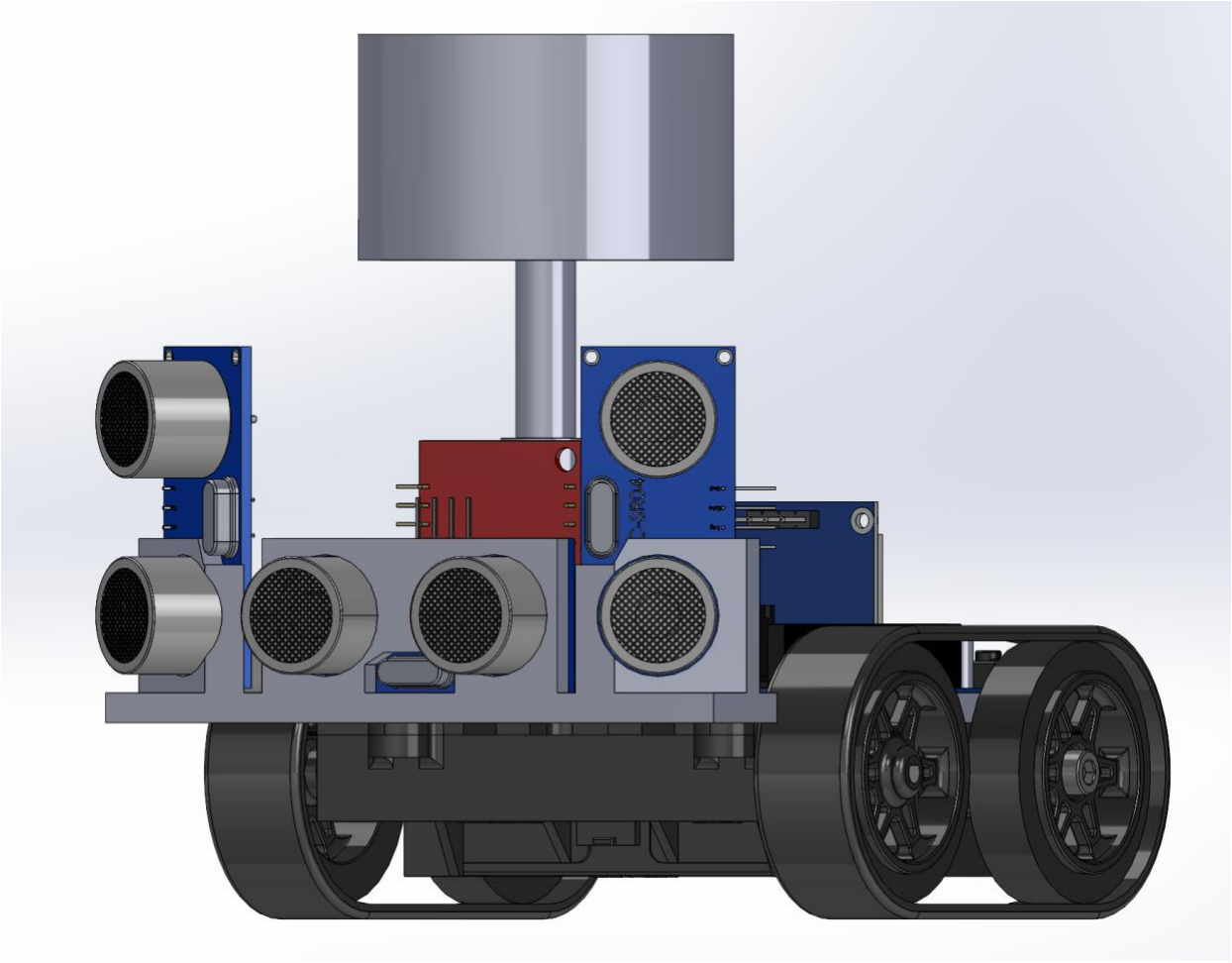
Autonomous Navigation Robot



I pledge my honor that I have abided by the Stevens Honor System

Drew Crossman(Civil Engineering) and
Om Gaikwad(Mechanical Engineering)

ENGR 112-W: Group 1

Executive Summary

The group was tasked with the designing of an Autonomous-Driving Robot that would be programmed to navigate the Stevens campus on tours and guide itself to specified destinations (targets) on the map (arena) using x and y coordinates provided via the Stevens MQTT network. The robot was to be tested via an arena with marked off locations, or targets, of which the robot had to locate and navigate itself to, using the LiDAR sensor system that was integrated into the arena to obtain the location of itself and each target.The robot had 3 minutes to successfully reach all 4 targets, along with having to avoid the obstacles that were laid out across the arena.

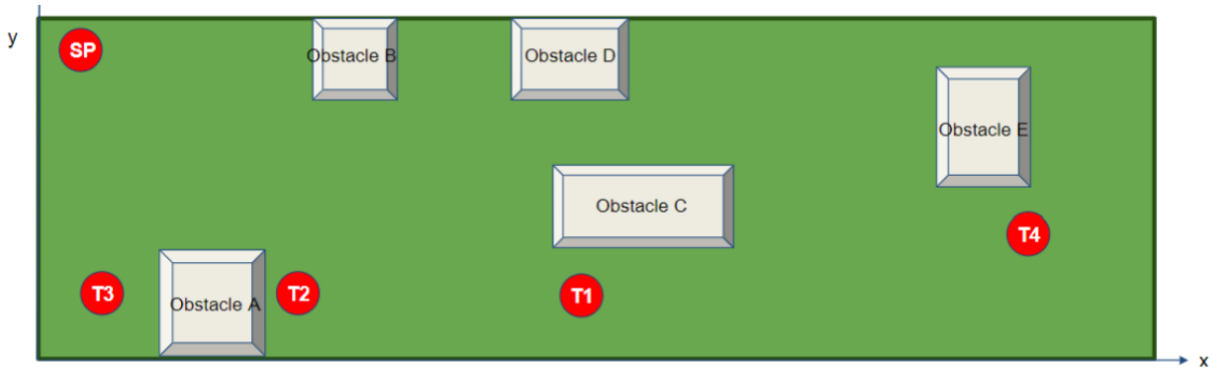


Figure 1.1: Overview of arena

The team was able to satisfy 100% of the client's demands, reaching all four targets in order. To construct the body of the robot, the team used the CAD software, SolidWorks, to create a 3D design of the robot base. Arduino IDE was used for the software development of the robot programming. Along with the software, the robot utilized a Zumo Chassis, 3 Ultrasonic Sensors, an OLED Display, a Feetech 2CH Servo motor controller, a D1R1 WeMos board, a printed circuit board (PCB), a battery pack, and wiring in order to create the circuits and function. The project was a hands-on activity that allowed the team to practice their skills in producing a desired product using 3D designing software, more advanced code development, and circuit creation.

Table of Contents

Introduction

1.1 Purpose (Crossman)

The purpose of this report is to transcribe the thinking and design process that went behind the production of the Autonomous Robot. As well as describe the process, it will overview the robot itself and go over every component involved to make the robot run. The system, mechanical, software, and electrical design and components will all be discussed, as well as an evaluation of the success of the project and the effectiveness of the team as a whole. Finally, future recommendations and possible improvements will be proposed in order to self reflect and prepare for any research down the road.

1.2 Project Objectives (Crossman)

The project required the team to design, fabricate, test, and refine an autonomous robot equipped with a motor subsystem, sensor suite including up to a set of 3 ultrasonic sensors, and mechanisms for over-the-air access to LiDAR determined position information. The team was required to develop a system design that included mechanical, electrical, and software elements integrated to effectively complete the robot's mission. That mission consisted of navigating a Test Arena to reach a series of strategically placed targets.

This project was used as an exercise to give the team a hands-on experience of the engineering design process. The objective of the exercise was to give the team practice in developing skills involving teamwork, professional collaboration amongst partners, job and work distribution, as well as oral and written communication skills. Along with supply the team with

practice in those skills, the project taught the team various technical applications, including the following:

1. Utilize Solidworks to design a 3D robot body.
2. Develop and program logic to navigate toward given coordinates while avoiding obstacles.
3. Design and create circuits between sensors and controllers in order to create a functioning system.
4. Solder and splice wiring to maximize space efficiency in circuit design.
5. Design, test and operate systems with ultrasonic sensors.
6. Program a microcontroller to periodically poll sensors, control motors, and acquire data.
7. Control and calibrate actuator movement using PWM signaling.
8. Display variables on the OLED Display panel using serial clock and serial data.
9. Perform data networking and communicating using MQTT protocol.
10. Develop and program logic to navigate toward given coordinates while avoiding obstacles.
11. Debug program parameters to enhance system operation via experimentation and logical thinking.

Carrying out the project using both the communicational and technical skills described above allowed the team to learn how to accurately and effectively proceed during the engineering design process.

1.3 Project Specifications (Crossman and Om)

The client tasked the team with designing an Autonomous-Driving Robot that would be programmed to navigate the Stevens campus on tours and guide itself to specified destinations (targets) on the map (See Fig 2.2) using x and y coordinates provided via the Stevens MQTT network. Along with the given task, the team had to adhere to a given set of specifications and desires the client requested.

