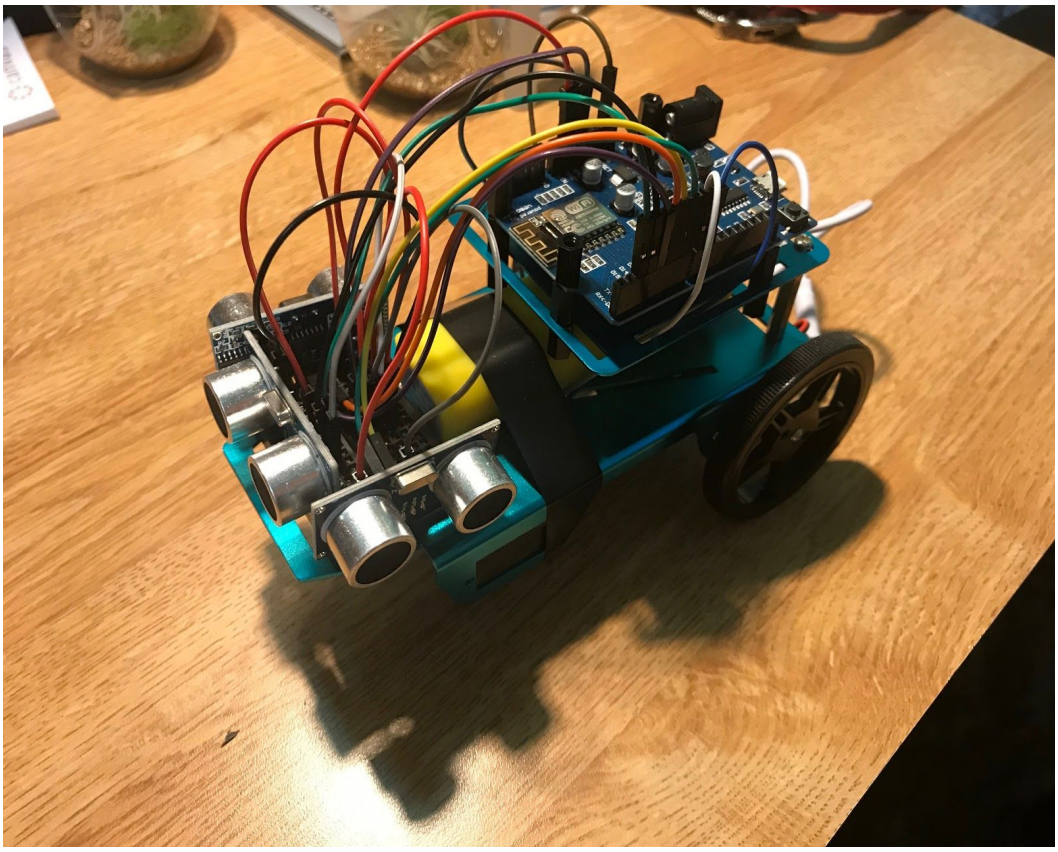
FINAL REPORT

Group 1



Ben Schablin, Joe Demateis, and Jakob Niglio

12/06/18

Fall 2018 E121-P Professor Barbetta

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

- Ben Schablin - Joe Demateis - Jakob Niglio

Abstract:

This project's objective was to code a robot using a WeMos R1 board to hit a series of targets with an arena. Requirements were given for the team to meet regarding the mechanical properties of the robot and the manner in which the robot completes the course within the arena. The team utilized a system of trial and error to determine the best way for the robot to accomplish the task. The code for the robot utilized a state machine in order to instruct the robot to complete specific tasks in the specified order in a way that was organized for both the robot and a reader of the code. After a month of trials and readjusting code, the team completed the course on the first attempt on the day of the final competition with a time of thirty-seven seconds. The process used by the team proved to be effective and resulted in a successful run receiving full credit.

Table Of Contents:

Introduction:

The purpose of this report was to provide an insight into what Group 1 accomplished this Fall Semester. There were many teams in this course, each having a unique robot. So, this report shows how the team built their robot and how they went about tackling all of the obstacles that were presented throughout the semester. There were endless ways one could have coded and built the robot to fit the way the group wanted it, so this is a summary of how Group 1 coded and designed it.

During the course of this semester, the team was challenged with making a robot car that functioned fully on its own. They were given a kit of parts, which they had to put together in a unique way that they thought was best. Before the final test, which will be discussed later, there were two characterization/calibrations. The tests were designed to figure out the behaviors of the motors and wheels. Also, they had to calibrate and characterize the ultrasonic ping sensors. The team did wheel characterization and calibration by counting the time it takes a wheel to rotate a full cycle at different speeds. This calibrations and characterizations allowed them to know the values to input into our code for different speeds for each wheel and how the sensor reads different distances. After learning more about the robot and how it behaved when given different values for different speeds and also what a sensor reads at certain distances, the team was given four challenges to put what they learned into action. The first was to make the robot travel straight for five feet and then stop within two inches of a wall. The second was to make the robot swerve and maneuver around an object in its way and continue going straight. The third was to parallel park the robot between two objects that were two feet apart. The final one was to go around the track with an obstacle in the middle. This was referred to as the NASCAR race. After the team completed these tasks, the final challenge was assigned. For

this, the team had to use all the skills they learned in the previous tasks. The task was to hit four different bumpers in a specific order from a specific starting spot and return to the spot after hitting all of them. The group was not allowed to hit the walls or the middle obstacle. Figure 1 depicts the layout of the arena and the path the team's robot needed to take to complete the course. The arena was a rectangle surrounded by four walls, which were mostly glass, that were about a foot high. In the middle of the arena, there was a two-foot-long obstacle. The team was specifically assigned with starting in spot 1, facing spot 4, and hitting bumpers in the order CBDA.

