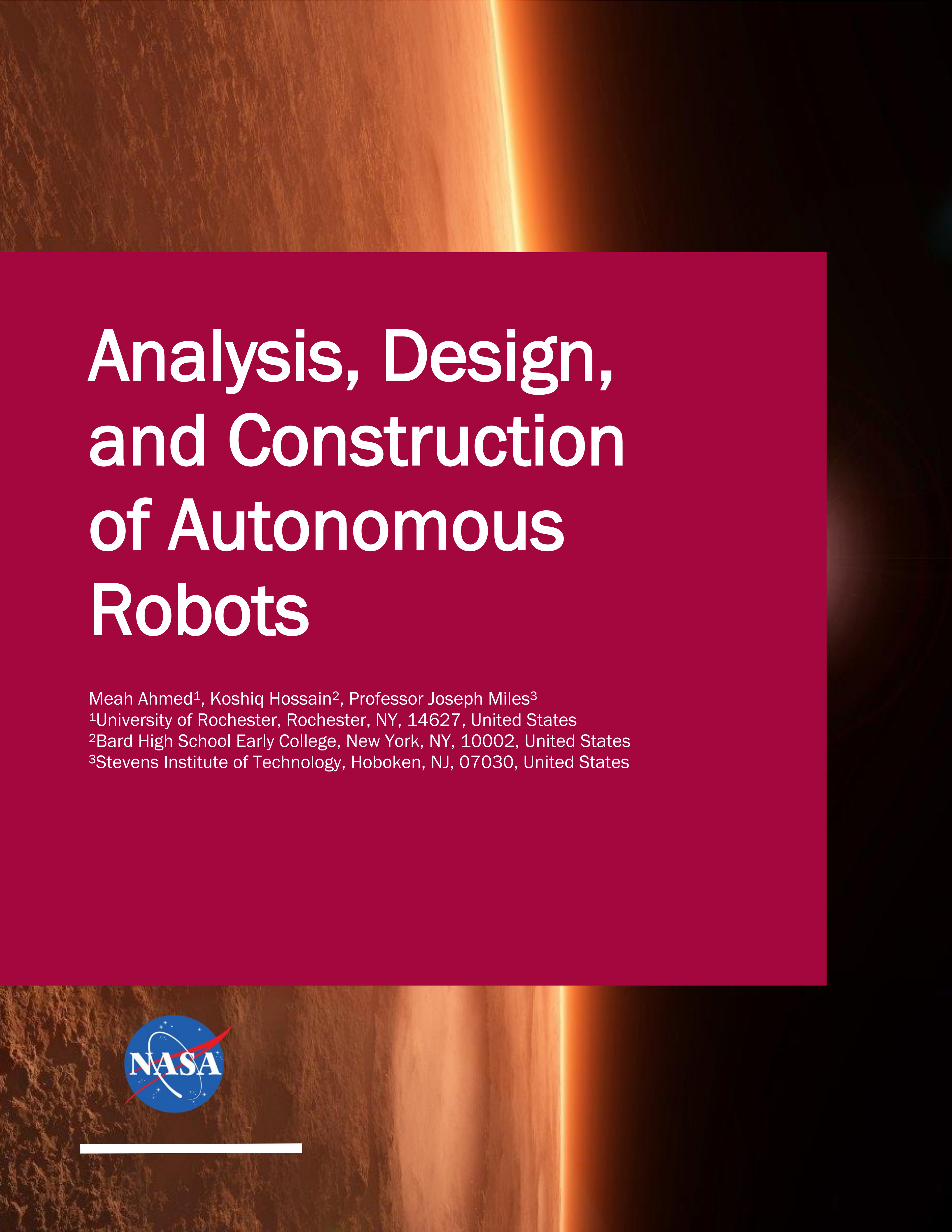


Analysis, Design, and Construction of Autonomous Robots

Meah Ahmed[1], Koshiq Hossain[2], Professor Joseph Miles[3]
[1]University of Rochester, Rochester, NY, 14627, United States
[2]Bard High School Early College, New York, NY, 10002, United States
[3]Stevens Institute of Technology, Hoboken, NJ, 07030, United States

Analysis, Design, and Construction of Autonomous Robots Capable of Exploring Martian Lava Tubes

Meah Ahmed[1], Koshiq Hossain[2], Professor Joseph Miles[3]
[1]University of Rochester, Rochester, NY, 14627, United States
[2]Bard High School Early College, New York, NY, 10002, United States
[3]Stevens Institute of Technology, Hoboken, NJ, 07030, United States

Abstract - Since the Apollo mission, scientists have begun questioning whether humans can live in planets within or beyond our solar system. In recent years, we have set their eyes on the colonization of Mars. Elon Musk, the founder of SpaceX, proposed the development of Mars transportation infrastructure to facilitate the eventual colonization of Mars. He aspires to have humans on Mars by the year 2024. However, the question still holds; can humans truly survive on Mars? Or anywhere beyond earth in general? Recent discoveries show cave like openings that are theorized to be remnants of lava tubes. These lava tubes are volcanic caverns on Mars and lunar surfaces that are formed because of fast-moving, basaltic lava, which reach high up to the surface, and cool down to produce tunnel like structures with large voids. The gravity on Mars is said to be about 38% of Earth's, which means Martian lava tubes are much larger compared to those of Earth's (because lava flow disperses throughout a much larger surface area with less gravitational constraints), and enormous enough to accommodate an entire city. On top of that, being underground helps it become a shield from cosmic radiation and micrometeorite flux. Martian Lava Tubes can potentially serve as the first human colony outside of Planet Earth.

Taking into consideration the cost of the project, the safety of human astronauts, and the fact that the tubes have lower visibility from orbit, sending humans straight away is not sagacious. Thus, scientists and engineers have been working on rovers that can collect data within the caves, such as temperature and humidity, to understand atmospheric construction, or the geographic infrastructure in general. One of the best current examples of that is NASA's Curiosity, the autonomous rover that collected surface and atmospheric data on Mars. Even though technological advancements modern space exploration seems very promising, the exploration of these tubes remains a demanding task, since geologists or engineers do not yet know the surrounding infrastructures meticulously. This project revolves around two 2 wheeled, Arduino based, autonomous (obstacle avoidance) roaming robots with the capabilities of collaboration. Considering that stronger controllers and motherboards can be used to develop even more sophisticated robots, the project uses open source hardware such as a WeMo's D1 R1 microcontroller board. An important theme of this project is also sustainability. Developing a singular robot as sophisticated as Curiosity requires a lot of time and money. Using Arduino based microcontrollers help demonstrate that open source microcontrollers that are cheap can potentially possess many functionalities for cave exploration. The robot is equipped with HC-SR04 ultrasonic sensors, which helps record spatial distance and avoid the many obstacles that gets in its way. The robots are also equipped with DHT11 temperature and humidity sensors which records the data in its environment. For calibration purposes, a 16x2 LCD was also used to physically demonstrate what the robots will be concurrently doing. The robots can successfully navigate within simple structures such as our cardboard constructed tunnel, which serves as a miniature example of the fascinating Martian lava tubes, while collecting and reporting necessary data.

Keywords – Autonomous, Light Seeking, Obstacle Avoiding Robots, Martian Lava Tubes

Table of Contents

Table of Figures

1 Introduction

The Geological Structure of Lava Tubes

Terrestrial structure has long been known for its fascinating formations, such as rivers, deltas, valleys, and so on. It's not until recent years that geologists and astronauts have laid their eyes on lava tubes that exist beyond Earth. According to "Lunar and Martian Lava Tube Exploration as Part of an Overall Scientific Survey", lava tubes are "natural caverns that form as the result of lava flow that over crusts to form subsurface flowing rivers of lava; as they drain an open conduit is left behind." Lava tubes, like the ones on Mars and lunar sub surfaces, are no strangers to Earth. Eruption, overflowing, and over crusting are the basic steps that lead to some of the largest lava tubes and volcanoes on other planets as well, leaving behind large voids as they drain.

In known cases and earthly processes, lava tubes are formed in accordance to the depiction above. As runny lava ponds flow through basaltic sub surfaces, the edges, linings, and the top cool down consecutively when exposed to cool air. This leaves a film like structure called "Regolith" for the tube to protect the running lava pond. The idea is very like a hot chocolate. Hot chocolate creates a layer of film at the top as it cools down, while the liquid remains underneath. As the lava is drained, a large void is left in the subsurface. It is believed that extraterrestrial sub surfaces, such as lunar land masses, and Mars, contains lava tubes like those of Earth. In fact, the consensus

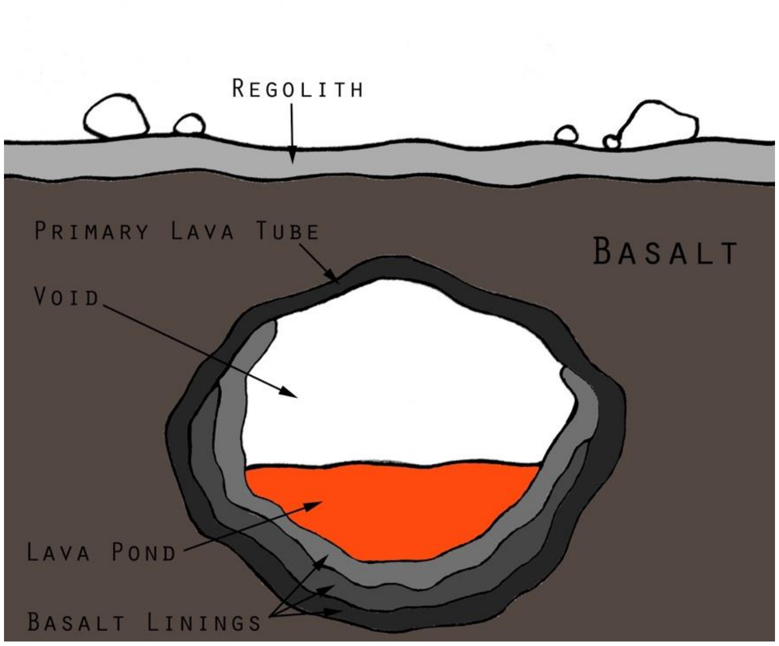


Figure 1

among Scientists is that those lava tubes are much larger. According to Andrew Daga, "The size of extraterrestrial riles and associated topographic ridges (which may represent sections of a tube that have not collapsed) suggest cross-sectional widths on the order of hundreds of meters, lengths of tens of kilometers, and roofs that are meters thick". For various reasons, such as difference of gravity and surface components (different types of rock layers), Martian and Lunar lava tubes are much bigger, and holds potential for large scale exploration and data collection.

The Importance of Martian Lava Tubes

Lava tubes hold great promise for exploration and advancement in extraterrestrial colonization. Per Daga, gaining entry to Martian lava tubes would allow geologists to study undisturbed native mineral compositions, bedrock structures, and the history of volcanism. In doing so, a lot of minerals that can't be excavated from top soils can be accessed from below. Natural events such as eruptions or magma flow can also be predicted. In terms of Martian missions, those lava tubes can also be a lair for laboratories or mission planning, because scientists predict that stable atmosphere and temperatures would serve to protect astronauts and scientists from solar proton event (SPE) radiation. Considering its vast size compared to those of Earth's, Martian lava tubes holds the potential to be the first extraterrestrial human colony, if properly explored and found to be survivable. For the same reason, scientists also believe that if life truly exists beyond earth, such lava tubes are a promising infrastructure to start the search. Because of predicted levels of constant temperature, water can exist and thrive. Like those of Earth, chemosynthetic organisms can thrive near volcanic grounds, ingesting minerals as a source of nutrients. Lava tubes can also protect these organisms from ultraviolet radiation and weather conditions beyond the subsurface of the lava tubes.

Currently, a lot of research is being performed to better understand how humans should approach extraterrestrial explorations, and lava tubes, although a broad avenue to consider, holds its limitations. Before gaining an entry point, scientists are looking to conduct orbital surveys or robotic missions to search for unclasped lava tube sites that are stable and safe. Aside from just safety, it is always statistically and scientifically possible that a lot of unknown factors are yet to be accounted for, so performing surveys are the first step, before developing human colonies within these large structures.

2 Related Works and Aspirations

One of the most noteworthy examples of previous NASA or space exploration missions that took place was the Mars Curiosity Rover. Curiosity pursued similar purposes such as today's extraterrestrial robotic projects; scientists wanted to study the bedrock structure and survey for survivable minerals or environments for evidence of microbial life. According to NASA, "It (Curiosity) has tools including 17 cameras, a laser to vaporize and study small pinpoint spots of rocks at a distance, and a drill to collect powdered rock samples". A three-dimensional illustration of Curiosity is shown below.

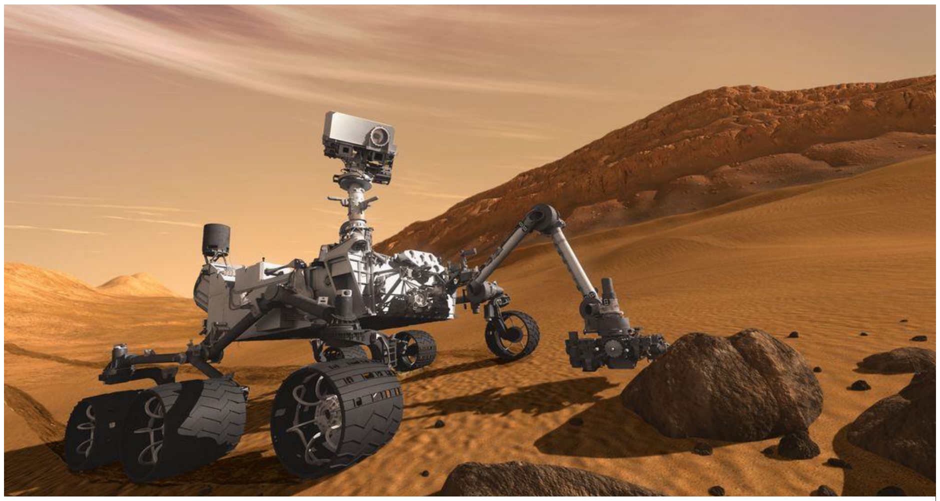


Figure 2

Another exemplary work, like the aims of ours, are collaborative lunar cave exploring robots. Carnegie Mellon's William (Red) Whittaker has been working on developing collaborative robots capable of exploring lunar caves, as part of the NASA Innovative Advanced Concepts (NIAC) program using earth-based analogs. Like other robotics engineers and scientists, Whittaker also believes that there are too many unknown factors to account for, such as temperature variations, structural integrity, and networking capabilities, before actually sending humans to inhabit and study extraterrestrial caves, and in our case, Martian lava tubes. NIAC is aiming for collaborative robots with the following behavioral pattern.

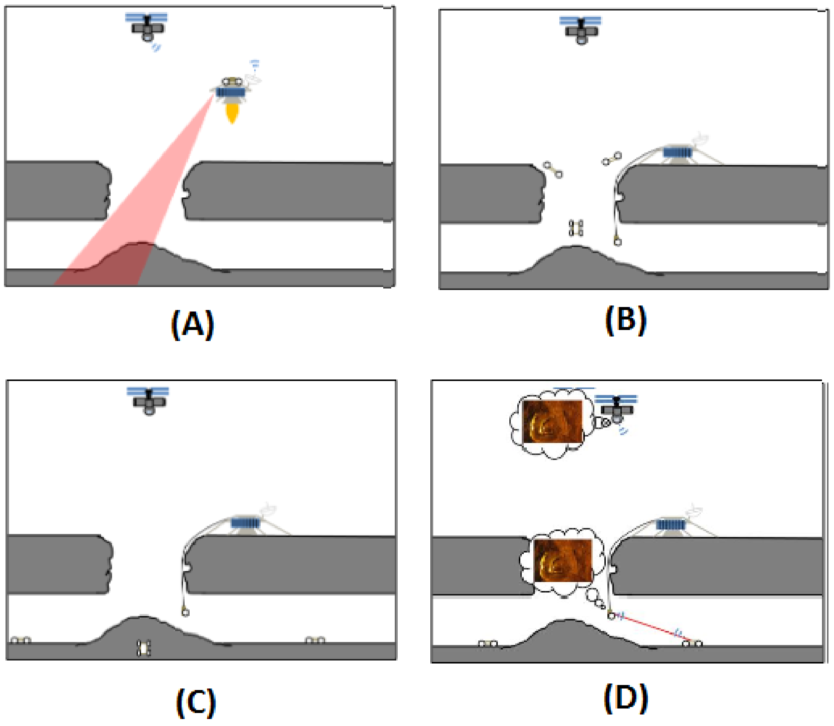


Figure 3

The two robots, "livewire" and "Cave Hopper", will prospectively be designed to autonomously move through bedrock obstacles and look for skylights or voids that they can descend to, retrieve data, and relay back to earth. Different problems are currently proposed with collaborative robots. For example, absolute location data mapping using GPS communication can be difficult, considering the distance from geocentric satellites, and the integrity of a cave's network. For the robots to pinpoint specific locations, Whittaker's research proposes that "orbiting satellites can be used to relay data and images back to earth". Our research adds on to those problems, regarding ways in which the robots **themselves** can communicate with each other.

3 Project Objectives

In honor of Curiosity's mission completion, and many influential research being performed to automate robots with sophisticated extraterrestrial exploration capabilities, this project aims to do the same. The following below are the explicit objectives of the project.

- Develop two robots capable of exploring Martian Lava tubes (Conceptually speaking)
- Code the robots using Arduino IDE for two styles of autonomy (obstacle avoidance)
- Research and comprehend the geology of lava tubes in general, to construct a cardboard tunnel maze with a similar design
- Collect data from the sensors and robotic activities as deemed necessary
- Develop a code for the robots to collaborate
- Utilize minimal resources to promote sustainability

When developing both robots, the goal was to develop them with capabilities of exploring Martian lava tubes. As they are only prototype robots, the project utilized material, sensors, and wheels that execute the desirable task, and would not actually be feasible robots to test on Mars right away. For that reason, the robots themselves are conceptual designs. For example, the metal material of the chassis may not be compatible with the weather conditions of Mars, as the high wind speeds and gusts can lead to erosion of the external structure. Another important example, which will be further expanded upon, are the different sensors being used. Some of the sensors have limited capabilities, such as slower refresh rates, inaccuracy, and a fragile build. Final robotic designs that will need to collect important spatial, structural, or atmospheric data must be much more fast, accurate, and of stronger materials to withstand the journey.

To develop the two robots, the project utilized open source hardware and microcontrollers. Thus, the robots were coded with the Arduino Integrated Development Environment (IDE) software. The IDE software uses C++ as its coding language, thus containing basic C++ capabilities as well, such as particular libraries, header files, and variables returning binary numbers. It was important for the researchers of this project to be well adapted to the software's language, and utilize **key coding skills** to bring about sophistication within the robots. Aside from using Arduino IDE, programs of higher caliber can be used, such as raspberry pi or python.

Understanding the significance and geological development of lava tubes was substantial. That manifested specific necessities for the robots' design and abilities. It's first important to understand the parameters and constraints of the lava tube, and what kind of data is critical, and what is not. If structural parameters and constraints are first understood, such as bedrock instability, large boulders, skylights etc., then the robots can be designed to withstand such variables. Depending on the purpose of the project, the kind of data retrieved is also important. For the purposes of this project (recognizing if Mars has capabilities to sustain life, or study the geology of Martian lava tubes in general), surface (spatial) and atmospheric data (temperature and humidity) is collected. Aside from that, understanding lava tubes was important to develop a sample testing platform. At the end of the project, a cardboard based lava tube like structure was created, with anticipated obstacles for the robots to avoid.

As mentioned previously, the robotic designs are conceptual, and prototypical. Although it accomplishes tasks that are on par with the purpose of exploring Martian lava tubes, utility of minimal resources (for the sake of sustainability) comes in conflict with developing the robots' final design using JUST minimal resources. For that reason, the kinds of sensors used in this project are not engineered to be perfect. Most of the sensors are of basic utility, and not as sophisticated as planned for final designs. This means that collecting data of the sensors and robotic activities are an important part of the project, to see if the robots can calibrate themselves adequately.

Compared to previous years of research, an important development for this year was 'collaboration'. Developing two robots is more than just an exercise for this project, and it was due to the driven necessity of collaboration. From the perspective of purely planning, collaboration meant many things for our project. From a larger viewpoint, collaboration meant either sharing data, or sharing physical gestures (motor movement). Important data, such as temperature and humidity could be shared, or the robots could possibly follow each other and even direct each other. Aside from the obvious sophistication that it brings to this robotics project, it is also a question of sustainability, as two collaborative robots of diverse mechanisms for each are much better than one expensive robot.

As we blast ourselves further into space, whether it be the James Webb telescope probing system, or Curiosity's next brother/sister, the question of economic prosperity comes. More resources and money becomes necessary when pondering upon projects of caliber that high. For prototypical and conceptual designs, utilizing minimal resources has become a productive facet of robotic projects. As also stated by Investopedia, "With a price tag of $2.5 billion, Curiosity has been on the receiving end of much skepticism and criticism as the media and public question: why did Curiosity cost so much and is this money well spent?". Considering that robotics projects must be provisioned with rocketry and landing gears, one can only imagine the amount of resources and money necessary to fulfill future projects. Many argue that Curiosity was not worth the money, as the project ended up crashing before collecting further data, while many argue that it was an important step towards extraterrestrial exploration. Although that's a subject matter that is to be approached with much more wisdom, it's indeed important to recognize that making such projects cost efficient is necessary. Thus, for this project, the students chose to pursue open source hardware and feasible material.

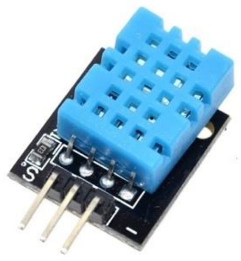
4 Implementations

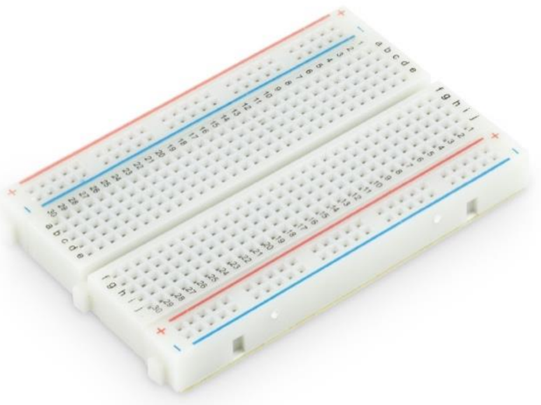
Provided Materials

Fun Fact: For the purposes of this project, which will be explained further on, two robots were built. One of the robots were named Romulus, and the other robot was named Remus. The names were inspired from ancient Roman mythology, where Romulus and Remus, sons of the Demi-god Hero Mars, discovered Rome.

Romulus Build

Parts	Description
	The *WeMos D1 R1 Board* is a microcontroller that consists of an ESP8266 chip and has a larger flash memory compared to an Arduino Uno. It consists of 11 digital I/O pins and 1 analogue (input) pin. The board can be connected using a Micro-B type USB cable. The ESP8266 helps with WiFi Communication by acting as a server or a client.

	The *Feetech 2WD Mobile Robot Platform* is a chassis body for the WeMos D1 R1 robot. It is the main infrastructure where all of the robot components are able to connect to one another and be able to move smoothly.
	The *HC-SR04 Ultrasonic Sensors* that comes with a 4-pin interface named as VCC, Trigger, Echo, and Ground. It is very useful for accurate distance measurement of the target object and mainly works as sound waves. The three Ultrasonic Sensors allow the robot to detect an object within the range of 270 degrees. It also helps to show the amount of space of the front, left, and right of the sensor.
	The *DHT11 Sensor* measures the Temperature and Humidity of a given environment. This device was placed on the breadboard on Romulus. Romulus serves as the robot that scouts and collects useful and necessary data. This data is then transferred to Remus's serial monitor through WiFi communication. This works because there is an ESP8266 built in to the WeMos D1 R1 microcontroller.

	Romulus only uses one *Feetech 2 SMC-2CH Motor Controller*. The Motor Controller is required to make the servos on the wheel's function correctly. This motor controller integrates adequately with SG-90 or FM-90 type servos, which are usually used for rotational tasks.
	Romulus only required two *TowerPro FM-90 Micro Servos*. The two servos were connected to the motor controller. The servos were connected to wheels which served as the only source of movement for the robot. This helped the robot move front, back, left, right and other directions as necessary.
	The *Breadboard* was an important instrument that was used to connect all the necessary components to the WeMos D1 R1 Board. The WeMos D1 R1 does not have enough space to connect three ultrasonic sensors, a motor, two servos, and a DHT11 sensor. The breadboard serves as a middle man (a network of circuitry) which would then connect to the WeMos D1 R1 microcontroller.

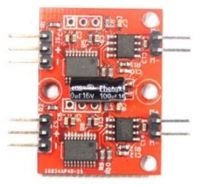
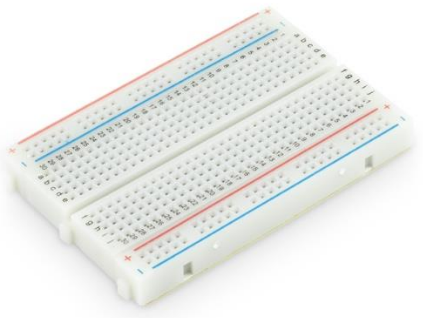
	The *Wheels* are connected to the servos which helps the robot move. It helps the robot move straight, left, right, and even backwards. The robot utilizes the wheels to take on an ideal course beyond obstacles.
	The *Battery,* also the *Portable Charger,* powers the robot and Arduino board. Without the battery, the robot wouldn't be able to work. The micro-USB cable connects the portable charger to the Arduino.
	The *Jumper Wires* connected the different pieces of hardware together. For instance, the Arduino connected to the Breadboard. The Breadboard was connected to the three ultrasonic sensors, DHT11 sensor, and the servo. The jumper wires help create a complete circuit, and are made of copper wires, which conducts the necessary volts of electricity between the board and the motors or sensors.
	The *Screws and Gears* were necessary to connect to the wheel. It also helped keep different parts intact. For instance, we used a screw to connect the Arduino to the Remote Car Chassis.

	An *ArduCam Mini 2MP Plus* module was provided. The purpose of this advanced sensor is to capture photos or livestream videos of its surroundings, at a rate of 150 frames per second, which allows the robot to look through its surroundings, and record spatial data.

Remus Build

Parts	Description
	The *WeMos D1 R1 Board* is a microcontroller that consists of an ESP8266 chip and has a larger flash memory compared to an Arduino Uno. It consists of 11 digital I/O pins and 1 analogue (input) pin. The board can be connected using a Micro-B type USB cable. The ESP8266 helps with Wi-Fi Communication.
	The *Feetech 2WD Mobile Robot Platform* is a chassis for the WeMos D1 R1 robot. It is a place where all the robot components are able to connect to one another and be able to move smoothly.