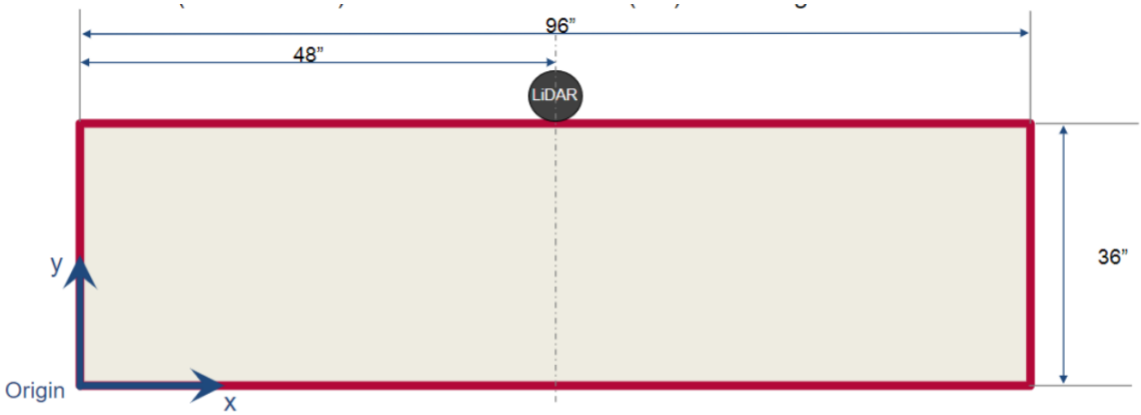


Figure 2.2: Layout of Arena and position of LiDar sensors (units are in inches)

The team was given 6 weeks, including in Lab time and outside hours, to update the Robot body base, 3D print, integrate a working circuit and system, develop the appropriate algorithm, and test the robot before the final evaluation of the robot during the last week. The robot base had to house the given sensors and all electrical components of the system, without the use of any tape or adhesive material. The mandatory sensors and electrical components included at least 3 Ultrasonic Sensors, an OLED Digital Display, a Feetech 2CH Servo motor controller, servo motors, a D1R1 WeMos board, a printed circuit board (PCB), and a battery pack. The client prohibited the use of a breadboard to create a circuit hence the use of the PCB. The client wanted the entire base to be printed in less than 4 hours, however, if the base were to be printed in the Lab during Lab hours, it had to be printed in 2 hours and 50 minutes, as the client did not want a team to be taking printer space up for the next class that came in. In

the new design, the team had to implement an area to install the GPS float, which would allow the LiDAR sensor to detect the robot's location, as well as make sure the location of the Ultrasonic Sensors on the base allowed for the detection of the various obstacles laid out in the Arena. In order to keep the robot organized and easy to replicate, the client desired that any wiring used to create the circuit be fitted with a color coding system. In regards to the software, the robot had to be programmed to navigate the Arena fully autonomously and avoid all obstacles as it travels to the designated targets. Inside of the code, the client desired there to be plenty of notes to explain each part of the code and make replication easy and possible. The entire robot had to be completed and ready for the final evaluation, which was given on the 14th week.

1.4 Mission (Crossman)

The mission of the group was to effectively fulfill the request and requirements of the client. The group set out to produce a polished final product that met the desires of the client while exercising principle problem solving techniques. The hope was to learn and grow in our capabilities of teamwork, brainstorming, collaboration, and time management, all while being successful with the given task. The mission of the project was to develop a fully Autonomous-Driving Robot that could navigate around an Arena, reaching targets that were laid out on the course, all while being able to avoid obstacles on its own. This would serve as a model of a possible real-world vehicle that can guide visitors on tours of the Steven's campus.

Team Introduction - Roles

TEAM 1	System and Mechanical Design	Software Development and Coding	Electrical and Circuit Design	Report Development	Oral Presentation
Team Member	Drew Crossman		Drew Crossman	Drew Crossman	Drew Crossman
	Om Gaikwad	Om Gaikwad		Om Gaikwad	Om Gaikwad

2. Requirements and Constraints, Crossman

As discussed briefly above, the team was given specific requirements and constraints to abide by by the client. In order to successfully fulfill the desires of the client, the team had to follow each of these requirements, and they are as follows:

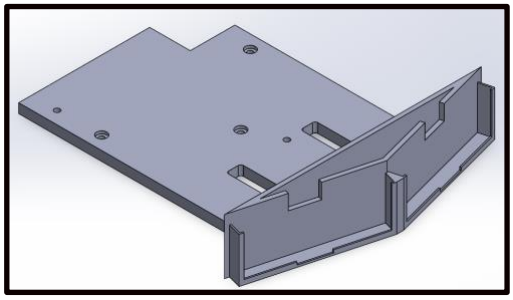
1. The Robot must be fully autonomous.
2. A total of 6 weeks, including in Lab time and outside hours, to update the Robot body base, 3D print, integrate a working circuit and system, develop the appropriate algorithm, and test the robot before the final evaluation of the robot during the last week.
3. Overall print time of 4 hours, however, in Lab printing is limited to 2 hours and 50 minutes.
4. The Robot body and sensor placement must fit in the arena and be able to fit through the obstacles while navigating.
5. The robot base has to house the given sensors and all electrical components of the system, without the use of any tape or adhesive material.

6. System must utilize at least 3 Ultrasonic Sensors, an OLED Digital Display, a Feetech 2CH Servo motor controller, servo motors, a D1R1 WeMos board, and a battery pack.
7. The use of a breadboard is prohibited, therefore, if needed, the use of a PCB and soldering is allowed.
8. Final base design must include an area to fit the GPS float to detect the robots location with the LiDAR sensor.
9. Body and circuit should be organized and neat in order to make any needed adjustments.
10. Wiring must follow a color coding system that makes it easy to identify where the wires attach to and what specific wires do.
11. Soldering is not mandatory, however, it is strongly encouraged to keep design space efficient and optimize the circuit.
12. The use of tape, glue, epoxy, or any other adhesive material (other than the supplied screws and bolts) to hold parts in place is prohibited.
13. The software must utilize an obstacle avoidance program in order to navigate through the Arena without crashing into the walls or obstacles.
14. The coding must have clear notes and explanations inside of it in order for it to be easily understood by others and for ease of making needed modifications.

All of these requirements and constraints needed to have been met in order to completely satisfy the request and desire of the clientele.

4. Conceptual Design (Crossman)

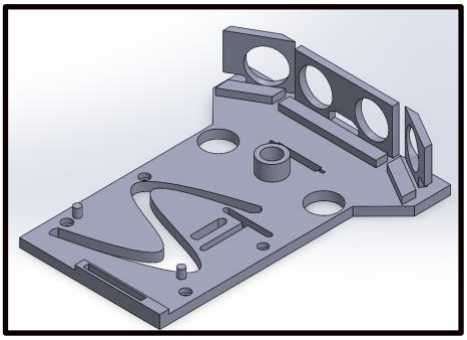
The team spent several weeks developing and improving different designs of the robot base that was to house the sensors and electrical system. The team was finally able to create and produce the final robot base that was to be used after rigorous prototyping and testing of different design iterations. The following will discuss and describe that process and the different design iterations that were created before the final design.

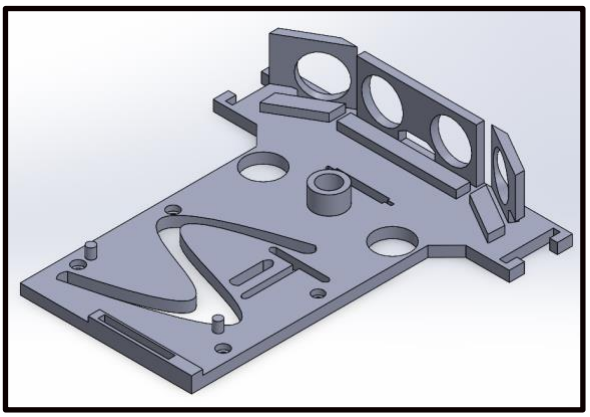


The first design iteration that was created and considered was the base that was used during the Left-handed loop test. The print time of this iteration was roughly 2 hours and 30 minutes. It was a simple design that involved 2 angled sensor brackets in the front of the body to house 2 Ultrasonic Sensors, 2 holes in the middle of the front of the base to allow for access to the motor controller wires, and screw holes to attach the base to the Zumma Chassis, and the WeMos board to the base. This design had to be discarded and updated because physically, it could not fit the needed sensors for the final evaluation and from performance point of view, the placement of the sensors

resulted in blind spots to the immediate right and left of the robot which led to occasional collisions.

The second design iteration that was created was the first iteration that was the basic layout of what the final base would be like. It involved a sleeker and lighter design, being 4mm thick, rather than 4.5mm thick in the previous design, and had a shorter print time of 2 hours and 10 minutes. It entailed 3 Ultrasonic Sensor brackets in the front, one pointing forward and the other 2 at slight angles outward. Behind each bracket was a backing that would secure the Ultrasonic Sensor in place. There was a slit in the middle of the front of the base (behind the brackets) that would house the motor controller, and behind that, a cylindrical pipe to secure the GPS float. On either side of the area for the GPS float were a hole, supplying access to the wires of the motors. In the center of the base was engraved SIT, serving both aesthetics purposes, as well as allowing for the wires of the battery pack to be stored under where the WeMos board was to be installed. The WeMos board sat on top of the SIT logo, being fascended by 2 prongs that were on either side of the SIT logo. In the very back was a slit to house the OLED Display. While this design was more efficient than the previous design and met all project requirements, there were many flaws in it. The first flaw was that the backings of the Ultrasonic sensor brackets were too close to the brackets themselves that the Ultrasonic Sensor was not able to be placed into the base. The second flaw was that the pipe to secure the GPS float was too short and didn't allow it to reach the height of the LiDAR sensor's view. This meant that the Robot's location was not able to be detected. Another flaw was that the prongs that held the WeMos board in place were too short, causing the board to become unsecured constantly. These flaws made it so that this iteration was unusable and, therefore, was discarded.