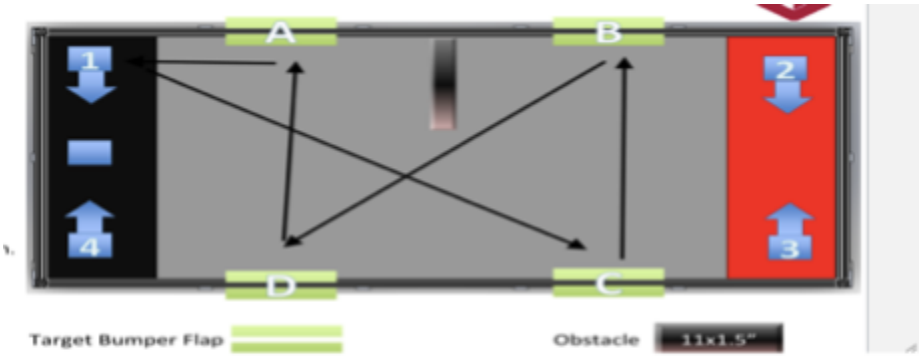


Figure 1. The arena layout and the path needed to complete the course by the team's robot.

To conquer all of these tasks and challenges, the team utilized a system of trial and error. This strategy was effective because it perfected each step and then built off of what was already known to work. This helped pinpoint troubleshooting and increase the effectiveness of the team's work.

Discussion:

Requirements

At the beginning of the project, the team was provided with a list of objectives that the robot must meet in order to successfully complete the course. Some of these objectives dealt with the mechanical design of the robot, including a height maximum of six inches and a restriction on eligible materials to only parts provided in the kit with some exception. In order to meet these physical requirements, the team constructed the robot in such a way that only a small board was placed above the robot's chassis. This small plate was mounted above the chassis to hold the WeMos board. This set-up allowed for the sensors to be mounted using a breadboard at the front of robot chassis, and for the battery to be placed on top of the chassis underneath the elevated plate.

Requirements regarding the manner in which the robot completed its task in the course were also provided to the team. These requirements included an order in which to hit the targets (which for this team was CBDA), an initial delay of ten to fifteen seconds once the robot was placed in the arena before moving, a requirement that the robot must be autonomous once placed in the arena, and a restriction against hitting objects in the course, such as walls, other than the intended targets. Various techniques within both the code and the physical design of the robot were utilized by the team in order to meet these requirements. In order to maintain the autonomy of the robot and still accomplish the task of the course, the team programmed the robot to operate utilizing three Ultrasonic Ping Sensors in order to allow the robot to orient itself in the arena without intervention by the team. The robot was also coded to use the readings of these sensors in order to hit the targets in the assigned order, as well as to avoid hitting walls within the arena. Within the setup portion of the robot's code, a delay of 11,000 milliseconds

was put in order to have the robot remain at rest for 10 seconds before beginning to maneuver the arena.

System Design:

System Design can be defined as how the systems interact with each other and how each system works. For this project, the main systems were the breadboard, the WeMos R1 board, the motors, and the battery. The main source of power was a portable battery charger which was plugged in directly to the WeMos board which everything was connected to, which gave everything else power. All of these systems were connected the body of the robot, the blue chassis. The Ultrasonic Ping sensors were plugged into the WeMos board with the wires. Each part of the pin for Ultrasonic Ping sensors had corresponding input on the WeMos board. However, one cannot plug Echo pin into D8, D9, or D10. Figure 2 shows this wiring system and Figure 3 shows an example of how the power flows from the battery to the motors through the robot's system.

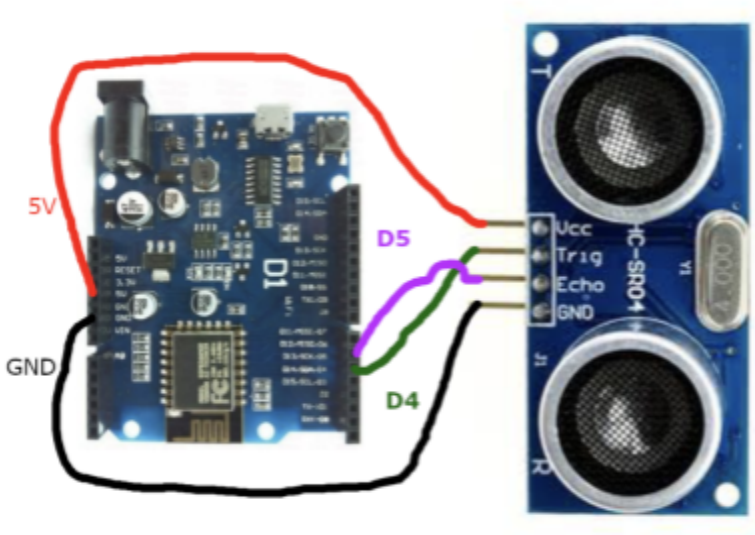


Figure 2. Wiring system showing communication between the ultrasonic ping sensor and the WeMos Board.

Figure 3. Diagram showing the path the power takes between the battery and the robot's motor.

The main system in the design would have to be the WeMos R1 board. It receives the code from the compiler Arduino and allows it to relay the messages to the motors. This enables the robot's wheel to go at different speeds and different directions. For different tasks, one would need to use different values in the code. It is basically a microcontroller similar to the Arduino Uno. It has integrated WiFi, which was not necessary for the building and use of our robot. It also operates at 3.3 volts. A WeMos R1 board was originally provided. However, very late in the process, the WeMos R1 stopped working. A WeMos R2 was originally provided as a replacement, but an additional was WeMos R1 board was found soon after and the team was able to avoid having to readjust their code and wiring to the different board type. The black holes in the two rows found at the top and bottom are the ports where wires can be plugged into. Each port has a specific objective it can do. For example, there are multiple 5-volt inputs where wires can be plugged into to transport power to another portion. Figure 4 shows the R1 style WeMos board utilized by the team.