	The *HC-SR04 Ultrasonic Sensors* that comes with 4 pin interface named as VCC, Trigger, Echo, and Ground. It is very useful for accurate distance measurement of the target object and mainly works on the sound waves. The HC-SR04 Sensor is connected to a Neck Brace which is also connected to a servo. This helps rotate the sensor left and right, which gives it a 180-degree range.
	The *HC-SR04 UltraSonic Mount Bracket* gives the UltraSonic Sensor 180 degrees view of what is ahead of it. It is connected to the servo which helps it move perfectly.
	The *16x2 Liquid Crystal Display (LCD)* is a screen that displays different texts which includes, "Hello from Remus", it also displays the data from the DHT11 in Romulus. This works because of the ESP8266 in the WeMos D1 R1 board.

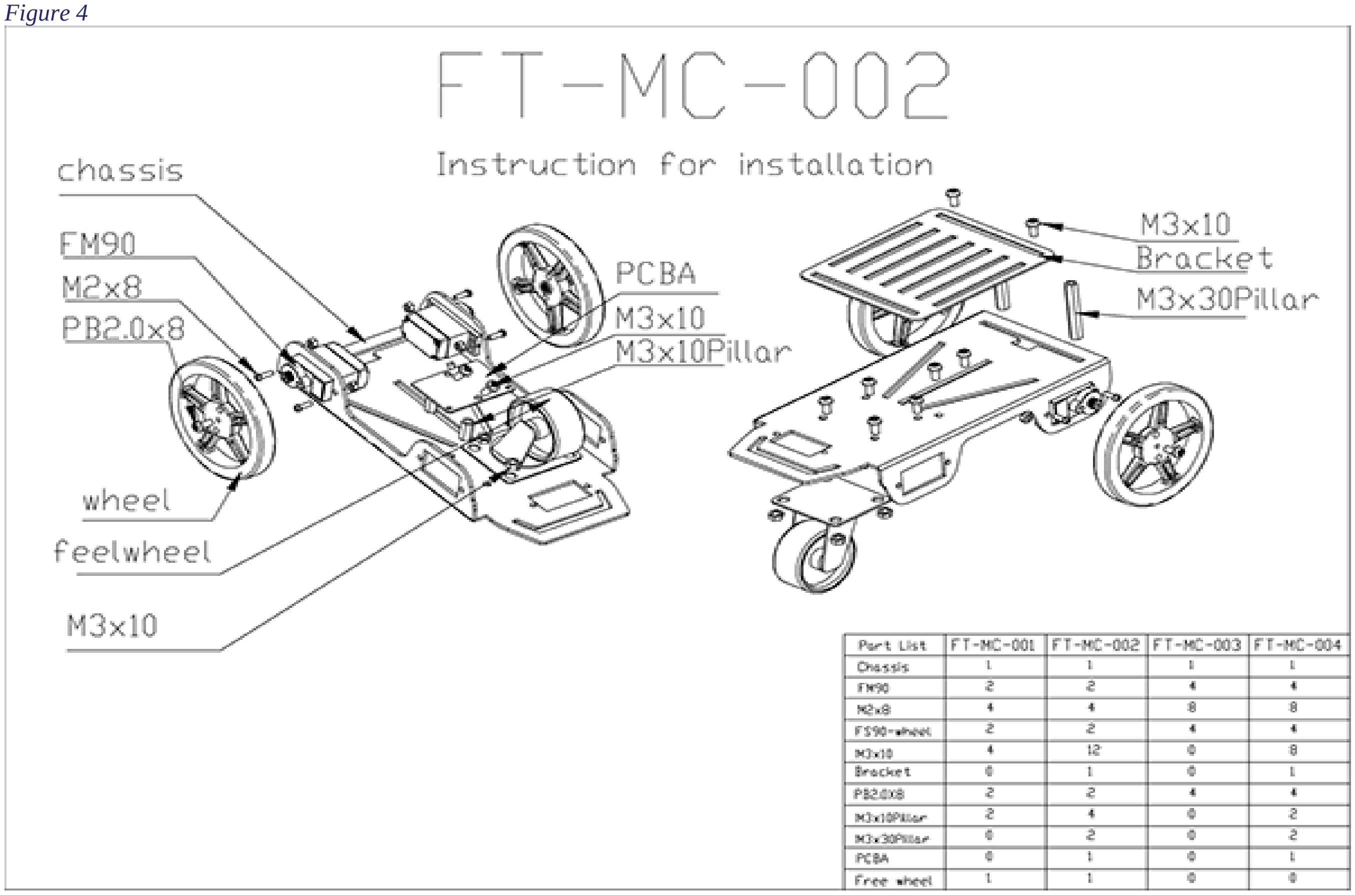
	Remus uses two *Feetech 2 SMC-CH Motor Controller*. The Motor Controller is required to make the servos function correctly. If this device wasn't in Remus then Remus's servos wouldn't be able to move, which means the wheels wouldn't move which breaks the whole purpose of the robot. Remus has two of these because the other connects to the servo which connects to the neck mount and HC-SR04 sensor.
	Romulus only required two *TowerPro FM-90 Micro Servos*. The two servos were connected to the motor controller. The servos were connected to wheels which served as the only source of movement for the robot. This helped the robot move front, back, left, right and other direction. Remus also has two servos that connects to the wheels. The additional servo is attached to the front of the robot, and the ultrasonic sensor bracket is mounted on it, for it to rotate and give it a 180-degree freedom.
	The *Breadboard* was an important instrument that was used to connect all the necessary components to the WeMos D1 R1 Board. The WeMos D1 R1 does not have enough space to connect three ultrasonic sensors, a motor, three servos, and a 16x2 LCD module. The breadboard serves as a middle man which would then connect to the WeMos D1 R1 microcontroller.

	The *Wheels* are connected to the servos which helps the robot move. It helps the robot move straight, left, right, and even backwards. They calibrate and move when it detects a larger distance using the ultrasonic sensors.
	The *Battery*, also the *Portable Charger*, powers the robot and Arduino board. Without the battery, the robot wouldn't be able to work. The micro-USB cable connects the portable charger to the Arduino.
	The *Jumper Wires* connected the different pieces of hardware together. For instance, the Arduino connected to the Breadboard. The Breadboard was connected to the ultrasonic sensor, LCD, and the servos.

	The *Screws and Gears* were necessary in order to connect to the wheel. It also helped keep different parts intact. For instance, we used a screw to connect the Arduino to the Remote Car Chassis.

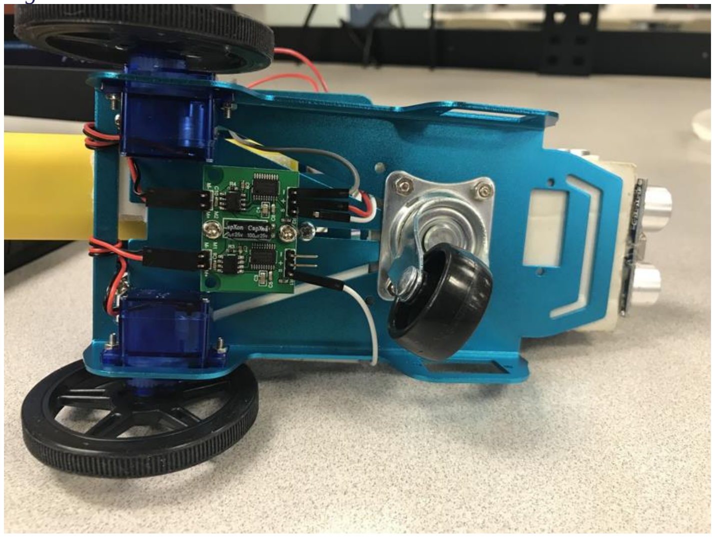
Additional Notes

Figure 4



Part List	FT-MC-001	FT-MC-002	FT-MC-003	FT-MC-004
Chassis	1	1	1	1
FM90	2	2	4	4
M2x8	4	4	8	8
FS90-wheel	2	2	4	4
M3x10	4	12	0	8
Bracket	0	1	0	1
PB2.0X8	2	2	4	4
M3x10Pillar	2	4	0	2
M3x30Pillar	0	2	0	2
PCBA	0	1	0	1
Free wheel	1	1	0	0

Figure 5



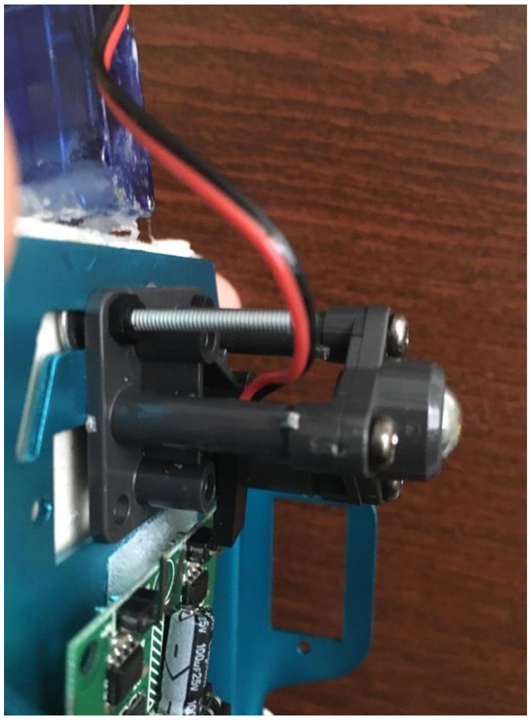


Figure 6

Initially, rotating wheels, or 'feel wheel' as shown in Figure 4 And Figure 5 were placed using screws towards the front bottom of the robots, for support. However, because of the wheel's poor rotating abilities, and high friction, this resulted in the robots moving in a slanted direction when coded to move strictly forward or backward in 90 degrees. As a group, it was decided that it is best to use the following type of roller (Figure 6) for both robots. It's a genuine roller, that was provided for the project by the mentor of the group.

Other important tools that were used are illustrated below:

- Glue Gun
- Duct Tape
- Super Glue
- Velcro
- Screws
- Screwdriver set
- Large Pieces of Cardboard
- Hula-Hoop
- Advanced (fast processing) Computers

Restrictions and Limitations

The robotic designs had to comply to a certain set of rules:

A. The robot’s chassis structure must not be modified. After building the general kit, the WeMos D1 R1 microcontroller board, Feetech 2 SMC-CH motor controller, and all the other necessary tools of assortment must be placed and organized on it.

B. No damage or modifications can be done on the WeMos D1 R1 microcontroller boards. Modifications include soldering, gluing, or addition of extra sensors.

C. No modifications can be done on the servo motors, or the wheels.

D. The size of the wheels cannot be altered, except for the front roller.

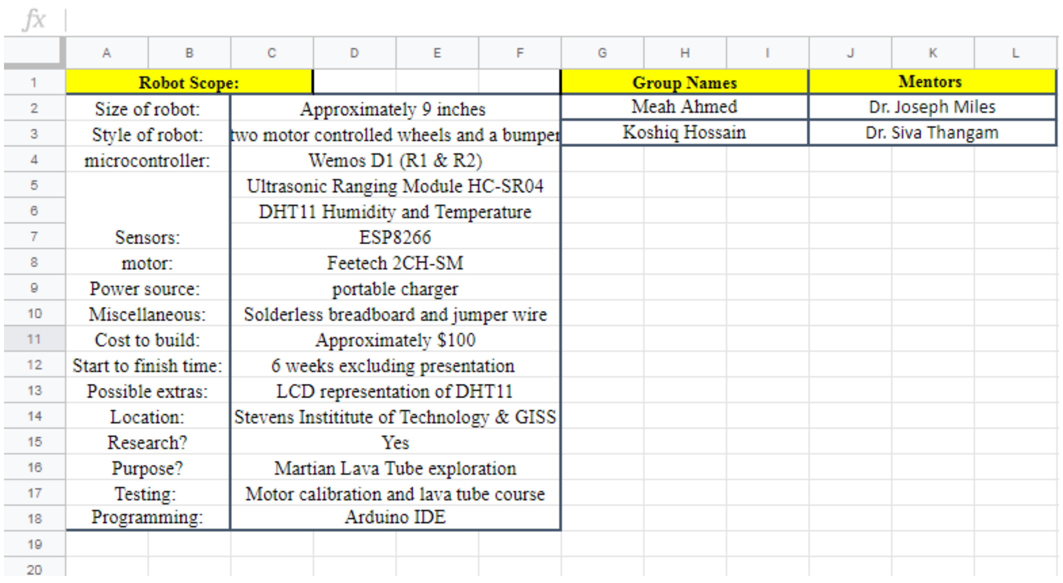
E. Two 9.6-volt Nickel Metal Hydride battery packs will be used to power both robots, and no different kinds of batteries must be used, as they may result in overpowering the board, or cause short circuits.

F. No additional sensors must be used, aside from the kind that integrate with the WeMos D1 R1 microcontroller board, as it must follow the theme of sustainability.

Project Planning

To initiate a clear-cut project plan, the methodology of declaring a scope, a work breakdown structure, and a GANTT chart was utilized.

The Following Below Demonstrates the Declaration of the Scope of the Project:

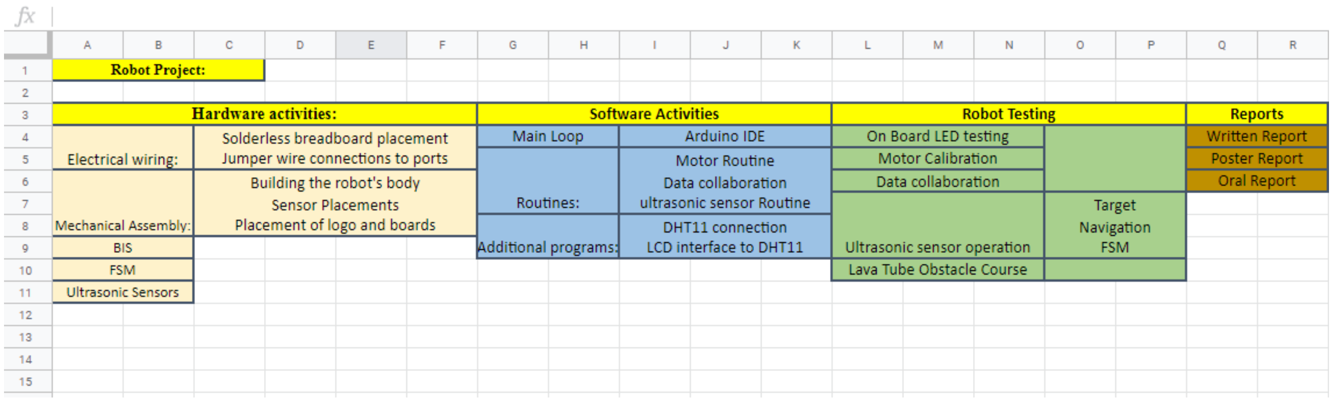


Robot Scope:		Group Names	Mentors
Size of robot:	Approximately 9 inches	Meah Ahmed	Dr. Joseph Miles
Style of robot:	two motor controlled wheels and a bumper	Koshiq Hossain	Dr. Siva Thangam
microcontroller:	Wemos D1 (R1 & R2)		
	Ultrasonic Ranging Module HC-SR04		
	DHT11 Humidity and Temperature		
Sensors:	ESP8266		
motor:	Feetech 2CH-SM		
Power source:	portable charger		
Miscellaneous:	Solderless breadboard and jumper wire		
Cost to build:	Approximately $100		
Start to finish time:	6 weeks excluding presentation		
Possible extras:	LCD representation of DHT11		
Location:	Stevens Instititute of Technology & GISS		
Research?	Yes		
Purpose?	Martian Lava Tube exploration		
Testing:	Motor calibration and lava tube course		
Programming:	Arduino IDE		

Figure 7

The purpose of a scope for any project in general is to give it a specific overview. Moreover, it also manifests general parameters that the project must follow. One can say confidently that the scope is a simile of a ‘goal’. In the case of our project, important parameters, or ‘goals’, were asserted, such as the kind of microcontroller used, the approximate price necessary for both robots, and many more.

The Following Below Demonstrates the Work Break-Down Structure of the Project:



	A–B	C–F	G–H	I–K	L–N	O–P	Q–R
1	**Robot Project:**						
2							
3	**Hardware activities:**		**Software Activities**		**Robot Testing**		**Reports**
4	Electrical wiring:	Solderless breadboard placement	Main Loop	Arduino IDE	On Board LED testing		Written Report
5		Jumper wire connections to ports	Routines:	Motor Routine	Motor Calibration		Poster Report
6	Mechanical Assembly:	Building the robot's body		Data collaboration	Data collaboration		Oral Report
7		Sensor Placements		ultrasonic sensor Routine	Ultrasonic sensor operation	Target	
8		Placement of logo and boards	Additional programs:	DHT11 connection		Navigation	
9	BIS			LCD interface to DHT11		FSM	
10	FSM				Lava Tube Obstacle Course		
11	Ultrasonic Sensors						
12							
13							
14							
15							

Figure 8

Work break-down structures are a general overview of all the tasks necessary to accomplish the entire project. One can confidently compare a work break-down structure to a ‘brain-storm’. It declares and collects a complete list of duties that must be executed in a consecutive manner from left to right. In the case of our project, the tasks have been divided into 4 major sectors that were accomplished consecutively, and the sectors are hardware activities, software activities, robot testing, and reports, respectively.

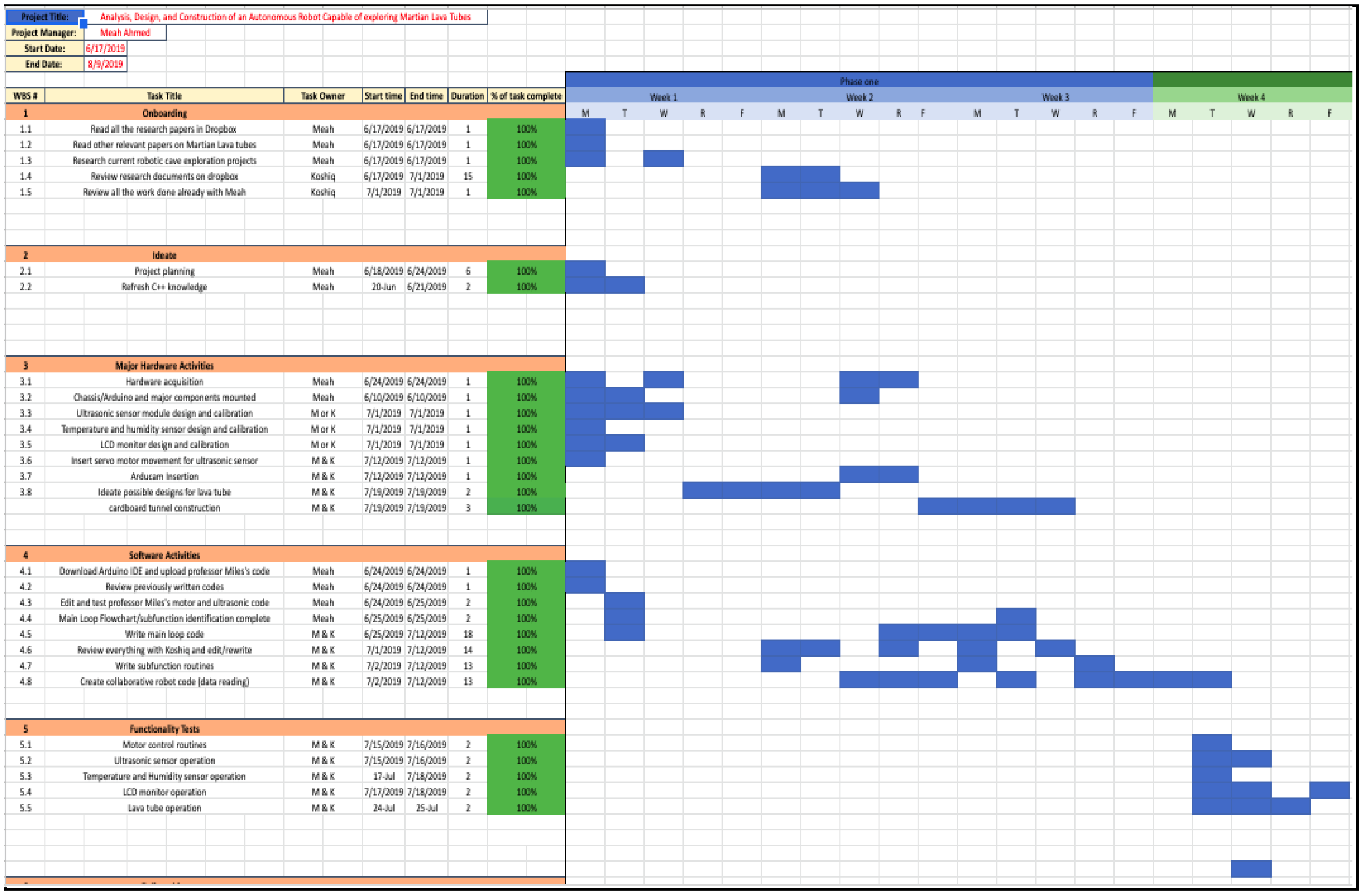
Hardware activities involve creating an electrical circuitry for the sensors and motors by utilizing breadboards, or the mechanical design of the robot (building the body and placing the components where it is necessary). Based off experience, this part of the project does not require much trial and error. Through different programs such as computer assisted design (CAD) and Fritzing, the general overview of the mechanical design and electrical circuitry was illustrated.

Software activities are simply the coding aspects of the project. By using Arduino IDE (The C++ coding software), different components of the whole code were approached through trial and error, and uploaded into the robots. Aside from Arduino IDE, the project also utilized Visuino. Arduino IDE utilizes sketches through scripture, and typed codes that return binary numbers. Visuino is designed to be more user friendly for engineers or students who are not familiar with C++ scripture. It allows one to create schematic diagrams of the electric circuitry, which translates the diagram to the necessary code, depending on the sensor being utilized, and the parameters given to the sensors.

Robot testing is for data collection purposes. Testing the robot means calibrating the motors, calibrating the sensors, and collecting collaboration data. It's the final product of a fully oiled project, as it is concrete proof that the robots would be functioning properly.

The final component of the work breakdown structure are the final reports. This includes important deliverables that will be used to communicate the project's work to the NASA community, and to the science community in general.

The Following Below Shows the GANTT Chart of Our Project:



Project Title:	Analysis, Design, and Construction of an Autonomous Robot Capable of exploring Martian Lava Tubes
Project Manager:	Meah Ahmed
Start Date:	6/17/2019
End Date:	8/9/2019

WBS #	Task Title	Task Owner	Start time	End time	Duration	% of task complete
1	Onboarding					
1.1	Read all the research papers in Dropbox	Meah	6/17/2019	6/17/2019	1	100%
1.2	Read other relevant papers on Martian Lava tubes	Meah	6/17/2019	6/17/2019	1	100%
1.3	Research current robotic cave exploration projects	Meah	6/17/2019	6/17/2019	1	100%
1.4	Review research documents on dropbox	Koshiq	6/17/2019	7/1/2019	15	100%
1.5	Review all the work done already with Meah	Koshiq	7/1/2019	7/1/2019	1	100%
2	Ideate					
2.1	Project planning	Meah	6/18/2019	6/24/2019	6	100%
2.2	Refresh C++ knowledge	Meah	20-Jun	6/21/2019	2	100%
3	Major Hardware Activities					
3.1	Hardware acquisition	Meah	6/24/2019	6/24/2019	1	100%
3.2	Chassis/Arduino and major components mounted	Meah	6/10/2019	6/10/2019	1	100%
3.3	Ultrasonic sensor module design and calibration	M or K	7/1/2019	7/1/2019	1	100%
3.4	Temperature and humidity sensor design and calibration	M or K	7/1/2019	7/1/2019	1	100%
3.5	LCD monitor design and calibration	M or K	7/1/2019	7/1/2019	1	100%
3.6	Insert servo motor movement for ultrasonic sensor	M & K	7/12/2019	7/12/2019	1	100%
3.7	Arducam insertion	M & K	7/12/2019	7/12/2019	1	100%
3.8	Ideate possible designs for lava tube	M & K	7/19/2019	7/19/2019	2	100%
	cardboard tunnel construction	M & K	7/19/2019	7/19/2019	3	100%
4	Software Activities					
4.1	Download Arduino IDE and upload professor Miles's code	Meah	6/24/2019	6/24/2019	1	100%
4.2	Review previously written codes	Meah	6/24/2019	6/24/2019	1	100%
4.3	Edit and test professor Miles's motor and ultrasonic code	Meah	6/24/2019	6/25/2019	2	100%
4.4	Main Loop Flowchart/subfunction identification complete	Meah	6/25/2019	6/25/2019	2	100%
4.5	Write main loop code	M & K	6/25/2019	7/12/2019	18	100%
4.6	Review everything with Koshiq and edit/rewrite	M & K	7/1/2019	7/12/2019	14	100%
4.7	Write subfunction routines	M & K	7/2/2019	7/12/2019	13	100%
4.8	Create collaborative robot code (data reading)	M & K	7/2/2019	7/12/2019	13	100%
5	Functionality Tests					
5.1	Motor control routines	M & K	7/15/2019	7/16/2019	2	100%
5.2	Ultrasonic sensor operation	M & K	7/15/2019	7/16/2019	2	100%
5.3	Temperature and Humidity sensor operation	M & K	17-Jul	7/18/2019	2	100%
5.4	LCD monitor operation	M & K	7/17/2019	7/18/2019	2	100%
5.5	Lava tube operation	M & K	24-Jul	25-Jul	2	100%

Figure 9

GANTT chart is a methodology like that of a work break-down structure. It also lists out major task milestones of tasks that need to be accomplished, but the major difference is that it is given the parameter of a timeline, schedule, and level of accomplishment. The GANTT chart, like a work break-down structure is also followed in a consecutive order. Aside from giving specific parameters of time that is necessary to finish the project, the entire session of the project is mapped out in such a manner that it allows the group members to keep track of when which task is completed, so that the pace is conserved thoroughly.

Note: An important thing to note about the 3 methodologies used to plan the project is that they were considered as **living documents.** This means that it was always updated whenever necessary, and looked upon for reminders or motivation. Living documents are a key facet to a productive project because they directly keep tabs of the guidelines.

Team Dynamics

The team for this project consisted of two students, and one mentor. The two students were Meah Ahmed, and Koshiq Hossain. Meah Ahmed is a University of Rochester undergraduate, majoring in biomedical engineering. Koshiq Hossain is a high school graduate from Bard High School Early college. Both students have diverse sets of previous experience with robotics, electrical circuitry, hardware, mechanical designs, and computer coding.

The mentor for this project was Professor Joseph Miles of Stevens Institute of Technology, who supervised the project, gave overall guidelines, and granted the necessary help or tools.

Meah started 2 weeks earlier than Koshiq on the project. Although both students worked together, Mead led the project's plan and initial designs. Both students divided the tasks equally, executed ideas by bouncing off each other's, and completed the deliverables by dividing duties.