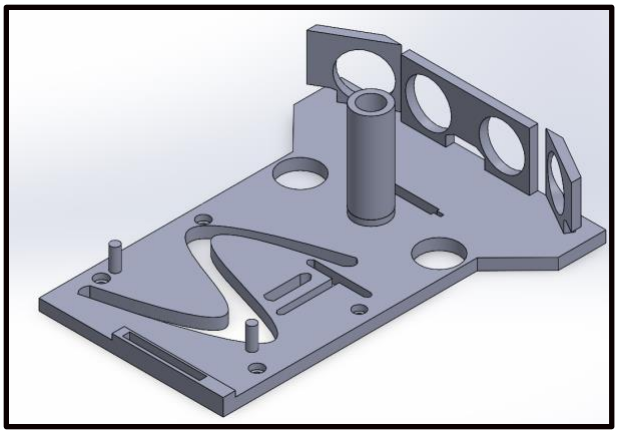




The third design iteration that was created was very similar to the second one, however, it had an extra component to it. In the front, on either side of the angled brackets was another bracket to house an Ultrasonic Sensor. This bracket, however, would secure a sensor vertically, pointing perpendicular to the robot itself. This was added to allow for multiple options of sensor location and angles down the road when developing the code and testing in the Arena. It should be noted that this iteration was designed alongside the second iteration. As a result, it had the same flaws as the second iteration and was discarded accordingly.

The fourth and last iteration that was created before creating the final boase design followed the same overall concept and design of the previous 2 iterations, however, many improvements and optimizations were made. First off, the thickness of the entire base was lowered from 4mm to 3mm. Secondly, the backings behind the brackets were removed to allow the Ultrasonic Sensors to be easily implanted into their designated area. Along with the backings being removed, the side brackets were also removed. This was done as the code was more developed while designing this iteration and it was realized that those 90 degree brackets were no longer needed. Along with that, both the GPS float pipe and the prongs to secure the WeMos board were lengthened to allow the GPS float to reach the needed height, and to allow

the WeMos board to be secured to the base without excessive movement. The overall print time of this base was decreased from the previous designs to about 2 hours.



Final Design Description

5.1 System Description and Architecture (Gaikwad)

Ultrasonic Sensors

In most of the design iterations for the project, 3 HC-SR04 ultrasonic sensors were used except for the iteration #1 where only two ultrasonic sensors were used. The ultrasonic sensors function as explained below:

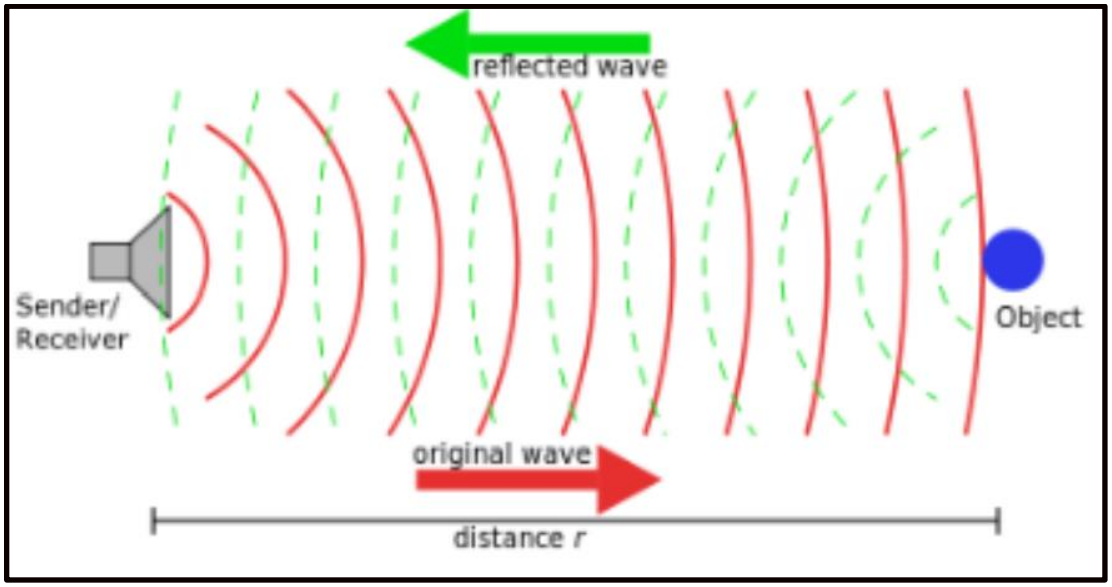


Figure 5.1.1: Functioning of Ultrasonic Sensors

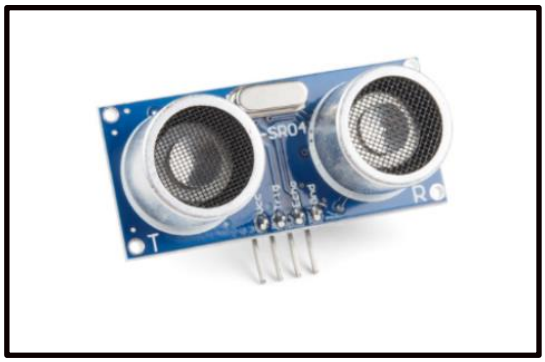


Figure 5.1.2: Ultrasonic Sensors HC-SR04

The pins responsible for the obstacle and distance detection are the Trig and Echo pings. The Trig pin sends out a sonic pulse while the Echo pin detects the signal reflected from the object if there is any. There is an arduino library file used called <ultrasonic.h> that does the measurements and gives the distance as the output. To measure the distance an object is from the sensor, the time taken for the sonic wave to bounce back is recorded and it calculated with the below formula:

$r = c / 2T$ (r = distance from object c = speed of sonic wave in air T = Time taken for sonic wave to bounce back)

Servo Motors

Two DC servo motors were used to power the front wheels of the robot. Servo Motor Controller (FT-SMC-4CH) was also used to control the speed and direction of rotation of the motors. By using the arduino library file <ESP8266WiFi.h>, Pulse Width Modulated (PWM) signals are sent to the controller to control motor spin speed and direction. The width (time difference) between pulses determines how the motor responds. An additional arduino library file <Servo.h>, is used to control the motor speeds based on the width of the pulses.

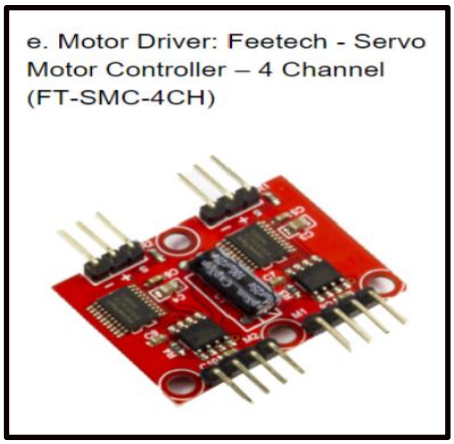


Figure 5.1.3: Servo Motor

Figure 5.1.4: Motor Controller

LiDar

The team used a baseline code provided to connect the WeMos board to the MQTT network for the purpose of receiving the location data of the robot's in terms of its x and y coordinates. The system architecture follows as below:

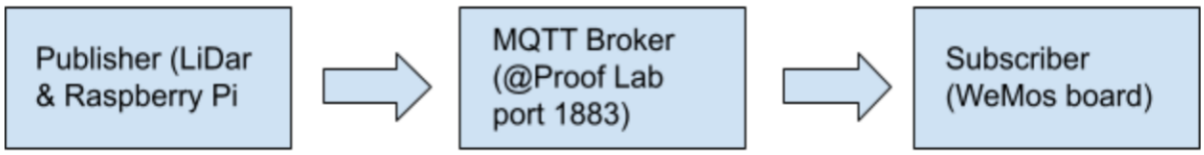


Figure 6.1.5: Subscribing LiDar

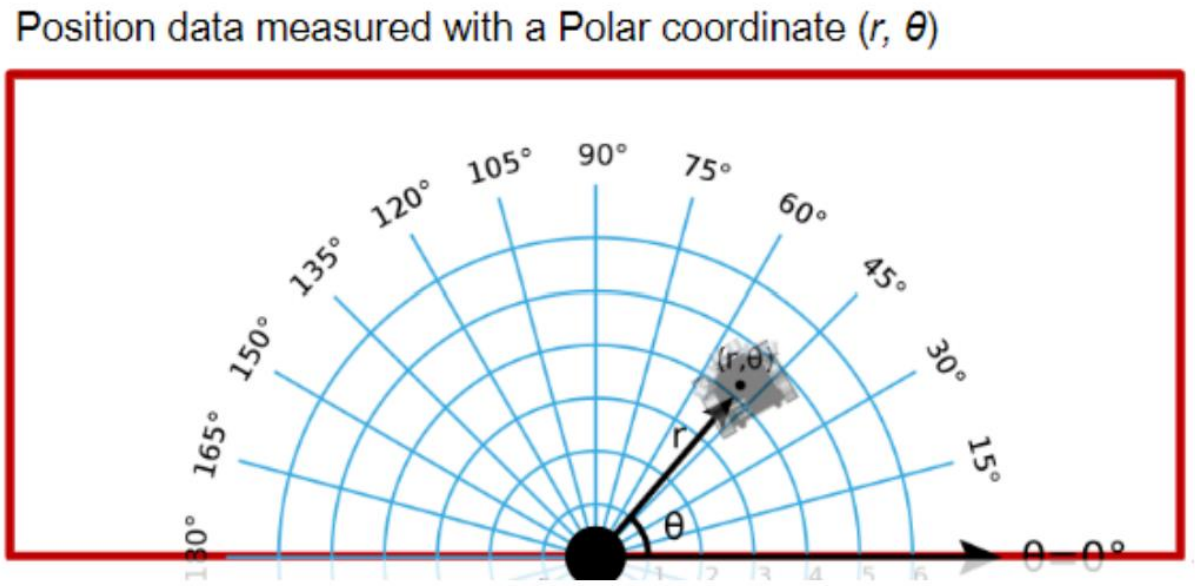


Fig 6.1.6 LiDar Polar Coordinates Positioning

A 2D laser scanner (YDLiDAR G4 LiDAR Laser Rangefinder) is used to find the robot's location by detecting the GPS float and calculating the distance using reflection of the lasers from the float. The LiDar sensor first obtains the robot's position in a Polar Coordinate System. The polar coordinates are converted into Cartesian coordinates before publishing the data to the MQTT broker. The cartesian coordinates are obtained with the following coordinates:

$x=r\cos(\Theta)$

$y=r\sin(\Theta)$

Where r is the distance of the robot to the LiDar sensor and Θ is the angle between r and the positive x-axis.