Figure 4. R1 style WeMos board used by the team to code the robot and connect the different aspects of the system together.

Figure 5 highlights the “brain” of the WeMos board. It entered the US Market in 2014. It is similar to the ATMEGA found in Arduino microcontrollers.

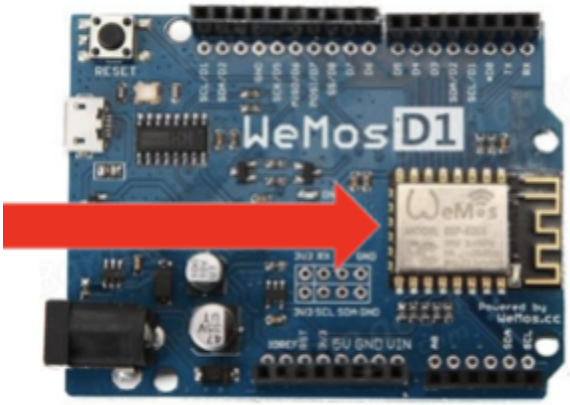


Figure 5. Highlights the “brain” of the WeMos board.

Another system in the design is the Ultrasonic Ping Sensors. The teams were provided

with five of them but ended up using only three. One in the front, one on the right side, and then one on the left side. The Ultrasonic Ping Sensors are similar to a submarine's sonar or a bat's echolocation. The sensor sends out a pulse and then receives the bounce back pulse. The pin that is labeled trigger is used to signal module to send out a sonic pulse. The pin that is labeled echo is used to detect and receive the bounce back of the signal. These Ultrasonic Ping Sensors used to make the robot be able to stop when it reaches a certain distance from the wall. To wire these sensors, we had to put them on the breadboard, which will be discussed later on, and use the wires to connect it correctly to the WeMos board. The echo and trigger pins had to connect to the input label D# (some number). The ground and volt had to be connected to their respective inputs on the WeMos board too. When using the sensors in coding on Arduino, it would be considered closed-loop programming. Which means that the sensors are constantly sending and receiving feedback, or the pulse, to the compiler and the WeMos board. Some specifications of the board are the working voltage is DC 5 volts, the working current is 15mA, the working frequency is 40 Hz, the maximum range is 4 meters, the minimum range is 2 centimeters, and the measuring angle is 15 degrees. The dimensions of the sensor are 45x20x15 millimeters. Figure 6 shows the Ultrasonic Ping Sensor and Figure 7 shows the process of reflection that allows them to work.

Figure 6. Ultrasonic Ping Sensor used to help robot be aware of its surroundings.

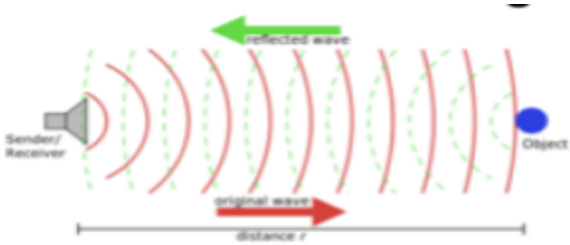


Figure 7. Displays the process of reflection which allows the sensors to determine their distance from an object

The motors of are one of the systems that were key to making the robot run too. The motors were powered by 5 volts and they were directly connected to the wheels. The wires ran through the bottom of the chassis to the WeMos board, where they were connected to a 5-volt port.

The breadboard was where the Ultrasonic Ping Sensors were connected to and where most of the wires were connected too. This allowed us to make circuits and circulate the power from the WeMos to the other systems. It is a great system because it does not require any soldering. Figure 8 shows the style of breadboard used by the team.

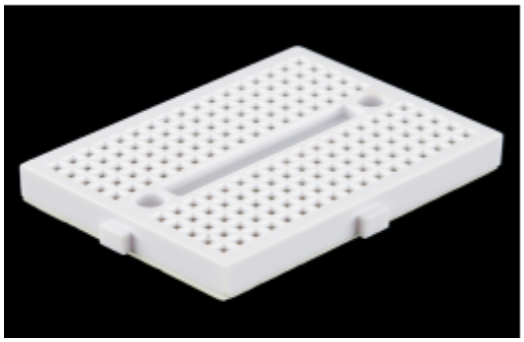


Figure 8. Breadboard used to efficiently wire the Ultrasonic Ping Sensors to the WeMos board.

Mechanical Design:

The group decided to first mount an ultrasonic sensor on the front of the robot by plugging it into the breadboard. The team thought that one sensor would be sufficient to perform the assigned tasks, however, it quickly became evident that more sensors were necessary to pass complete each assignment. The first additional sensor was a left sensor that was mounted on the robot's left side. This was added in order to use it for a wall-follow function where the robot would be able to adjust itself to remain a specified distance away from the wall. This left sensor was added during the NASCAR test because the robot needed to hug the inside wall, using the left sensor, as well as still take readings with the front sensor to know when it needed to make a turn.

