E-121: Final Report

December 3, 2019

Authors: Matthew Roberts, Christopher Kruger, Wenhan Zheng

Group 5

Section J

Abstract

The purpose of this class was to learn how to construct a robot with set instructions for different tasks. We built robots from parts and arranged them in our own interests and then coded them in the Arduino programming app. After constructing the robot, we then were assigned to perform different tasks with the robot. These tasks included stopping 2 inches within a wall and making a 180 degree turn, going around a wall in the middle of an arena, and touching 4 different paddles in a set order inside of a larger arena. With these objectives, we have learned how to program a robot to perform different tasks with no direct human interaction. We also learned how to think efficiently and algorithmically due to the robot having set limitations and needing to perform set instructions.

1.0 Introduction

The purpose of this report is to display the results of the group's robot and the design process of the group as the robot was assembled and programmed for each task. This report delves into the group's failures and successes involved in both the assembly of hardware and software during the E-121 design class.

The goal of the robot project was to build-up to the completion of the Robot Arena by completing increasingly difficult tests throughout the E-121 design class. The goal of this overarching project was to help the group understand how components come together to form a completed project, while also guiding the group in a way that maintained independent learning.

There were two additional projects to help the group succeed in the arena: The Road Test and The Nascar Challenge. The goal of the road test was to help the group calibrate and understand the sensors and code of the group's robot. This challenge was to code the robot to move in a straight line until two inches from a wall, turn 180 degrees, and return to its original position. The purpose of the Nascar challenge was to again advance the code of the group in a manner that will yield greater understanding in preparation for the Robot Arena challenge. The goal of the Nascar Challenge was to have the robot make four left runs around a central wall and return to its initial position without hitting either the perimeter walls or internal wall placed in the middle of the track. The groups' robot was unable to complete the NASCAR challenge, upon multiple weeks of trouble shooting Professor Russo gave his assembled robot to the group. All descriptions are the groups' interpretations and uses of this robot. This leads to the Robot Arena Challenge.

The goal of the Robot Arena Challenge is to hit four bumpers on the wall of the Arena in a group-specific order. Additionally, the robot begins at 1 of 4 random starting points as presented in Figure 1. The group had 2 minutes to successfully complete the challenge. The robot is not allowed to touch anything other than the bumpers, including perimeter walls and the center obstacle wall. In Addition, the robot must be fully autonomous, acting without the group's intervention. The group's sequence was bumpers BDAC, in successive order.

The group's approach included a large amount of planning, testing, and trial and error. Before the group placed the robot in the arena, thought was put into how to best solve the problem. For example, before the code for the Arena was written, the group brainstormed how the robot would theoretically move. This theoretical plan was then converted to the software. The planning phase allowed the group to write and organize a plan before attempting to apply the group's thoughts to the final product. In the initial stages, problems were identified by looking at separate pieces at one time. For example, first the group examined the different components of the hardware, and, if that did not yield a solution, examined the software for possible issues. If the group faced an issue in the arena, the code was tweaked and then the robot was retested.

Additionally, the group attempted to maintain independence with any problem that arose. The group approached the professor for help only when all other options were exhausted.

2.0 Discussion

2.1 Requirements

The main objectives of the robot mostly involved tasks of movement and sensing. Each of these tasks required hardware and software developments to successfully complete. The first test was the Road Test. At this point, all boards were mounted in a manner that did not conflict with wheel movement. Additionally, a sensor had to be mounted to the front so the robot would be able to detect when it was 2 inches from the wall. Next was the NASCAR challenge. In the challenge, the group initially believed additional sensors would be required, and the other two sensors were mounted. However, these sensors were not completely necessary, and the robot was able to complete the loop with just the front sensor. The final test was the Robot Arena. For this, all three sensors were found to be necessary for the robot to be able to hit each paddle in the proper order.

Overall, the robot had to accomplish two things: move independently and be able to sense its location. These two factors were central to each of the tasks, and subsequently to the design of the robot itself. The robot's sense of location stemmed from the sensors connected to its frame. The group found that three sensors was optimal for the tasks required of the robot. One placed in the front, one on the front left, and one placed on the front right of the frame. This created a blind spot behind the robot, but the group found that this particular blind spot did not affect the performance of the robot during any of the tests. The lack of a back sensor then freed space that was used to hold the battery and easily connect to the main board to provide power to the robot. The design of the robot was created and changed to fit each objective. Through this fluid design process, the group was able to shape the design of their robot to given specifications, but in a constructive manner rather than destroying and restarting each time. The final result of the process was, the group believes, as optimal as the group could produce.

2.2.1 Mechanical Design

The robot is designed with each of the necessary boards on separate planes for easier attachment with necessary hardware. There are various elevations for the boards to avoid conflict. The first elevation is the bracket, on which the Wemos board is attached using 3x10 standoffs. Next, the motor chip is attached to the front of the bracket using 3x30 standoffs. This allowed for the motor chip to be wired to the motor from the back and to the Wemos board from the front

Three components are connected to the front of the robot: the bread board, one of the three sensors, and a bumper. The front sensor is necessary for the robot to be able to gauge how far walls are in front of it. The bread board was necessary for each sensor, as it allows for a secure connection. The bumper is in place, so the sensor is secure and avoids damage during testing phases. Additionally, the sensor is taped as it was initially loosely hanging, cause problems with how the sensor gathered information.

The main features of the left and right side of the robot are the left and right wheels and the left and right sensors. These sides are essentially symmetrical. These sensors were loose due to them hanging from the bread board. Due to this, electrical tape was used to secure it and guarantee accurate readings. A rubber band was placed around the left and right wheels. This allowed for additional grip to avoid slippage due to a lack of traction with the Arena floor.. The highest chip is the motor chip, which was connected to the motors. The furthest left board was the bread board. Wires were connected between the sensors and Wemos board with minimal curves to avoid unnecessary complications.

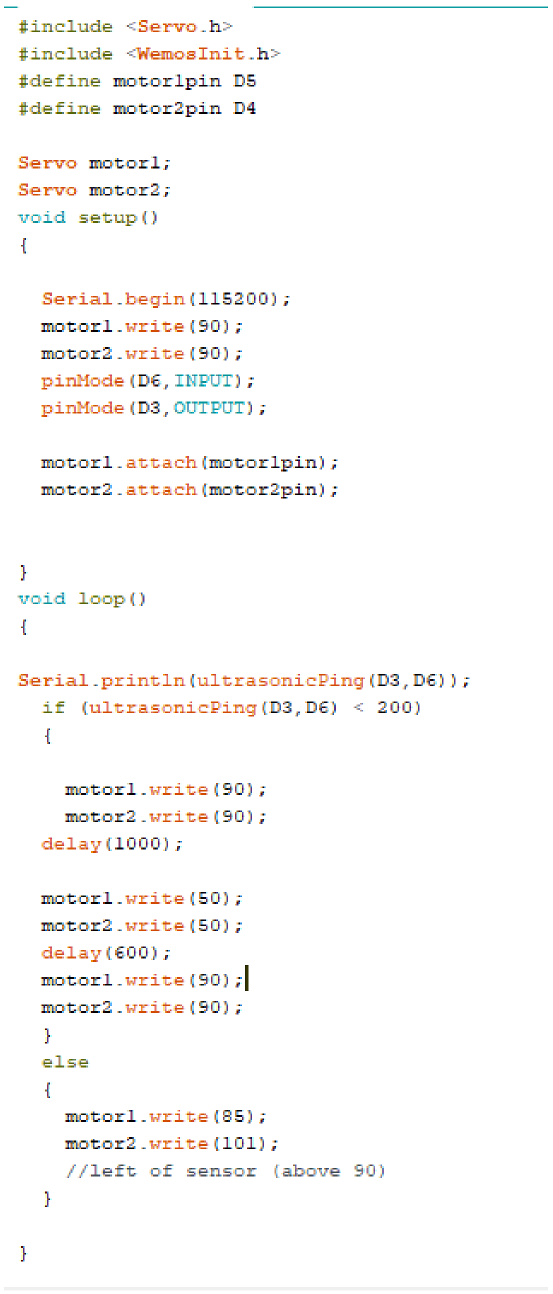
The battery was placed in the gap in the back of the robot. This allowed for the most direct connection between the Wemos board and its power source. The boards are mounted in ways to avoid conflict between their wires. For example, the motor chip is able to pass over the Wemos board because the Wemos board is only wired on its sides. The robot was mounted to maximize space in order to minimize conflict and keep the assembly organized.