OLED Display Screen

I^2C Serial Communication Protocol is used to relay data to the OLED display from the WeMos board. The messages are first broken into two types of frames:

1. Address Frame: The address frame indicates the peripheral to which the message is being sent whereas the data-frames are 8-bit data messages passed from controller to the peripheral or vice versa.
2. Data Frame: Data is placed on the SDA line after SCL goes low, and is sampled after the SCL line goes high. The “SSD1306.h” arduino library file was used for this process.

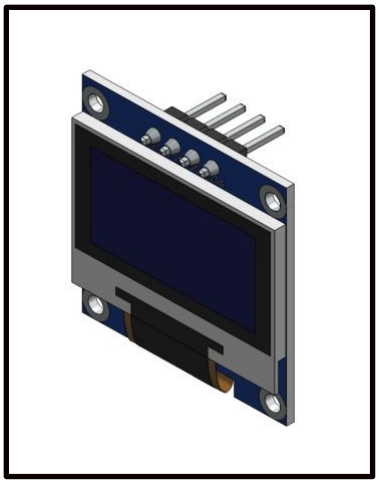


Figure 5.1.7 OLED Display

5.2 Mechanical Design, (Gaikwad)

The main goal when designing the robot was to keep the overall layout compact with minimal weight which would put less stress on the motors.Taking care of the center of mass was essential to ensure a smooth movement of the robot along the tracks.

The positioning of the sensors was a very important step in the design. The team decided on the layout with three sensors with one pointing in the front and the other two about 30 degrees to the right and left of the one in the middle (Fig 5.). Brackets were designed to hold the sensors in place (Fig 5.2)

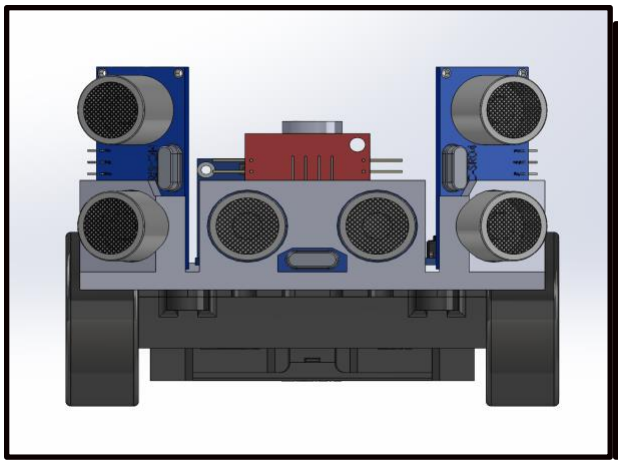


Fig 5.2.1 Positioning of the sensors

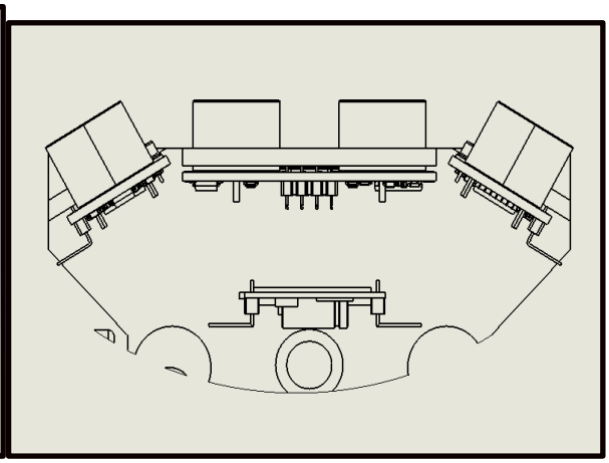


Fig 5.2.2 Top view of the sensor brackets

A cylindrical bracket was designed right above the beginning of the zumo chassis. The WeMos Board was designed to be placed on the back of the robot behind the GPS float. for the GPS float. This made sure that the center of gravity of the whole system was well positioned to avoid any structural imbalances in the robot. A rectangular indentation was designed at the back end of the design for the OLED Display.

Fig. 5.2.3 shows the interference detection of the design; some parts like the OLED display and the motor controller had a slight difference in their physical dimensions and dimensions in theirSolidWorks part files. Hence, the team had to take physical dimensions with the help of a vernier caliper to make sure that the actual parts would fit in the design that would further be printed. The following figure contains the picture of the stl file of the design part and its print material and time.

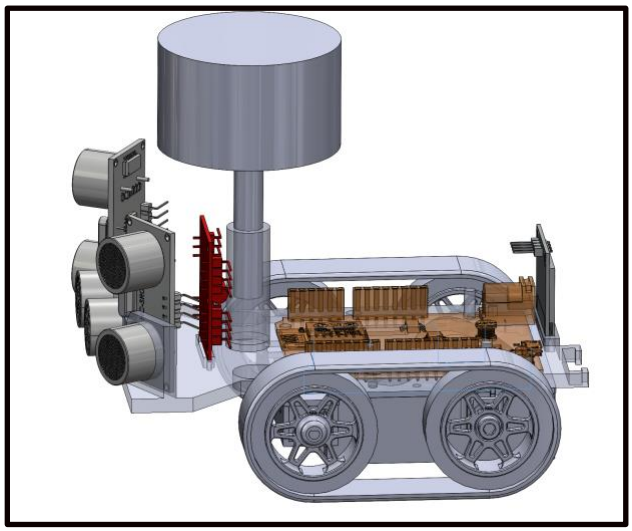
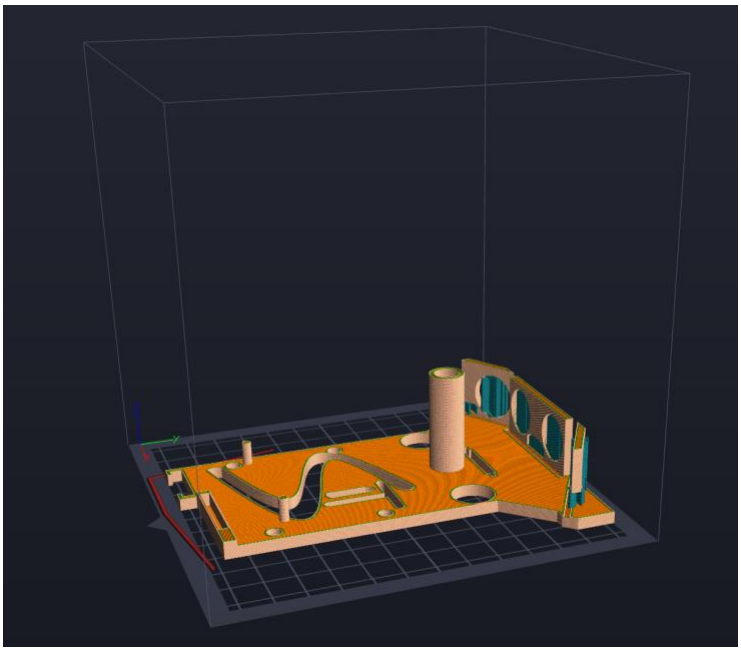


Fig. 5.2.3 Interference detection



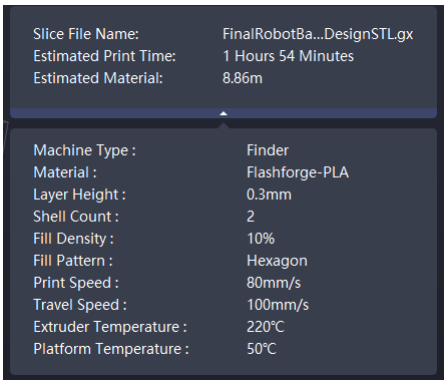


Fig 5.2.5 Print Properties (Flash Print 5)

Fig. 5.2.4 Picture of the STL print slice preview

10% hexagonal infill was used to optimize the print time and it did not cause any issues during assembly as the parts needed to be attached were not heavy and the orientation of the different parts already ensured that that center of mass would be well positioned.

5.3 Electrical and Circuit Design (Crossman)

The Robot's system was run by an electrical circuit that worked by transferring the readings from the Ultrasonic Sensors to the WeMos board, which was programed then to send signals to the servo motors (which controlled the Robot's movements) of what to do, i.e. go

forward, turn, or stop. In order to do this, the Robot's circuit consisted of 3 Ultrasonic Sensors, an OLED Digital Display, a Feetech 2CH Servo motor controller, servo motors, a D1R1 WeMos board, a PCB, and a battery pack. The circuit was constructed as follows:

- The battery pack was connected directly to the D1R1 Wemos board to supply power to the entire Robot.
- The PCB was placed on top of and connected to (utilizing solder) the WeMos board.
- Connected to the PCB was every individual wire that connected to the Ultrasonic Sensors, OLED Display, and motors, including wires connecting them to the 5V pins, GND pins, and their respective action pins.

A diagram of the circuit can be seen in Fig 6.3.1.