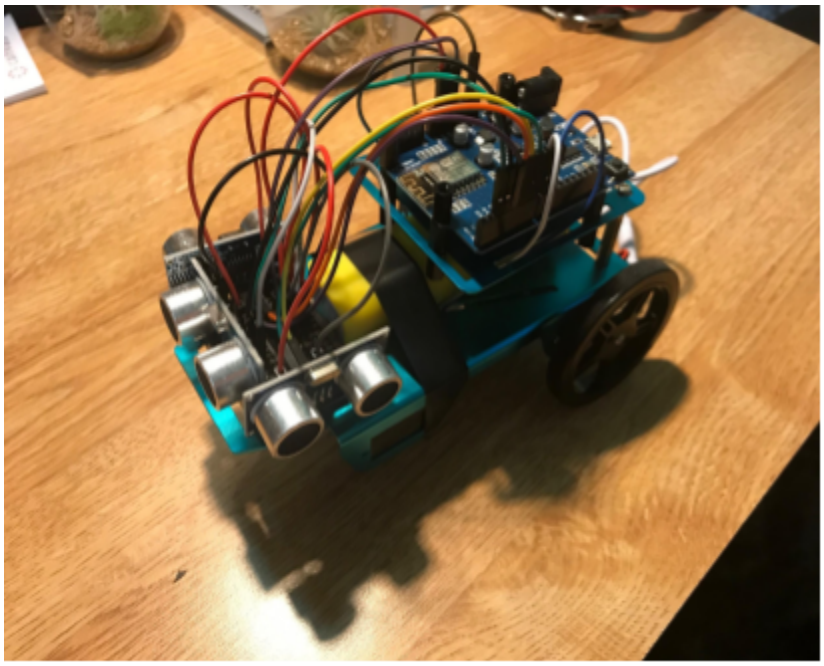


Figure 9. View of completed robot.

Two sensors were still not enough for the robot once the final project was revealed. In order to travel in the arena, the wall-follow function in the code would need to work with the wall on both sides of the robot. This sensor was added on the opposite side of the breadboard that the previous sensor was mounted. The wall-follow function was able to be applied to the right sensor and this allowed the robot to be able to roam around the arena easier than if there were fewer sensors.

The battery was held in place using a strip of electrical tape, as seen above in Figure 9. The battery also helps hold up the Wemos board as well. The Wemos board is held in place by four screws and the USB port faces the rear of the robot so that the battery cord does not have to stretch far to plug in. As shown in Figure 10 below, the caster wheel was taped in place because it was consistently interfering with the accuracy of turns during testing.

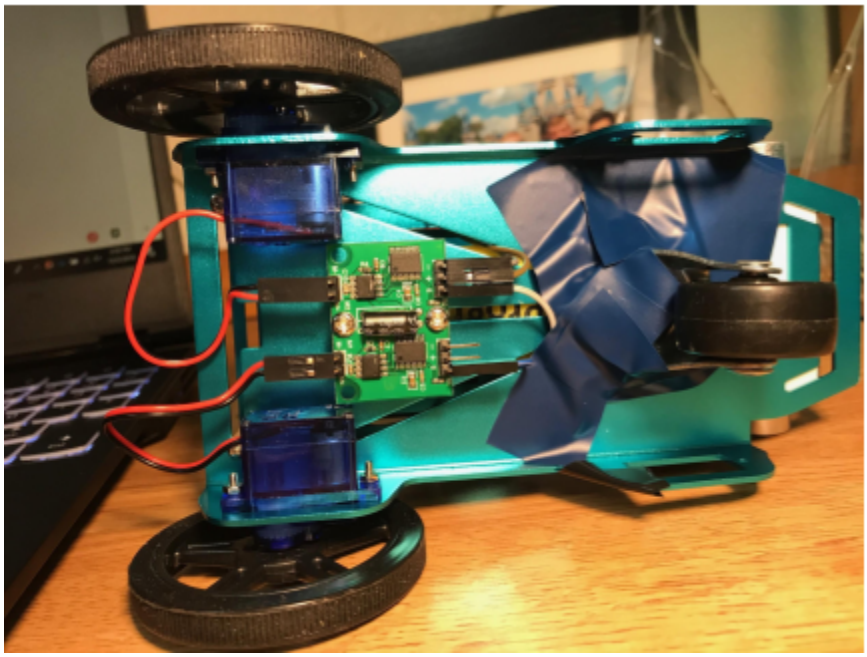


Figure 10. Bottom view of the robot. Shows the taping of the caster to improve the accuracy of the robots movements.

Software Design and Coding:

The team utilized multiple strategies in order to write a code that effectively provided the robot with clear instructions to perform its task while also making it easy for someone to read and understand. One major way in which the team did this was utilizing functions within the code to perform regularly repeated motions, such as following a wall or turning a specific direction. The code for these functions was written in the space between the robot's setup function and main function, in which the functions were called in order for the robot to perform the action each function instructed it to.

Two of these functions were functions that allowed the robot to move forward in a relatively straight direction by following the wall adjacent to either side of the robot, one for the left side of the robot and one for the right side. These functions were referred to as "follow wall functions," and were developed by the team during the completion of the NASCAR left-hand loop portion of the road test. These functions instructed the robot to take in readings from the ultrasonic ping sensor on one of its sides and compare it to an arbitrary value. This value could change each time the function was called during the main loop function of the robots code. If the reading from the sensor was less than the value that the code instructed the robot to be from the wall, the robot would slow down the wheel farthest from the wall in order for it to move farther away from the wall and thus increase the value being read by the sensor. On the other hand, if the reading from the sensor was greater than the value that the code provided, the robot would slow down the wheel closer to the wall in order for it to move closer to the wall and decrease the value being read by the sensor. This process is illustrated in Figure 11 and Figure 12, which depict the flowcharts for the "FollowWallLeftForward" and "FollowWallRightForward" functions, respectfully.

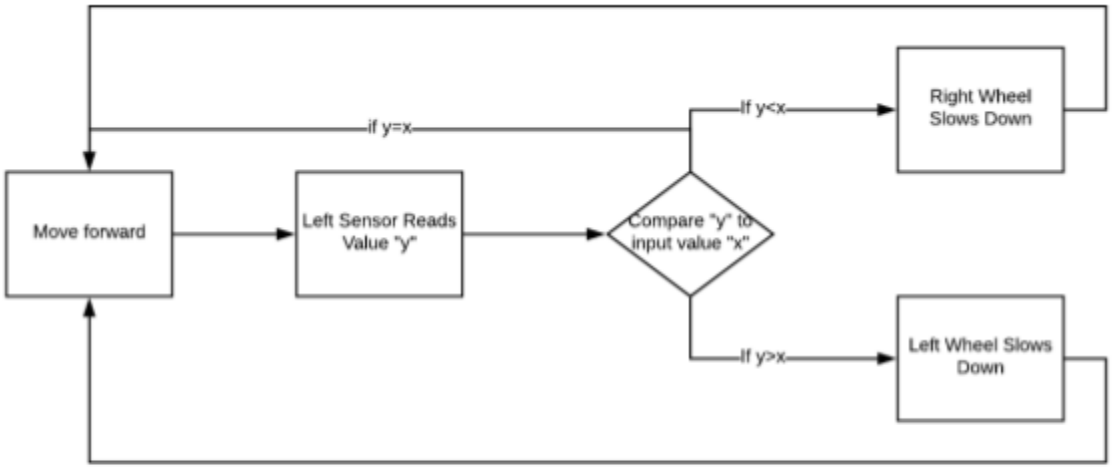


Figure 11. Flowchart for the code for the “FollowWallLeftForward” function.