2.2.3 Software Design and Coding

The first program our team created was getting our robot to move forward as long as it needed to until the front sensor detected the wall within roughly two inches of the robot. The units for the front sensor were not in any common units like centimeters or millimeters so the team had to constantly do trial and error to find a working number that corresponds to roughly two inches. Once the sensor detected it was within two inches of the wall, it stopped for one second. It then spun around 180 degrees and kept moving forward again until the sensors detected the robot was within two inches of another wall. As for this program, there were no special features, the team purposely made it simple to combat any possible errors. Check figure 2.2.3_1 for the full code, figure 2.2.3_2 for the flow chart of the code, and figure 2.2.3_3 for the diagram of the mission.

The second program our team created was getting our robot to move counter clockwise around an arena with a long metal wall in the middle. The team began writing this program by thinking about the logic the robot will need to progress in this task. The team at first decided to make it use only the front sensor, checking it’s distance to the wall and within around 4 inches it would turn left. Unfortunately, our sensors did not work. Once replacing them, the team found out that it wasn’t the sensor but the actual arduino board. Once fixing the arduino board, the team then repeatedly mixed and matched different parts to see what actually worked and what did not. After replacing the wires countless time, the arduino board three times, the motor chip twice, and the frame four times, the team finally decided to fully replace our robot and take loan of a robot that was assured to work by another professor. Once the team got the new robot, the team decided the team would hard code the robot to perform the task. The team thought it could go straight for a few seconds and then turn every couple of seconds. This deemed a harder challenge than just using the sensors, so then the team decided to use the front sensor again for this challenge. Some notable features of the code is using all three sensors to troubleshoot positioning but only using the front sensor for the logic on whether or not it would turn. Figure 2.2.3_4 is the code, figure 2.2.3_5 is the flow chart logic, and figure 2.2.3_6 is the diagram of the mission.

The third program was much more complex, involving sensors and hard coding. Due to being behind from having to replace our robot which took roughly three weeks to finalize, the team was already two weeks behind on the code for the third project. The team knew the team was only going to have time to potentially finish the code for just one position on the board shown on figure 2.2.3_9. The team began by coding a unique feature to use all two of the three sensors to detect where in the arena it is positioned at. Although the code did work, it is commented out so the team could focus on making the code for the logic of moving across the board. The objective for this involved starting at a random spot of four choices, and moving to different paddles in a set order, ours being BDAC. The team had to code the logic not only to detect where it was on the board, but to move to the respective paddles in that order no matter the position it started in while not touching any walls or barriers. Due to a lack of time, the team focused on the logic if it were to start at position 2. As for moving around the arena, the team made it hard coded to save time and complexity. It moved forward, turned right, moved forward again, and then turned right to move forward again just to hit the first paddle being B. As seen in the code in figure 2.2.3_7, the code is commented to understand the logic needed for hitting each paddle. The team only had enough time to code for position 2 so that is all the team had. The team also didn't have enough time to code for the fourth paddle, but the team were able to hard code the robot, starting at position 2, to hit the first three paddles (being B, D, and A). A unique piece from the code was the robot being able to detect where on the board/what position it is starting from. Although the team did not use it, it did help initially with the logic on where it would start. Another interesting feature was instead of it going backward at 180 degrees and doing some form of a K-turn, the team made the robot do this: whenever it needed to go backwards to move in a different direction, the team made it go backwards in a curve so that it could move forward once done. If the team wanted it to move backwards then turn left to move right from a birds eye view, the team would designate it to move like in the figure 2.2.3_10. If the team wanted it to move backwards then turn right to make it turn left from a birds eye view, then the team would make the robot move backwards curving right then move forward The code is in 2.2.3_7, the flowchart is in 2.2.3_8, the diagram for the arena is figure 2.2.3_9, and the described action of turning backwards in in figure 2.2.3_10.



```
#include <Servo.h>
#include <WemosInit.h>
#define motor1pin D5
#define motor2pin D4

Servo motor1;
Servo motor2;
void setup()
{

  Serial.begin(115200);
  motor1.write(90);
  motor2.write(90);
  pinMode(D6,INPUT);
  pinMode(D3,OUTPUT);

  motor1.attach(motor1pin);
  motor2.attach(motor2pin);


}
void loop()
{

Serial.println(ultrasonicPing(D3,D6));
  if (ultrasonicPing(D3,D6) < 200)
  {

    motor1.write(90);
    motor2.write(90);
  delay(1000);

  motor1.write(50);
  motor2.write(50);
  delay(600);
  motor1.write(90);
  motor2.write(90);
  }
  else
  {
    motor1.write(85);
    motor2.write(101);
    //left of sensor (above 90)
  }

}
```

Figure 2.2.3_1 (left)

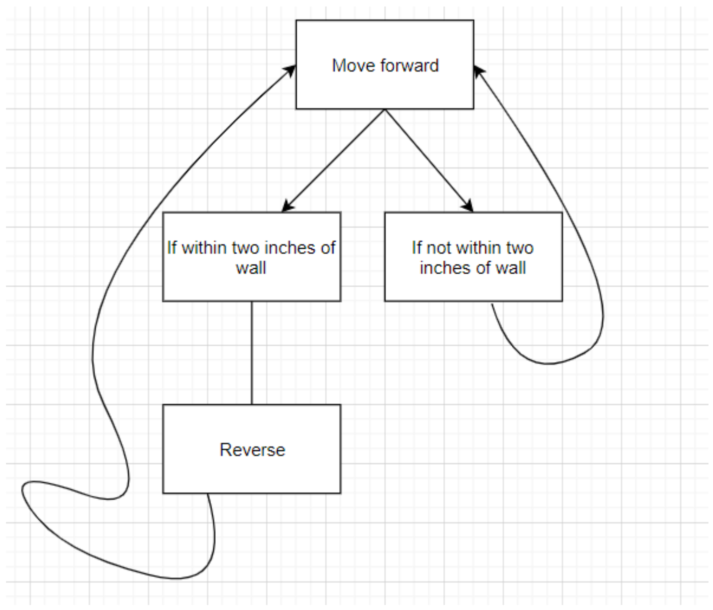

Move forward
If within two inches of wall
If not within two inches of wall
Reverse

Figure 2.2.3_2 (above)