4.1.1 Division of Labor:

Meah Ahmed	**Koshiq Hossain**
Mechanical, software, and electrical construction of the robot	Assist with electrical construction
Troubleshooting	Troubleshooting
Tunnel Prop Construction	Tunnel Prop Construction
Report Write Up	Report Edits
Presentation Edits	Presentation Design and Write

	up
Poster Edits	Poster Design and Write up

Figure 10

5 Methods

Mechanical Design

The Final Mechanical Design of Remus is Illustrated Below:

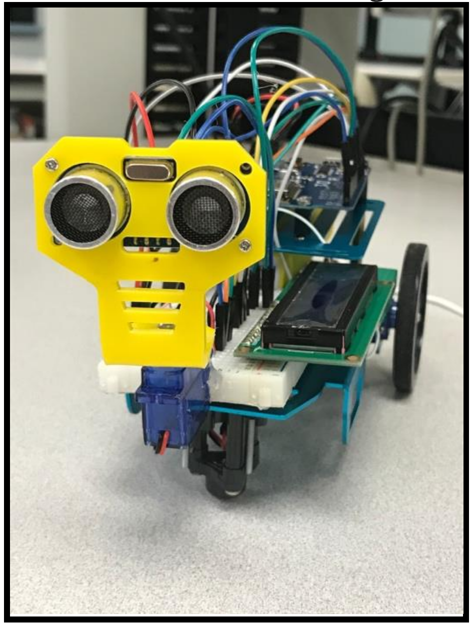


Figure 11

After the blue aluminum chassis structure was built, using screws, bolts, and pillars, as followed by the directions of Figure 4, the breadboard was placed directly on the bottom shelf of the platform. The bottom of the breadboard is made of an adhesive that allows it to stick directly onto the aluminum chassis. The servos for the wheels were also placed using the schematic shown by Figure 4. The wheels were then screwed in to the servo. It was carefully placed in, making sure that the servos did not break as the screws plunged in. Screws were also used to place the front roller onto the bottom of the robot, for support as it moves around. To place the servo on the front of the robot, super glue was used. Then screws were used to place the ultrasonic sensor mounting bracket onto the servo, being careful not to plunge too deep into the servo and break it. One HC-SR04 ultrasonic sensor was placed directly onto the mounting bracket, with 4 screws that fit gently onto the 4 screw holes. The combination of the servo and the mounting bracket with the ultrasonic sensor allows the robot to have a 180-degree field of vision. The WeMos D1 R1 was screwed on the top shelf of the chassis platform, using 3 pillars and screws. Two towards the back of the top shelf, as shown in Figure 4, and one near the front center. It was important to improvise and utilize sensible engineering practice for screwing and placements, and highly depended on the physical weight of the circuit. The battery pack was placed using Velcro. The motor controllers were placed on the bottom of the robot, as shown below in Figure 11, one next to the two servos for the wheels, and the other next to the front servo for the ultrasonic sensor mounting bracket. The motor controllers were placed using screws and Velcro.

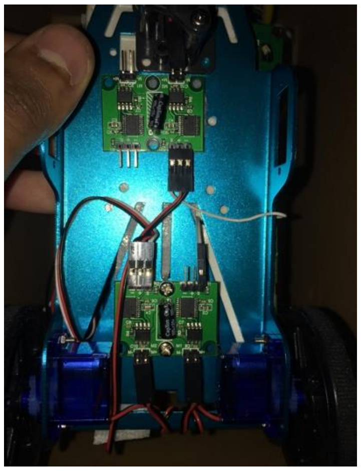


Figure 12

Finally, the LCD was placed onto the main breadboard. The LCD has exactly 16 pins that fits directly into the circuit of the breadboard.

The Final Mechanical Design of Romulus is Shown Below:

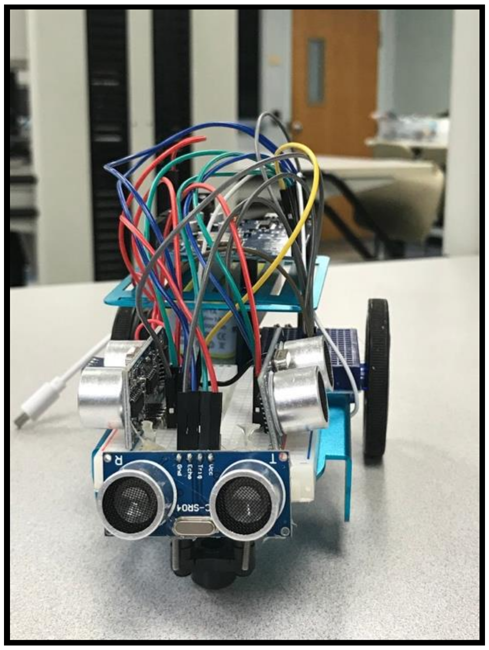


Figure 13

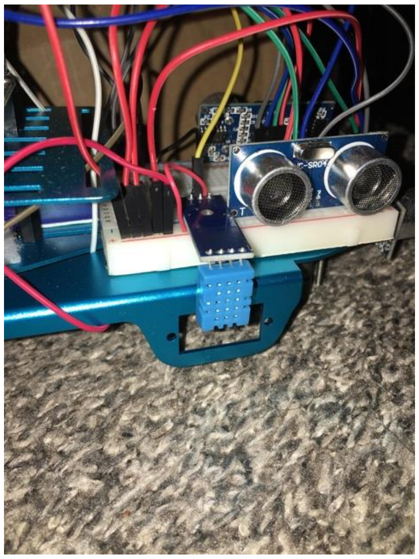


Figure 14

The build design for Romulus's chassis platform is the same as Remus's. The breadboard placement, wheel servo motor placement, wheel placement, and WeMos D1 R1 microcontroller board placement follows the schematics of Figure 4. The key difference, however, is that there is no ultrasonic sensor mounting bracket, an additional servo, and an additional motor controller. There is, however, three ultrasonic sensors for the Romulus design. Two ultrasonic sensors were placed on the right of the robot, and left of the robot. The HC-SR04 ultrasonic sensor has 4 pins, all of which fit directly onto the breadboard. The front ultrasonic sensor that was placed, however, was placed using a Velcro. The three ultrasonic sensors give it an absolute 180-degree field of vision. In addition, the DHT11 temperature and humidity sensor, which consists of 3 pins, fit directly onto the breadboard of Romulus.

Overall, when calibrated, Remus weighs approximately 20 grams, and Romulus weighs approximately 28 grams. Despite the difference of weight for both robots, they still move with similar accelerations and velocity, when coded to do so.

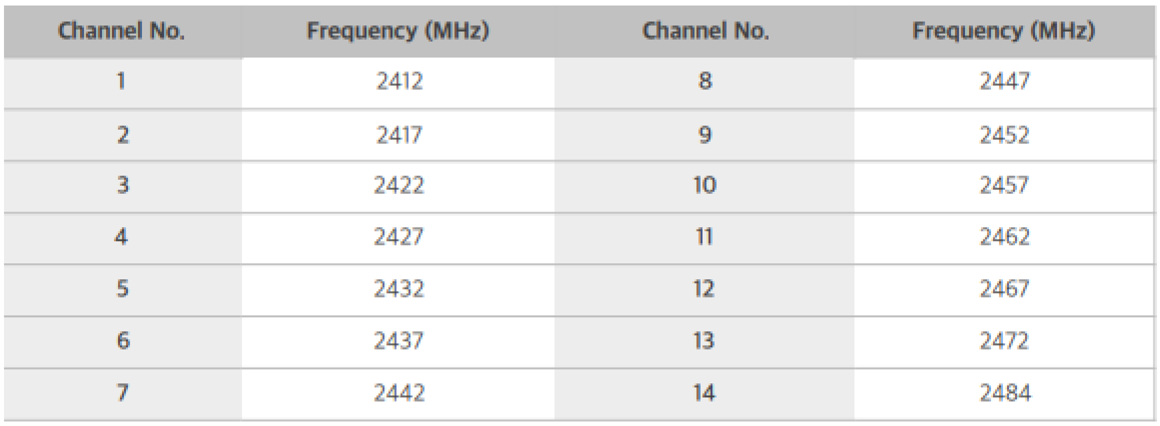
5.1.1 The Controllers

The 3 main types of controllers, which includes the WeMos D1 R1 microcontroller, ESP-8266 microprocessor, and the Feetech 2 SMC-CH motor controller.

The WeMos D1 R1 board is the brain of the robot, and integrates directly to the Arduino IDE platform. It connects to the microcontroller for wheel movement, and the different sensors. One of the constraints of using the WeMos D1 R1 board, which will be discussed later, is the fact that it is substantially retired. This means that it can't keep up with a lot of the modern hardware, such as sensors or motors. It also consists of only 1 analog pin, but many digital pins (exactly 11). The benefit of using the WeMos D1 R1 microcontroller is the default soldered ESP-8266 microprocessor, which initiates ideas for collaboration.

The motor controller, placed next to the servos, integrates directly to the servo wheel motors, or any application of servo rotation. The motor controller should not be metaphorically aligned to the "brain" of the robot, rather the muscles. The muscles connect to the brain (WeMos board), and functions as told to do so. On one side of the microcontroller, there exists 4 total positive and negative charged pins for VCC and ground powers, respectively, to power the microcontroller from the board. And on the same side where those pins exist, the motor controller also consists of two pins for digital inputs, marked R1 and R2, to receive binary code from the microcontroller and direct the servos. On the other side, there is a total of 4 positive and negative charged pins that integrates the motor controller to the servo.

The ESP-8266 microprocessor is soldered onto the WeMos D1 R1 microcontroller, which sped up the process of the project. In most cases, the ESP-8266 microprocessor can be bought separately as a controller module, and integrated into the microcontroller. The ESP-8266 microprocessor utilizes WiFi principals. It has the capability of sending and receiving Wi-Fi signals. According to Expressif, the ESP-8266 specifies 14 channels for low power Wi-Fi communication in the ISM (unlicensed) band, as shown in Figure 14. These channels are spaced 5MHz apart, except ch.14, which is spaced 12MHz away from ch.13. In 802.11 b/g/n modes, the typical bandwidth requirement per channel is 20MHz and a guard band of 2MHz is added to that. Thus, there is potential overlap in operating frequency range when two transmitters operate in the same airspace.



Channel No.	Frequency (MHz)	Channel No.	Frequency (MHz)
1	2412	8	2447
2	2417	9	2452
3	2422	10	2457
4	2427	11	2462
5	2432	12	2467
6	2437	13	2472
7	2442	14	2484

Figure 15

5.1.2 *The Sensors and Motors*

- The HC-SR04 ultrasonic sensor consists of 4 pins. The 4 pins are VCC, ground, echo, and trigger. The VCC and ground pins of the ultrasonic sensor is used to generate electricity into the sensor, and power it for proper function. The trigger send pin sends out ultrasonic sound signals of 20 kilohertz, and the echolocation pin receives the signal back when it bounces back from an obstacle, and prints onto the serial monitor the duration of time in microseconds to receive the signal back after sending it. The larger the duration, the further away the obstacle from the sensor, and the smaller the duration, the closer the obstacle to the sensor. There is approximately a 20-microsecond delay in between the window of time a signal is received, read, refreshed, and sent back again, which results in inefficiency. The duration can easily be converted to distances, such as centimeters, or inches, using the following conversions:

Sound travels at approximately 340 meters per second. This corresponds to about 29.412μs (microseconds) per centimeter. To measure the distance the sound has travelled we use the formula: Distance = (Time x Speed of Sound) / 2. The "2" is in the formula because the sound must travel back and forth. First the sound travels away from the sensor, and then it bounces off a surface and returns. The easy way to read the distance as centimeters is to use the formula: Centimeters = ((Microseconds / 2) / 29). For example, if it takes 100μs (microseconds) for the ultrasonic sound to bounce back, then the distance is ((100 / 2) / 29) centimeters or about 1.7 centimeters.

- The 16x2 LCD consists of 16 pins as shown in Figure 14. Ground and VCC pins are used to generate electricity, and power the LCD. Contrast pin creates a contrast between the lightings to depict characters onto the screen. The contrast is maintained using a 20-ohm potentiometer. RS, RW and EN pin are for calibration purposes. D0 through D7 pins are digital input pins that connect directly to the WeMos microcontroller board, resulting in digital integration, and transfer of binary code from Arduino IDE. The backlight (+) and backlight (-) are also pins with the purpose of generating electricity, and powering the lighting capabilities of the LCD. The purpose

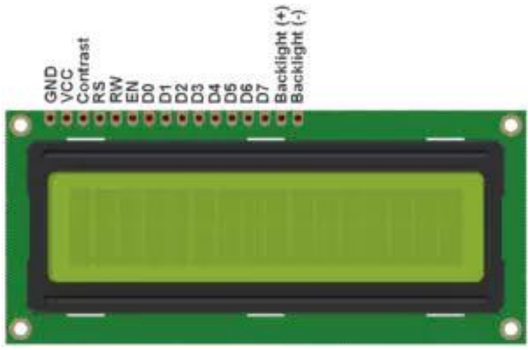


Figure 16

- The FM-90 servo is a miniature motor, and is placed for applications of rotation, whether it be for the wheels, or the ultrasonic sensor mounting bracket rotation. It consists of simply VCC (+) powered wire, and Ground (-) powered wire, which connects directly to the motor controller. When coded, the servo can move at different speeds either forward, or backward.
- The DHT-11 Temperature and Humidity sensor uses basic diode chemistry applications to detect the surrounding heat levels for temperature in Celsius, and saturation point comparisons for relative humidity. Relative humidity implies the percentage of moisture in the surrounding atmosphere, in comparison to the local saturation point of Earth (as a standard). 100% relative humidity implies raining. Whereas 0% humidity means absolute dry air. The sensor consists of 3 pins. VCC, ground, and data. VCC (+) and ground (-) are to generate electricity and power the sensor, whereas the data pin is a digital input pin that connects directly to the WeMos D1 R1 microcontroller.

Electrical Design

To execute the electrical design for both robots, a software platform known as Fritzing was used. Fritzing is an open source hardware sketching software that allows one to create schematic diagrams of the electrical circuitry.

The Electrical Circuitry for Remus is Shown Below:

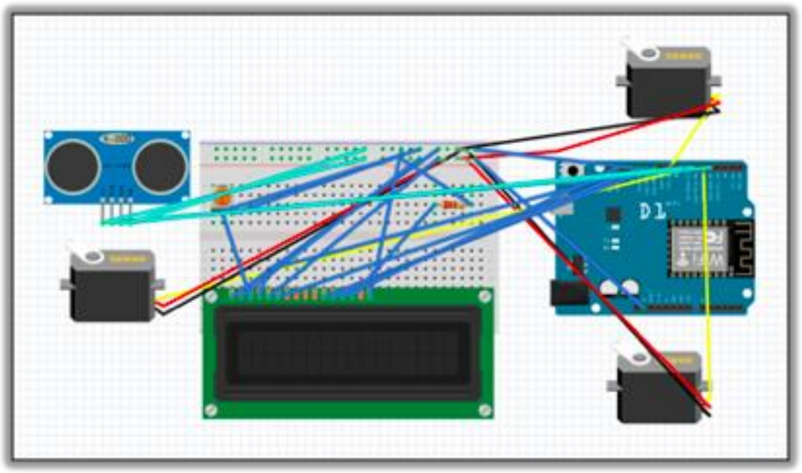


Figure 17

The Electrical Circuitry for Romulus is Shown Below:

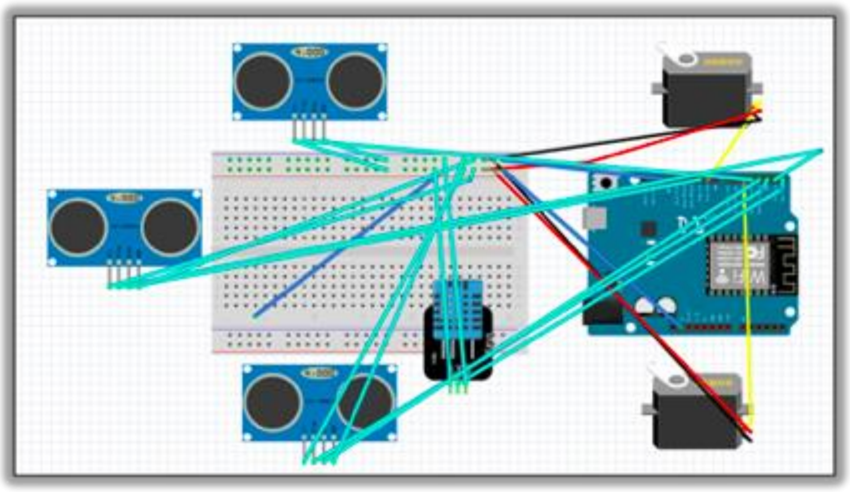


Figure 18

Note: The color in the Fritzing diagram was done before the actual circuitry, to set up a design a plan. Therefore, the wire colors are not the same as the ones in Fritzing.

5.1.3 Pin Assignments and Circuitry Schematics

The Pin Assignments (Color Coded with the Real Jumper Wires and Font) for Remus is Shown Below in Accordance to Figure 16:

Sensor/Motor	**WeMos Assignment**
HC-SR04 ultrasonic sensor **(VCC, Ground, Trigger, Echo)**	**5V, GND, D4, D7** The VCC (+) wires are red, ground (-) wires are black, trigger wires are green, and echo wires are blue
Ultrasonic sensor mounting bracket servo motor	D6 The servo is integrated to the microcontroller through the R2 pin, and into D6 of WeMos. The wire color for servos are white.
Right wheel servo motor	D5 The servo is integrated to the microcontroller through the R2 pin, and into D5 of WeMos. The wire color for servos are white.
Left wheel servo motor	D9 The servo is integrated to the microcontroller through the R1 pin, and into D9 of WeMos. The wire color for servos are white.
16x2 LCD	LCD pin: WeMos assignment VSS or GND: **GND** VDD or VCC: **5V**

	V0 or contrast	**20 ohm potentiometer's middle pin (potentiometer must also be powered through 5V and GND wires)**
	RS	**D12**
	RW	**GND**
	E or En	**D11**
	D0	Nothing
	D1	Nothing
	D2	Nothing
	D3	Nothing
	D4	**D8**
	D5	**D13 (White Wire)**
	D6	**D14**
	D7	**D0**
	A or Backlight (+)	**5V through a 220 ohm resistor**
	K or Backlight (-)	**GND**

The Pin Assignments (Color Coded with the Real Jumper Wires and Font) for Romulus is Shown Below in Accordance to Figure 17:

Sensor/Motor	**WeMos Assignment**
Front HC-SR04 ultrasonic sensor **(VCC, Ground, Trigger, Echo)**	**5V, GND, D1, D10** The VCC (+) wires are red, ground (-) wires are black, trigger wires are green, and echo wires are blue
Right HC-SR04 ultrasonic sensor **(VCC, Ground, Trigger, Echo)**	**5V, GND, D4, D7** The VCC (+) wires are red, ground (-) wires are black, trigger wires are green, and echo wires are blue
Left HC-SR04 ultrasonic sensor **(VCC, Ground, Trigger, Echo)**	**5V, GND, D2, D3** The VCC (+) wires are red, ground (-) wires are black, trigger wires are green, and echo wires are blue
Right wheel servo motor	D5

	The servo is integrated to the microcontroller through the R2 pin, and into D5 of WeMos. The wire color for servos are white.
Left wheel servo motor	D9 The servo is integrated to the microcontroller through the R1 pin, and into D9 of WeMos. The wire color for servos are white.
DHT-11 temperature and humidity sensor **(VCC, Ground, Data)**	**5V, GND, D0**

Note: It's important to perform trial and error on the digital pins that are integrated with each sensor to the WeMos microcontroller board. For our particular board, the pin assignments shown above makes the robot function.

Software Design

5.1.4 Software Flowchart

A flow-chart methodology was utilized to visualize the entire software design. There were multiple steps to the process, and it required much trial and error. But the overall process is shown below throughout the entire session of the project.

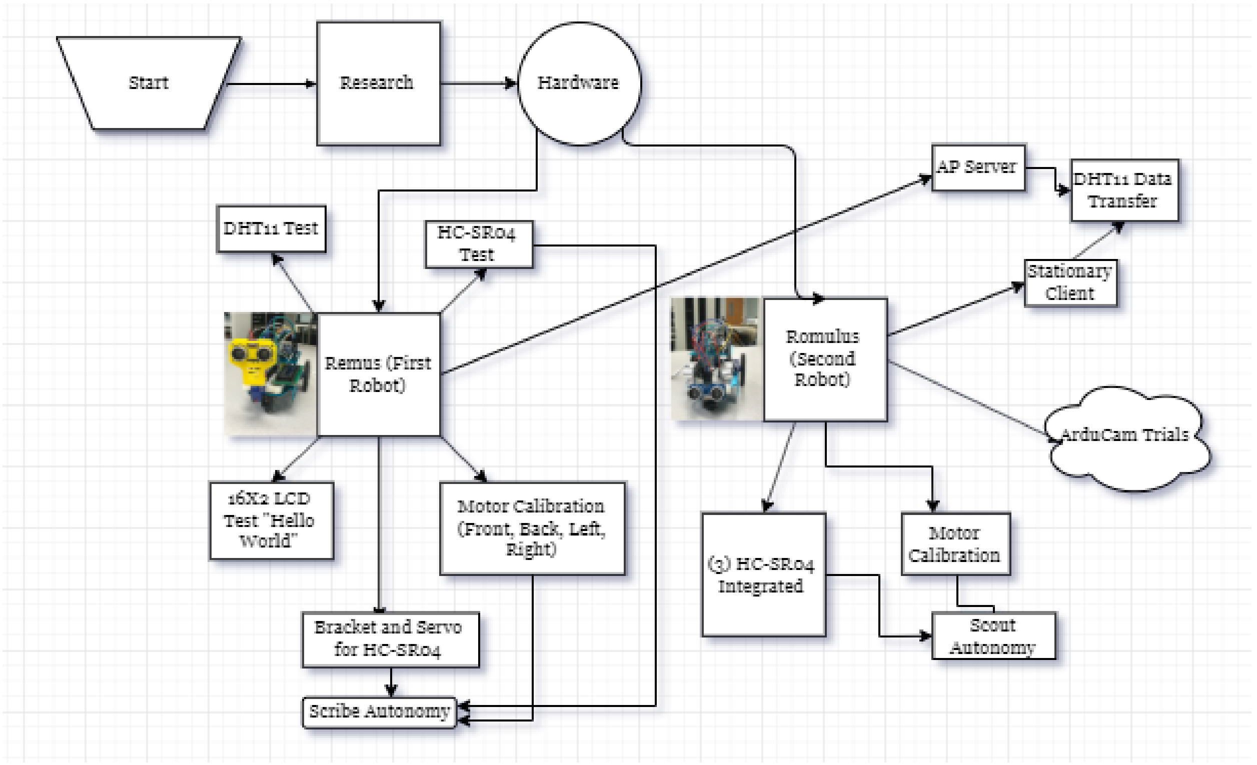


Figure 19

Prior to the actual software designing, research was done from previous years' works, and various other similar projects. One of the most important works studied deeply was the research done by Olufikunayo Famutimi and Cristian Juarez-Morales.

The following below illustrates their robot:

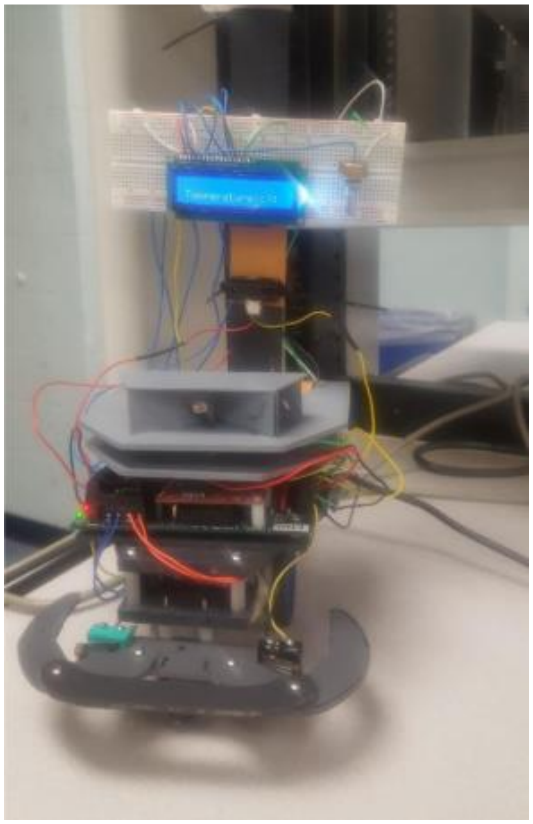


Figure 20

The design for this robot is vastly different than the design produced by this project. The external body is built from different styles of material, and the robot consists of a bumper. This particular robot design also used a light sensor to detect obstacles, or follow directions. The report stated that the light sensor is the most efficient method of autonomy. The light sensor has poor refresh rates, and interferes frequently with external lights. However, the codes for the robot's motor servo controls and LCD were important.

Following the flow-chart methodology of Figure 18, the hardware activities were then completed after the research, such as the mechanical design and electrical design.

During the first phase of the project, Remus was designed. After designing Remus, the DHT11 temperature and humidity sensor was calibrated with the DHT-11 test code, and tested for simple directions such as reading the temperature in Celsius and reading the relative humidity in percentage. The results were printed onto the serial monitor. The HC-SR04 ultrasonic sensor was calibrated with the HC-SR04 code to give readings of duration in microseconds, and converted into basic distances such as centimeters, inches, and feet using conversions. The results were also printed onto the serial monitor. The 16x2 LCD was coded, and calibrated to print out simple characters such as "hello world". The results from the DHT-11 sensor and the ultrasonic sensor were also coded to print onto the LCD, and translated into characters of data, to calibrate the LCD. The wheel servo motors were also calibrated to do basic movement of the robot, such as front, back, left, and right. As mentioned previously, each servo can be coded to move strictly only forward, or backward, at different speeds. The servo motors are coded in a manner that allows the motor to move right and left. To move the robot right (make a right turn), the left motor is coded to move forward, whereas the right motor is coded to stop moving completely, resulting in the right turning motion. To move the robot left (make a left turn), the right motor is coded to move forward, whereas the left motor is coded to stop moving completely, resulting in the left turning motion. So far, the robot was coded for simple movements, for purposes of calibration and basic assessment. The ultrasonic sensors were not yet integrated with the wheel servo motors, resulting in no capabilities of intelligent obstacle avoidance.

In the next phase, Romulus was built. Romulus consisted of 3 ultrasonic sensors, therefore the code for obstacle avoidance was implemented with the servo wheel motors. Using clever C++ methodology, a style of autonomy was developed for Romulus. The members of the project recognized this style of obstacle avoiding autonomous wheel movement as “scout autonomy”. After this phase of designing autonomous motor movement, Remus’s autonomous codes were developed. The ultrasonic sensor mounting bracket was placed near the front of the robot with only 1 ultrasonic sensor, and it was coded to be integrated with the motors to look around for obstacles, and move autonomously. This style of autonomy was recognized as “scribe autonomy” by the group members.