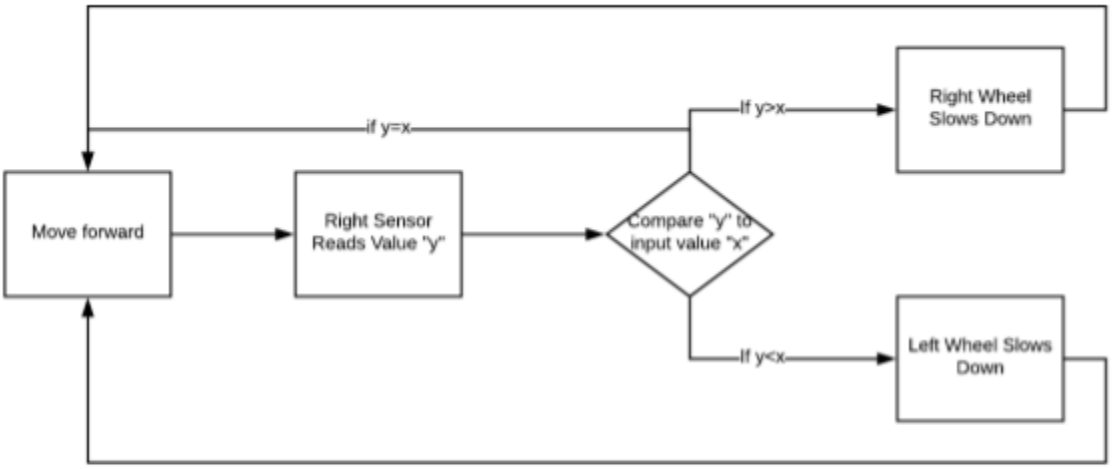


Figure 12. Flowchart showing the code for the “FollowWallRightForward” function.

Other functions that were written include turning functions, which were simpler than follow wall functions. These functions involve one wheel spinning in one direction while the other

wheel spins in the opposite direction. When called in the main loop function, the function required a time input, and that time determined how far the robot turned. Figure 13 shows the flowchart for the “LeftTurn” function.

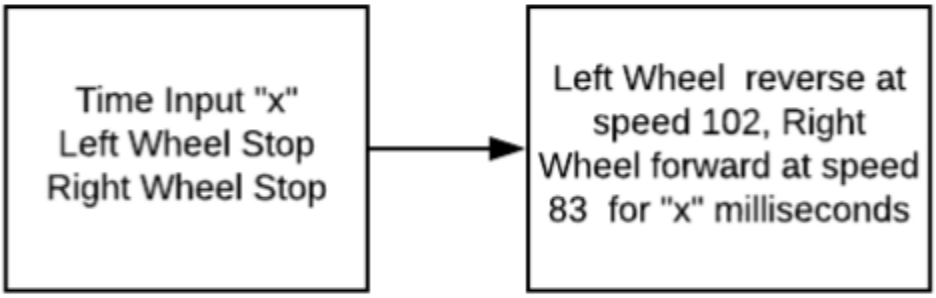


Figure 13. Flowchart showing the code for the “LeftTurn” function.

One unique feature that the team incorporated into the “mainloop” portion of the code is the utilization of state machine system in order to organize the way in which the code is executed by the robot. This process worked by including a series of if statement testing the value of the integer variable “state” ex) if state ==1 {}. Each of these states incorporated a specific portion of the motions that the robot needed to perform in order to complete the course within the arena, and at the end of each state the value of the variable “state” is changed to the next number, so that when the mainloop functions looped back to the beginning, it ran the code written in the if statement that is linked to that new value of the “state” variable. Originally, the team incorporated several motions within each state of the state machine. However, this led the robot's motions to become inaccurate. In an attempt to refine the robot's motions, the team broke up each state of the state machine to contain smaller amounts of motions per state. This improved the robot's motions, but some maneuvers were still inconsistent in their execution by the robot. To combat this, the team incorporated some short delays within the code in order to

give a short break between major steps in the completion of the course. This proved to be effective, as the robot then performed the tasks that were assigned to each state with higher accuracy when each state contained instructions for a smaller amount of motions.

The robots motions were designed by the team to hit the targets in the order C-B-D-A. The code was written to accomplish this when the robot begins at position one, as the team was instructed by the professor for the robot to begin at the starting position diagonal to the first target in their assigned sequence. Because of this, the robot's motions that rely on sensor readings are coded for the robot to follow a designated path that begins at position one. The complete path of the robot's motions to complete the course in this manner is depicted in Figure 14.