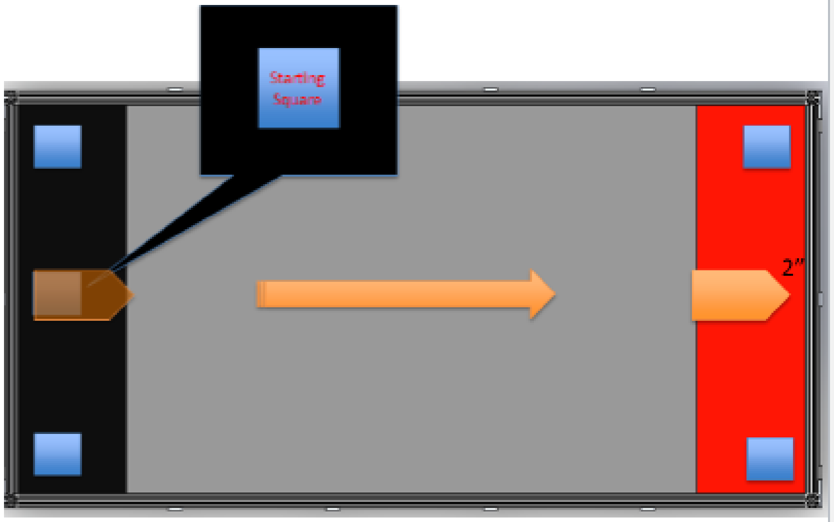
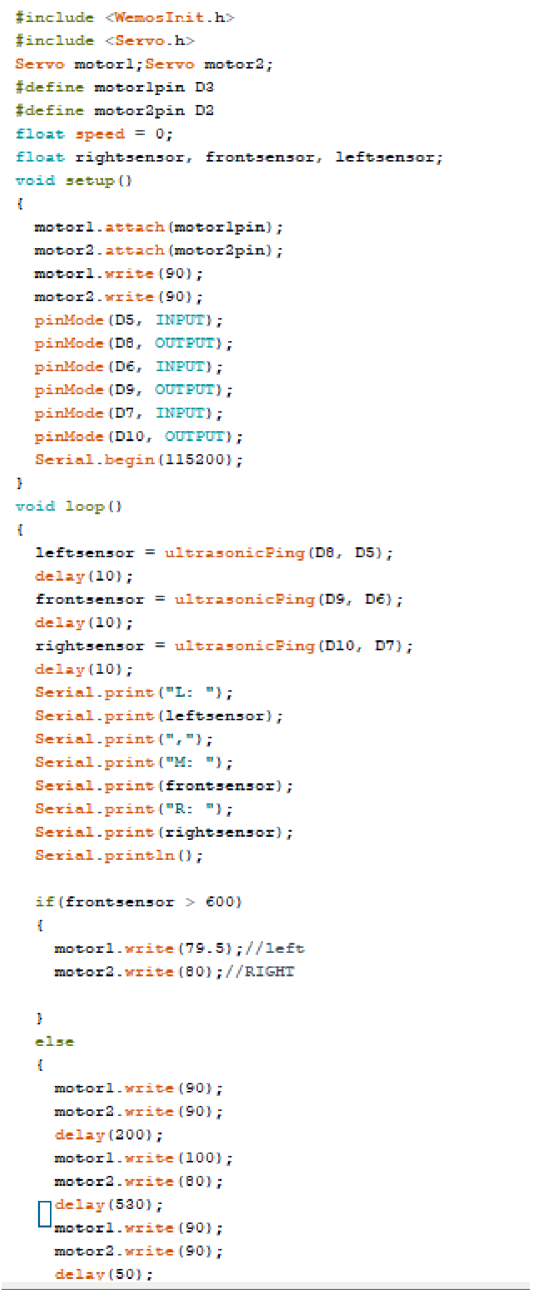


Figure 2.2.3_3 (left)



```
#include <WemosInit.h>
#include <Servo.h>
Servo motor1;Servo motor2;
#define motor1pin D3
#define motor2pin D2
float speed = 0;
float rightsensor, frontsensor, leftsensor;
void setup()
{
  motor1.attach(motor1pin);
  motor2.attach(motor2pin);
  motor1.write(90);
  motor2.write(90);
  pinMode(D5, INPUT);
  pinMode(D8, OUTPUT);
  pinMode(D6, INPUT);
  pinMode(D9, OUTPUT);
  pinMode(D7, INPUT);
  pinMode(D10, OUTPUT);
  Serial.begin(115200);
}
void loop()
{
  leftsensor = ultrasonicPing(D8, D5);
  delay(10);
  frontsensor = ultrasonicPing(D9, D6);
  delay(10);
  rightsensor = ultrasonicPing(D10, D7);
  delay(10);
  Serial.print("L: ");
  Serial.print(leftsensor);
  Serial.print(",");
  Serial.print("M: ");
  Serial.print(frontsensor);
  Serial.print("R: ");
  Serial.print(rightsensor);
  Serial.println();

  if(frontsensor > 600)
  {
    motor1.write(79.5);//left
    motor2.write(80);//RIGHT

  }
  else
  {
    motor1.write(90);
    motor2.write(90);
    delay(200);
    motor1.write(100);
    motor2.write(80);
    delay(530);
    motor1.write(90);
    motor2.write(90);
    delay(50);
```

Figure 2.2.3_4 (left)

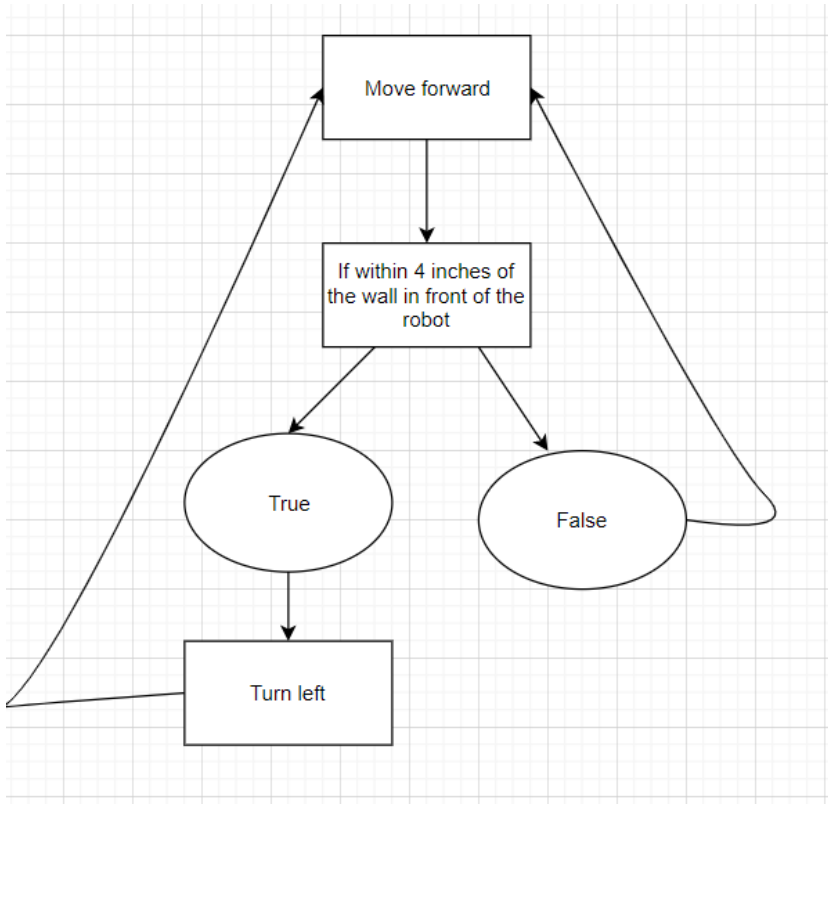


Figure 2.2.3_5 (left)