In order to implement collaboration between the two robots, the ESP-8266 microprocessors on both of the robots’ WeMos boards were utilized. Initially, the goal for this project was to create two robots that would be able to physically follow each other. The scout would clear the way, and direct the scribe to follow and collect data. However, the members of the project realized that stringing codes for servos with the Wi-Fi functions of the ESP-8266 requires more complexity beyond current knowledge. The most plausible level of collaboration would be sending data back and forth. Sending data means sending binary number variables from one serial monitor onto the other through Wi-Fi or Ad-Hoc technology. In this case, data means results, such as ultrasonic readings, or DHT-11 readings. The fact that such readings are variables returning binary numbers makes it an elementary task to transfer from one robot’s serial monitor onto the other robots’. Following the theme of sustainability and efficiency, the members of the project found it an essential idea to place the DHT-11 sensor on Romulus, and the LCD on Remus. Using ESP-8266 principals, the DHT-11 data is from Romulus onto Remus’s LCD. Why was the LCD placed on Remus? And why was the DHT-11 placed on Romulus, and not Remus? When the three ultrasonic sensors of Romulus are placed with the LCD for one WeMos microcontroller board, the group realized through trial and error that the LCD results in errors such as Hieroglyphic characters, as shown below in Figure 20:

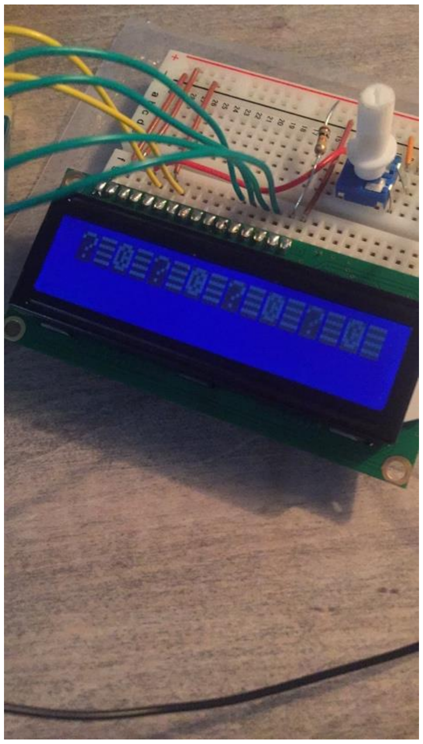


Figure 21

Rather than a software issue, it is predicted that this is a hardware issue. One ultrasonic sensor already uses a large amount of electricity to be powered. This makes the HC-SR04 ultrasonic sensor model a relatively weaker model. So, utilizing three ultrasonic sensors and an LCD results in fluctuation of the levels of electricity being transferred, and thus, it leads to the LCD printing out gibberish. It was more prudent to utilize the LCD onto the robot with less ultrasonic sensors, which was Remus, in comparison to Romulus.

It may seem like a witless exercise to send the DHT-11 data from one robot to the other, when one can simply put a DHT-11 sensor onto every single robot they desire, and read the data they want from the anticipated location of the robot. However, it is a groundbreaking exercise for the following two reasons:

- Following Whittaker's research, it was found that robots have difficulty with networking through earthly satellites. So, he studied that it is best to send out orbital satellites to Mars first, and then relay data back to earth. An important technology that must be implemented in the future is efficient collaboration between the two robots. Wi-Fi technology is a simple yet extremely handy tool for this problem. Wi-Fi network and collaboration does not require earthly satellites, as ESP-

8266 Wi-Fi signals are like free agents. Only two of these microprocessors are necessary to initiate collaboration of sending Wi-Fi signals, and receiving them. In addition, Wi-Fi signals are a viable method on the surface and atmosphere of Mars, despite harsh weather conditions on the subsurface.

- To promote sustainability, conservation of data is extremely important. Curiosity completed its mission before collecting all the necessary data, which was a large amount of time, money, and technology that went in vain. Robots are, at the end of the day, a physical medium. Physical structures are always battling with the forces of its physical environment, which means damages will always occur, and servicing will always be necessary. For distances that stretch as far as Mars, servicing is not a possible solution always, or at least not currently. If data could be relayed back and forth to many miniature robots capable of collaboration, creating a network of data, even though one or two robots break, a lot of the important data can be conserved and relayed back to earth.

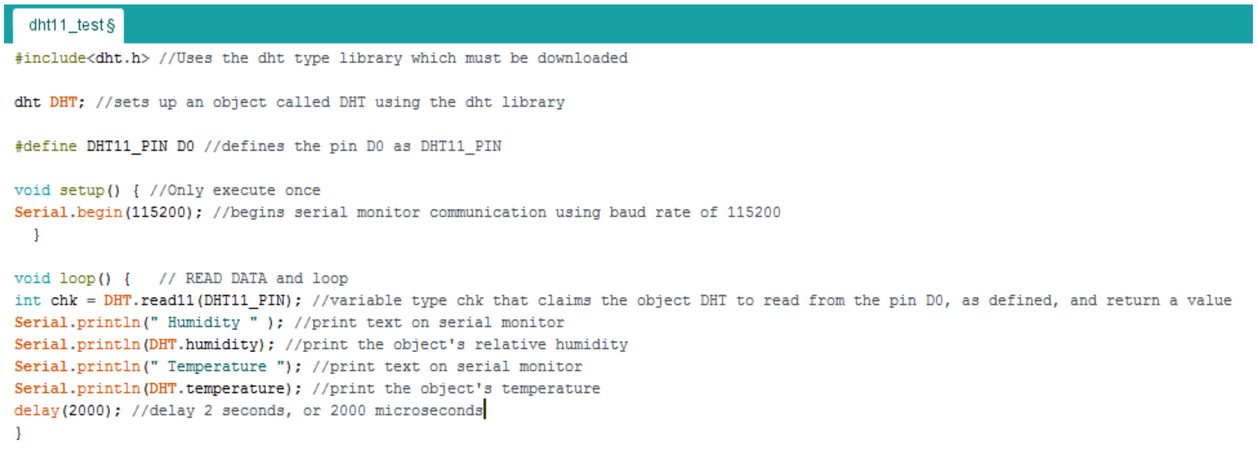
ESP-8266 modules and microprocessors are extremely effective, because, as mentioned previously, they do not require earthly satellites or a middleman to operate. It would be useful to create a third-party middleman through two modules, such as a computer that can read the default IP address of 192.168.4.1 or 192.168.4.2, but for the case of this project, it wasn't necessary. The ESP-8266 microprocessor's Wi-Fi microprocessors can calibrate into two modes: the soft access point server mode, and the stationary client mode. The soft access point server sends out the Wi-Fi signals, and the stationary client receives the data. It is the same concept as the way Wi-Fi works in common household scenarios. The Wi-Fi router is integrated with a global network, and opens a server that sends out Wi-Fi signals to client telephones, laptops, or TVs. In doing so, bytes of data and information are transferred.

Another important part of this project which resulted in unfortunate failure, after much trial and error, was the ArduCam module. The ArduCam module was also intended to be placed on Romulus, considering the initial plans of scout and scribe was executed, where Romulus clears the way and Remus follows. So, we found it important to place a camera on Romulus, as it would speculate the field first. However, despite proper hardware and electrical circuitry, and even proper coding, the ArduCam did not work properly, and resulted in a failure.

5.1.5 The Code

The following Figures demonstrate all the necessary codes, with comments on each code (marked with "//") demonstrating the actual function of each code.

5.1.5.1 THE DHT11 TEST

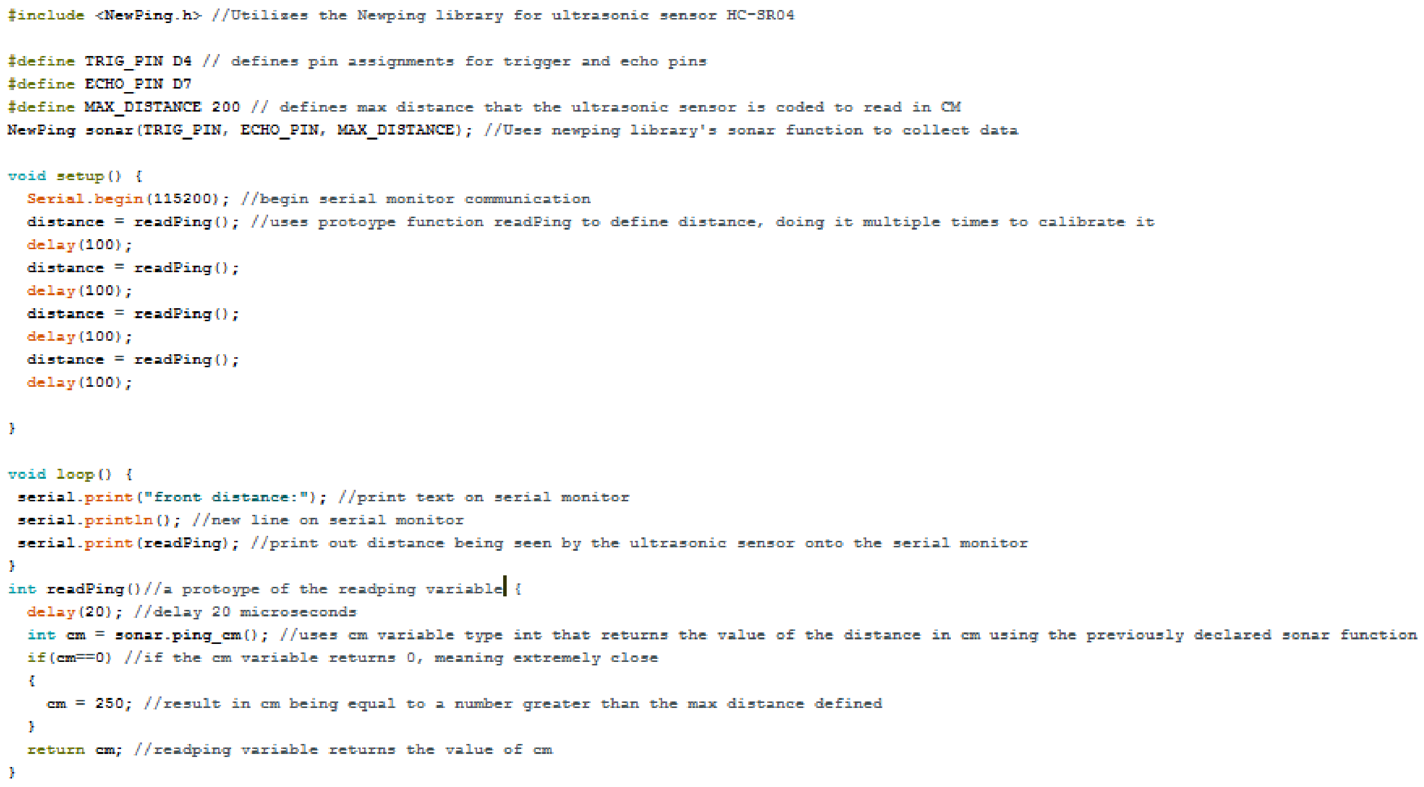


```
#include<dht.h> //Uses the dht type library which must be downloaded

dht DHT; //sets up an object called DHT using the dht library

#define DHT11_PIN D0 //defines the pin D0 as DHT11_PIN

void setup() { //Only execute once
Serial.begin(115200); //begins serial monitor communication using baud rate of 115200
  }

void loop() {   // READ DATA and loop
int chk = DHT.read11(DHT11_PIN); //variable type chk that claims the object DHT to read from the pin D0, as defined, and return a value
Serial.println(" Humidity " ); //print text on serial monitor
Serial.println(DHT.humidity); //print the object's relative humidity
Serial.println(" Temperature "); //print text on serial monitor
Serial.println(DHT.temperature); //print the object's temperature
delay(2000); //delay 2 seconds, or 2000 microseconds
}
```

Figure 22

This code results in the DHT-11 reading out temperature data in Celsius, and humidity in relative percentage, with a baud rate of 115200 on the serial monitor, looping every 2 seconds. This was coded to calibrate the sensor, before actually utilizing it for the robot's data reading tasks.

5.1.5.2 THE ULTRASONIC SENSOR TEST

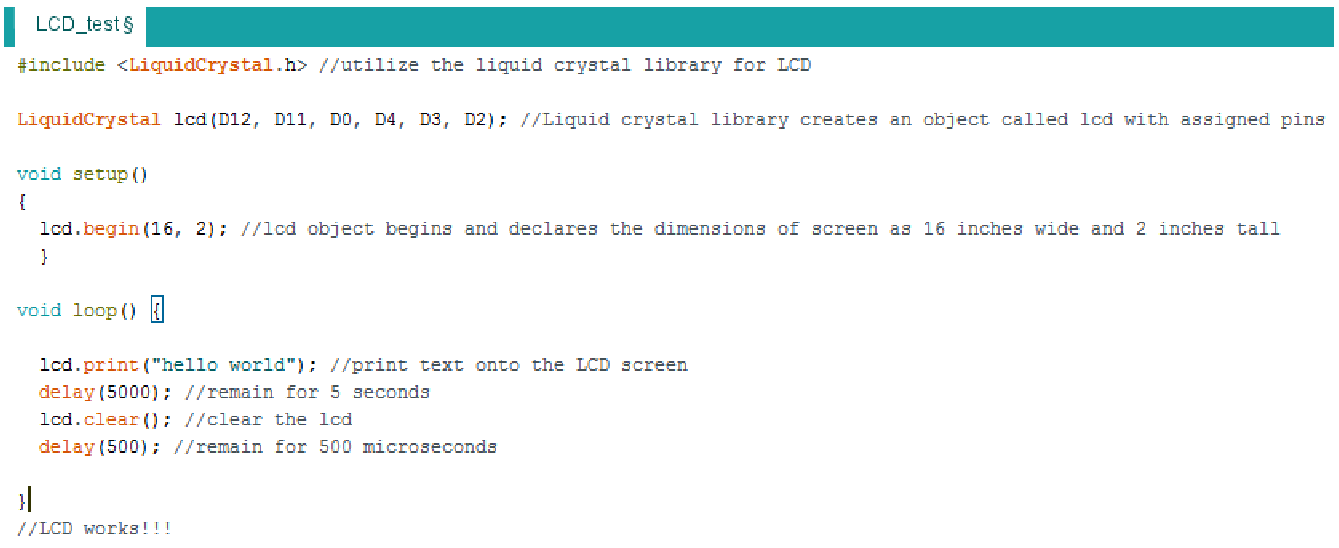


```
#include <NewPing.h> //Utilizes the Newping library for ultrasonic sensor HC-SR04

#define TRIG_PIN D4 // defines pin assignments for trigger and echo pins
#define ECHO_PIN D7
#define MAX_DISTANCE 200 // defines max distance that the ultrasonic sensor is coded to read in CM
NewPing sonar(TRIG_PIN, ECHO_PIN, MAX_DISTANCE); //Uses newping library's sonar function to collect data

void setup() {
  Serial.begin(115200); //begin serial monitor communication
  distance = readPing(); //uses protoype function readPing to define distance, doing it multiple times to calibrate it
  delay(100);
  distance = readPing();
  delay(100);
  distance = readPing();
  delay(100);
  distance = readPing();
  delay(100);

}

void loop() {
 serial.print("front distance:"); //print text on serial monitor
 serial.println(); //new line on serial monitor
 serial.print(readPing); //print out distance being seen by the ultrasonic sensor onto the serial monitor
}
int readPing()//a protoype of the readping variable {
  delay(20); //delay 20 microseconds
  int cm = sonar.ping_cm(); //uses cm variable type int that returns the value of the distance in cm using the previously declared sonar function
  if(cm==0) //if the cm variable returns 0, meaning extremely close
  {
    cm = 250; //result in cm being equal to a number greater than the max distance defined
  }
  return cm; //readping variable returns the value of cm
}
```

Figure 23

The following code results in the ultrasonic sensor reading a distance with its default factory delay, and it prints the result out onto the serial monitor. This was coded to calibrate the ultrasonic sensor, before actually utilizing it for autonomy.

5.1.5.3 THE LCD TEST

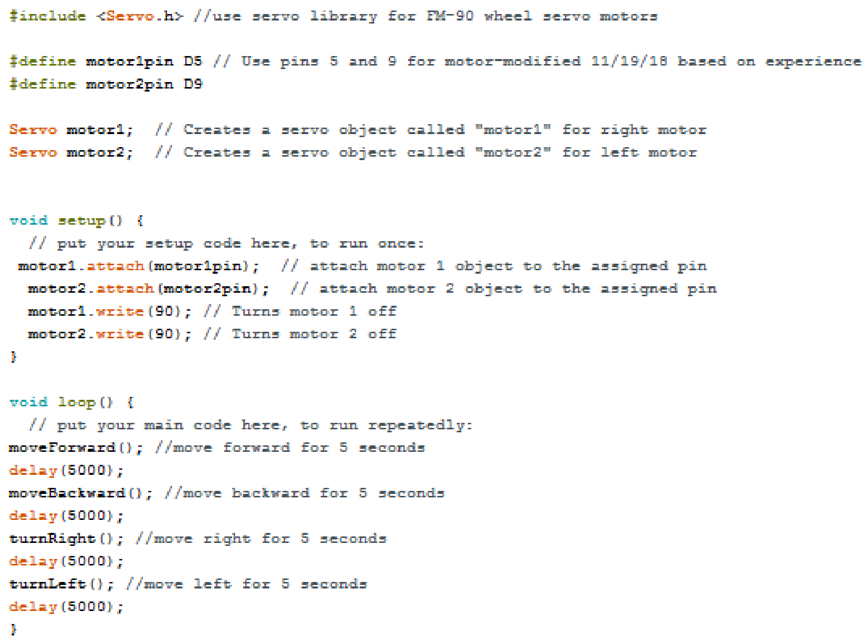

LCD_test§

```
#include <LiquidCrystal.h> //utilize the liquid crystal library for LCD

LiquidCrystal lcd(D12, D11, D0, D4, D3, D2); //Liquid crystal library creates an object called lcd with assigned pins

void setup()
{
  lcd.begin(16, 2); //lcd object begins and declares the dimensions of screen as 16 inches wide and 2 inches tall
  }

void loop() {

  lcd.print("hello world"); //print text onto the LCD screen
  delay(5000); //remain for 5 seconds
  lcd.clear(); //clear the lcd
  delay(500); //remain for 500 microseconds

}
//LCD works!!!
```

Figure 24

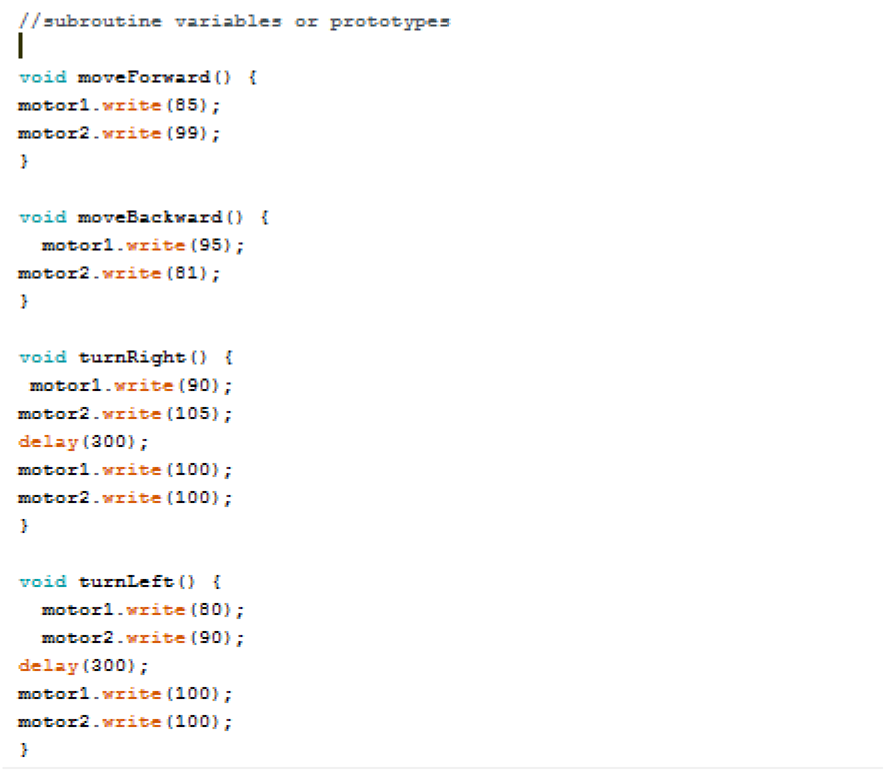
The following code results in the LCD displaying simple characters titled "Hello World!", remaining for 5 seconds, and then disappearing, and reappearing again. This was used to calibrate the LCD, before actually placing it onto the robot for action.

5.1.5.4 MOTOR CALIBRATION



```
#include <Servo.h> //use servo library for FM-90 wheel servo motors

#define motor1pin D5 // Use pins 5 and 9 for motor-modified 11/19/18 based on experience
#define motor2pin D9

Servo motor1;  // Creates a servo object called "motor1" for right motor
Servo motor2;  // Creates a servo object called "motor2" for left motor


void setup() {
  // put your setup code here, to run once:
 motor1.attach(motor1pin);  // attach motor 1 object to the assigned pin
  motor2.attach(motor2pin);  // attach motor 2 object to the assigned pin
  motor1.write(90); // Turns motor 1 off
  motor2.write(90); // Turns motor 2 off
}

void loop() {
  // put your main code here, to run repeatedly:
moveForward(); //move forward for 5 seconds
delay(5000);
moveBackward(); //move backward for 5 seconds
delay(5000);
turnRight(); //move right for 5 seconds
delay(5000);
turnLeft(); //move left for 5 seconds
delay(5000);
}
```

Figure 25

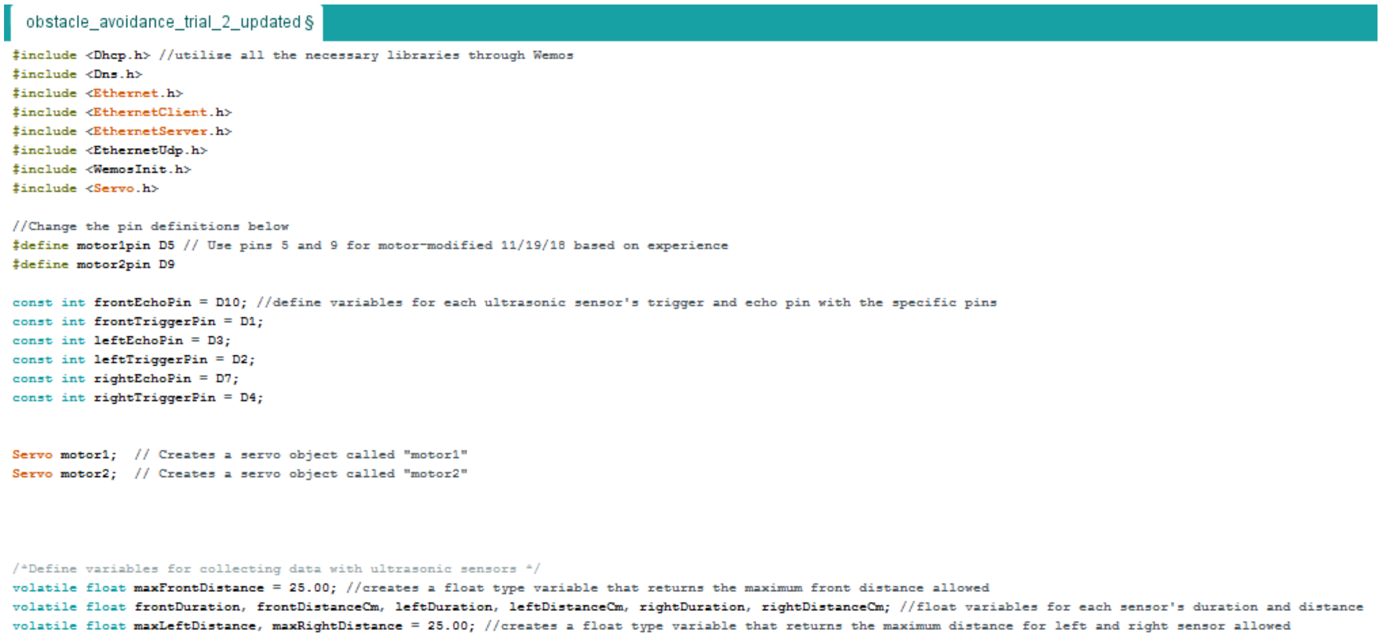


```
//subroutine variables or prototypes

void moveForward() {
motor1.write(85);
motor2.write(99);
}

void moveBackward() {
  motor1.write(95);
motor2.write(81);
}

void turnRight() {
 motor1.write(90);
motor2.write(105);
delay(300);
motor1.write(100);
motor2.write(100);
}

void turnLeft() {
  motor1.write(80);
  motor2.write(90);
delay(300);
motor1.write(100);
motor2.write(100);
}
```

Figure 26

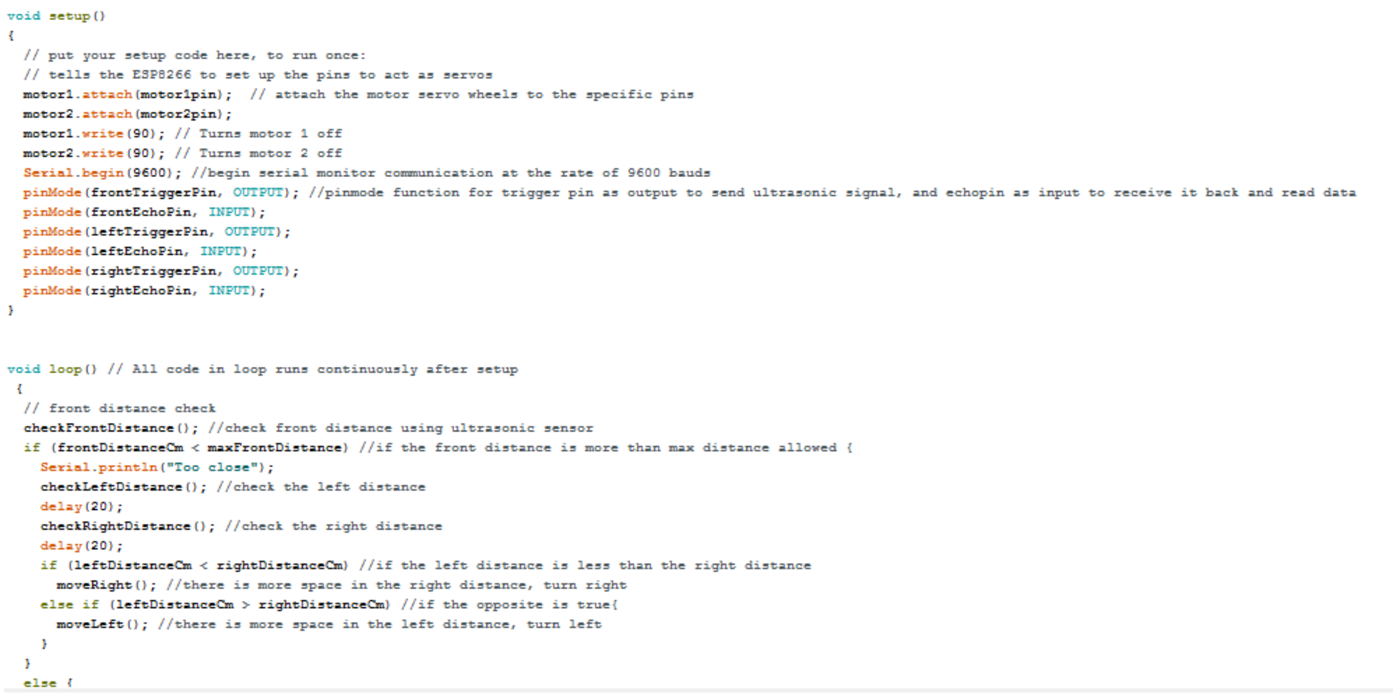
The following code results in simple motion for the robot, such as forward, backward, turning left, and turning right. A deeper explanation is necessary beyond self-explanatory comments for the subroutines of servo motor control. Commanding a motor to write to 90 makes a motor stop moving completely. For some servos, moving forward means writing the code for the object to 90 and above, and for some servos, moving backward means writing the code for the object to 90 and below. The max for 90 and above would be 180 degrees of motion, whereas for 90 and below would be 0 degrees of motion. For some servos, however, the opposite is true. 90 and above results in moving backward, and 90 and below results in moving forwards. It requires trial and error to Figure out how each servo acts. For the latter case, for example, writing the servo as exactly 0 would mean moving forwards at the default max speed of the FM-90 servo, which is 40 cm/s. Whereas writing the servo as exactly 180 would mean moving backwards at the same max speed. Adjusting it to half would result in the speed of 20 cm/s for either direction. It was extremely important to perform trial and error, and retrieve an optimal speed that the robot could travel for this project.