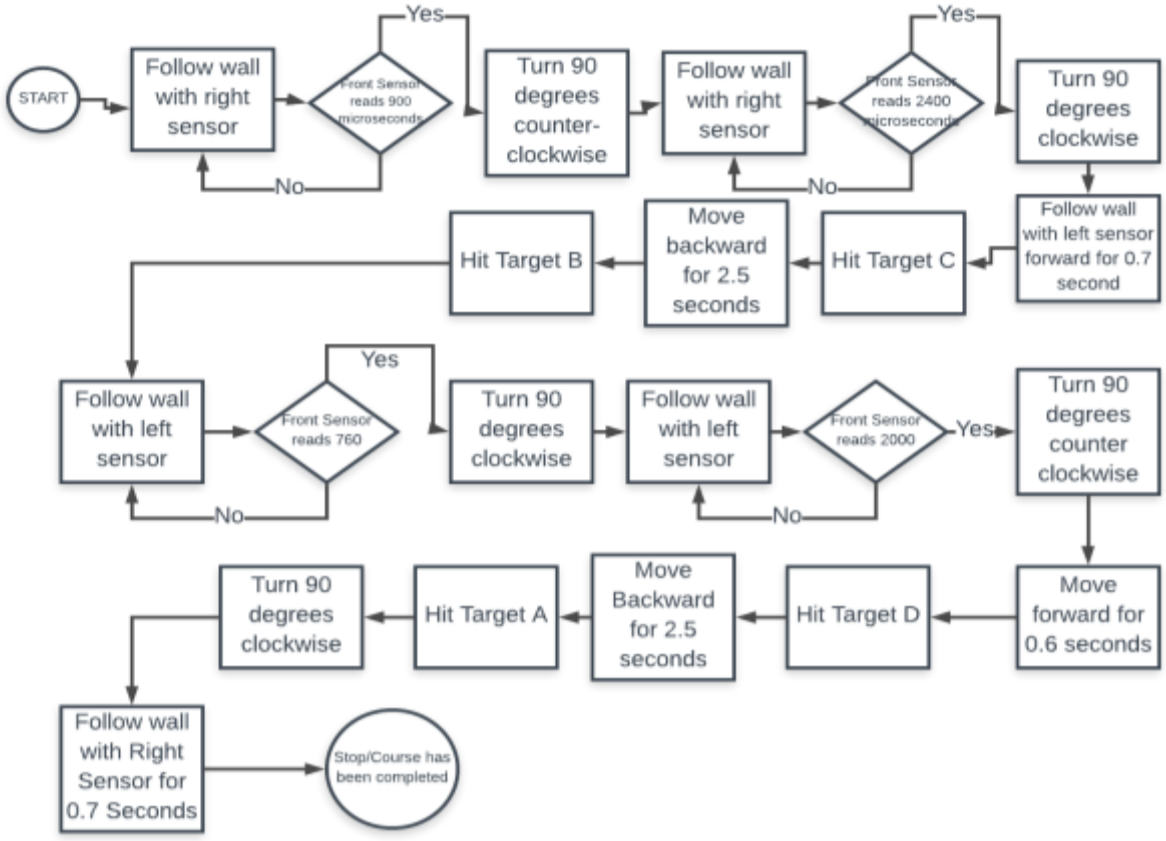


Figure 14. Flowchart for the complete projected path of the robots motions within the arena.

The team programmed their robot to run on a combination of both open-loop and closed-loop functions in order to accomplish the task of completing the arena, as can be seen within the flowchart. Portions of the code where closed-loop programming was utilized are where the robot's actions are dependent on a sensor reading to instruct the robot to perform the next task in its code. These moments can be seen in the flowchart where there are diamonds that correspond to the robot reading a certain sensor value. The robot repeats the action that comes before that diamond in the flowchart until that sensor reading is read, and then goes onto the next move it is instructed to perform. This is found in the box after the diamond that contains the sensor reading condition. These instances where the robot receives a sensor reading often correspond to moments where the value of the “state” variable is changed and the preceding action is often the first instruction of the next state in the series of state machines. An example of this within the code can be seen in Figure 15.

```
else if (state == 8){
  followWallLeftForward(1100);
  if (frontSensor <= 2000)
    state = 9;
}
```

Figure 15. State 8 of the mainloop function which utilizes closed-loop programming to instruct the robot to move between targets B and D.

Examples of open-loop programming were utilized by the team for instances where the distance that needed to be traveled or the time needed to perform the action was relatively short. These instructions within the code relied upon timing to instruct the robot on how long to perform a task an when to move onto the next task. This was done due to higher accuracy for predicting the exact location of the robot within the course within this method. Over long

distances, open-loop programming is helpful because it gets the robot to move in a desired direction and stop in a specified area within the arena. However, when trying to move short distances, particular moments when the robot is hitting the targets, the closed-loop method became unreliable due to the limitation of the sensors. The sensors are not accurate within a range of two centimeters to an object, so hitting an object head-on required the use of timing to guarantee the robot perform the way it is intended to. An example of this within the code can be seen in Figure 16.

Furthermore, there are instances where direct motor values were coded within the mainloop function of the code rather than utilizing follow wall functions. These often occurred in the same areas of the code as closed-loop programs due to the higher accuracy they provided. When performing the motor characterization testing in the beginning part of the semester, the time made adjustments to accommodate the wheels moving at slightly different speeds. However, over long distances, it was hard for the team to accommodate for this and the robot often drifted to the side rather than moving straight. This led the team to develop, and heavily utilize, follow wall functions. However,

```
//hits D and A
  else if (state == 9) {
    leftTurn(700);
    right.write(80);
    left.write(81);
    delay(600);

    right.write(90);
    left.write(90);
    delay(500);

    right.write(105);
    left.write(107);
    delay(2500);

    right.write(90);
    left.write(90);
    delay(500);
    state = 10;
  }
```

Figure 16. State 9 of the code for the mainloop function which utilizes open-loop programming to bring the robot between targets D and A. Also shows the use of direct coding of motors rather than using follow wall functions to move the robot forward.

although these functions allowed the robot to get to a location directly in front of it, the path the robot took was not perfectly straight. It was a swaying pattern. When trying to utilize these functions to perform the maneuvers to hit the target at close distances, these swaying motions sometimes resulted in the robot hitting the wall to the side of the target rather than the target itself. The team then tried programming the motors directly, and because the distances traveled in these moments were short enough that the drift did not largely impact the trajectory of the robot, this method proved to be more accurate and reliable than utilizing follow wall functions.