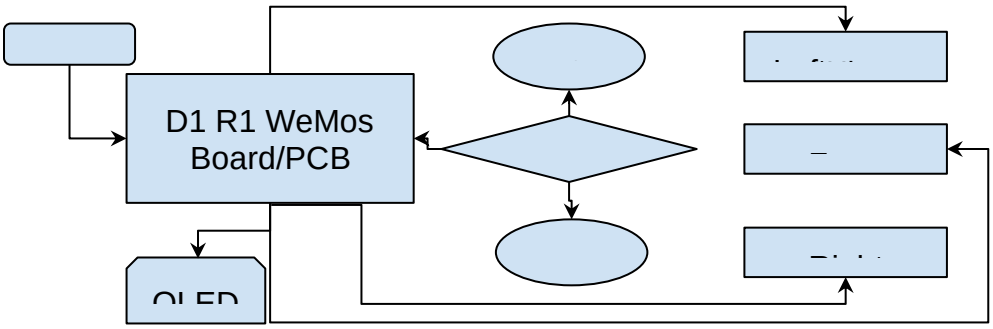


Fig 6.3.1 Circuit Diagram

A PCB was used instead of a breadboard in order to keep the circuitry space efficient, and zip ties were used to tie specific groups of wires together, creating a more organized and clean circuit, which can be seen in Fig 6.3.2.

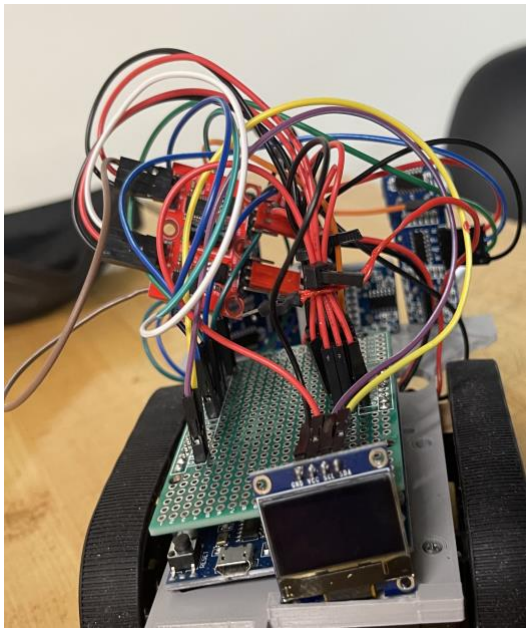


Fig 6.3.2 Circuit Picture

In order to keep everything organized and easily understandable, the wiring that constructed the circuit followed a color coding system, which can be seen in the table that follows Table 6.3.1.

Table 6.3.1

PIN NAME	**WIRE COLOR**
5V	RED
GND	BLACK
Vin	ORANGE
TRIG	BLUE
ECHO	GREEN
SDA	YELLOW
SCL	PURPLE
Left Motor	WHITE
Right Motor	BROWN

Along with the wiring, sensors, and other devices, the circuit was connected through the use of soldering. The team utilized soldering to attach each individual wire to the PCB that was connected to the WeMos board. This supplied the the entire circuit with power by allowing

everything to be connected to the WeMos board, of which was directly connected to the battery pack inside of the Zumma Chassis.

6.4 Software Design and Coding, (Gaikwad)

To successfully complete the task assigned by the client, the robot had to not only reach the targets but also had to detect, calculate and create a path to the target with proper obstacle avoidance. Arduino IDE was used to write, compile and upload code to the robot. The basic software objective of the code was clear:

1. Using the location data received from the lidar sensor to plan a path to the next target and stop for 2 seconds for confirmation and then repeating the process until it reaches the final target.
2. Constantly detecting obstacles and avoiding collisions and damages.

Once the team had a basic layout of code objectives, variables and functions were introduced in the code (See Fig.6.4.1). The distance variables were for every ultrasonic sensor. Defining variables to store previous and spin variables for turning.

```
int n=0;
int distance_left = 0;
int distance_right = 0;
int distance_front = 0;
double prev_x = -1; // The previous coordinate
double prev_y = -1; // The previous coordinate
int distance_to_target;
double angle;
int vectorProduct;
int spin=1;
```

Fig. 6.4.1 Introduction of variables

A list was created for storing the target values. There were two different lists for x and y coordinates (See fig. 6.4.2). Some pseudo targets (Fig. 6.4.2 highlighted) were added to make sure the robot is on the right track. Next, he team developed a mathematical model that calculates the vectors (See fig. 6.4.3) using the location data received from the lidar sensors and then calculates the cross product and the angle to the next target which further helps the robot on the duration and magnitude of the required turn.

```
//Setting up the target destination, xt[]={A,B,C,D ~}
int xt[] = {1400,650,350,150,150,2000};
int yt[] = {150 ,150,500,500,150,700};
```

Fig. 6.4.2 List to store targets

```
double vectorA=sqrt((x-prev_x)*(x-prev_x) + (y-prev_y)*(y-prev_y));
double vectorB=sqrt((xt[n]-x)*(xt[n]-x) + (yt[n]-y)*(yt[n]-y));

int dot = (x-prev_x)*(xt[n]-x) + (y-prev_y)*(yt[n]-y);

// Use dot product to calculate angle
angle = degrees(acos((dot)/(vectorA*vectorB)));
vectorProduct = (x-prev_x)*(yt[n]-y)-(y-prev_y)*(xt[n]-x);
```

Fig. 6.4.3 Vector Product and Angle calculation

```
    if(distance_front < 50)
    {
// Use vector product to determine which way
// is more convenient to turn
      if(vectorProduct < 0)
      {
        spin=1;
        motorR.write(110);
        motorL.write(110);
        delay(350);
        motorR.write(115);
        motorL.write(65);
        delay(400);
        motorR.write(115);
        motorL.write(115);
        }
      else
      {
        spin=1;
        motorR.write(110);
        motorL.write(110);
        delay(350);
        motorL.write(115);
        motorR.write(65);
        delay(400);
        motorR.write(115);
        motorL.write(115);
        }
    }
```

Object Avoidance

Next challenge was obstacle detection and avoidance (See fig. 6.4.3. For this we used if condition statements. Under the bigger if condition, if the vector product is less than zero, the code will start using the distance readings for obstacle detection. If the front sensor detects an obstacle less than the stop distance, the code will analyze and calculate which side to turn to based on the readings of the other two (left and right) sensors. If all of the sensors detect an obstacle, the robot will go in reverse and then ***Fig. 6.4.3*** repeats the process.

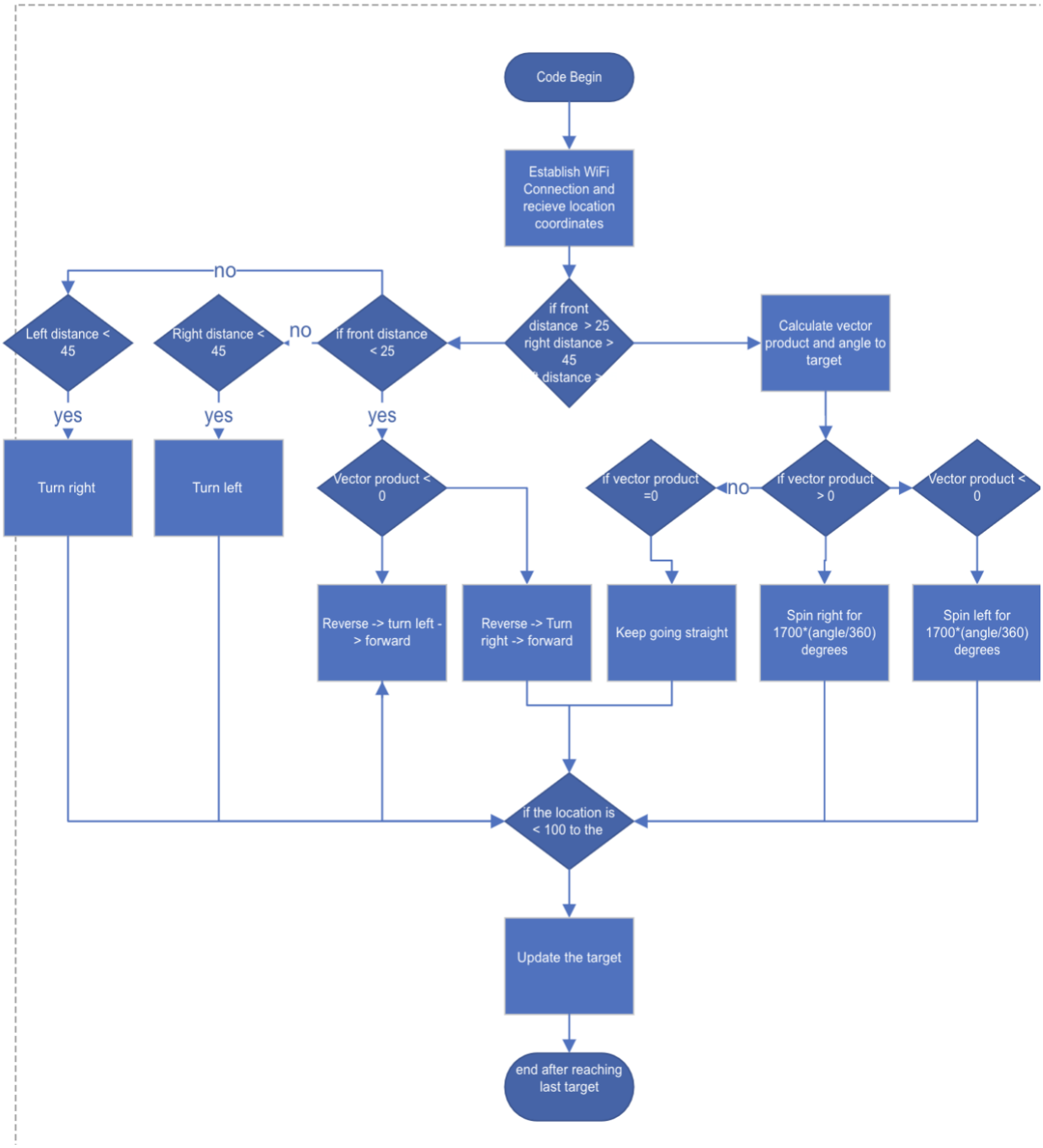
Finally, the code has a function to stop for 2 seconds and update if the robot is in the 100 cm range of the target, updates the next target coordinates and follows the whole process again (See Fig. 6.4.4)

```
distance_to_target = sqrt(sq(x-xt[n])+sq(y-yt[n]));
// Check if Robot is at target
if (n<6)
{
  if (n==0||n==1||n==4||n==5)
  {
    if (distance_to_target<=100)
    {
      motorR.write(90);
      motorL.write(90);
      delay(2000);
      display.clear();
      String str_1 = "Target Reached!"; //Display that target has been reached
      display.drawString(0, 0, str_1);
      display.flipScreenVertically();
      display.display();
      n++;
      }
    }
    else if (distance_to_target<=400)
    {
    if (n == 1) {
    motorR.write(180);
    motorL.write(0);
    delay(150); }
      n++;
      }
   else{
    n++; }
   }


  if (n>5) {            //Stops after hitting all targets
  motorR.write(90);
  motorL.write(90);
  delay(2000000);
  }
```