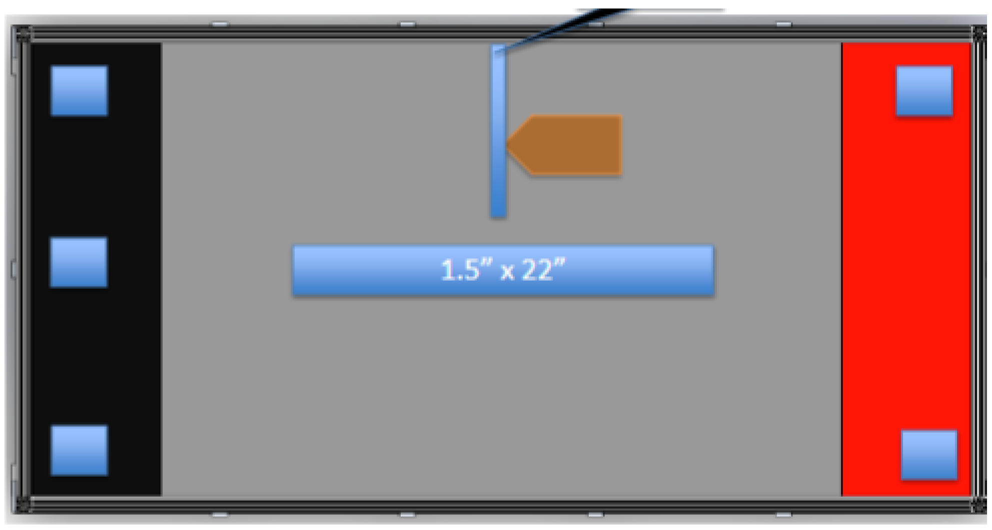
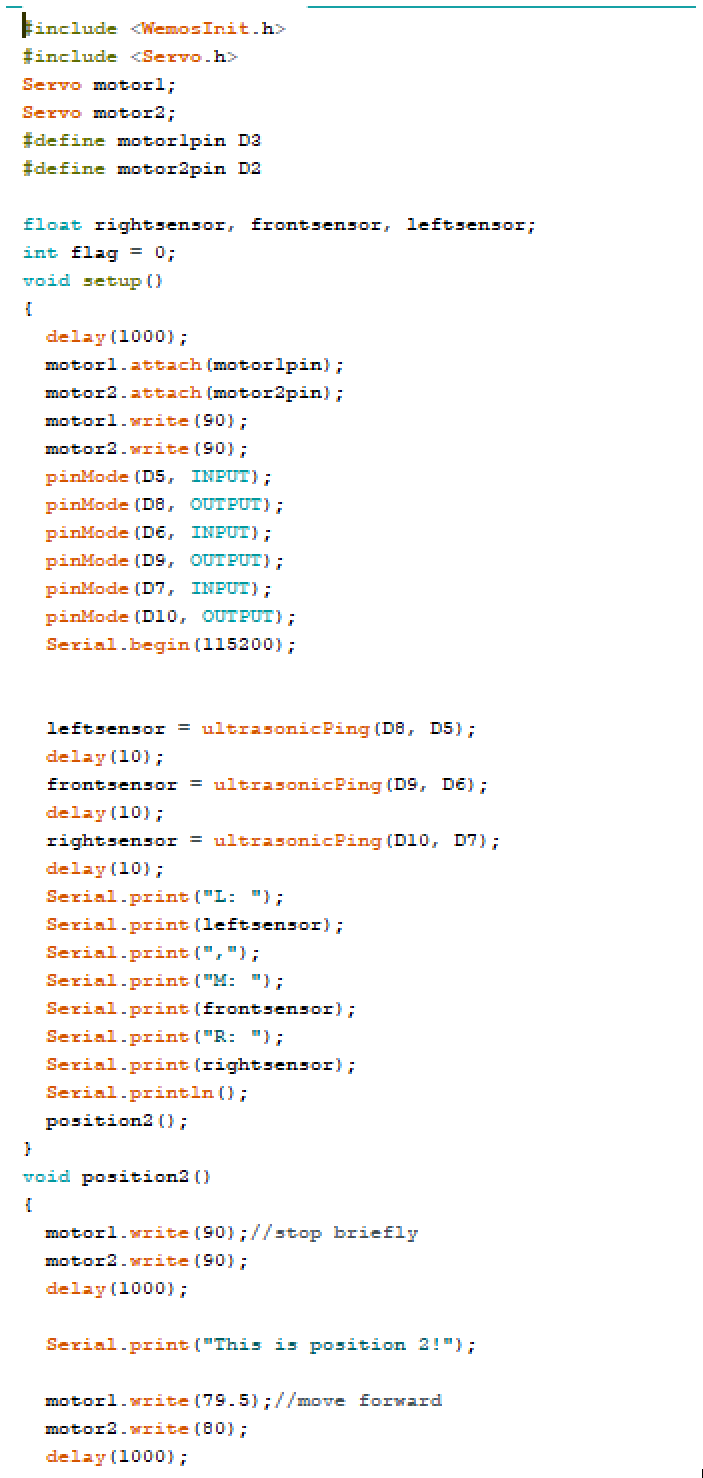
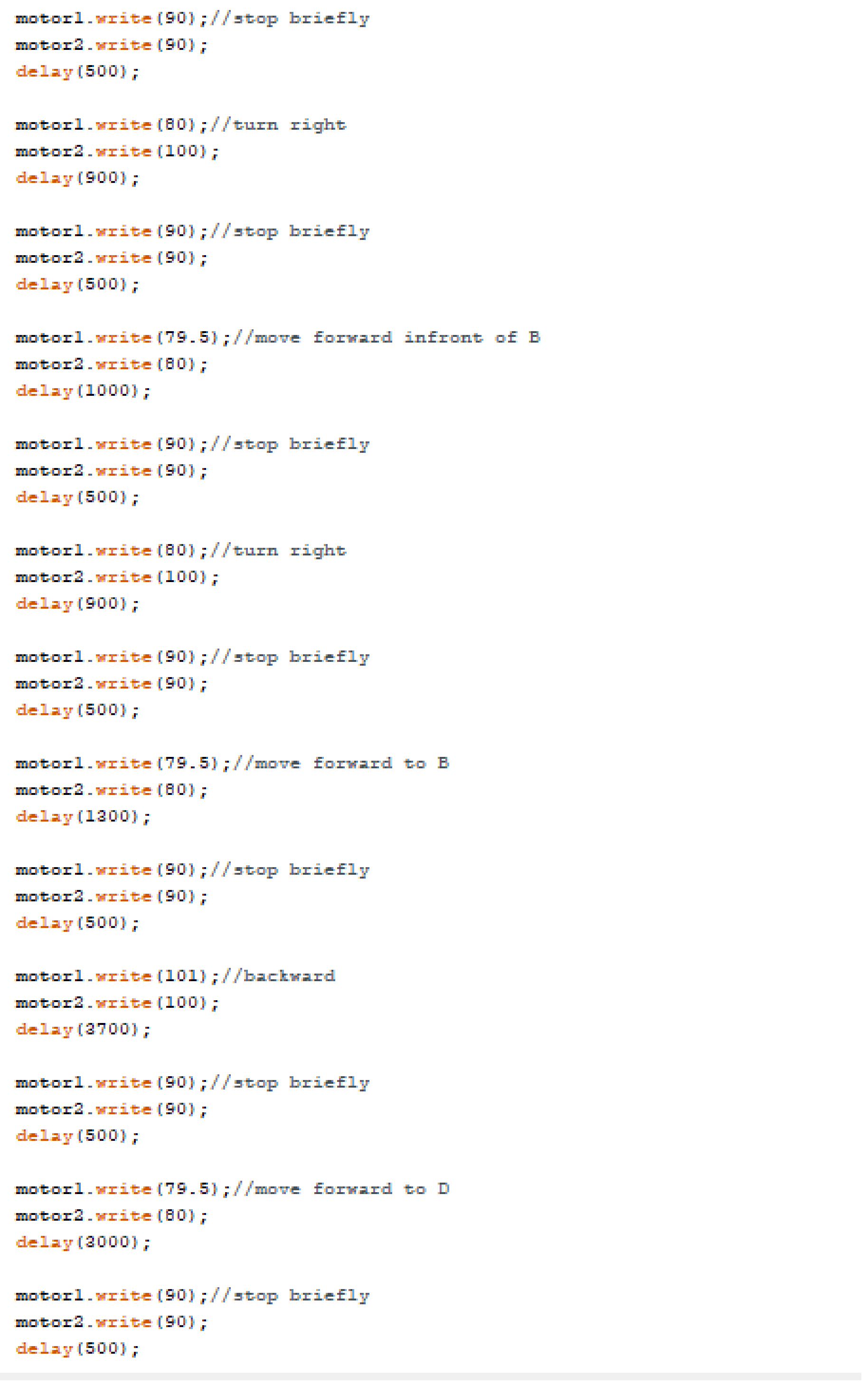


Figure 2.2.3_6 (above)