Electrical and Wiring Design:

Wemos Boards are WiFi development boards used to program all different kinds of electrical devices. In the scope of this project, the Wemos Board is used to control the robot. This board is wired to the motors and the sensors on the robot.

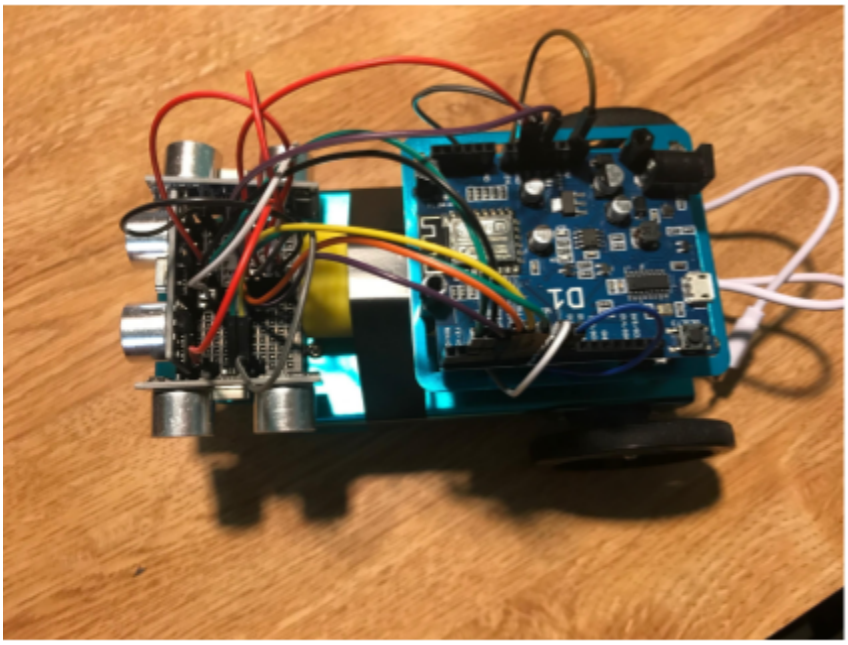


Figure 17. Top View of the Robot. Shows need for the breadboard in the front to properly wire all components of the system to the WeMos board.

As shown in Figure 17, there were not enough ports on the Wemos Board to support everything that needed to be wired to the board. For example, there are wires that connect to the breadboard from the 5 volt port on the Wemos Board. The breadboard is then used to extend the number of wires that can access the 5 volt port on the Wemos Board. The same tactic is used on for the ground port of the Wemos Board. An example of this wiring design can be found in Figure 18.

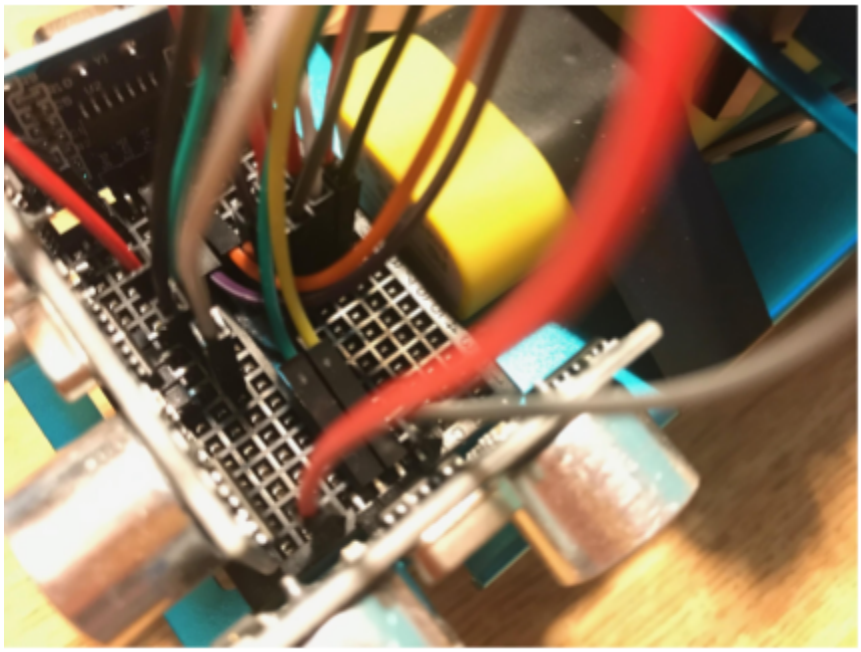


Figure 18. Upclose picture of wiring on the breadboard. Wiring style was used to overcome to limited amount of 5V ports on the WeMos board.

To connect the sensors, the middle pins were bent so that wires could be connected and the other, outer pins were plugged into the breadboard. To connect these pins, wires were connected to corresponding spots on the breadboard. The system used to color code the wires was pretty basic, although not much thought was put into it. Red was typically used for the 5-volt

ports. The group used a gray wire for the ground port. The other color wires were used for the different pins needed for the sensors. Figure 19 shows which input ports on the WeMos board were used for each sensor and motor input and output.

D9	D8	D7	D6	D5	D4	D3	D2
Left Motor	Right Motor	Left Sensor Trigger	Left Sensor Echo	Right Sensor Trigger	Right Sensor Echo	Front Sensor Echo	Front Sensor Trigger

Figure 19. Chart showing the ports in the WeMos board for each wiring connection for the motor and sensors.