Fig. 6.4.5 Updating current target

FLOW CHART


Code Begin
Establish WiFi Connection and recieve location coordinates
if front distance > 25 right distance > 45
Calculate vector product and angle to target
if front distance < 25
Right distance < 45
Left distance < 45
no
no
yes
yes
yes
Turn right
Turn left
Vector product < 0
if vector product =0
no
if vector product > 0
Vector product < 0
Reverse -> turn left -> forward
Reverse -> Turn right -> forward
Keep going straight
Spin right for 1700*(angle/360) degrees
Spin left for 1700*(angle/360) degrees
if the location is < 100 to the
Update the target
end after reaching last target

6.5 Assembly and Prototyping Specifications, Gaikwad

The assembly of the actual robot after printing the prototype was fully accurate based on the 3D assembly on SolidWorks. It was possible for the right and the left sensors to be placed either horizontally or vertically. After testing it was noted that placing the sensors horizontally would make it difficult for the robot to pass through some obstacles due to its width. Hence, the team used the design with vertical placement of sensors as the final working prototype.

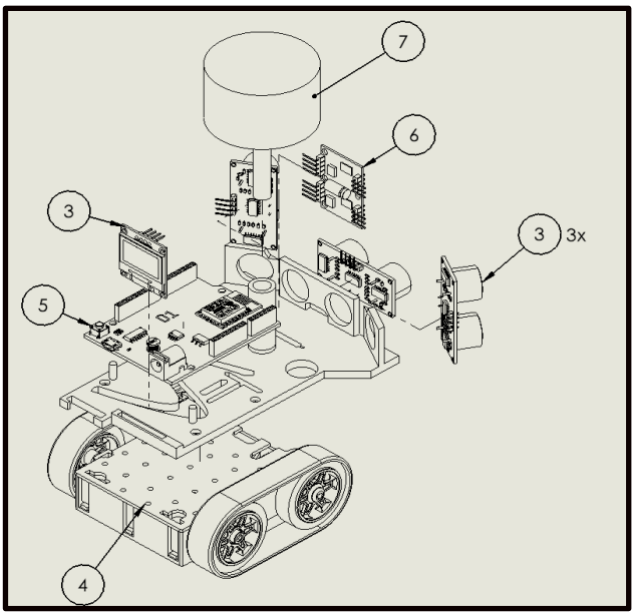


Fig. 6.5.1 Exploded drawing view of the robot assembly

ITEM NO.	PART NUMBER	QTY.
1	FinalRobotBase_Arena Design	1
2	ENGR122_HCSR04	3
3	ENGR122_OLED_Displa y	1
4	ENGR122_ZumoChassis	1
5	ENGR122_WeMosD1	1
6	ENGR122_FeeTech-SMC-4CH	1
7	22s_gps_float_design	1

Fig. 6.5.2 List of parts used in the robot assembly

The robot base (Part No.1) was designed with little holes which would eventually be screwed into the zumo chassis (Part No.4). There were two cylindrical pillars for easy holding

and removal of the WeMos board so that the battery could be charged without much part displacements.

Fig. 6.5.3 Right, front and left view of the assembly

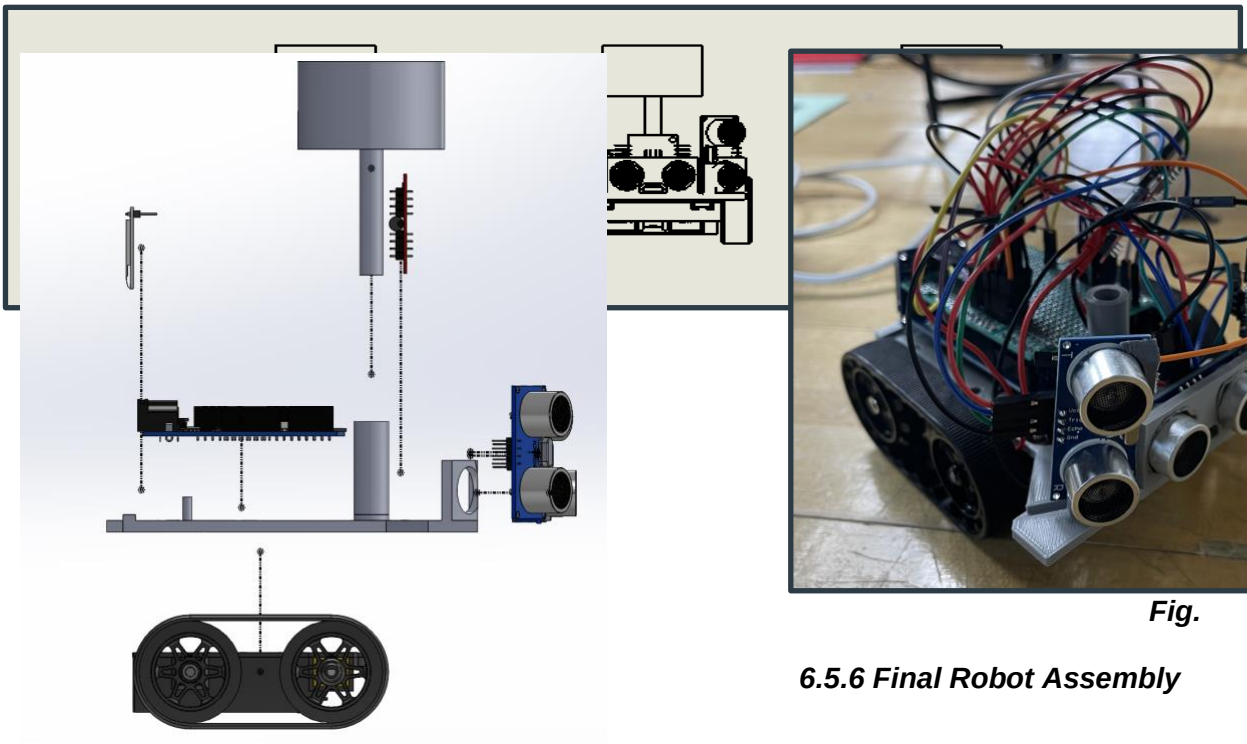
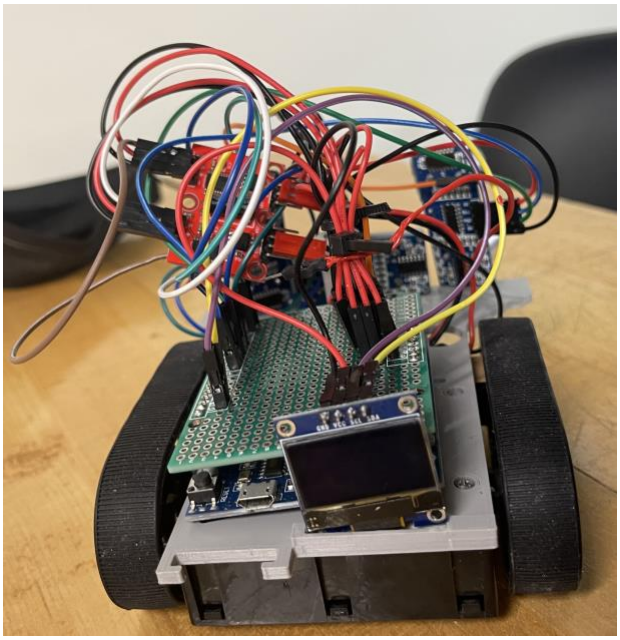


Fig. 6.5.5 Exploded Side view of the assembly.

