting in the 2019 Summer workshop, continuing in the 2019 Fall semester and 2020 Spring semester. Working alongside fellow students Will Pastor and Nick Gahman, Corey contributed to researching FPGA devices to complement Noah and Sal's work on object detection and collision avoidance.

###