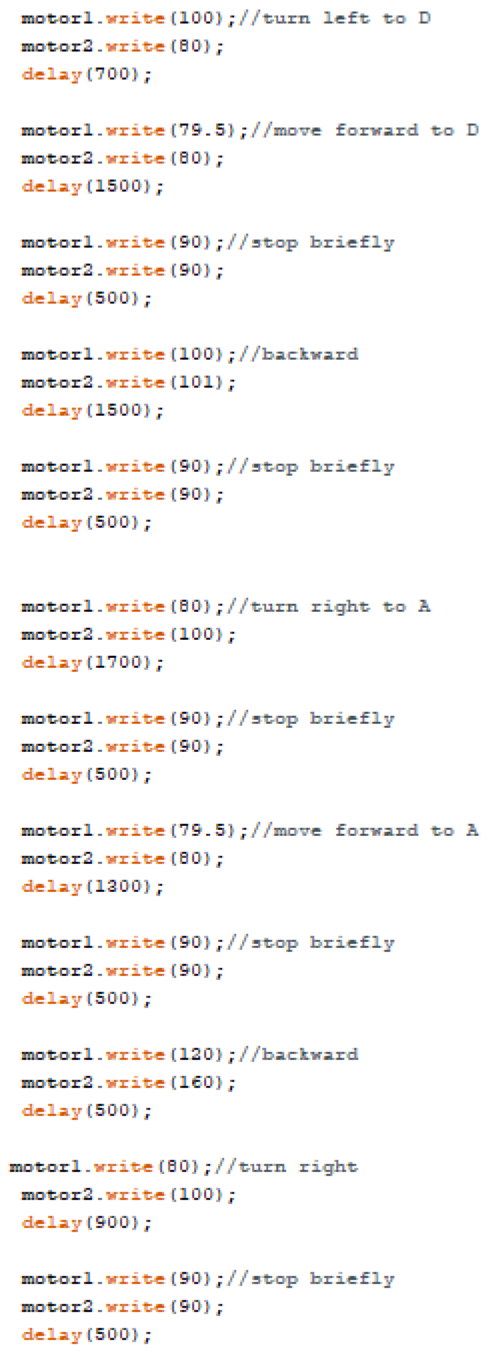
```
#include <WemosInit.h>
#include <Servo.h>
Servo motor1;
Servo motor2;
#define motor1pin D3
#define motor2pin D2

float rightsensor, frontsensor, leftsensor;
int flag = 0;
void setup()
{
  delay(1000);
  motor1.attach(motor1pin);
  motor2.attach(motor2pin);
  motor1.write(90);
  motor2.write(90);
  pinMode(D5, INPUT);
  pinMode(D8, OUTPUT);
  pinMode(D6, INPUT);
  pinMode(D9, OUTPUT);
  pinMode(D7, INPUT);
  pinMode(D10, OUTPUT);
  Serial.begin(115200);


  leftsensor = ultrasonicPing(D8, D5);
  delay(10);
  frontsensor = ultrasonicPing(D9, D6);
  delay(10);
  rightsensor = ultrasonicPing(D10, D7);
  delay(10);
  Serial.print("L: ");
  Serial.print(leftsensor);
  Serial.print(",");
  Serial.print("M: ");
  Serial.print(frontsensor);
  Serial.print("R: ");
  Serial.print(rightsensor);
  Serial.println();
  position2();
}
void position2()
{
  motor1.write(90);//stop briefly
  motor2.write(90);
  delay(1000);

  Serial.print("This is position 2!");

  motor1.write(79.5);//move forward
  motor2.write(80);
  delay(1000);
```

Figure 2.2.3_7 (left)



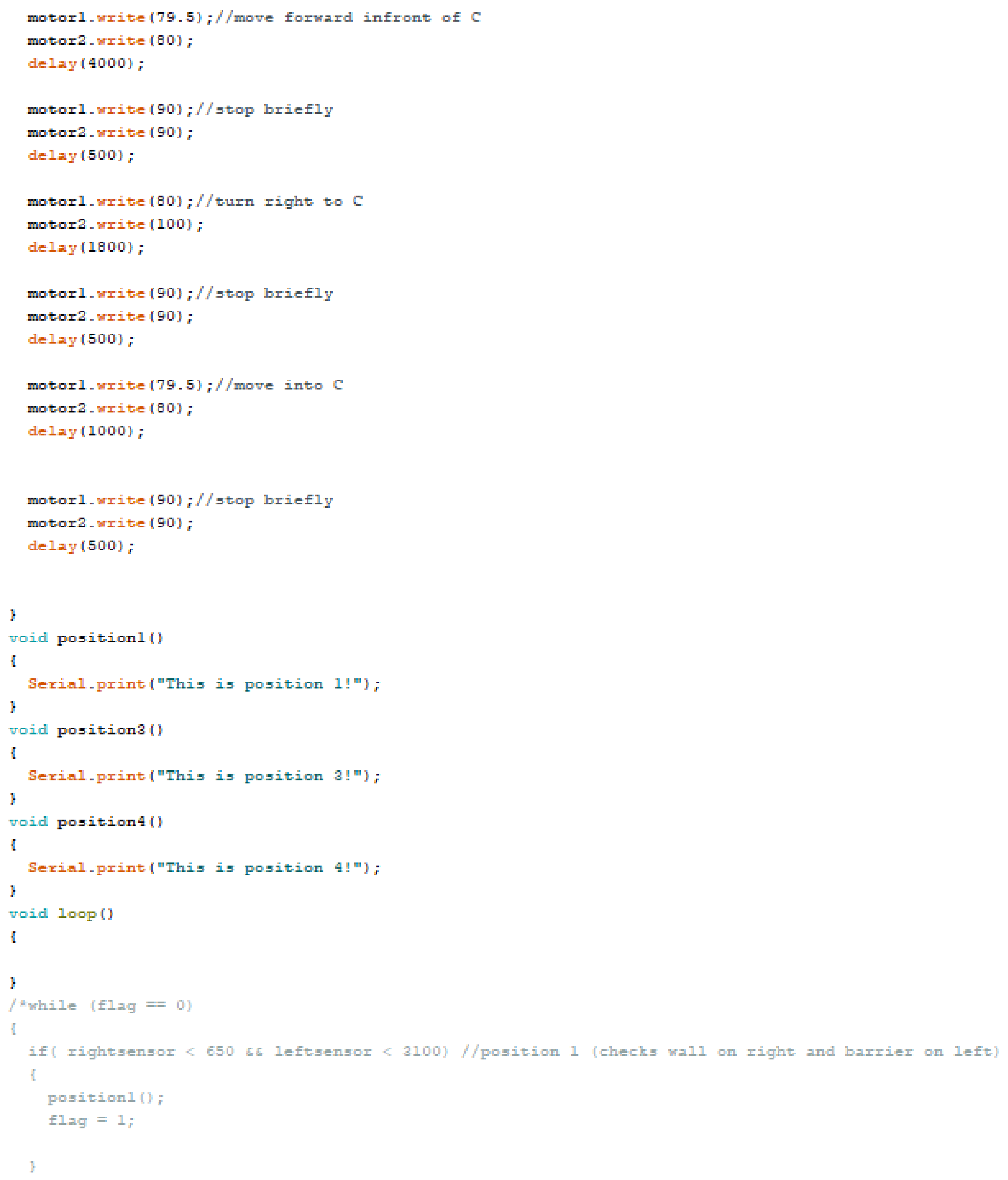
```
motor1.write(90);//stop briefly
motor2.write(90);
delay(500);

motor1.write(80);//turn right
motor2.write(100);
delay(900);

motor1.write(90);//stop briefly
motor2.write(90);
delay(500);

motor1.write(79.5);//move forward infront of B
motor2.write(80);
delay(1000);

motor1.write(90);//stop briefly
motor2.write(90);
delay(500);

motor1.write(80);//turn right
motor2.write(100);
delay(900);

motor1.write(90);//stop briefly
motor2.write(90);
delay(500);

motor1.write(79.5);//move forward to B
motor2.write(80);
delay(1300);

motor1.write(90);//stop briefly
motor2.write(90);
delay(500);

motor1.write(101);//backward
motor2.write(100);
delay(3700);

motor1.write(90);//stop briefly
motor2.write(90);
delay(500);

motor1.write(79.5);//move forward to D
motor2.write(80);
delay(3000);

motor1.write(90);//stop briefly
motor2.write(90);
delay(500);
```

Figure 2.2.3_7 [continued] (above)