Fig. 6.5.6 Final Robot Assembly



Fig, 6.5.7 Final assembly (back view)

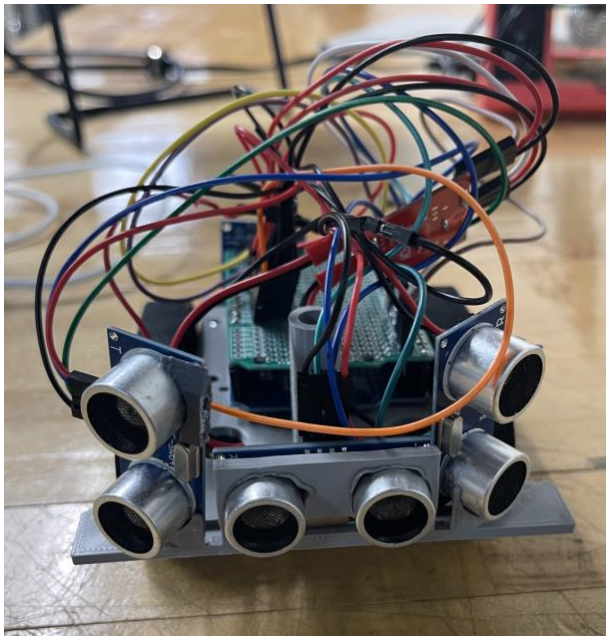


Fig. 6.5.8 Final Assembly (front view)

6.6 Performance Reporting, (Gaikwad)

The design as well as the performance of the final prototype was efficient and accurate as the team anticipated. The robot was able to satisfy all the requirements and complete the tasks assigned by the client. All four targets were reached in 1 minute 33 seconds. There were no mechanical changes needed for the robot during final testing but we had to add minor pseudo targets when the robot was on the way to target 3 and 4 that highly aided in completing the tasks more effectively.

Our tests consistently showed that the robot is efficient in both obstacle avoidance and path finding. See the images below for the report of each target completed.

Iter	Robot X Po	Robot Y position	Target Coordinate	Flag	Total Score
1	118	527	150, 150	Exploring to X 1400 Y 150 - 0iteration Target : 1	0
2	133	543	150, 150	Exploring to X 1400 Y 150 - 1iteration Target : 1	0
3	133	543	150, 150	Exploring to X 1400 Y 150 - 2iteration Target : 1	0
4	105	610	150, 150	Exploring to X 1400 Y 150 - 3iteration Target : 1	0
5	186	521	150, 150	Exploring to X 1400 Y 150 - 4iteration Target : 1	0
6	330	281	150, 150	Exploring to X 1400 Y 150 - 5iteration Target : 1	0
7	286	362	150, 150	Exploring to X 1400 Y 150 - 6iteration Target : 1	0
8	226	412	150, 150	Exploring to X 1400 Y 150 - 7iteration Target : 1	0
9	267	550	150, 150	Exploring to X 1400 Y 150 - 8iteration Target : 1	0
10	431	467	150, 150	Exploring to X 1400 Y 150 - 9iteration Target : 1	0
11	667	375	150, 150	Exploring to X 1400 Y 150 - 10iteration Target : 1	0
12	885	292	150, 150	Exploring to X 1400 Y 150 - 11iteration Target : 1	0
13	1092	192	150, 150	Exploring to X 1400 Y 150 - 12iteration Target : 1	0
14	1329	118	1400, 150	Target delivered : #1	25
15	1409	107	1400, 150	Exploring to X 650 Y 150 - 14iteration Target : 2	25
16	1399	88	1400, 150	Exploring to X 650 Y 150 - 15iteration Target : 2	25
17	1323	61	1400, 150	Exploring to X 650 Y 150 - 16iteration Target : 2	25
18	1353	63	1400, 150	Exploring to X 650 Y 150 - 17iteration Target : 2	25
19	1341	44	1400, 150	Exploring to X 650 Y 150 - 18iteration Target : 2	25
20	1337	112	1400, 150	Exploring to X 650 Y 150 - 19iteration Target : 2	25
21	1199	46	1400, 150	Exploring to X 650 Y 150 - 20iteration Target : 2	25
22	1097	113	1400, 150	Exploring to X 650 Y 150 - 21iteration Target : 2	25
23	866	42	1400, 150	Exploring to X 650 Y 150 - 22iteration Target : 2	25
24	704	97	650, 150	Target delivered : #2	50
25	616	156	650, 150	Exploring to X 150 Y 150 - 24iteration Target : 3	50
26	616	156	650, 150	Exploring to X 150 Y 150 - 25iteration Target : 3	50
27	584	101	650, 150	Exploring to X 150 Y 150 - 26iteration Target : 3	50
28	641	30	650, 150	Exploring to X 150 Y 150 - 27iteration Target : 3	50
29	666	19	650, 150	Exploring to X 150 Y 150 - 28iteration Target : 3	50
30	826	10	650, 150	Exploring to X 150 Y 150 - 29iteration Target : 3	50
31	939	38	650, 150	Exploring to X 150 Y 150 - 30iteration Target : 3	50
32	947	134	650, 150	Exploring to X 150 Y 150 - 31iteration Target : 3	50
33	779	255	650, 150	Exploring to X 150 Y 150 - 32iteration Target : 3	50
34	597	244	650, 150	Exploring to X 150 Y 150 - 33iteration Target : 3	50
35	538	250	650, 150	Exploring to X 150 Y 150 - 34iteration Target : 3	50
36	545	187	650, 150	Exploring to X 150 Y 150 - 35iteration Target : 3	50
37	556	149	650, 150	Exploring to X 150 Y 150 - 36iteration Target : 3	50
38	540	60	650, 150	Exploring to X 150 Y 150 - 37iteration Target : 3	50
39	544	50	650, 150	Exploring to X 150 Y 150 - 38iteration Target : 3	50
40	572	26	650, 150	Exploring to X 150 Y 150 - 39iteration Target : 3	50
41	572	26	650, 150	Exploring to X 150 Y 150 - 40iteration Target : 3	50
42	609	62	650, 150	Exploring to X 150 Y 150 - 41iteration Target : 3	50
43	714	54	650, 150	Exploring to X 150 Y 150 - 42iteration Target : 3	50
44	701	164	650, 150	Exploring to X 150 Y 150 - 43iteration Target : 3	50
45	633	282	650, 150	Exploring to X 150 Y 150 - 44iteration Target : 3	50
46	613	292	650, 150	Exploring to X 150 Y 150 - 45iteration Target : 3	50
47	452	357	650, 150	Exploring to X 150 Y 150 - 46iteration Target : 3	50
48	242	337	650, 150	Exploring to X 150 Y 150 - 47iteration Target : 3	50
49	109	222	150, 150	Target delivered : #3	75

49	109	222	150, 150	Target delivered : #3	75
50	109	222	150, 150	Exploring to X 2000 Y 700 - 49iteration Target : 4	75
51	109	222	150, 150	Exploring to X 2000 Y 700 - 50iteration Target : 4	75
52	94	359	150, 150	Exploring to X 2000 Y 700 - 51iteration Target : 4	75
53	226	526	150, 150	Exploring to X 2000 Y 700 - 52iteration Target : 4	75
54	481	423	150, 150	Exploring to X 2000 Y 700 - 53iteration Target : 4	75
55	733	503	150, 150	Exploring to X 2000 Y 700 - 54iteration Target : 4	75
56	954	533	150, 150	Exploring to X 2000 Y 700 - 55iteration Target : 4	75
57	1037	515	150, 150	Exploring to X 2000 Y 700 - 56iteration Target : 4	75
58	896	479	150, 150	Exploring to X 2000 Y 700 - 57iteration Target : 4	75
59	952	424	150, 150	Exploring to X 2000 Y 700 - 58iteration Target : 4	75
60	1063	578	150, 150	Exploring to X 2000 Y 700 - 59iteration Target : 4	75
61	1252	610	150, 150	Exploring to X 2000 Y 700 - 60iteration Target : 4	75
62	1480	614	150, 150	Exploring to X 2000 Y 700 - 61iteration Target : 4	75
63	1726	604	150, 150	Exploring to X 2000 Y 700 - 62iteration Target : 4	75
64	1949	646	2000, 700	Test is done, Good Job	158.5

Score Calculation

A rubric and algorithm was made known for transparency of grading policy.The above score has been calculated using the algorithm mentioned below.

Performance	Goal	Scoring Rubric
Target Reaching	Reaching 4 targets in a given order	Reaching one target: 25 points (100 points max.)
Collision Avoidance	Collision-free Driving	-5 points/crash (Maximum of 10 crashes allowed)
Time limit	3 Mins	mission completed less than 3 mins: +0.5 points/sec mission completed more than 3 mins: -0.5 points/sec

Here is the grading algorithm

```
base_score = 50;   //This base score will be manually added by your instructor
Arena_score = test_results_received_from_the_SmartArenaSystem();
number_of_collisions = manually_counted_by_your_instructor();
final_competition_score = base_score + Arena_score + (-5 * number_of_collisions);

if (final_competition_score >= 150) {
    final_competition_score = 150;
}
```

6.7 Conclusion and Recommendations, (Gaikwad)

This project has been a huge learning experience of all of its team members. Everyone has worked hard and invested their time,knowledge and effort into making this project successful. Working on this project has improved engineering skills, social skills, team building and cooperation which is a very vital skill to have to be successful in the industry. The journey of designing the prototype was a huge indication of how much the CAD skills of the members improved throughout the entire project. Nevertheless, the software development was an eye opening experience as it not only taught the importance of programming in engineering but also that combining the knowledge acquired from different fields like mathematics and applying it into the algorithm can help build a more complex code than a beginner coder would expect.

Even after a successful completion of the task, the team believes that there is still room for potential improvement. The wiring can be more well-organized. The code could be more optimized if four sensor sensors were used instead of three. In such a case if one sensor was placed in the back, the overall performance could be improved especially in corner situations. Originally the code would make small reverse movements to find a better turning side, but with four sensors, multiple reverse turns could be avoided and aid the robot to find a way out of such situations in one or relatively less turns. However, the team could not use four sensors because of time constraints as it required an additional library and some wiring modifications because the WeMos provided can conventionally only hold three 4-pin Ultrasonic sensors.