5.1.5.5 ROMULUS AUTONOMY


obstacle_avoidance_trial_2_updated §

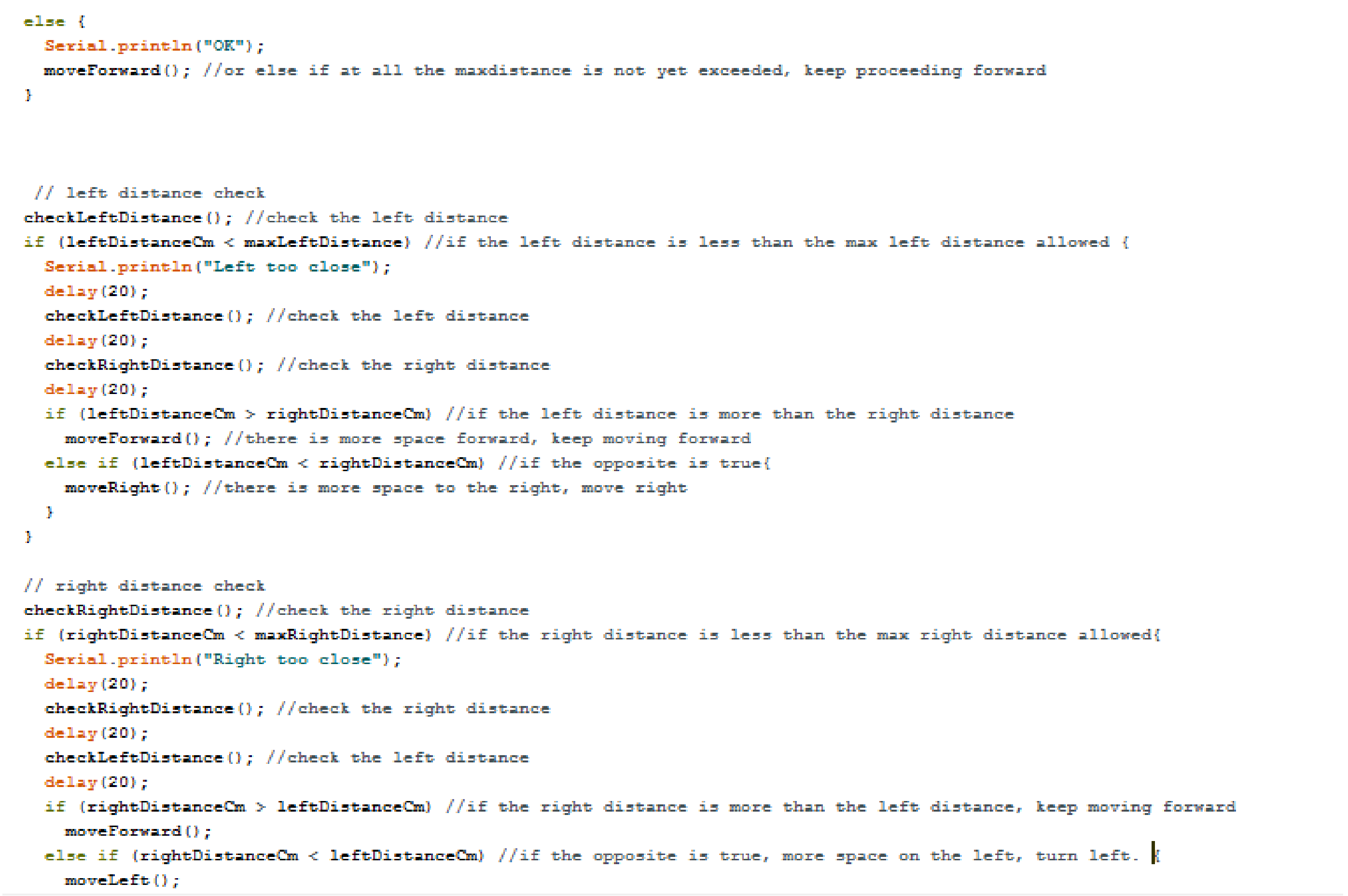
```
#include <Dhcp.h> //utilize all the necessary libraries through Wemos
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <EthernetServer.h>
#include <EthernetUdp.h>
#include <WemosInit.h>
#include <Servo.h>

//Change the pin definitions below
#define motor1pin D5 // Use pins 5 and 9 for motor-modified 11/19/18 based on experience
#define motor2pin D9

const int frontEchoPin = D10; //define variables for each ultrasonic sensor's trigger and echo pin with the specific pins
const int frontTriggerPin = D1;
const int leftEchoPin = D3;
const int leftTriggerPin = D2;
const int rightEchoPin = D7;
const int rightTriggerPin = D4;


Servo motor1;  // Creates a servo object called "motor1"
Servo motor2;  // Creates a servo object called "motor2"




/*Define variables for collecting data with ultrasonic sensors */
volatile float maxFrontDistance = 25.00; //creates a float type variable that returns the maximum front distance allowed
volatile float frontDuration, frontDistanceCm, leftDuration, leftDistanceCm, rightDuration, rightDistanceCm; //float variables for each sensor's duration and distance
volatile float maxLeftDistance, maxRightDistance = 25.00; //creates a float type variable that returns the maximum distance for left and right sensor allowed
```

Figure 27



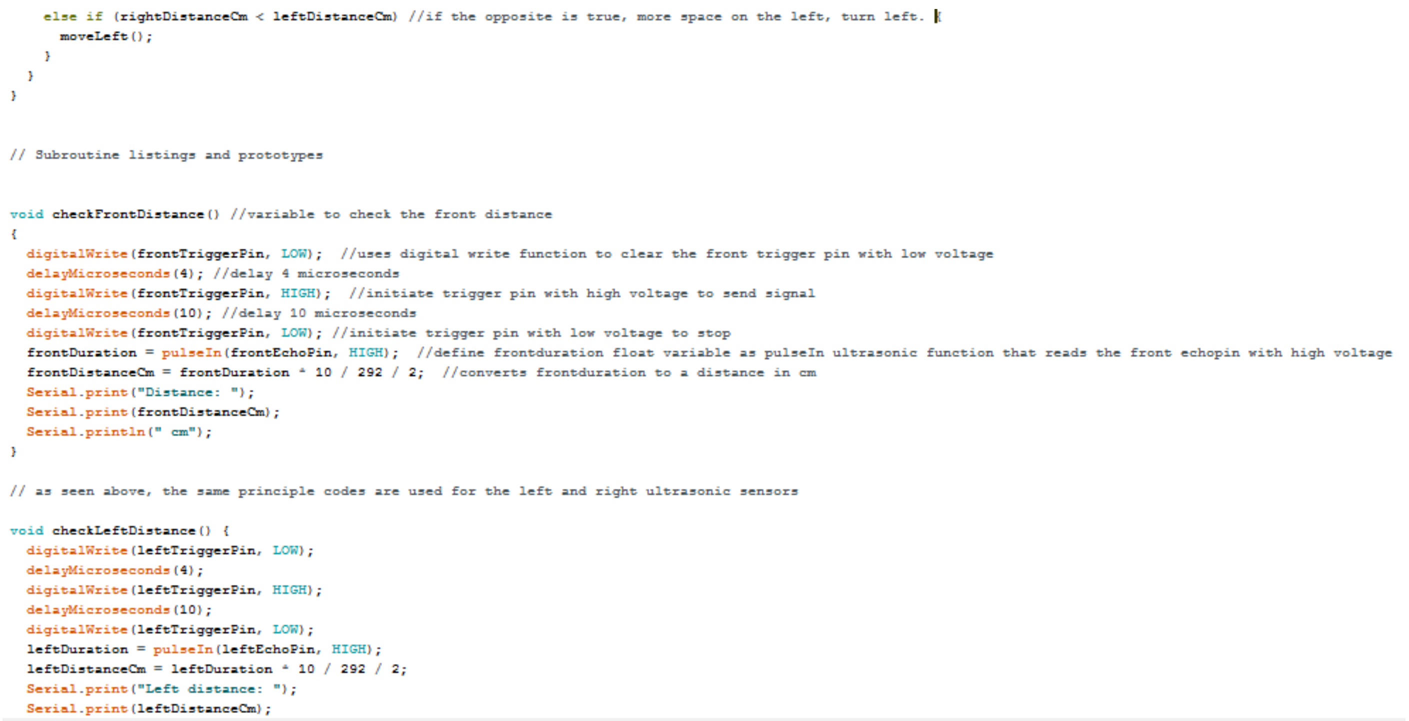
```
void setup()
{
  // put your setup code here, to run once:
  // tells the ESP8266 to set up the pins to act as servos
  motor1.attach(motor1pin);  // attach the motor servo wheels to the specific pins
  motor2.attach(motor2pin);
  motor1.write(90); // Turns motor 1 off
  motor2.write(90); // Turns motor 2 off
  Serial.begin(9600); //begin serial monitor communication at the rate of 9600 bauds
  pinMode(frontTriggerPin, OUTPUT); //pinmode function for trigger pin as output to send ultrasonic signal, and echopin as input to receive it back and read data
  pinMode(frontEchoPin, INPUT);
  pinMode(leftTriggerPin, OUTPUT);
  pinMode(leftEchoPin, INPUT);
  pinMode(rightTriggerPin, OUTPUT);
  pinMode(rightEchoPin, INPUT);
}


void loop() // All code in loop runs continuously after setup
 {
  // front distance check
  checkFrontDistance(); //check front distance using ultrasonic sensor
  if (frontDistanceCm < maxFrontDistance) //if the front distance is more than max distance allowed {
    Serial.println("Too close");
    checkLeftDistance(); //check the left distance
    delay(20);
    checkRightDistance(); //check the right distance
    delay(20);
    if (leftDistanceCm < rightDistanceCm) //if the left distance is less than the right distance
      moveRight(); //there is more space in the right distance, turn right
    else if (leftDistanceCm > rightDistanceCm) //if the opposite is true{
      moveLeft(); //there is more space in the left distance, turn left
    }
  }
  else {
```

Figure 28



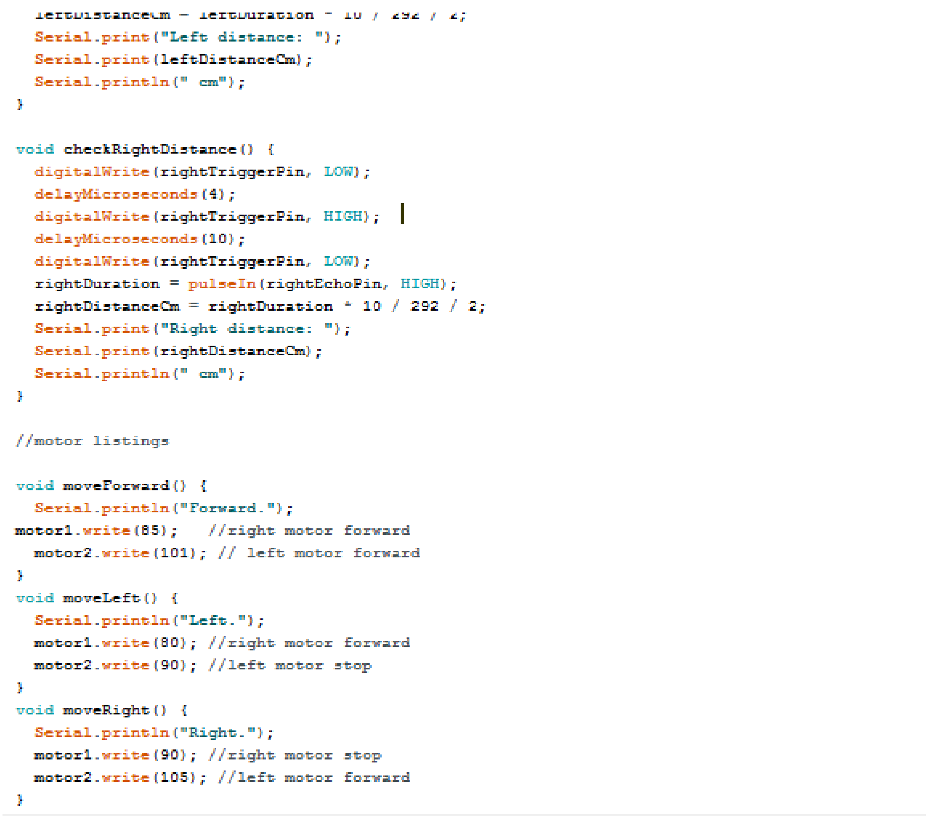
```
 else {
   Serial.println("OK");
   moveForward(); //or else if at all the maxdistance is not yet exceeded, keep proceeding forward
 }



  // left distance check
 checkLeftDistance(); //check the left distance
 if (leftDistanceCm < maxLeftDistance) //if the left distance is less than the max left distance allowed {
   Serial.println("Left too close");
   delay(20);
   checkLeftDistance(); //check the left distance
   delay(20);
   checkRightDistance(); //check the right distance
   delay(20);
   if (leftDistanceCm > rightDistanceCm) //if the left distance is more than the right distance
     moveForward(); //there is more space forward, keep moving forward
   else if (leftDistanceCm < rightDistanceCm) //if the opposite is true{
     moveRight(); //there is more space to the right, move right
   }
 }

 // right distance check
 checkRightDistance(); //check the right distance
 if (rightDistanceCm < maxRightDistance) //if the right distance is less than the max right distance allowed{
   Serial.println("Right too close");
   delay(20);
   checkRightDistance(); //check the right distance
   delay(20);
   checkLeftDistance(); //check the left distance
   delay(20);
   if (rightDistanceCm > leftDistanceCm) //if the right distance is more than the left distance, keep moving forward
     moveForward();
   else if (rightDistanceCm < leftDistanceCm) //if the opposite is true, more space on the left, turn left. {
     moveLeft();
```

Figure 29



```
   else if (rightDistanceCm < leftDistanceCm) //if the opposite is true, more space on the left, turn left. {
     moveLeft();
   }
  }
}


// Subroutine listings and prototypes


void checkFrontDistance() //variable to check the front distance
{
  digitalWrite(frontTriggerPin, LOW);  //uses digital write function to clear the front trigger pin with low voltage
  delayMicroseconds(4); //delay 4 microseconds
  digitalWrite(frontTriggerPin, HIGH);  //initiate trigger pin with high voltage to send signal
  delayMicroseconds(10); //delay 10 microseconds
  digitalWrite(frontTriggerPin, LOW); //initiate trigger pin with low voltage to stop
  frontDuration = pulseIn(frontEchoPin, HIGH);  //define frontduration float variable as pulseIn ultrasonic function that reads the front echopin with high voltage
  frontDistanceCm = frontDuration * 10 / 292 / 2;  //converts frontduration to a distance in cm
  Serial.print("Distance: ");
  Serial.print(frontDistanceCm);
  Serial.println(" cm");
}

// as seen above, the same principle codes are used for the left and right ultrasonic sensors

void checkLeftDistance() {
  digitalWrite(leftTriggerPin, LOW);
  delayMicroseconds(4);
  digitalWrite(leftTriggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(leftTriggerPin, LOW);
  leftDuration = pulseIn(leftEchoPin, HIGH);
  leftDistanceCm = leftDuration * 10 / 292 / 2;
  Serial.print("Left distance: ");
  Serial.print(leftDistanceCm);
```

Figure 30

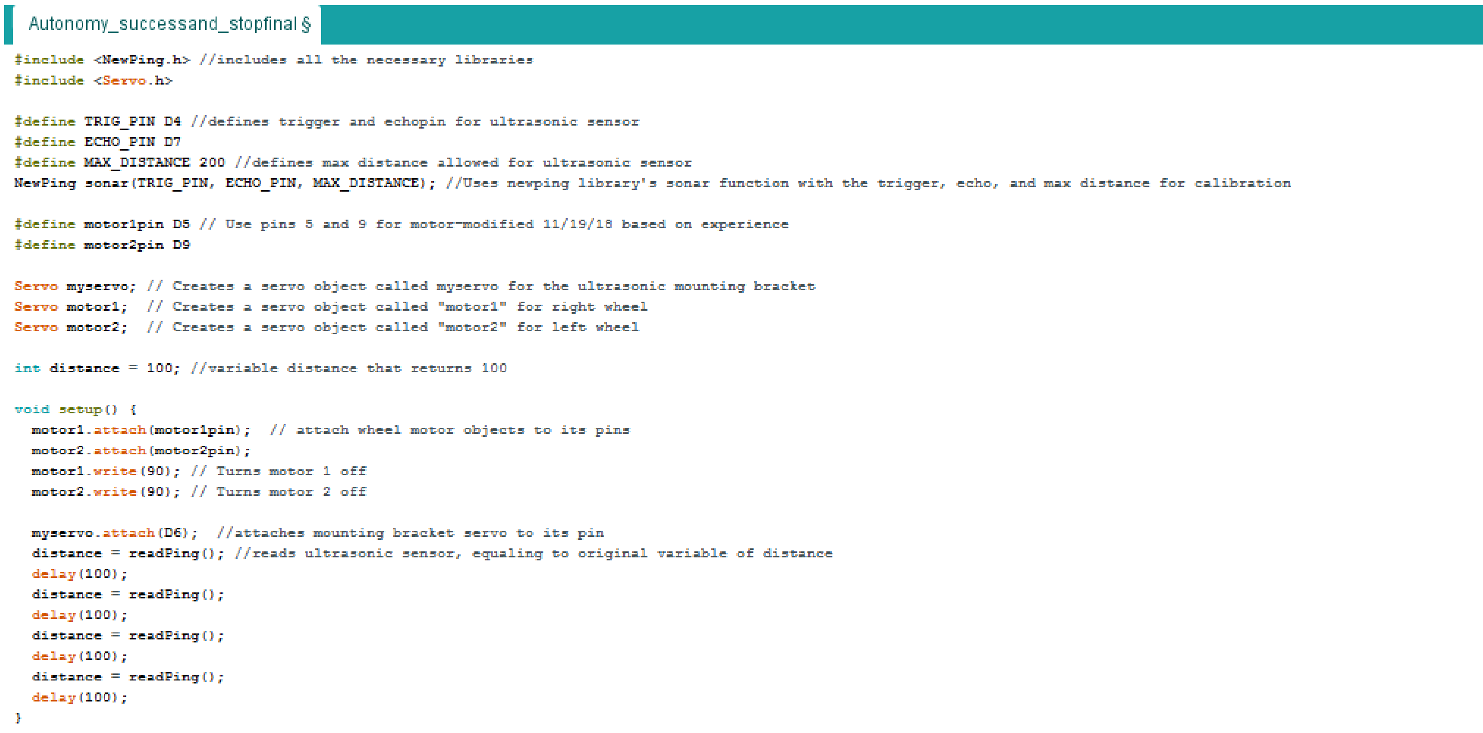


```
  leftDistanceCm = leftDuration * 10 / 292 / 2;
  Serial.print("Left distance: ");
  Serial.print(leftDistanceCm);
  Serial.println(" cm");
}

void checkRightDistance() {
  digitalWrite(rightTriggerPin, LOW);
  delayMicroseconds(4);
  digitalWrite(rightTriggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(rightTriggerPin, LOW);
  rightDuration = pulseIn(rightEchoPin, HIGH);
  rightDistanceCm = rightDuration * 10 / 292 / 2;
  Serial.print("Right distance: ");
  Serial.print(rightDistanceCm);
  Serial.println(" cm");
}

//motor listings

void moveForward() {
  Serial.println("Forward.");
motor1.write(85);   //right motor forward
  motor2.write(101); // left motor forward
}
void moveLeft() {
  Serial.println("Left.");
  motor1.write(80); //right motor forward
  motor2.write(90); //left motor stop
}
void moveRight() {
  Serial.println("Right.");
  motor1.write(90); //right motor stop
  motor2.write(105); //left motor forward
}
```

Figure 31

The following Figures from 26 to 30 determine the final phase of the trial and error for the code for Romulus's obstacle avoidance autonomy by using 3 ultrasonic sensors. The code utilizes clever if, else, and else if statements to regenerate data retrieved by the three ultrasonic sensors placed, with minimal delays, resulting in a fluent motion and obstacle avoidance. In addition, the code also uses all the basic principles learned from motor calibration and ultrasonic sensor calibration codes. The robot never stops moving, and constantly avoids obstacles when seen directly, which helps it salvage its name of "scout autonomy". It is fast, fluent, and smooth! Because it contains 3 units of ultrasonic sensors, it has 180 degrees of absolute field of vision. However, that does not make it an ultimately efficient robot for obstacle avoidance autonomy. At the end of the day, the robot has three units, with two trigger pods for signal sending. It can view at exactly 0, $0.5\boldsymbol{\pi}$, and $\boldsymbol{\pi}$ radians, and theoretically not the surrounding angles. This results in a vast inefficiency for the robot's autonomous capabilities.

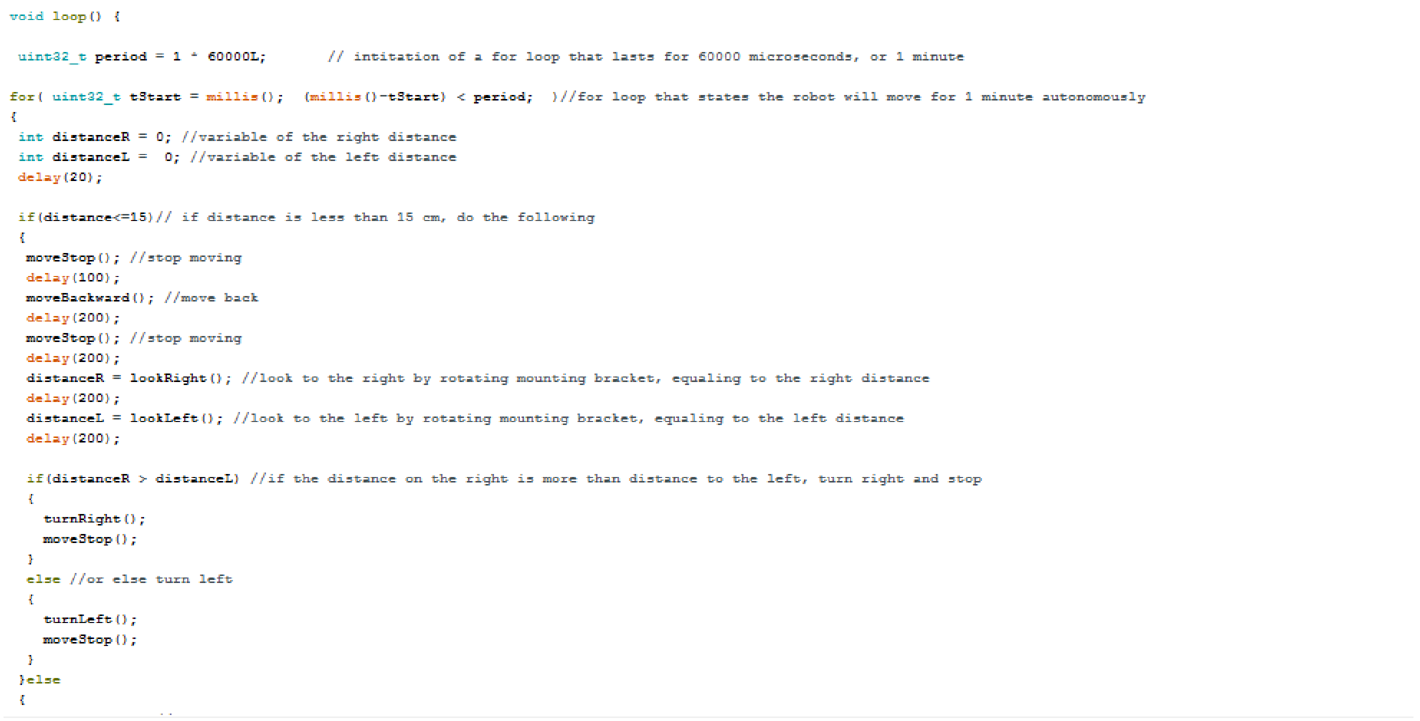
5.1.5.6 REMUS AUTONOMY



Autonomy_successand_stopfinal §

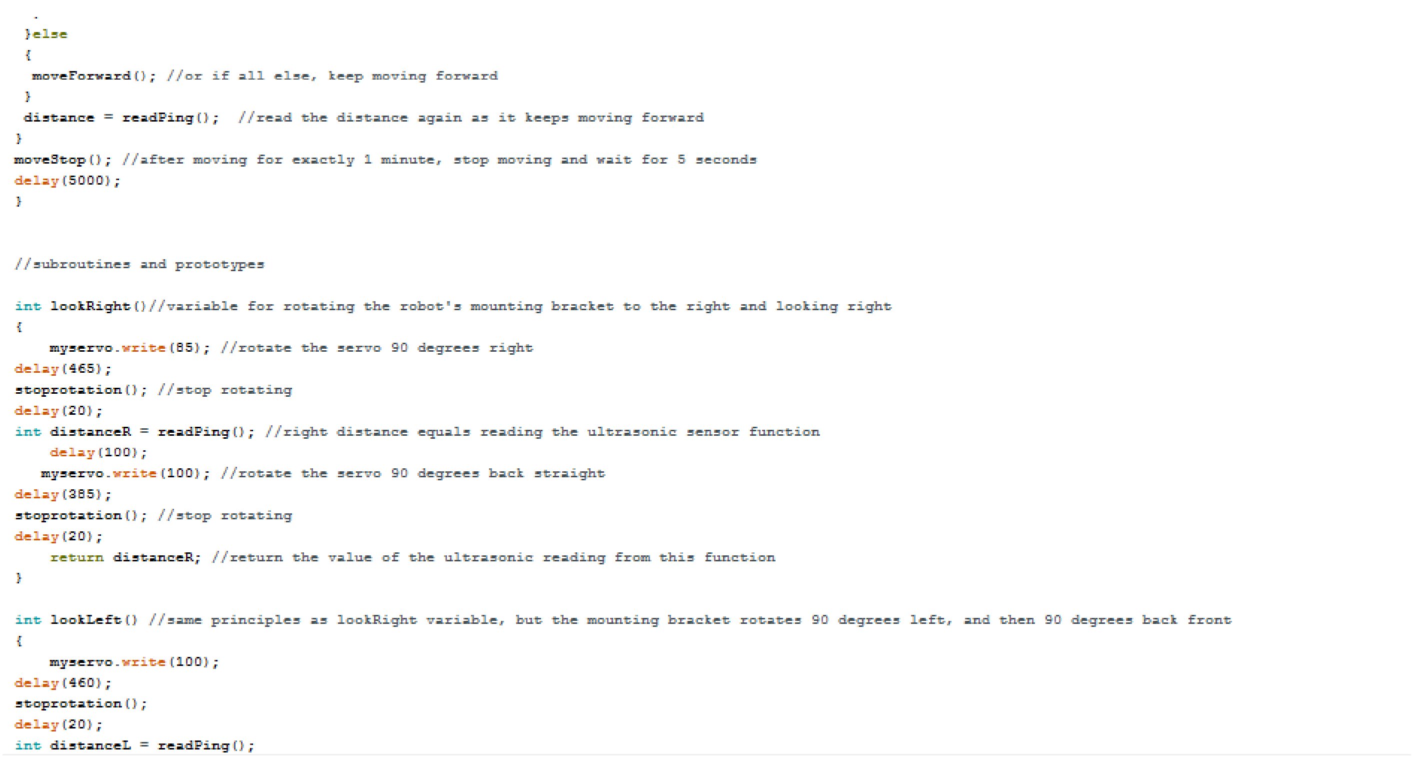
```
#include <NewPing.h> //includes all the necessary libraries
#include <Servo.h>

#define TRIG_PIN D4 //defines trigger and echopin for ultrasonic sensor
#define ECHO_PIN D7
#define MAX_DISTANCE 200 //defines max distance allowed for ultrasonic sensor
NewPing sonar(TRIG_PIN, ECHO_PIN, MAX_DISTANCE); //Uses newping library's sonar function with the trigger, echo, and max distance for calibration

#define motor1pin D5 // Use pins 5 and 9 for motor-modified 11/19/18 based on experience
#define motor2pin D9

Servo myservo; // Creates a servo object called myservo for the ultrasonic mounting bracket
Servo motor1;  // Creates a servo object called "motor1" for right wheel
Servo motor2;  // Creates a servo object called "motor2" for left wheel

int distance = 100; //variable distance that returns 100

void setup() {
  motor1.attach(motor1pin);  // attach wheel motor objects to its pins
  motor2.attach(motor2pin);
  motor1.write(90); // Turns motor 1 off
  motor2.write(90); // Turns motor 2 off

  myservo.attach(D6);  //attaches mounting bracket servo to its pin
  distance = readPing(); //reads ultrasonic sensor, equaling to original variable of distance
  delay(100);
  distance = readPing();
  delay(100);
  distance = readPing();
  delay(100);
  distance = readPing();
  delay(100);
}
```