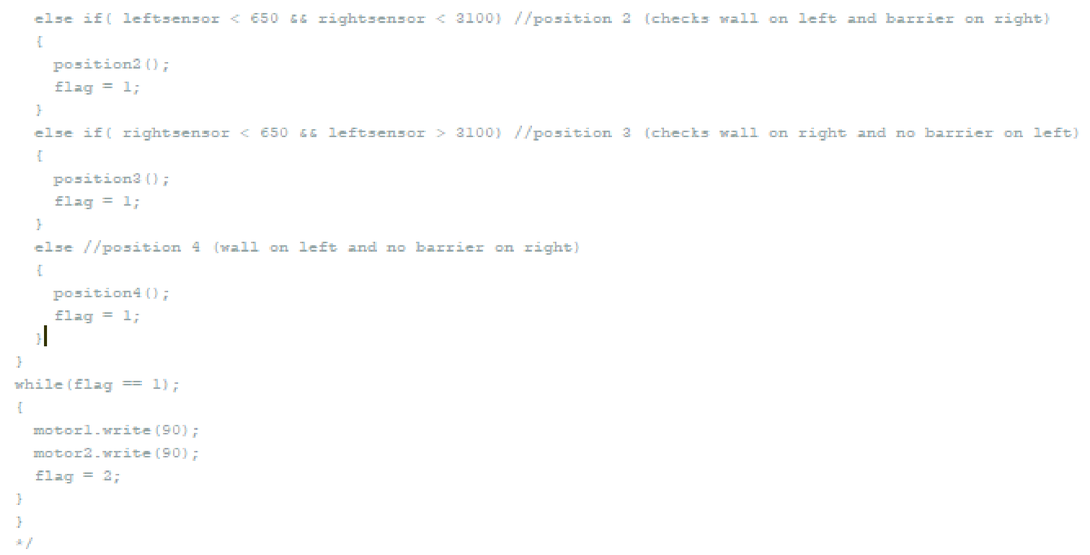
```
 motor1.write(100);//turn left to D
 motor2.write(80);
 delay(700);

 motor1.write(79.5);//move forward to D
 motor2.write(80);
 delay(1500);

 motor1.write(90);//stop briefly
 motor2.write(90);
 delay(500);

 motor1.write(100);//backward
 motor2.write(101);
 delay(1500);

 motor1.write(90);//stop briefly
 motor2.write(90);
 delay(500);


 motor1.write(80);//turn right to A
 motor2.write(100);
 delay(1700);

 motor1.write(90);//stop briefly
 motor2.write(90);
 delay(500);

 motor1.write(79.5);//move forward to A
 motor2.write(80);
 delay(1300);

 motor1.write(90);//stop briefly
 motor2.write(90);
 delay(500);

 motor1.write(120);//backward
 motor2.write(160);
 delay(500);

motor1.write(80);//turn right
 motor2.write(100);
 delay(900);

 motor1.write(90);//stop briefly
 motor2.write(90);
 delay(500);
```

Figure 2.2.3_7 [continued] (above)



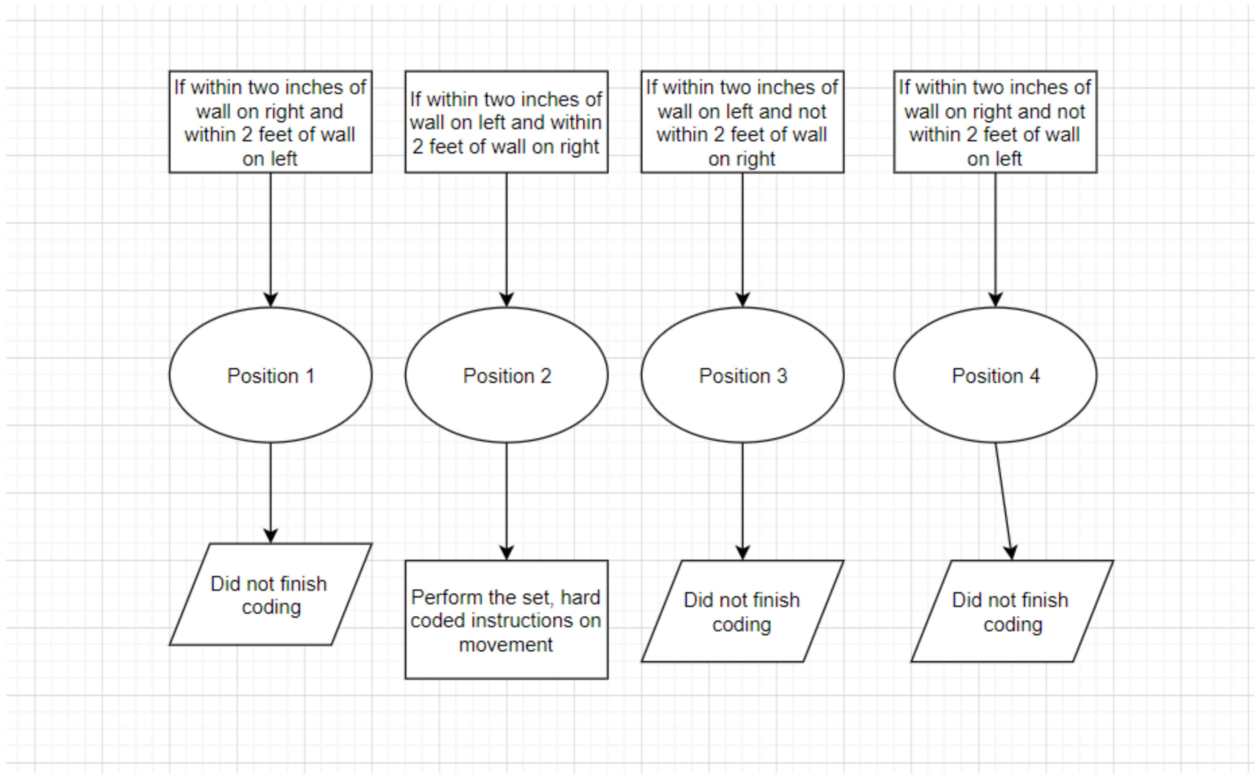
```
  motor1.write(79.5);//move forward infront of C
  motor2.write(80);
  delay(4000);

  motor1.write(90);//stop briefly
  motor2.write(90);
  delay(500);

  motor1.write(80);//turn right to C
  motor2.write(100);
  delay(1800);

  motor1.write(90);//stop briefly
  motor2.write(90);
  delay(500);

  motor1.write(79.5);//move into C
  motor2.write(80);
  delay(1000);


  motor1.write(90);//stop briefly
  motor2.write(90);
  delay(500);


}
void position1()
{
  Serial.print("This is position 1!");
}
void position3()
{
  Serial.print("This is position 3!");
}
void position4()
{
  Serial.print("This is position 4!");
}
void loop()
{

}
/*while (flag == 0)
{
  if( rightsensor < 650 && leftsensor < 3100) //position 1 (checks wall on right and barrier on left)
  {
    position1();
    flag = 1;

  }
```

Figure 2.2.3_7 [continued] (above)



```
  else if( leftsensor < 650 && rightsensor < 3100) //position 2 (checks wall on left and barrier on right)
  {
    position2();
    flag = 1;
  }
  else if( rightsensor < 650 && leftsensor > 3100) //position 3 (checks wall on right and no barrier on left)
  {
    position3();
    flag = 1;
  }
  else //position 4 (wall on left and no barrier on right)
  {
    position4();
    flag = 1;
  }
}
while(flag == 1);
{
  motor1.write(90);
  motor2.write(90);
  flag = 2;
}
}
*/
```

Figure 2.2.3_7 [continued] (above)



If within two inches of wall on right and within 2 feet of wall on left
If within two inches of wall on left and within 2 feet of wall on right
If within two inches of wall on left and not within 2 feet of wall on right
If within two inches of wall on right and not within 2 feet of wall on left
Position 1
Position 2
Position 3
Position 4
Did not finish coding
Perform the set, hard coded instructions on movement
Did not finish coding
Did not finish coding

Figure 2.2.3_8 (above)