In conclusion, the most important lesson to be learned is that there are always more improvements and optimizations that can make a more efficient product and the skills and knowledge to recognize them is through hard work, teamwork, accepting mistakes and learning from them and ultimately making the improvements. The team is highly satisfied and proud with the outcome of this project and thanks to the professional assistance provided by the department and of course, the team members to be a responsible and effective member.

Code

```
#include <Ultrasonic.h>
#include <Servo.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <ESP8266WiFi.h>
#include <Wire.h>

#include "PubSubClient.h"
```

```
#include "info.h"
#include "WiFiManager.h"
#include "SSD1306.h"

//MQTT Communication associated variables
String inString = "";
int testvariable;
char payload_global[100];
boolean flag_payload;
int k = 0; // done target constant

//MQTT Setting variables
const char* mqtt_server = "155.246.62.110";
const int mqtt_port = 1883;
const char* MQusername = "jojo";
const char* MQpassword = "hereboy";

//WiFi Define
WiFiClient espClient;
info board_info;
PubSubClient client(espClient);

////////////////////////////////////////Robot Logic Variables//////////////////////////////////////////////////
//////////////////////////Students change this section for their modification/////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////// Wifi Settings variables // Remote students change to your network parameters
//////////////////////
const char* ssid = "Stevens-IoT";
const char* password = "nMN882cmg7";

////////// If on-campus, remember to add your WeMos MAC address to Stevens network
//////////////////////////

//////********CHANGE FOR EACH ARENA***********////////
//const char* MQtopic = "louis_lidar1"; // Topic for Arena_1 (EAS011 - South)
//const char* MQtopic = "louis_lidar2"; // Topic for Arena_2 (EAS011 - North)

// Define the DC motor contorl signal pins
#define motorRpin D0  //GPIO pin setting for motorR
#define motorLpin D2  //GPIO pin setting for motorL

Servo motorR;        //Create servo motorR object to control a servo
Servo motorL;        //Create servo motorL object to control a servo

// Define the Ultrasonic sensor pins
Ultrasonic ultrasonic_front(D8, D5); // Ultasonic sensor, front (trig, echo)
Ultrasonic ultrasonic_right(D9, D6); // Ultasonic sensor, right (trig, echo)
Ultrasonic ultrasonic_left(D10, D7); // Ultasonic sensor, left (trig, echo)

// Define the OLED display pins D14 SDA, D15 SCL
```

```
SSD1306  display(0x3C, D14, D15); //Address set here. Pins defined as D2 (SDA/Serial Data) and D5 (SCK/Serial Clock).

//////////////////////////// Define the variables needed for your algorithm ////////////////////////////////////

int n=0;
int distance_left = 0;
int distance_right = 0;
int distance_front = 0;
double prev_x = -1; // The previous coordinate
double prev_y = -1; // The previous coordinate
int distance_to_target;
double angle;
int vectorProduct;
int spin=1;



////////////////////////////////////Robot Logic Variables End/////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////

// Setup the wifi, Don't touch
void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  randomSeed(micros());
}

// MQTT msg callback
void callback(char* topic, byte* payload, unsigned int length) {
  for (int i = 0; i < length; i++) {
    payload_global[i] = (char)payload[i];
  }
  payload_global[length] = '\0';
  flag_payload = true;
}

// MQTT reconnection setup
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);

    // Attempt to connect
    if (client.connect(clientId.c_str(),MQusername,MQpassword)) {
      client.subscribe(MQtopic);
```

```
    }
  }
}

////////////////////////////// SETUP LOOP. Don't Touch //////////////////////////////////////

void setup() {

  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
  display.clear();
  display.drawString(0, 0, "Connected to Wifi");
  display.flipScreenVertically();
  display.display();
  motorR.attach(motorRpin);   //motorR is attached using the motorRpin
  motorL.attach(motorLpin);   //motorL is attached using the motorLpin

  // Display Setup
  delay(1500);
  display.clear();
  display.drawString(0, 0, "Starting Robot...");
  display.flipScreenVertically();
  display.display();
  motorR.write(90);
  motorL.write(90);
```

```
  delay(1500);


  motorR.write(115);
  motorL.write(115);
}

////////////////////////////////// MAIN LOOP ////////////////////////////////////////

void loop() {
  //subscribe the data from mqtt server
  if (!client.connected()) {
      reconnect();
  }
  const char *msg = "target";
  char temp1[50];
  sprintf(temp1,"%d",k);
  const char *c = temp1;


  client.publish( msg , c);
  client.loop();
  String payload(payload_global);


  int testCollector[10];
  int count = 0;
  int prevIndex, delimIndex;
```

```
  prevIndex = payload.indexOf('[');
  while( (delimIndex = payload.indexOf(',', prevIndex +1) ) != -1){
    testCollector[count++] = payload.substring(prevIndex+1, delimIndex).toInt();
    prevIndex = delimIndex;
  }
  delimIndex = payload.indexOf(']');
  testCollector[count++] = payload.substring(prevIndex+1, delimIndex).toInt();

  // Robot location x,y from MQTT subscription variable testCollector
  float x, y;
  x = testCollector[0];
  y = testCollector[1];

  //Setting up the target destination, xt[]={A,B,C,D ~}
  int xt[] = {1400,650,350,150,150,2000};
  int yt[] = {150,150,500,500,150,700};


//////////////////////////////////////////Robot Logic Begin////////////////////////////////////////////////////
//////////////////////////Students change this section for their modification/////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////

  ////////////////////// find the closest obstacle ////////////////////////////////////

  // Read the distances from the Utrasonic sensors. Output unit is mm
  distance_left = ultrasonic_left.read(CM)* 10;
```

```
distance_front = ultrasonic_front.read(CM)* 10;
distance_right = ultrasonic_right.read(CM)* 10;

  // Display the x,y location in the OLED
  display.clear(); // Clear the buffer
  String str_1 = "x: " + String(x);
  String str_2 = "y: " + String(y);
  String str_3 = "Target: " + String(n+1) + " (" + String(xt[n]) + "," + String(yt[n]) + ")";
  display.drawString(0, 0, "Current Location");
  display.drawString(0, 15, str_1);
  display.drawString(0, 30, str_2);
  display.drawString(0, 45, str_3);
  display.flipScreenVertically();
  display.display();


  ////////////////////////////////////////////// Robot Path Planning Logic ////////////////////////////////////////////
  if (!(distance_front<25||distance_left<45||distance_right<45))
  if  (!(prev_x==x&&prev_y==y)&&x!=0&&spin==0)
  {

    double vectorA=sqrt((x-prev_x)*(x-prev_x) + (y-prev_y)*(y-prev_y));
    double vectorB=sqrt((xt[n]-x)*(xt[n]-x) + (yt[n]-y)*(yt[n]-y));

    int dot = (x-prev_x)*(xt[n]-x) + (y-prev_y)*(yt[n]-y);
```

```
// Use dot product to calculate angle
angle = degrees(acos((dot)/(vectorA*vectorB)));
vectorProduct = (x-prev_x)*(yt[n]-y)-(y-prev_y)*(xt[n]-x);

// Use vector product to determine  which way to turn
if (vectorProduct>0)
{
  spin=1;
  motorR.write(65);
  motorL.write(115);
  delay(1700*(angle/360));
  motorR.write(115);
  motorL.write(115);
  }
else
{
  spin=1;
  motorR.write(115);
  motorL.write(65);
  delay(1700*(angle/360));
  motorR.write(115);
  motorL.write(115);
  }
}
else
{
```

```
      spin=0;
      }
    ////////////////////////////////////////////// Robot Path Planning Logic End ////////////////////////////////////////////
     prev_x=x;
     prev_y=y;
    ////////////////////////////////////////////// Robot Object Detection ////////////////////////////////////////////
    if(distance_front < 25)
    {
// Use vector product to determine which way
// is more convenient to turn
      if(vectorProduct < 0)
      {
        spin=1;
        motorR.write(110);
        motorL.write(110);
        delay(350);
        motorR.write(115);
        motorL.write(65);
        delay(400);
        motorR.write(115);
        motorL.write(115);
        }
      else
      {
        spin=1;
        motorR.write(110);
```

```
    motorL.write(110);
    delay(350);
    motorL.write(115);
    motorR.write(65);                    ////////////////////////////////////////////////////////////////////
    delay(400);
    motorR.write(115);
    motorL.write(115);
    }
}
else if(distance_right < 45)
{
  spin=1;
  motorL.write(65);
  motorR.write(115);
  delay(600);
  motorR.write(115);
  motorL.write(115);
  }
else if(distance_left < 45)
{
  spin=1;
  motorL.write(115);
  motorR.write(65);
  delay(600);
  motorR.write(115);
  motorL.write(115);
```

```
  }
else
{
  spin=0;
  motorR.write(115);
  motorL.write(115);
  }

//////////////////////////////////////////// Robot Object Detection End ////////////////////////////////////////////
//////////////////////////////////////////// Check if at Location ////////////////////////////////////////////

distance_to_target = sqrt(sq(x-xt[n])+sq(y-yt[n]));
// Check if Robot is at target
if (n<6)
{
  if (n==0||n==1||n==4||n==5)
  {
    if (distance_to_target<=100)
    {
      motorR.write(90);
      motorL.write(90);
      delay(2000);
      display.clear();
      String str_1 = "Target Reached!"; //Display that target has been reached
      display.drawString(0, 0, str_1);
      display.flipScreenVertically();
```

```
    display.display();
    n++;
    }
  }
  else if (distance_to_target<=400)
  {
  if (n == 1) {
  motorR.write(180);
  motorL.write(0);
  delay(150); }
    n++;
    }
 else{
  n++; }
 }


 if (n>5) {          //Stops after hitting all targets
 motorR.write(90);
 motorL.write(90);
 delay(99999);
 }
```

//Robot Logic End//

//

//

```
}
```