Figure 32



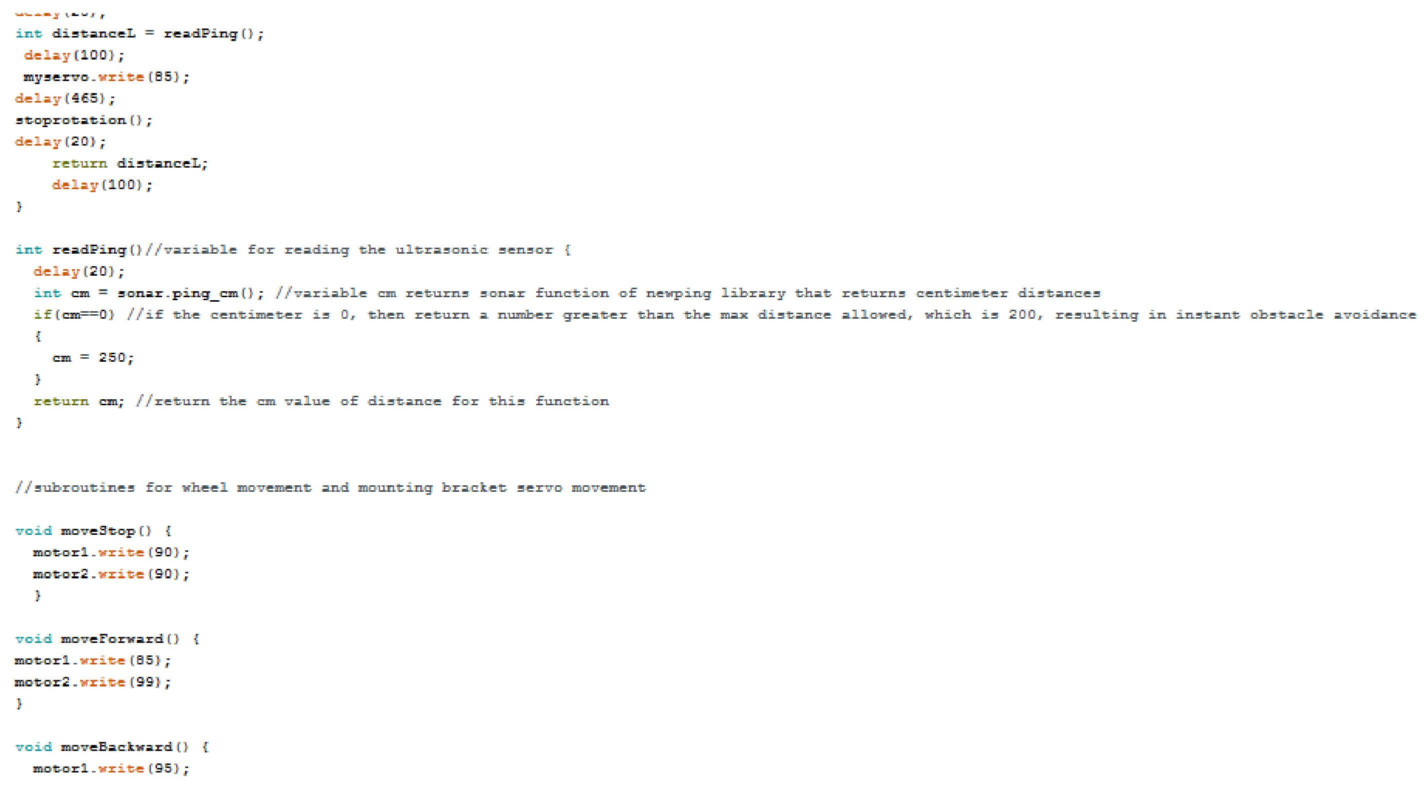
```
void loop() {

 uint32_t period = 1 * 60000L;       // intitation of a for loop that lasts for 60000 microseconds, or 1 minute

for( uint32_t tStart = millis();  (millis()-tStart) < period;  )//for loop that states the robot will move for 1 minute autonomously
{
 int distanceR = 0; //variable of the right distance
 int distanceL =  0; //variable of the left distance
 delay(20);

 if(distance<=15)// if distance is less than 15 cm, do the following
 {
  moveStop(); //stop moving
  delay(100);
  moveBackward(); //move back
  delay(200);
  moveStop(); //stop moving
  delay(200);
  distanceR = lookRight(); //look to the right by rotating mounting bracket, equaling to the right distance
  delay(200);
  distanceL = lookLeft(); //look to the left by rotating mounting bracket, equaling to the left distance
  delay(200);

  if(distanceR > distanceL) //if the distance on the right is more than distance to the left, turn right and stop
  {
    turnRight();
    moveStop();
  }
  else //or else turn left
  {
    turnLeft();
    moveStop();
  }
 }else
 {
```

Figure 33



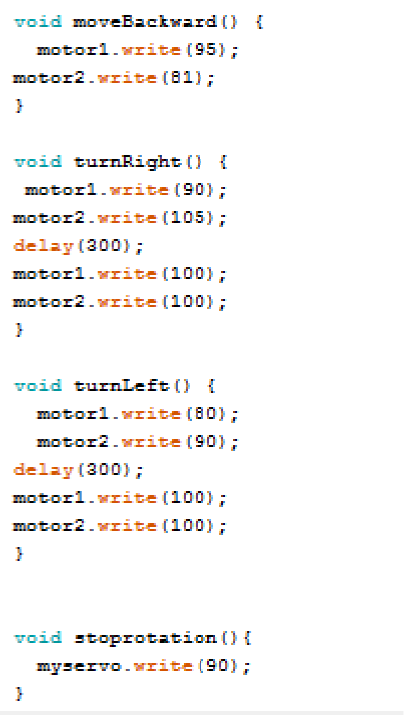
```
 }else
 {
  moveForward(); //or if all else, keep moving forward
 }
 distance = readPing();  //read the distance again as it keeps moving forward
}
moveStop(); //after moving for exactly 1 minute, stop moving and wait for 5 seconds
delay(5000);
}


//subroutines and prototypes

int lookRight()//variable for rotating the robot's mounting bracket to the right and looking right
{
    myservo.write(85); //rotate the servo 90 degrees right
delay(465);
stoprotation(); //stop rotating
delay(20);
int distanceR = readPing(); //right distance equals reading the ultrasonic sensor function
    delay(100);
   myservo.write(100); //rotate the servo 90 degrees back straight
delay(385);
stoprotation(); //stop rotating
delay(20);
    return distanceR; //return the value of the ultrasonic reading from this function
}

int lookLeft() //same principles as lookRight variable, but the mounting bracket rotates 90 degrees left, and then 90 degrees back front
{
    myservo.write(100);
delay(460);
stoprotation();
delay(20);
int distanceL = readPing();
```

Figure 34



```
int distanceL = readPing();
 delay(100);
 myservo.write(85);
delay(465);
stoprotation();
delay(20);
    return distanceL;
    delay(100);
}

int readPing()//variable for reading the ultrasonic sensor {
  delay(20);
  int cm = sonar.ping_cm(); //variable cm returns sonar function of newping library that returns centimeter distances
  if(cm==0) //if the centimeter is 0, then return a number greater than the max distance allowed, which is 200, resulting in instant obstacle avoidance
  {
    cm = 250;
  }
  return cm; //return the cm value of distance for this function
}


//subroutines for wheel movement and mounting bracket servo movement

void moveStop() {
  motor1.write(90);
  motor2.write(90);
  }

void moveForward() {
motor1.write(85);
motor2.write(99);
}

void moveBackward() {
  motor1.write(95);
```

Figure 35

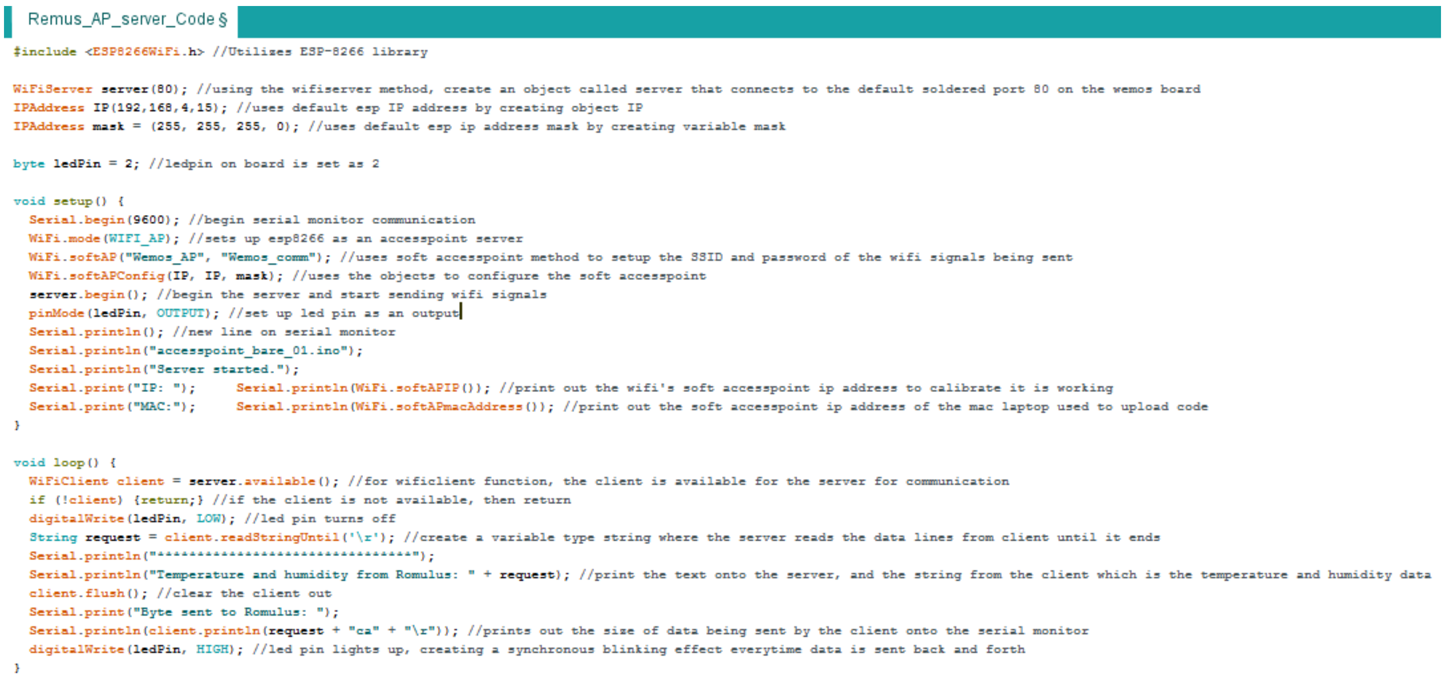


```
void moveBackward() {
  motor1.write(95);
motor2.write(81);
}

void turnRight() {
 motor1.write(90);
motor2.write(105);
delay(300);
motor1.write(100);
motor2.write(100);
}

void turnLeft() {
  motor1.write(80);
  motor2.write(90);
delay(300);
motor1.write(100);
motor2.write(100);
}


void stoprotation(){
  myservo.write(90);
}
```

Figure 36

The following Figures from 31 to 35 determine Remus's code for autonomy. This style of autonomy has a newer coding methodology that was utilized to improve the final code. A for loop is used where the robot moves around for exactly 1 minute, and then after 1 minute of movement, the robot stops and waits 5 seconds. This is an important improvement because this would allow the two robots to move around autonomously on a certain path for a certain amount of time, stop, read the necessary data, and then share the data through Wi-Fi collaboration. Remus's style of autonomy is completely different than Romulus's style of autonomy. It's much slower, much less fluent, and in some cases, much less accurate. Unlike Romulus, the robot comes to a complete stop when it detects an obstacle, and calibrates for the best possible course. The calibration for the best possible course is also not precise, because the servo attempts to rotate around and cover the same ground as 3 units of ultrasonic sensors, like on Romulus. That is not always perfect because the servo movement can vary based on voltage, code, and physical constraints such as the weight of the mounting bracket. This leads to a lot of inefficiency when trying to read exactly 180 degrees of field of vision.

5.1.5.7 REMUS ACCESS POINT SERVER CODE

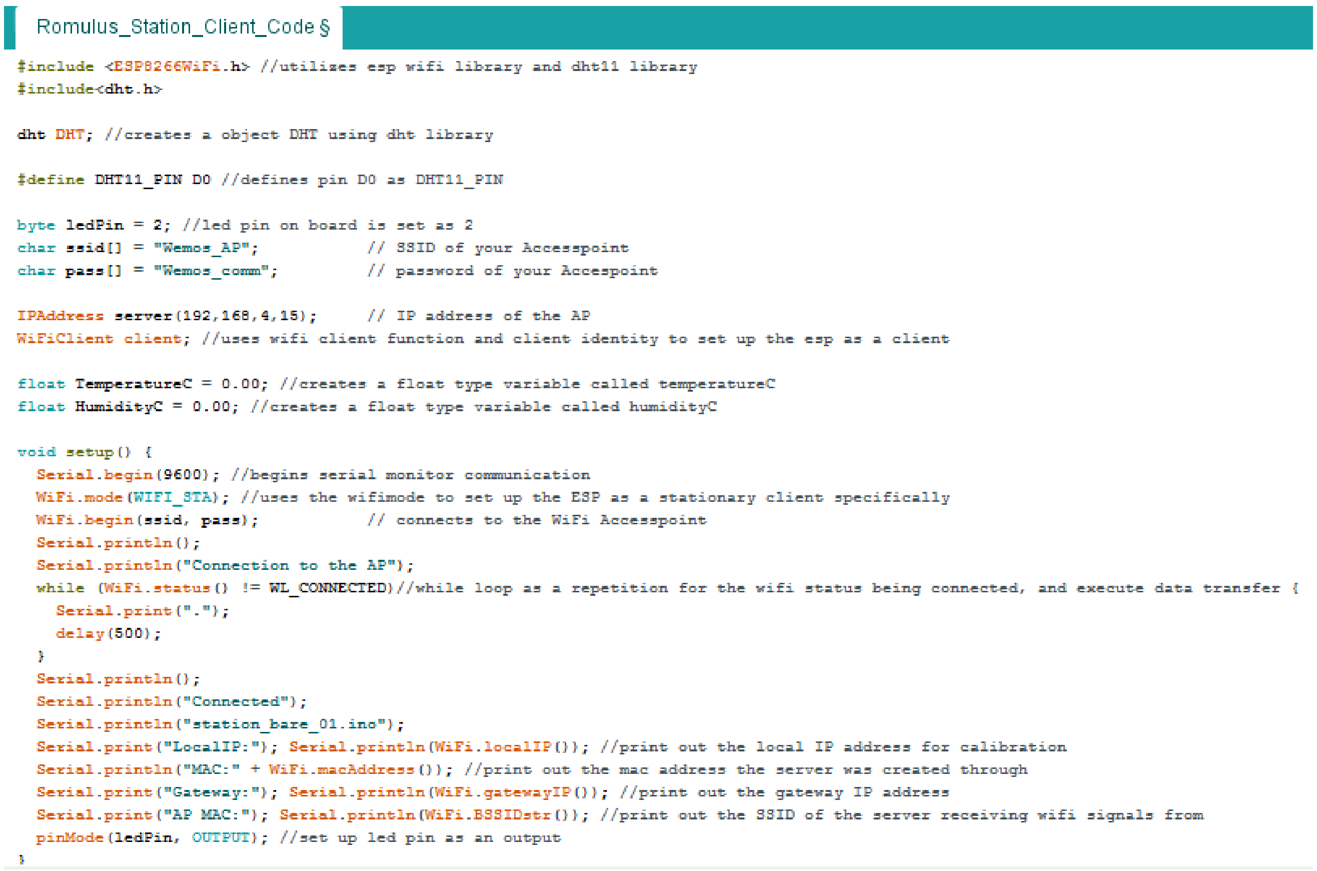


```
#include <ESP8266WiFi.h> //Utilizes ESP-8266 library

WiFiServer server(80); //using the wifiserver method, create an object called server that connects to the default soldered port 80 on the wemos board
IPAddress IP(192,168,4,15); //uses default esp IP address by creating object IP
IPAddress mask = (255, 255, 255, 0); //uses default esp ip address mask by creating variable mask

byte ledPin = 2; //ledpin on board is set as 2

void setup() {
  Serial.begin(9600); //begin serial monitor communication
  WiFi.mode(WIFI_AP); //sets up esp8266 as an accesspoint server
  WiFi.softAP("Wemos_AP", "Wemos_comm"); //uses soft accesspoint method to setup the SSID and password of the wifi signals being sent
  WiFi.softAPConfig(IP, IP, mask); //uses the objects to configure the soft accesspoint
  server.begin(); //begin the server and start sending wifi signals
  pinMode(ledPin, OUTPUT); //set up led pin as an output
  Serial.println(); //new line on serial monitor
  Serial.println("accesspoint_bare_01.ino");
  Serial.println("Server started.");
  Serial.print("IP: ");     Serial.println(WiFi.softAPIP()); //print out the wifi's soft accesspoint ip address to calibrate it is working
  Serial.print("MAC:");     Serial.println(WiFi.softAPmacAddress()); //print out the soft accesspoint ip address of the mac laptop used to upload code
}

void loop() {
  WiFiClient client = server.available(); //for wificlient function, the client is available for the server for communication
  if (!client) {return;} //if the client is not available, then return
  digitalWrite(ledPin, LOW); //led pin turns off
  String request = client.readStringUntil('\r'); //create a variable type string where the server reads the data lines from client until it ends
  Serial.println("********************************");
  Serial.println("Temperature and humidity from Romulus: " + request); //print the text onto the server, and the string from the client which is the temperature and humidity data
  client.flush(); //clear the client out
  Serial.print("Byte sent to Romulus: ");
  Serial.println(client.println(request + "ca" + "\r")); //prints out the size of data being sent by the client onto the serial monitor
  digitalWrite(ledPin, HIGH); //led pin lights up, creating a synchronous blinking effect everytime data is sent back and forth
}
```

Figure 37

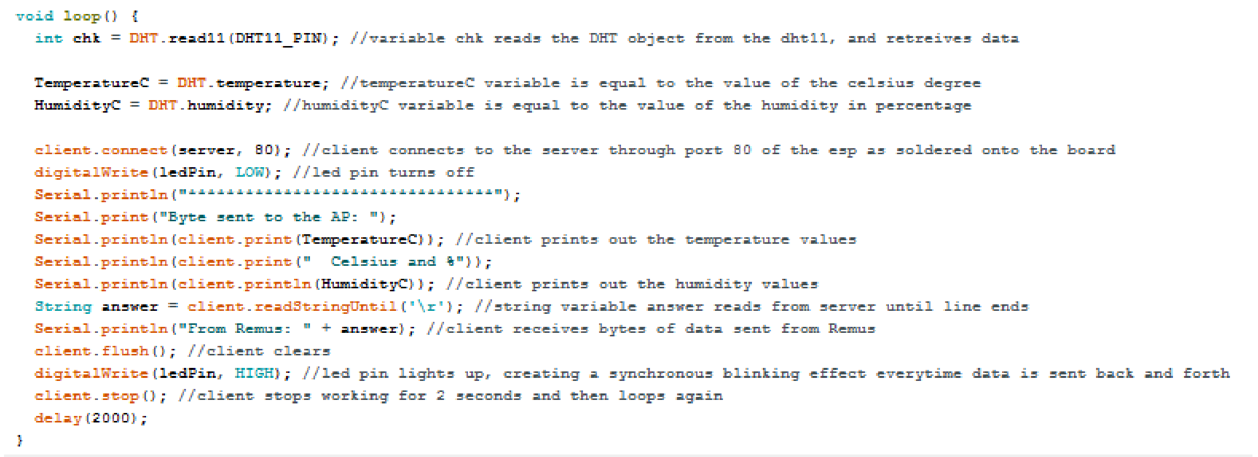
The following code in Figure 36 represents Remus's capabilities of setting up as a Wi-Fi server, sending out signals, and then receiving DHT-11 temperature and humidity data from the stationary client Romulus. Aside from that, it also receives information of the bytes of data being sent from Romulus. The most sophisticated and appealing part of this code, aside from creating a network using just two microprocessors, is the fact that code works in such a way that both robots will blink their WeMos boards with synchrony each time data is sent back and forth, which allows the engineer or scientist to assess if it is indeed working.

5.1.5.8 ROMULUS STATIONARY CLIENT CODE


Romulus_Station_Client_Code §

```
#include <ESP8266WiFi.h> //utilizes esp wifi library and dht11 library
#include<dht.h>

dht DHT; //creates a object DHT using dht library

#define DHT11_PIN D0 //defines pin D0 as DHT11_PIN

byte ledPin = 2; //led pin on board is set as 2
char ssid[] = "Wemos_AP";           // SSID of your Accesspoint
char pass[] = "Wemos_comm";         // password of your Accespoint

IPAddress server(192,168,4,15);     // IP address of the AP
WiFiClient client; //uses wifi client function and client identity to set up the esp as a client

float TemperatureC = 0.00; //creates a float type variable called temperatureC
float HumidityC = 0.00; //creates a float type variable called humidityC

void setup() {
  Serial.begin(9600); //begins serial monitor communication
  WiFi.mode(WIFI_STA); //uses the wifimode to set up the ESP as a stationary client specifically
  WiFi.begin(ssid, pass);           // connects to the WiFi Accesspoint
  Serial.println();
  Serial.println("Connection to the AP");
  while (WiFi.status() != WL_CONNECTED)//while loop as a repetition for the wifi status being connected, and execute data transfer {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.println("Connected");
  Serial.println("station_bare_01.ino");
  Serial.print("LocalIP:"); Serial.println(WiFi.localIP()); //print out the local IP address for calibration
  Serial.println("MAC:" + WiFi.macAddress()); //print out the mac address the server was created through
  Serial.print("Gateway:"); Serial.println(WiFi.gatewayIP()); //print out the gateway IP address
  Serial.print("AP MAC:"); Serial.println(WiFi.BSSIDstr()); //print out the SSID of the server receiving wifi signals from
  pinMode(ledPin, OUTPUT); //set up led pin as an output
}
```

Figure 38



```
void loop() {
  int chk = DHT.read11(DHT11_PIN); //variable chk reads the DHT object from the dht11, and retreives data

  TemperatureC = DHT.temperature; //temperatureC variable is equal to the value of the celsius degree
  HumidityC = DHT.humidity; //humidityC variable is equal to the value of the humidity in percentage

  client.connect(server, 80); //client connects to the server through port 80 of the esp as soldered onto the board
  digitalWrite(ledPin, LOW); //led pin turns off
  Serial.println("**********************************");
  Serial.print("Byte sent to the AP: ");
  Serial.println(client.print(TemperatureC)); //client prints out the temperature values
  Serial.println(client.print("  Celsius and %"));
  Serial.println(client.println(HumidityC)); //client prints out the humidity values
  String answer = client.readStringUntil('\r'); //string variable answer reads from server until line ends
  Serial.println("From Remus: " + answer); //client receives bytes of data sent from Remus
  client.flush(); //client clears
  digitalWrite(ledPin, HIGH); //led pin lights up, creating a synchronous blinking effect everytime data is sent back and forth
  client.stop(); //client stops working for 2 seconds and then loops again
  delay(2000);
}
```

Figure 39

The following code through Figure 37 and 38 determine Romulus's code for setting up as a client that receives Remus's Wi-Fi signals, and sends DHT-11 data back to Remus. It also receives information regarding the bytes of data sent from Remus when data is sent back and forth. In doing so, the LED on the board will blink every time data is sent back and forth.