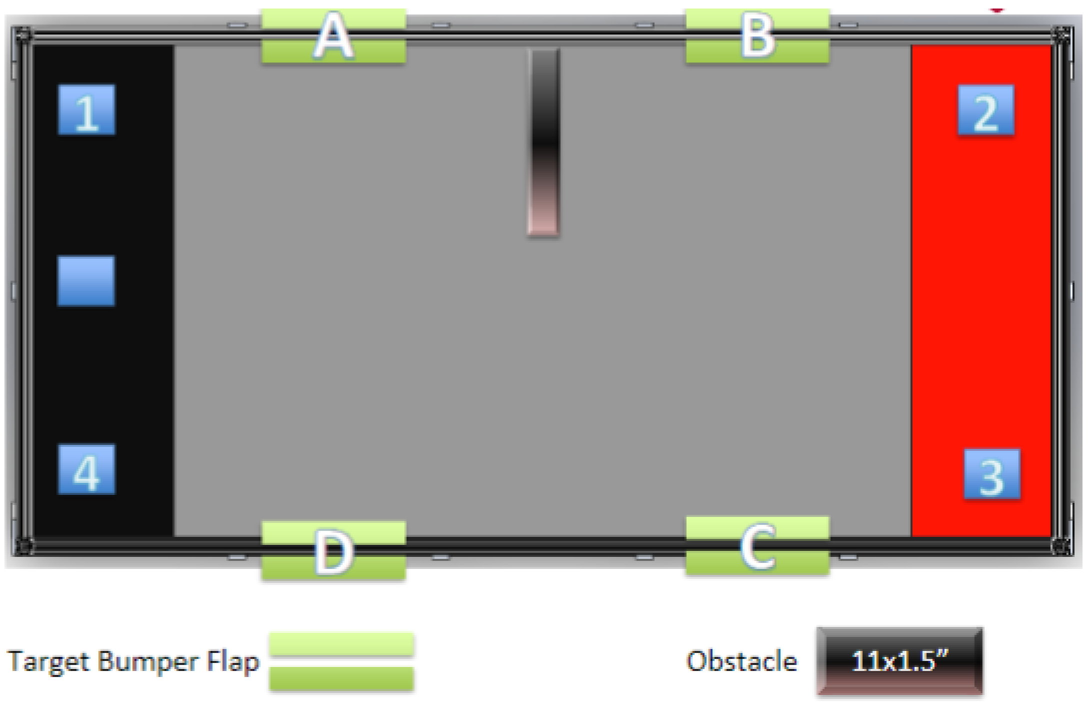


Figure 2.2.3_9 (above)

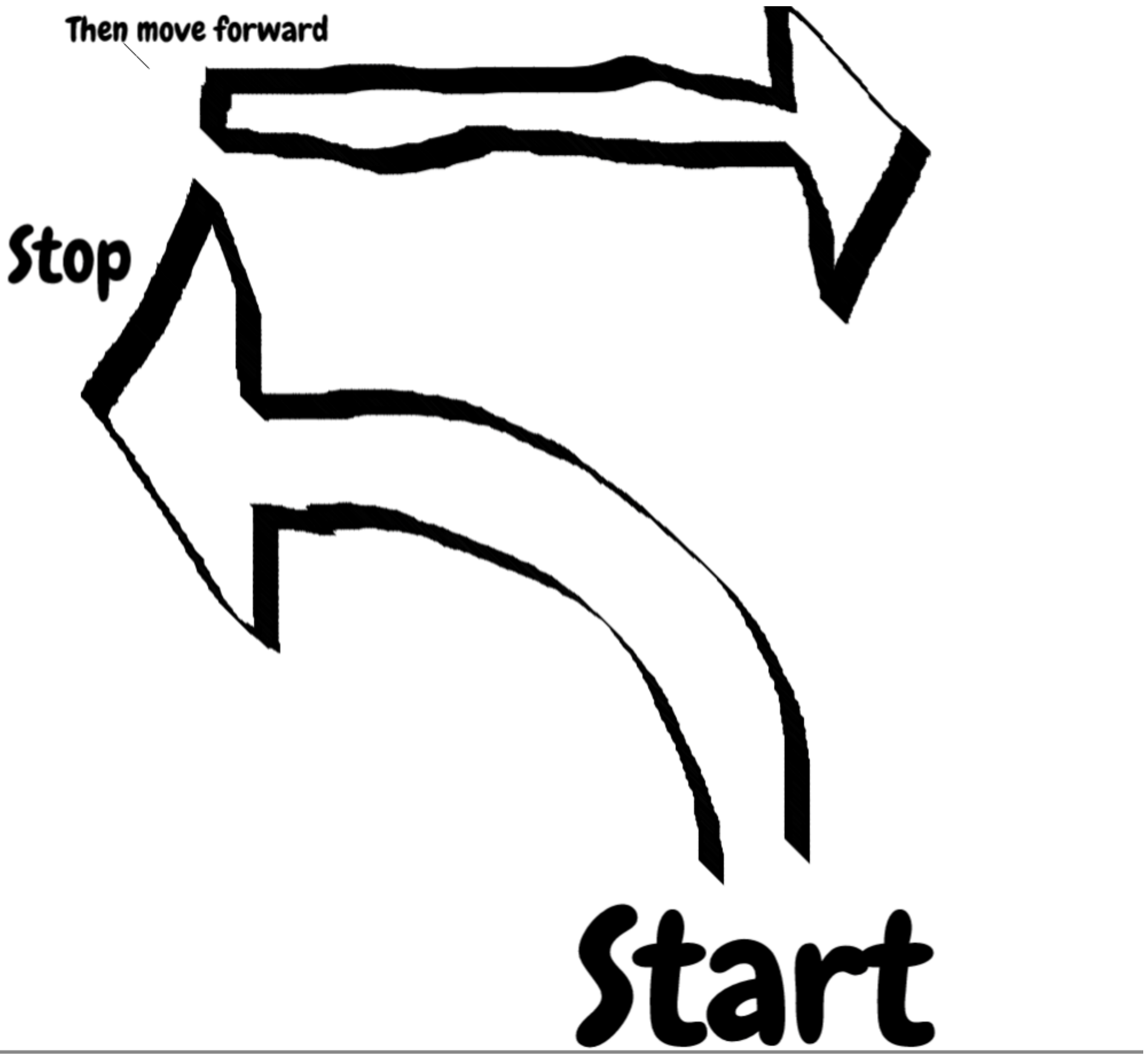


Figure 2.2.3_10 (above)

2.2.4 Electrical and Wiring Design

2.2.4.1

WeMos Board

The WeMos D1 is a ESP8266 WiFi based board that uses the Arduino layout with an operating voltage of 3.3V. The group was able to use the board to execute code and cause the robot to perform certain actions. The board was the central connector between all parts of the hardware including the motor chip and sensors.

(https://cyaninfinite.com/getting-started-with-the-wemos-d1-esp8266-wifi-board/)

2.2.4.2

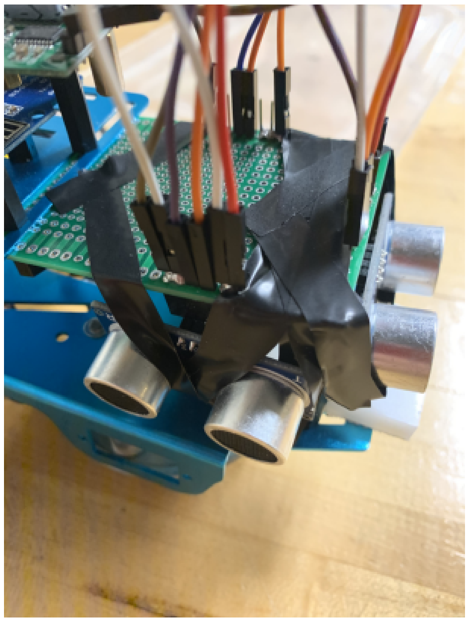
Connections and Issues

The group used a WeMos D2 board. This board was the center of the robot, and controlled the motor chip and took in information from the sensors. It was necessary that wires connected to the WeMos board had secure connections. An extender was mounted so all wires could be connected, as shown in Figure 2.2.4_1. Due to the limited possible connections between the WeMos board and hardware, the group attempted to extend wires. First, soldering two wires together to lengthen was used, but this led to loose connections. The group then tried to tape the connected wires to create a tighter bond, this also failed. Finally, the group had to mount two wires together with dual wire heads, which worked in a stable manner, as shown in Figure 2.2.4_2.

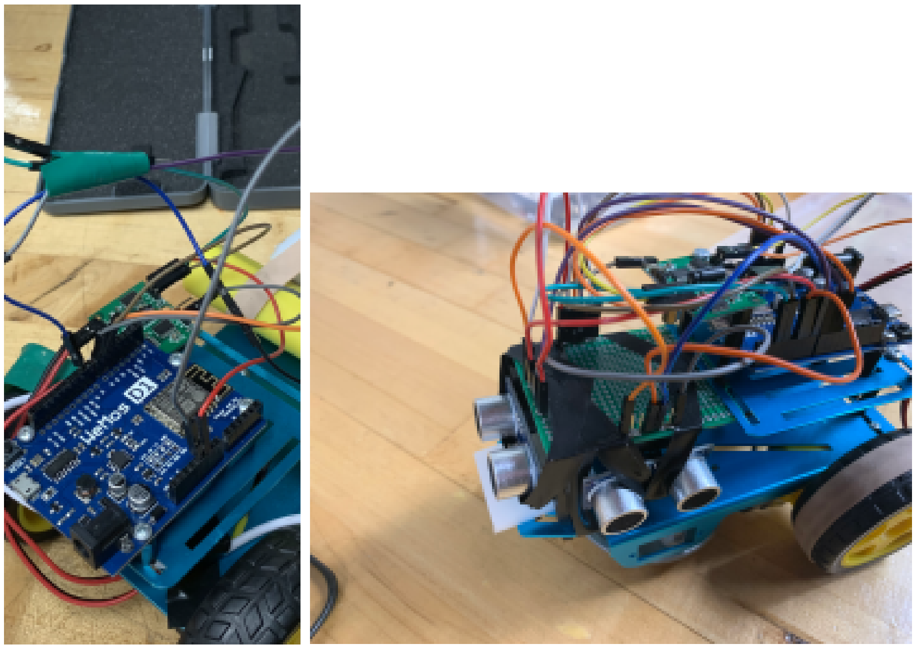
2.2.4.3

Sensor and Motor

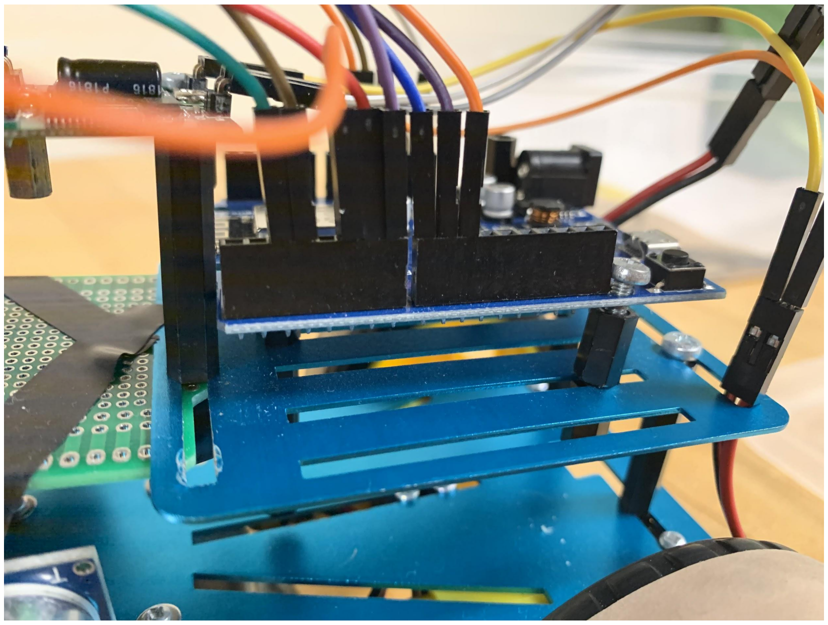
The group used the sensor to allow the robot to detect its distance from walls and determine location. This was used in challenges which involved the turning of the robot. Additionally, in the final competition, it was necessary for the robot to determine location to successfully hit the paddles in the proper order. Wires were used to connect the sensors and the WeMos board. The sensors were placed on the front of the robot and wires were soldered to the breadboard for a more stable connection, as shown in Figure 2.2.4_3. There were many wire connection issues faced by the group, especially inhibiting the group's performance in the Nascar Challenge. Eventually it was necessary to replace the robot fully due to issues with the hardware, including being unable to diagnose which wire/connection was consistently faulty. The motor chip was connected to the motors and WeMos board. Four wires were used to connect the wheels and motors chip, and connections between the motor chip and WeMos board were also created, as shown in Figure 2.2.4_4.



The green board is the extended board, and we mount three sensors on it(figure 2.2.4_1)



We try to tape the wire in order to let it connected well, also we try to mount three wires together then tape its(figure 2.2.4_2)


(figure 2.2.3_4)

2.2.5 Integration Testing and Evaluation

There were a lot of problems that showed up. In the beginning, our group just simply entered the code step by step. In the second mission, the group tried to use the sensor to let the robot figured out when to make a turn, but it didn't work so well. After many replacements of sensors and soldering wires did the connection become stable, allowing the robot to perform adequately. In the third mission, the team used the Serial Monitor to figure out where the robot was, then follow the sequence the robot was assigned. As a team, we were able to create the code to detect where it was located, but we only had the logic in the code to move from one starting area due to a lack of time. Since the robot didn't work for a large amount of time, the team went through a lengthy process of changing the code, changing out the frame, changing the motor chip, changing the board, changing the wheels, until finally we decided it was best to inherit an already working robot. The team had to rewrite the code to make it compatible, but it worked out in the end.We also recorded every test to look back on to see what had to be changed. With the code not having specific units for movement and speed, the serial monitor was used as a benchmark for roughly estimating when to turn and other logic of the sort. The team would hold the robot two inches away from the wall and make a note of what number the serial number gave to have a rough estimate on how to adjust accordingly. The serial Monitor showed the distance from the wall to help us determine when to let the robot make a turn.

2.2.6 Final Competition

The group was assigned the paddle sequence BDAC. To achieve this the group chose to start from a chosen position. The group chose to begin from position two. To do this the robot was coded to take predetermined actions based on the starting point to hit each bumper in order. The robot successfully waited the mandatory 15 seconds before starting. The robot was able to move to and hit bumper B, bumper D, and bumper A in the correct order. However, the robot was unable to hit bumper D, instead the robot veered off course and became trapped in a corner of the arena, which ended the trial. These actions occurred in about 20 seconds, not taking the initial 15 into account. These results indicate a relative, but not whole success, as the robot was able to independently hit 3 of the 4 bumpers in the proper order. However, this was partially due to the group's decision to start in a predetermined, as opposed to random, starting position.

Most likely the issue involved either the wheels or the software of the robot. The wheels gave the group issue throughout the testing process, as the group could not find the values that would allow the robot to turn at a 90-degree angle for every test. It typically worked during some trials seemingly at random. This would result in the robot veering off track with no way to self-correct itself. This could explain the error that occurred during the final test.

An alternative explanation could be an issue with the software. The robot was not adequately programmed with any self-correction in case of error. The group had planned to program the robot to perfectly execute commands rather than plan for possible errors. This would result in the robot to continue executing code regardless of whether it was on track or not. Therefore, when veering off-course the robot continued into the corner instead of correcting and executing code to reach paddle C.

2.3 Conclusion

The group was able to give each challenge a valid attempt. The group holds that although they needed to replace their robot, the problems that arose were due to faulty wires and factors outside of the group's control. The group maintains their hardware design would have successfully completed the challenges at the same caliber as was reached if it were able to adequately execute the group's code. The group maintains that although unable to complete the Final Competition, the robot was overall successful. The robot was able to successfully complete the other challenges, and it did complete a majority of the Final Competition. The group learned to distribute work based on strengths and learned the design process involves a majority of testing and redesign and tweaking to reach success.

The group recommends that that all equipment must be tested prior to assembly to guarantee a fully functioning project. The group believes that the process does enhance teamwork and teaches students to accept failure, even if these lessons do not seem apparent or necessary in the beginning of the design process. The group recommends that each member has a role that maximizes the use of their strengths, while still being able to work as a single unit.

Overall, the group faced many successes and failures during each of the tests. This allowed the group to create the best robot the group could, even if it was not fully successful. The group maintains that their robot was a success.