6 Results and Testing Phase

Lava Tube Trials

6.1.1 Lava Tube Maze Structure

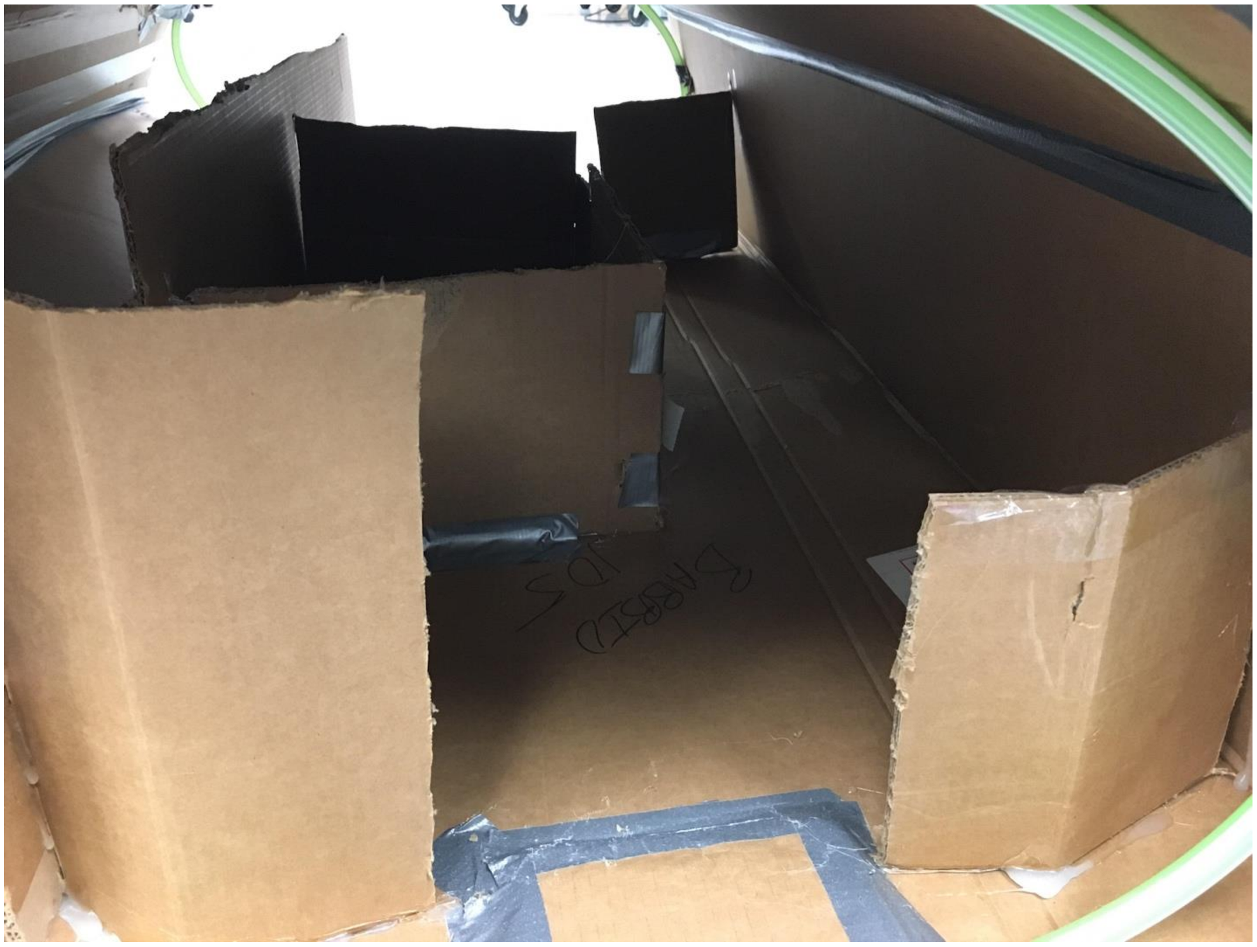


Figure 40

Figure 39 represents the front end of the lava tube, and the maze structure that follows it. The robots come out of this end, so it is considered as the exit point.

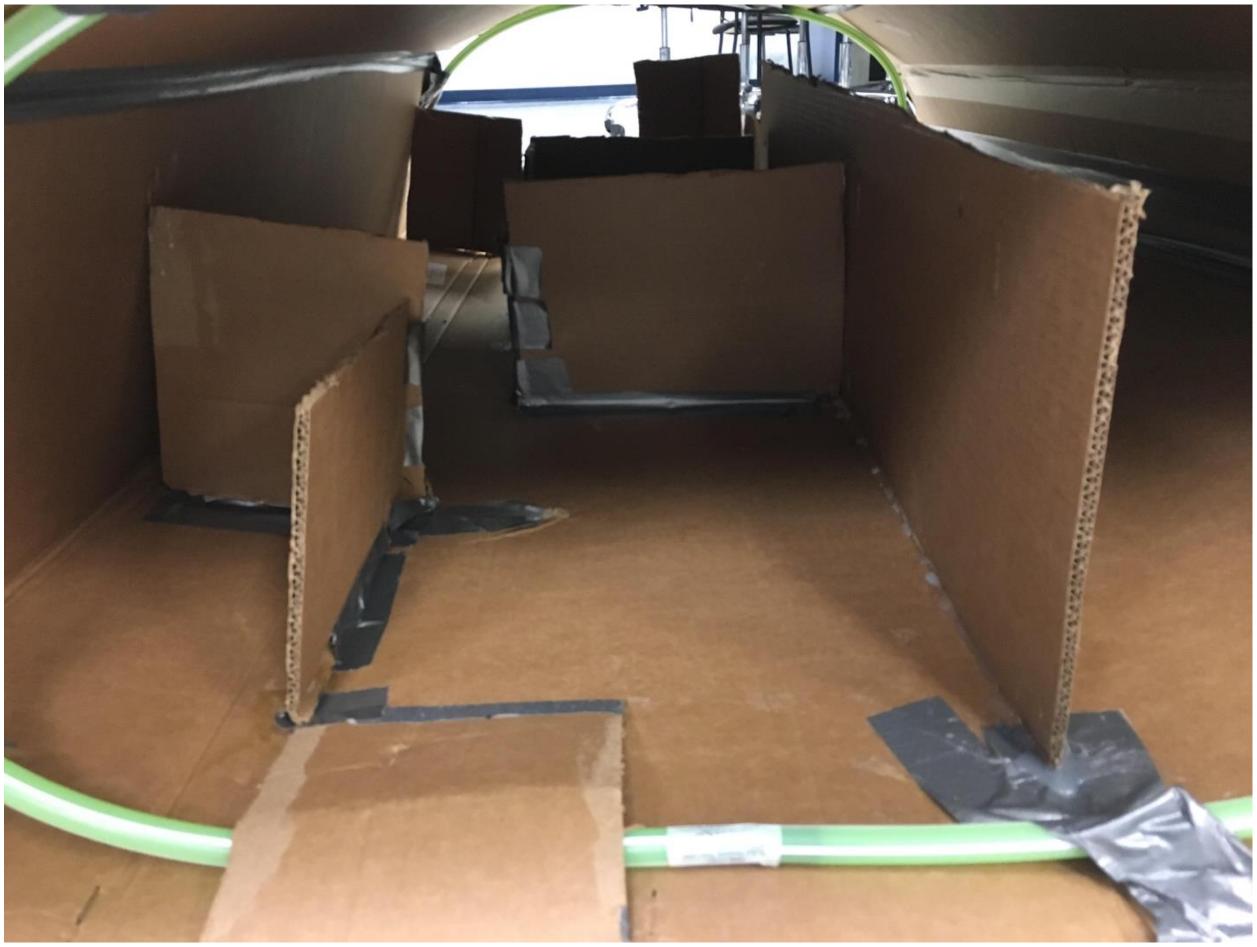


Figure 41

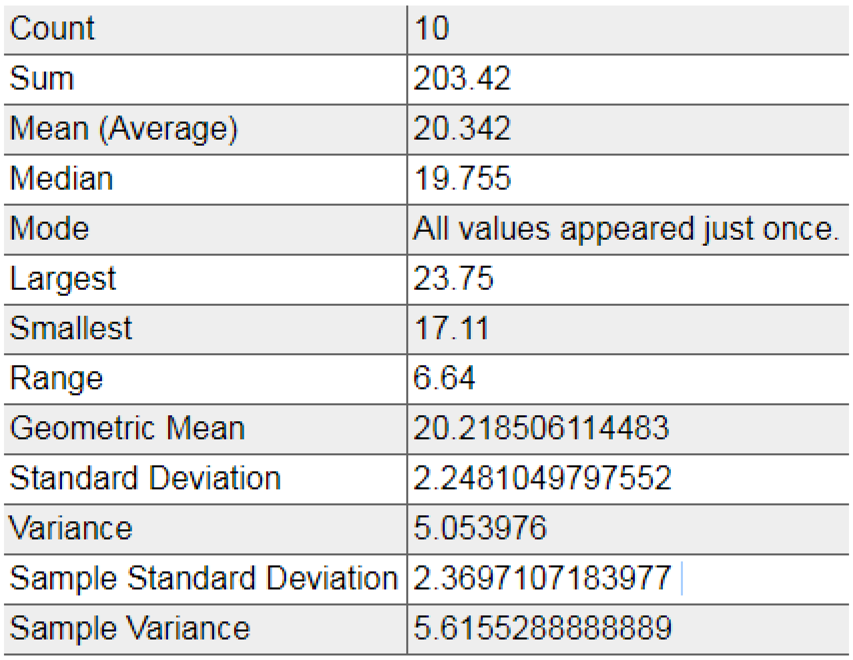
Figure 40 represents the entrance end of the lava tube, where the robots are released and then tackles the maze as obstacles.

Romulus Lava Tube Trial Time (seconds)	Remus Lava Tube Trial Time (seconds)
18.37	31.56
17.98	33.48
17.11	33.00
19.25	36.89
23.75	26.34
22.51	39.56
23.14	44.89

18.91	25.92
20.26	46.73
22.14	23.12

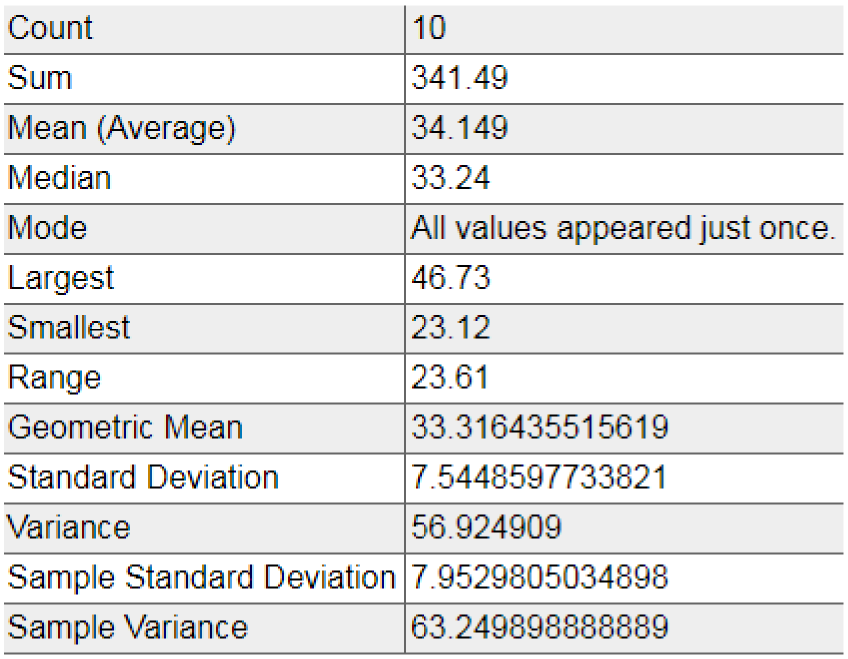
Figure 42

Calculation for Romulus's Data:



Count	10
Sum	203.42
Mean (Average)	20.342
Median	19.755
Mode	All values appeared just once.
Largest	23.75
Smallest	17.11
Range	6.64
Geometric Mean	20.218506114483
Standard Deviation	2.2481049797552
Variance	5.053976
Sample Standard Deviation	2.3697107183977
Sample Variance	5.6155288888889

Calculation for Remus's Data:



Count	10
Sum	341.49
Mean (Average)	34.149
Median	33.24
Mode	All values appeared just once.
Largest	46.73
Smallest	23.12
Range	23.61
Geometric Mean	33.316435515619
Standard Deviation	7.5448597733821
Variance	56.924909
Sample Standard Deviation	7.9529805034898
Sample Variance	63.249898888889

Per Figure 41, the first test conducted was the lava tube course trials. Both robots were released through the lava tube maze's entry point as shown in Figure 40, with approximately the same speed as written on the code. Although it is important to know the servo's factory speed settings of 40 cm/s as the max speed, the speed was not a significant factor for this data collection, because an optimal and slow velocity was important to go through the maze. It was important, however, for both robots to have the same exact speed, so the code was executed to do so. As seen from the data and calculations, the Romulus trials contains less variability of numbers, a smaller range, and smaller standard deviation than Remus's. This implies directly that Romulus's autonomy is much more consistent. This may be since 3 units of ultrasonic sensors are much more accurate than relying on a servo to cover the same 180-degree field of vision as 3 units of ultrasonic sensor. Remus's trials in the lava tube maze is much slower, as seen from the data's higher second counts, this is since the robot moves, stops, and calibrates, and may sometimes move directly towards another object until it stops again and calibrates. As mentioned previously, Romulus's autonomy is unlike Remus's, as it is much more fluent, and there are no stops. There are pros and cons to both of their motor movements.

DHT11 Test

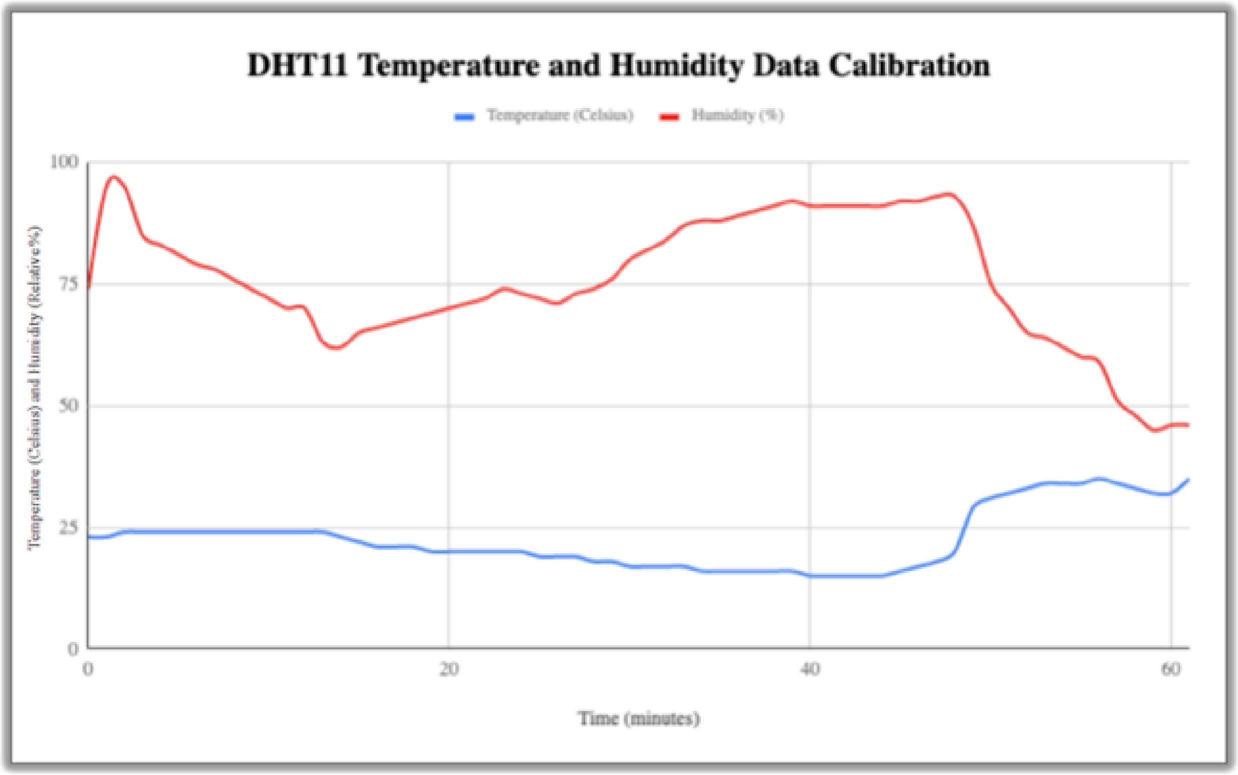
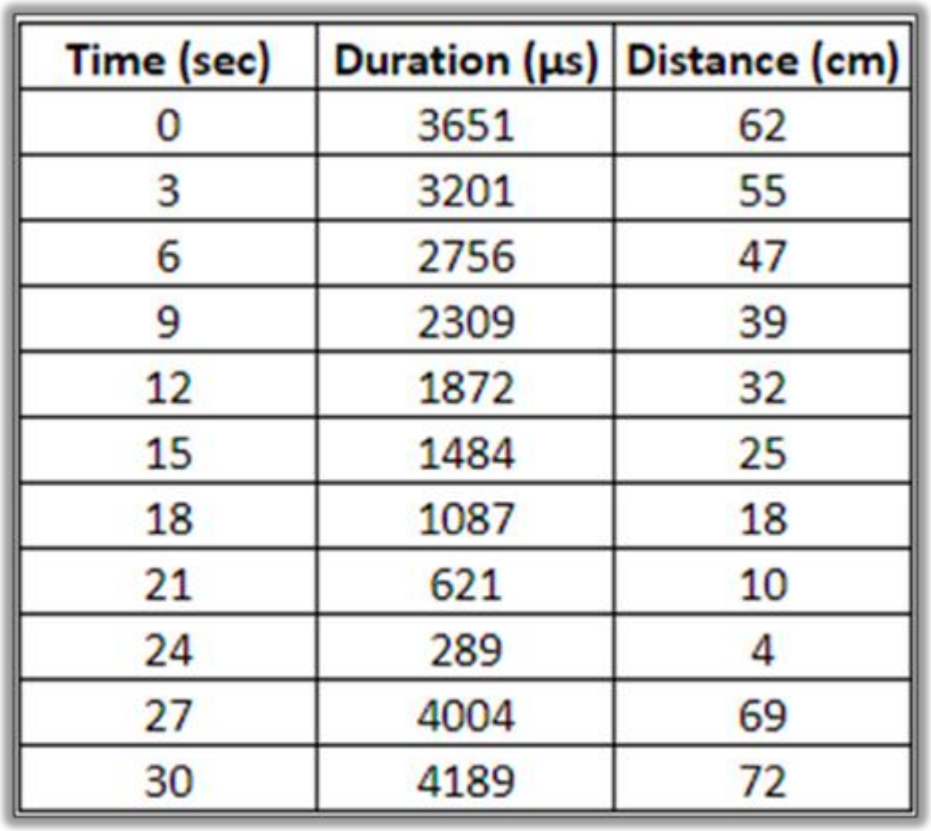


Figure 43

To assess that the DHT-11 sensor works accurately, a DHT-11 sensor test was performed. The data was printed onto the serial monitor from a sample circuit, and then initially, as seen around the first 20 minutes, the sensor was touched with a damp tissue, which led to an increase in humidity, but no change in temperature. The sensor was left in a cold room for the next 20 minutes, while the humidity went down to its original state. After the 20-minute mark, the sensor was placed next to a high-speed air conditioner, which led to an increase in humidity, and a steady decrease in temperature. Finally, after the 40-minute mark, the sensor was placed outside, next to the sun, during a dry and hot day. This lead to a sharp jump in temperature, and a sharp decline of humidity. It was, indeed, understood that the DHT-11, although an inexpensive tool, gives sensible readings.

Obstacle Avoidance Ultrasonic Sensor Test

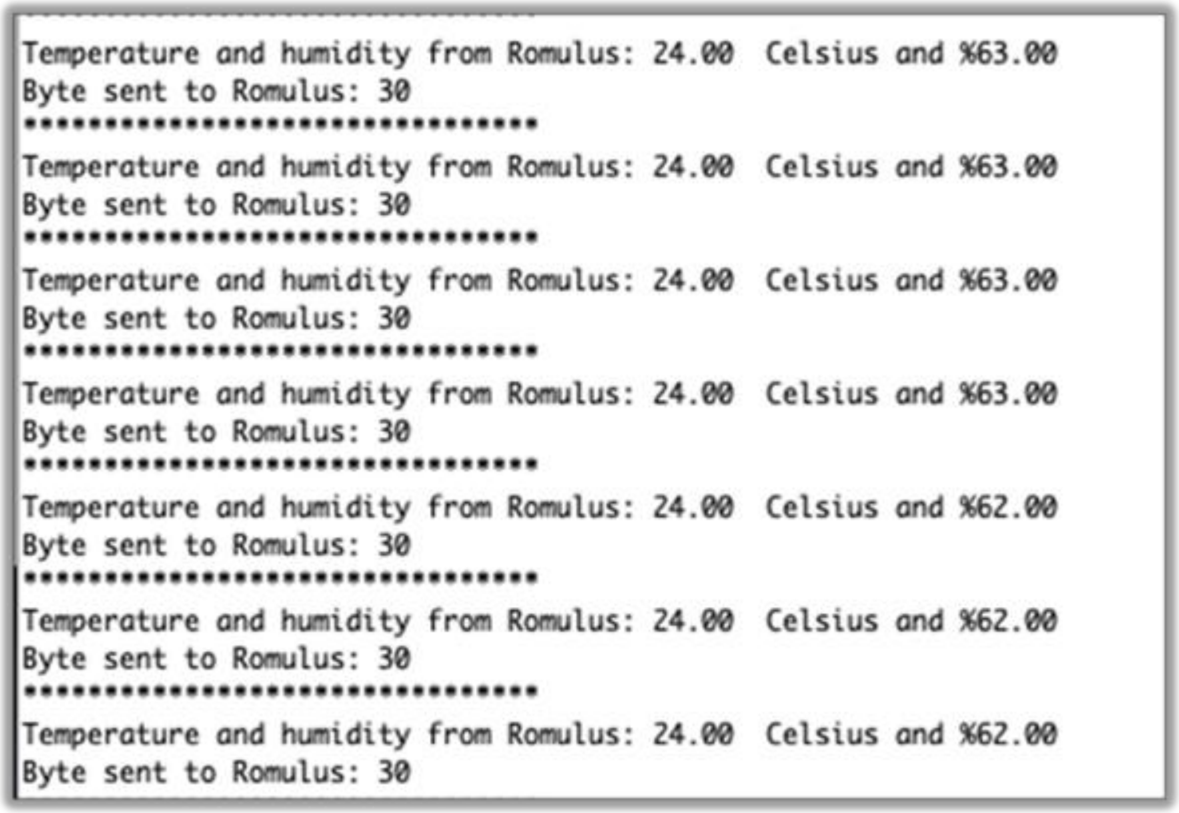


Time (sec)	Duration (µs)	Distance (cm)
0	3651	62
3	3201	55
6	2756	47
9	2309	39
12	1872	32
15	1484	25
18	1087	18
21	621	10
24	289	4
27	4004	69
30	4189	72

Figure 44

The following data demonstrates a 30 second session of Romulus as a sample to demonstrate the HC-SR04 ultrasonic sensor's obstacle avoidance capabilities. Following the scout autonomy code, which results in a 3 second delay for the serial monitor to print out the distance readings, the front sensor was used to create Figure 43. Between time 0 seconds to 24 seconds, Romulus approaches an obstacle at its default coded speed, which results in the duration and the distance decreasing steadily. The duration decreases because it takes less time for the ultrasonic signal to be sent and bounce back. Romulus then proceeds to make a turn to a space with more distance during time 27 seconds, which leads to distances and durations that are much larger, allowing the robot to keep moving forward until there is another obstacle. The 20-microsecond delay causes a vast amount of inefficiency for the robot's obstacle avoidance capabilities, and in a lot of cases, sometimes the ultrasonic sensors do not work perfectly, as they have slow refresh rates, and may lag. The ultrasonic sensor also fails to cover other diverse angles beyond where it is placed a whole unit. Although this data represents a successful trial, implying that the HC-SR04 ultrasonic is a sustainable tool, it is a 100% reliable. It is also important to note that the 20-microsecond delay of the ultrasonic sensor is NOT the same as the 3 second delay of printing out the data onto the serial monitor.

Wi-Fi Collaboration Test



```
Temperature and humidity from Romulus: 24.00  Celsius and %63.00
Byte sent to Romulus: 30
********************************
Temperature and humidity from Romulus: 24.00  Celsius and %63.00
Byte sent to Romulus: 30
********************************
Temperature and humidity from Romulus: 24.00  Celsius and %63.00
Byte sent to Romulus: 30
********************************
Temperature and humidity from Romulus: 24.00  Celsius and %63.00
Byte sent to Romulus: 30
********************************
Temperature and humidity from Romulus: 24.00  Celsius and %62.00
Byte sent to Romulus: 30
********************************
Temperature and humidity from Romulus: 24.00  Celsius and %62.00
Byte sent to Romulus: 30
********************************
Temperature and humidity from Romulus: 24.00  Celsius and %62.00
Byte sent to Romulus: 30
```

Figure 45

The following Figure is a simple demonstration of the capabilities for both Romulus and Remus to collaborate. They are both coded to blink each time two lines of data are sent, separated by dots of line. Both robots are coded to blink on the WeMos board each time the data is sent back and forth. There is approximately a 1 second delay between each time both robots blink and send data back and forth. Depending on the surrounding environment, the temperature and the humidity will obviously change. However, the bytes of data are usually not changed, since the robots are constantly sending back and forth the same type of data. The bytes of data would theoretically only change if different types of stringed data are sent between the client and the server.

Errors:

- Servo breaking and calibrating was a provoking error for this project. As mentioned previously, FM-90 servos are not ideal motors to put on wheels. Wheels carry the weight of the whole robot, and FM-90 servos are too miniature to take upon that much force, especially when it moves in different vectors and takes upon more stress. During the lava tube trials, if the robots ever ran into an obstacle because of coding issues, or ultrasonic sensor inefficiency, the members of the project would often try pulling the robot back while the robot's motors kept moving forward. This resulted in the servos breaking, because it would not be able to bear moving both forward and backward at the same time. It did not slow down the momentum of the project by a great amount, but it led to more time being spent on servicing, repairing, and troubleshooting. Coding for the servos are also not exactly perfect. For some servos, the range of writing the speed is between 0 and 180. Whereas for servos, as odd as it is, the range is between -5 and 185. This led to difficulty trying to code both robots to have the same speed. When trying to code Remus's ultrasonic mounting bracket servo to look 90 degrees to the right, back to the front, 90 degrees to the left, and back to the front again, the code would not execute perfectly. For the first 3 loops, it may work, but then the robot may start looking 100 degrees to the right and 80 degrees to the left, leading to EXTREME inefficiency for obstacle avoidance. In the future, using different motors may be a better choice, despite the efforts of sustainability.
- Integrating the battery pack with the WeMos board and all the sensors/motors, as mentioned previously, can lead to some voltage issues. When a lot of sensors or motors are utilized at once, the board may not be able to take the load, and could possible short circuit or spawn errors on the sensors/motor. It may be a better idea to use a different microcontroller board in the future, or different sources of electrical power.
- The WeMos D1 R1 board is a retired board, unlike a lot of the open source hardware microcontrollers that exist in the market today. This causes trouble integrating the board with modern technology, such as cameras, sensors, or motors. It may be a better idea to use a different kind of microcontroller board in the future

- The Arducam 2MP plus mini was an ambitious part of this project, and a hopeful prospect. To integrate the ArduCam to the WeMos board, a new software technology known as Visuino was trialed. The following Figure below illustrates the Visuino code that was utilized to integrate the DHT-11 pin to the WeMos board:

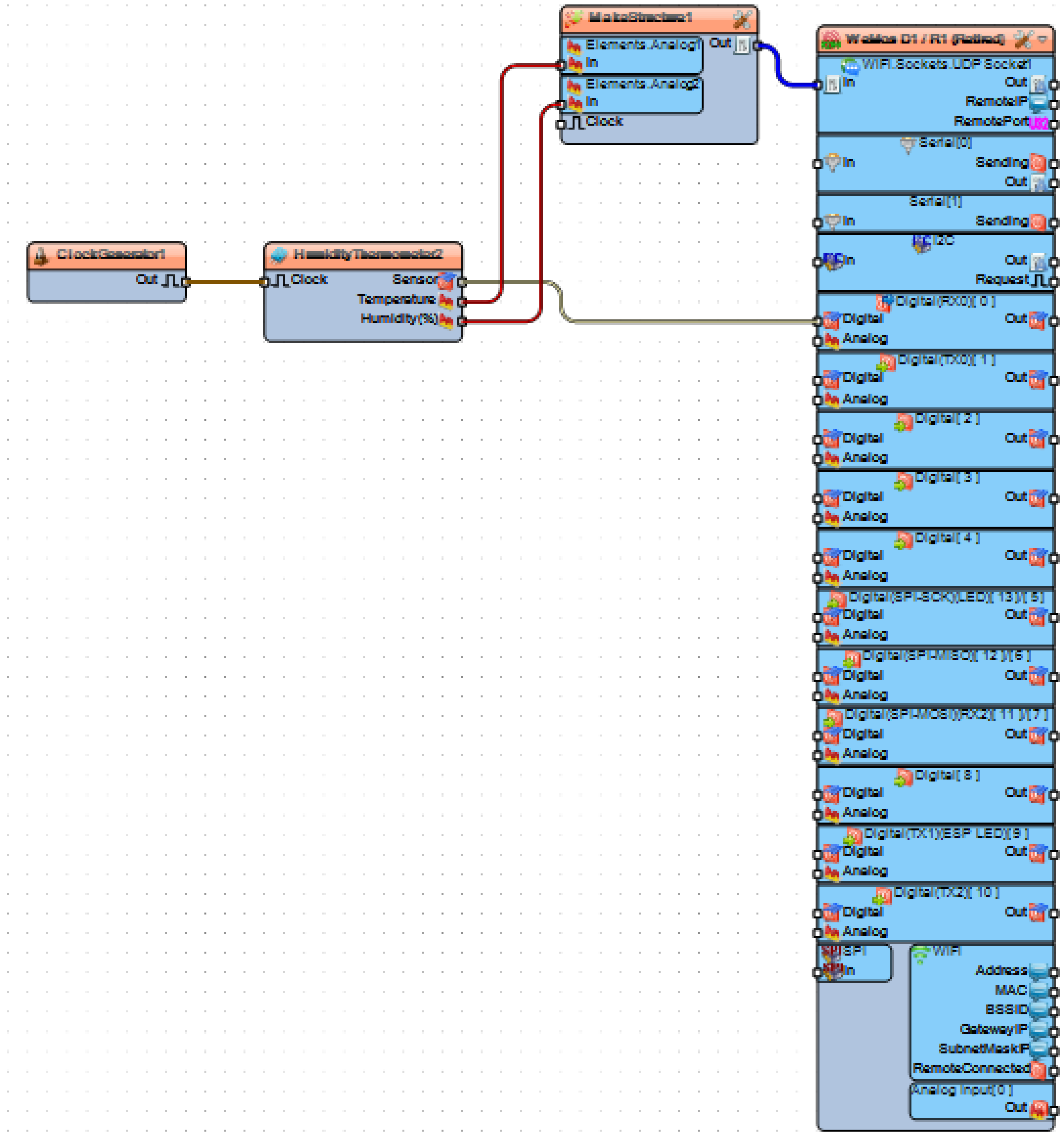


Figure 46

As seen above in Figure 45, Visuino allows one to create a complete circuit of the sensors used with the board, give the parameters to the sensors, and then generate a complete Arduino IDE based code to upload to the board. Unfortunately, Visuino does not include a lot of modern technological hardware as well, and the Arducam happens to fall under that category. Although a code was developed for the Arducam, it did not integrate properly with the WeMos board and led to corrupted images, such as dark pixels and broken frames. The most probable explanation for this conundrum is the fact that the WeMos board is a retired board, and does not integrate properly with a lot of modern technology. In the future, it is important to research the hardware carefully before purchasing it, and making sure that it would integrate and function properly with the microcontroller board in possession. It is also important to utilize Visuino for the benefit it provides, and it can save a lot of time from scripting individual lines of codes for sensors and motors!

7 Conclusion

Autonomous Robots are a key asset in space exploration. They help astronauts better understand the environment they will be exploring. On top of that, with the utilization of many kinds of sensors, scientists will be able to collect useful data that will increase their knowledge on space exploration. At the end of the project, the objectives were conquered. The members were able to create two fully autonomous robots that are capable of obstacle avoidance. The robots are equipped with HC-SR04 ultrasonic sensors, which helps record spatial distance and avoid the many obstacles that gets in its way. Romulus is also equipped with DHT-11 temperature and humidity sensors that records the data in its environment. For calibration purposes, a 16x2 LCD was also used for Remus to physically demonstrate what the robots would be concurrently doing. The robots can successfully navigate within simple structures such as our cardboard constructed tunnel, which serves as a miniature example of the fascinating Martian lava tubes, while collecting and reporting necessary data. With the use of the ESP8266 microprocessor, the members were also able to establish a network where one robot, Romulus, would connect to the other, Remus. This helped the members send valuable data from one robot to another. This will help initiate prospective projects where instead of using two autonomous robots, there can be multiple robots with various sensors; each one for unique functions. Each robot would be able to send data to the scout robot, or to data storing clouds, creating a vast network that leads to minimal loss of data and resources. The aspect of sustainability was also achieved, and is relatively a new theme that the members believe should be conducted for testing phases of projects.

In the future, the members hope to use a LIDAR (Light Detection and Ranging) sensor which sends out infrared waves instead of sonar. This will be even better because scientists will be able to map out the Martian Lava Tubes. This Infrared technology also sends out waves all at once which then maps diverse angles of the field of its vision, instead of specific units. This means that the LIDAR technology consists of a faster refresh rate that allows it to constantly read out data. This will allow the robot to move a wider range of motion and move more fluently. The members also hope to use stronger servos and wheels because the terrain on Mars is rocky and hard. The servos we have currently are not capable of moving in Mars. With all said, this is the start of a New Era in open source robotics and sustainable efforts for extraterrestrial exploration.

8 Literature Cited

A. Dage: *Lunar and Martian Lava Tube Exploration as Part of an Overall Scientific Survey*, A White Paper submitted to the Planetary Sciences Decadal Survey 2013-2022

B. Baid, Akash, and Dipankar Raychaudhuri. “Understanding Channel Selection Dynamics in Dense Wi-Fi Networks.” *IEEE Communications Magazine*, vol. 53, no. 1, 2015, pp. 110–117., doi:10.1109/mcom.2015.7010523.

C. “Overview.” *NASA*, NASA, https://mars.nasa.gov/msl/mission/overview/.

D. *Insights.globalspec.com*, https://insights.globalspec.com/article/7008/robotic-exploration-and-settlement-of-lunar-caves.

E. *LAVA TUBES*, http://earthsci.org/processes/geopro/lava_tubes/lava_tubes.html.

F. “Figure 2f from: Irimia R, Gottschling M (2016) Taxonomic Revision of Rochefortia Sw. (Ehretiaceae, Boraginales). Biodiversity Data Journal 4: e7720. Https://Doi.org/10.3897/BDJ.4.e7720.” doi:10.3897/bdj.4.e7720.figure2f.

G. Folger, Jean. “Why Curiosity Cost $2.5 Billion.” *Investopedia*, Investopedia, 12 Mar. 2019, https://www.investopedia.com/financial-edge/0912/why-curiosity-cost-2.5-billion.aspx.

H. Europlanet Media Centre. "Lava tubes: Hidden sites for future human habitats on the Moon and Mars." ScienceDaily. ScienceDaily, 25 September 2017. <www.sciencedaily.com/releases/2017/09/170925112842.htm>.

I. Heward, Anita. “Lava Tubes as Hidden Sites for Future Human Habitats on Moon and Mars:’ *Phys.org – News and Arficles on Science and Technology*, 29 Sept. 2017, phys.org/news/2017-09-lava-tubes-hidden-sites- future.html

J. “How to Set Up the DHT11 Humidity Sensor on an Arduino.” *Circuit Basics*, 22 June 2018, www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/.

K. Kalita, Himangshu & Morad, Steven & Ravindran, Aaditya & Thangavelautham, Jekan. (2018). Path planning and navigation inside off-world lava tubes and caves. 1311-1318. 10.1109/PLANS.2018.8373521.

L. Kardys, Gary. “Robotic Exploration and Settlement of Lunar Caves.” *Https://Www.globalspec.com/*, 6 Nov. 2017, insights.globalspec.com/article/7008/robotic-exploration-and-settlement-of-lunar-caves.

M. Meystel, A. Autonomous Mobile Robots: Vehicles with Cognitive Control. United States: World Scientific Pub. Co. Teaneck, NJ, 1987.Print.

N. Miles, Joseph. "Design Project Student Overview." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

O. Miles, Joseph. "Rules of Engagement." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

P. Miles, Joseph. "Resources Provided." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

Q. Miles, Joseph. "Connecting to the Arduino Board." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

R. Miles, Joseph. "Downloading and Configuring Arduino." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

S. Miles, Joseph. "Connecting to the Arduino." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

T. Miles, Joseph. "Arduino Pin Assignments." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

U. Miles, Joseph. "Defined Subroutine Input." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

V. Miles, Joseph. "Engineering Design I." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

W. Miles, Joseph. "Updating LED Test." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

X. Miles, Joseph. "Using Flowcharts and Structure Arduino." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

Y. Miles, Joseph. "Predefined Subroutine Functions" Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

Z. Miles, Joseph. "Robot Bumper" Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

AA. Miles, Joseph. "Using Flowcharts and Structure Arduino." Engineering Design I. Stevens Institute of Technology, New Jersey. 28 Jul 2017.

BB. Nedelkovski, Dejan. "DHT11 & DHT22 Sensor Temperature and Humidity Tutorial." *HowToMechatronics*, howtomechatronics.com/tutorials/arduino/dht11-dht22-sensors-temperature-and-humidity-tutorial- using-arduino/.

CC. Sloat, Sarah. "Mars Colonists Could live in Lava Tubes Beneath the Surface." Inverse, 28 Sept. 2017, www.inverse.com/article/36777-mars-moon-human-colony-lava-tubes.

DD. SunFounder. "Lesson 8 LCD1602." *Https://Www.sunfounder.com/*, 16 Feb. 2017, www.sunfounder.com/learn/Super-Kit-V2-0-for-Arduino/lesson-8-lcd1602-super-kit.html.

EE. William Whittaker, "Technologies enabling exploration of skylights, lava tubes and caves," NASA, U.S., Report, no. NNX11AR42G, 2012.

9 Acknowledgments

We would like to thank the following groups and individuals for all their help and efforts in making this project possible. They helped with supporting us with valuable resources including a place to work, funding, and knowledge. The National Aeronautics and Space Administration helped fund the project and gave us the opportunity to work for the government agency. The Goddard Institute for Space Studies provided us with the internship and invited us to various enrichment programs and a generous stipend. On top of that, we would like to thank Goddard Space Flight Center. The Goddard Space Flight Center hosted a poster viewing event in which we were able to participate in. We would like to thank the Stevens Institute of Technology for giving us a place to work on our project. We would also like to thank Dr. Siva Thangam and Dr. Joseph Miles who supervised the whole project and provided us with the necessary material to conduct the project. The Opportunity Network provided us with the internship. Lastly, we would like to thank Ayodamola Okunseinde from The New School who helped us understand how to collaborate the two robots and gave us valuable resources regarding the robot project.

10 Appendix

Romulus Final Code

```
/Includes all the necessary libraries//
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <EthernetServer.h>
#include <EthernetUdp.h>
#include <WemosInit.h>
#include <Servo.h>
#include <ESP8266WiFi.h>
#include<dht.h>

dht DHT; //uses dht temperature and humidity sensor library to create an object called "DHT"


//Defines wheel motor pins//
#define motor1pin D5 // defines pins for right motor wheels
#define motor2pin D9 // defines pins for left motor wheels
#define DHT11_PIN D0 //defines pins for the dht11 sensor

byte ledPin = 2; //creates a variable that integrates the soldered esp8266's led pin 2 on the wemos
board, so that the led blinks everytime the data from dht11 is read
char ssid[] = "Wemos_AP";           // SSID of Accesspoint (Remus)
char pass[] = "Wemos_comm";         // password of your Accesspoint

IPAddress server(192,168,4,15);     // IP address of the Accesspoint
WiFiClient client;                  // ESP8266 is set up as a stationary client, instead of an accesspoint
server, which is Remus

float TemperatureC = 0.00; // creates variable type float to read temperature from dht11
float HumidityC = 0.00;  //creates variable type float to read humidity from dht11

const int frontEchoPin = D10; //initializes variable for front ultrasonic sensor echo receiver pin
const int frontTriggerPin = D1; //initializes variable for front ultrasonic sensor trigger send pin
const int leftEchoPin = D3; //initializes variable for left ultrasonic sensor echo receiver pin
const int leftTriggerPin = D2; //initializes variable for left ultrasonic sensor trigger send pin
const int rightEchoPin = D7; //initializes variable for right ultrasonic sensor echo receiver pin
const int rightTriggerPin = D4; //initializes variable for right ultrasonic sensor trigger send pin


Servo motor1;  // Creates a servo object called "motor1" for right motor wheels
Servo motor2;  // Creates a servo object called "motor2" for left motor wheels


/*Define variables for collecting data with ultrasonic sensors */
volatile float maxFrontDistance = 25.00;
volatile float frontDuration, frontDistanceCm, leftDuration, leftDistanceCm, rightDuration,
rightDistanceCm;
volatile float maxLeftDistance, maxRightDistance = 25.00;

void setup()
{
  // put your setup code here, to run once:
  motor1.attach(motor1pin);  // right motor object attaches to the right motor pin defined earlier
  motor2.attach(motor2pin);  // left motor object attaches to the left motor pin defined earlier
  motor1.write(90); // Turns motor 1 off to calibrate it properly
  motor2.write(90); // Turns motor 2 off to calibrate it properly
  Serial.begin(9600); //Begin Serial Monitor Communication
  /* pinmode function initializes each ultrasonic sensor's
   *  trigger send as an output
   *  echo receive as an input back to the wemos board
   */
  pinMode(frontTriggerPin, OUTPUT);
  pinMode(frontEchoPin, INPUT);
  pinMode(leftTriggerPin, OUTPUT);
  pinMode(leftEchoPin, INPUT);
  pinMode(rightTriggerPin, OUTPUT);
  pinMode(rightEchoPin, INPUT);

    WiFi.mode(WIFI_STA); //sets up wifi mode as stationary for esp8266
  WiFi.begin(ssid, pass);           // connects to the WiFi accesspoint
  Serial.println();                 //new line on serial monitor
  Serial.println("Connection to the AP");

  /* a while loop that iterates wifi connection function
   */
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println(); //new line on serial monitor
  Serial.println("Connected");
  Serial.println("station_bare_01.ino");
  Serial.print("LocalIP:"); Serial.println(WiFi.localIP()); //prints out local IP address of the accesspoint
  Serial.println("MAC:" + WiFi.macAddress()); //prints out the mac computer's IP address
  Serial.print("Gateway:"); Serial.println(WiFi.gatewayIP()); //prints out gateway IP address
  Serial.print("AP MAC:"); Serial.println(WiFi.BSSIDstr());   //prints out accesspoint mac computer's
IP address
  pinMode(ledPin, OUTPUT); //declares the variable ledpin as an output to blink when data is sent
}


void loop() // All code in loop runs continuously after setup
 {
  int chk = DHT.read11(DHT11_PIN); //sets up a variable that uses DHT object and read dht11
function to read from the assigned dht11 pin

 TemperatureC = DHT.temperature; //declares the variable to be of the value that returns temperature
in celsius
  HumidityC = DHT.humidity;      //declares the variable to be of the value that returns humidity in
relative percentage



  client.connect(server, 80); //connects client to the server and the default port 80 as soldered onto the
esp8266 board
  digitalWrite(ledPin, LOW); //the led pin is not lit up
  Serial.println("********************************");
  Serial.print("Byte sent Remus: ");
  Serial.println(client.print(TemperatureC)); //prints out temperature onto the server's serial monitor
  Serial.println(client.print("  Celsius and %"));
  Serial.println(client.println(HumidityC)); //prints out humidity onto the server's serial monitor
  String answer = client.readStringUntil('\r'); //creates a string that allows the the client ot read
everything from the server
  Serial.println("Data to Remus: " + answer); //prints out the size of data sent ot remus
  client.flush(); //clears the client, to loop again
  digitalWrite(ledPin, HIGH); //led lits up when data is sent, causing blinking in synchrony with Remus
  client.stop(); //client signals are put to a stop
  delay(2000); //delays for 2 seconds

uint32_t period = 1 * 60000L;       //intiation of a for loop to make the robot autonomous for 1 minute

for( uint32_t tStart = millis();  (millis()-tStart) < period;  ){
  // checks front distance
  checkFrontDistance();
  if (frontDistanceCm < maxFrontDistance) {
    checkLeftDistance();
    delay(20);
    checkRightDistance();
    delay(20);
    if (leftDistanceCm < rightDistanceCm)
      moveRight();
    else if (leftDistanceCm > rightDistanceCm) {
      moveLeft();
    }
  }
  else {
    moveForward();
  }



   // checks left distance
  checkLeftDistance();
  if (leftDistanceCm < maxLeftDistance) {
    checkLeftDistance();
    delay(20);
    checkRightDistance();
    delay(20);
    if (leftDistanceCm > rightDistanceCm)
      moveForward();
    else if (leftDistanceCm < rightDistanceCm) {
      moveRight();
    }
  }

  // checks right distance
  checkRightDistance();
  if (rightDistanceCm < maxRightDistance) {
    checkRightDistance();
    delay(20);
    checkLeftDistance();
    delay(20);
    if (rightDistanceCm > leftDistanceCm)
      moveForward();
    else if (rightDistanceCm < leftDistanceCm) {
      moveLeft();
    }
  }
}

moveStop();
delay(5000); //after moving for 1 minute, stop the robot for 5 seconds

}




// Subroutine listings !!! subroutines clarify variables that are prototyped early on//


void checkFrontDistance() {
  digitalWrite(frontTriggerPin, LOW);  //trigger pin is set low to clear out any bytes of data
  delayMicroseconds(4);  //delay for 4 microseconds
  digitalWrite(frontTriggerPin, HIGH);  //trigger pin is set high to send out the ultrasonic rays
  delayMicroseconds(10); //the ultrasonic rays are sent out for 10 seconds
  digitalWrite(frontTriggerPin, LOW); //trigger pin is turned back off
  frontDuration = pulseIn(frontEchoPin, HIGH);  //echo receiver pin receives the ultrasonic sensors
back and calculates the time it takes to receive it back
  frontDistanceCm = frontDuration * 10 / 292 / 2;  //that time is converted to a distance in cm
}
void checkLeftDistance() {
  digitalWrite(leftTriggerPin, LOW);
  delayMicroseconds(4);
  digitalWrite(leftTriggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(leftTriggerPin, LOW);
  leftDuration = pulseIn(leftEchoPin, HIGH);
  leftDistanceCm = leftDuration * 10 / 292 / 2;
}
void checkRightDistance() {
  digitalWrite(rightTriggerPin, LOW);
  delayMicroseconds(4);
  digitalWrite(rightTriggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(rightTriggerPin, LOW);
  rightDuration = pulseIn(rightEchoPin, HIGH);
  rightDistanceCm = rightDuration * 10 / 292 / 2;
}


void moveForward() {

motor1.write(85);   //right motor full speed forward
motor2.write(101); // left motor full speed forward
}

void moveStop() {

motor1.write(90);   //right motor full stop
motor2.write(90); // left motor full stop
}

void moveLeft() {

  motor1.write(80);  //right motor forward for a left turn
  motor2.write(90);  //left motor stopped
}
void moveRight() {

  motor1.write(90);  //right motor stopped
  motor2.write(105); //left motor forward for a right turn
}
```

Remus Final Code

```
//Includes all the necessary libraries//
#include <NewPing.h>
#include <Servo.h>
#include <LiquidCrystal.h>
#include <ESP8266WiFi.h>

WiFiServer server(80); //configures wifi port as default soldered port 80 on esp8266
IPAddress IP(192,168,4,15); //configures default esp8266 IP address
IPAddress mask = (255, 255, 255, 0); //configures mask IP address of esp8266

byte ledPin = 2; //creates variable byte that configures default esp8266 led as pin 2

#define TRIG_PIN D4 //defines ultrasonic sensor trigger send pin
#define ECHO_PIN D7 //defines ultrasonic echo receive pin
#define MAX_DISTANCE 200 //defines variable for max distance read by ultrasonic sensor
NewPing sonar(TRIG_PIN, ECHO_PIN, MAX_DISTANCE); //newping library uses sonar function
which configures the trigger, echo, and max distance in order to set up the ultrasonic sensor

#define motor1pin D5 // defines pins for right motor wheels
#define motor2pin D9 // defines pins for left motor wheels

Servo myservo; //creates servo object for the servo bracket containing ultrasonic sensor
Servo motor1;  // Creates a servo object called "motor1" for right motor wheels
Servo motor2;  // Creates a servo object called "motor2" for left motor wheels

boolean goesForward=false;
int distance = 100; // creates variable distance that returns 100
int speedSet = 0;

LiquidCrystal lcd(D12, D11, D0, D4, D3, D2); //liquid crystal lcd library configures lcd pins

void setup() {

  Serial.begin(9600); //initiates serial monitor communication
  WiFi.mode(WIFI_AP); //uses esp8266 wifi mode as accesspoint
  WiFi.softAP("Wemos_AP", "Wemos_comm"); //uses soft accesspoint method using ssid and
password
  WiFi.softAPConfig(IP, IP, mask); //configures soft accesspoint with IP and mask addresseS
  server.begin(); //begins esp8266 server
  pinMode(ledPin, OUTPUT); //sets the default esp8266 blink as an output
  Serial.println(); //new line in serial monitor
  Serial.println("accesspoint_bare_01.ino");
  Serial.println("Server started.");
  Serial.print("IP: ");     Serial.println(WiFi.softAPIP()); //print out soft accesspoint IP address
  Serial.print("MAC:");     Serial.println(WiFi.softAPmacAddress()); //print out soft accesspoint mac
laptop address

  lcd.begin(16, 2); //begins lcd calibration and configures it as a 16 x 2 wide and long lcd screen

  motor1.attach(motor1pin);  //right motor object attaches to the right motor pin defined earlier
  motor2.attach(motor2pin);  //left motor object attaches to the left motor pin defined earlier
  motor1.write(90); // Turns motor 1 off to calibrate it properly
  motor2.write(90); // Turns motor 2 off to calibrate it properly

  myservo.attach(D6); //attaches myservo servo bracket object to the pin
  distance = readPing(); //declares distance variable as readping function prototype
  delay(100); //delay for 100 microseconds
  distance = readPing(); //keep looping in order to get sensible readings
  delay(100);
  distance = readPing();
  delay(100);
  distance = readPing();
  delay(100);
}

void loop() {
  lcd.print("Hello world!");
  delay(3000);
  lcd.clear();
  delay(500);

  lcd.print("I am Remus");
  delay(2000);
  lcd.clear();
  delay(500);

  lcd.print("Here's my partner");
  delay(3000);
  lcd.clear();
  delay(500);

  lcd.print("Romulus!");
  delay(2000);
  lcd.clear();
  delay(500);

  lcd.print("Reading Data...");
  delay(3000);
  lcd.clear();
  delay(500);



 WiFiClient client = server.available(); //client's existence implies server remus is also available for
communication
  if (!client) {return;} //if client is not available, then stop
  digitalWrite(ledPin, LOW); //esp8266 stops blinking
  String request = client.readStringUntil('\r'); //string variable request reads data from romulus
  Serial.println("********************************");
  Serial.println("Temperature and humidity from Romulus: " + request); //prints out data onto the serial
monitor
  client.flush(); //client refreshes
  Serial.print("Byte sent to Romulus: ");
  Serial.println(client.println(request + "ca" + "\r")); //prints out to client the size of data sent
  digitalWrite(ledPin, HIGH); //esp8266 lights up everytime data is sent to create blinking effect


 uint32_t period = 1 * 60000L;       //intiation of a for loop to make the robot autonomous for 1 minute
```

```
for( uint32_t tStart = millis();  (millis()-tStart) < period;  ){
 int distanceR = 0; //sets up right distance variable that returns 0
 int distanceL =  0; //sets up left distance variablet that returns 0
 delay(20); //delays 20 microseconds

 if(distance<=15)
 {
  moveStop();
  delay(100);
  moveBackward();
  delay(200);
  moveStop();
  delay(200);
  distanceR = lookRight();
  delay(200);
  distanceL = lookLeft();
  delay(200);
//when front distance is less than 15, move back, stop, and then check left and right distance to decide
best possible outcome

  if(distanceR > distanceL)
  {
    turnRight();
    moveStop();
  }
  else
  {
    turnLeft();
    moveStop();
  }
 }else
 {
  moveForward();
 }
 distance = readPing();
}
moveStop(); //after moving around autonomously for 1 minute, stop and delay for 5 seconds
delay(5000);
}

int lookRight() //protyped function to make the servo look right and read a distance
{
    myservo.write(85); //turn robot's neck right
delay(465);
stoprotation(); //stop!
delay(20);
int distanceR = readPing(); //sets up distance to the right as a function readping
    delay(100);
   myservo.write(100); //turn left to center it back
```

```
delay(385);
stoprotation(); //stop!
delay(20);
    return distanceR; //returns distance to the right from this function
}

int lookLeft() //prototyped function to make the servo look left and read a distance
{
    myservo.write(100); //turn robot's neck to the left
delay(460);
stoprotation(); //stop!
delay(20);
int distanceL = readPing(); //sets up distance to the left as a function readping
 delay(100);
 myservo.write(85); //turn right to center it back
delay(465);
stoprotation(); //stop!
delay(20);
    return distanceL; //returns distance to the left from this function
    delay(100);
}

int readPing() //readping function prototype that results in ultrasonic sensor readings
{
  delay(20);
  int cm = sonar.ping_cm(); //variable cm returns newping function sonar.ping in centimeter values that
reads actual distances
  if(cm==0) //if cm is so close that it is 0, then return value 250
  {
    cm = 250;
  }
  return cm; //return the cm value from the function readping
}

void moveStop() //stop the motor wheels
{
  motor1.write(90);
  motor2.write(90);
  }

void moveForward() //push motor wheels forward
{
motor1.write(85);
motor2.write(99);
}

void moveBackward() //push motor wheels backward
{
  motor1.write(95);
motor2.write(81);
```

```
}

void turnRight() //turn right and forward to a certain distance to avoid basic obstacles
{
 motor1.write(90);
motor2.write(105);
delay(300);
motor1.write(100);
motor2.write(100);
}

void turnLeft() //turns left and forward to a certain distance to avoid basic obstacles
{
  motor1.write(80);
  motor2.write(90);
delay(300);
motor1.write(100);
motor2.write(100);
}


void stoprotation() //stop the neck servo bracket rotating
{
  myservo.write(90);
}
```