Integration Testing and Evaluation:

The approach used to test and integrate all of the robot components was most likely similar to other groups' tactics. Essentially, trial and error was the most prevalent approach. After an error was spotted, the group decided to revert to the code and determine the problem from there. Before any sensors were mounted to the robot, they were tested with the Serial Monitor to make sure the sensors were adequately reading the distance. The Serial Monitor was also used to test how many microseconds were in an inch so that the correct distance could be programmed for the robot to test for when making decisions. Later in the project, the team simply placed the robot in a desired spot in the arena and used the Serial Monitor to determine the number of microseconds the robot needed to be away from the wall.

Eventually, the group found that the sensors were unreliable at larger distances. This caused the team to keep the distances of the robot from the wall to be as short as possible. The group also used this knowledge to program the number of microseconds to be a smaller amount, less than what the anticipated distance was. This was because of the group realizing

that the robot would make a turn farther than the distance actually specified in the code. By making these adjustments, the robot acts fairly consistently and is able to perform the tasks necessary to succeed. As seen in Figure 20, the sensors were very inconsistent. In this case, the robot was the same distance away from the box and had readings ranging from 693 microseconds to 715 microseconds. The inconsistency of these sensors was a challenge for the group to overcome.

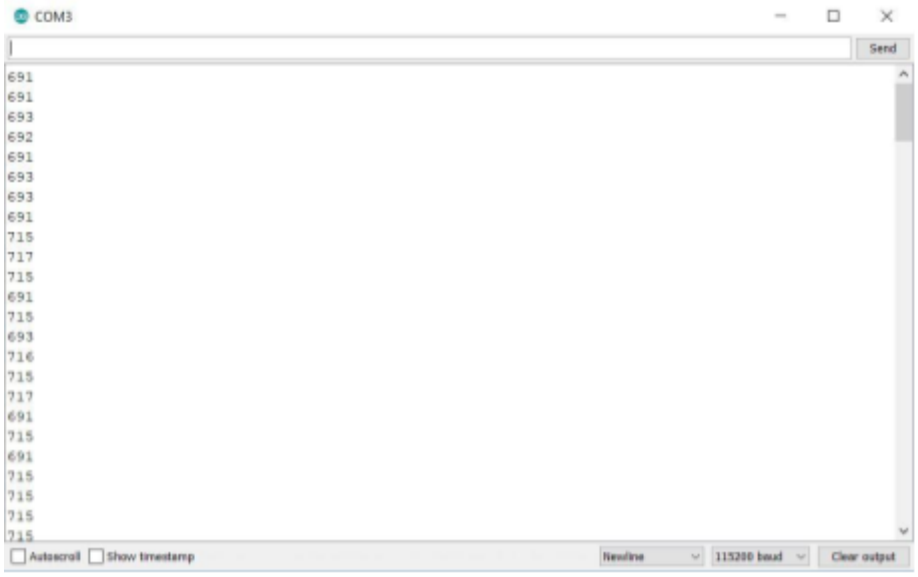


Figure 20. Serial Monitor showing the output of the front sensor when the robot was still. Shows the inconsistency the group needed to overcome.

To determine the speed of each motor, the team put a piece of string on a part of a wheel and counted the revolutions for certain increments of time. This was used as a base to estimate the motor speed for moving the robot, however, more adjustments were required for the robot to move as desired. Figure 21 shows the findings of the motor calibration for motor one and was used to calibrate the motors to properly assess what parameters at which to code the robot throughout the course. The robot speed is in inches per second.

Motor One	
Parameter	Robot Speed
0	19
15	19
30	19
45	19
60	19
75	15.4
80	10.4
85	5.2
89	0.8
90	0
91	0
100	5.5
105	11
110	15.6
120	18.2
140	18.2
160	18.2
180	18.2

Figure 21. Chart showing the findings of the motor calibration test for motor one (right motor). Parameter is the value coded in the program, and robot speed is the speed in inches per second for the given parameter.

Final Competition:

On December 4th, the team competed in Section P's final competition during the last class of the semester. For this competition, different aspects of the robot's performance were evaluated in order to earn different amounts towards the final grade of the project. Hitting the first target, for Group 1 target C, was worth 40 points. Each subsequent target after that was worth 10 points each. Returning to the starting location, for Group 1 location one, was also worth 10 points. The final 20 points were awarded as style points, which were given under the condition that the robot successfully maneuvered the course without hitting any unintended objects, such as walls or the mid-arena obstacle.

The team's robot successfully completed all of the above-mentioned aspects on its first

run the day of the competition. After the initial eleven second delay as required in the project objectives, the robot spent a total time of thirty-seven seconds between leaving the starting location and returning to it after hitting the final target in its sequence. No objects other than the four intended targets were hit by the robot, resulting in the awarding of the full amount of style points. This successful completion resulted in the awarding of the full 100 points for the assignment grade for Group 1.

The group was successful in the competition due to the work put in to ensure that all the different components of the robots code worked individually of each other. This greatly aided with troubleshooting when problems did arise throughout the process of working on the robot and allowed the team to develop a better understanding of how their robot worked.

Conclusion and Recommendations:

In conclusion, there were many obstacles throughout building and coding this robot. However, through trial and error, the team was able to work around them and fix them. The component of the semester that the group has the most trouble with was the NASCAR loop traversal, which gave the team some difficulty. They did not finish in the top three, the least amount of time it took to complete the course, which received the most points. This challenge allowed the team to learn more about the way the robot operates and allowed them to make changes that allowed them to be successful in the final competition. The team's design was very successful because it did not prohibit the way they went to complete the tasks.

The major recommendation that the team had was in regard to the equipment. The two main wheels gave them a lot of trouble because they were constantly slipping. This made it very difficult to turn the right way, which in turn would throw off everything else. Also, the caster wheel was very awkward because if it was not originally straight, the robot would move not according to the code, because the caster wheel led it in a different direction. Because of this, the team would recommend that other teams in the future also tape down their caster wheel early on, as to not waste time trying accommodating their code to the wheel's inconsistencies. Additionally, the team would also recommend utilizing a state machine to code their robot. This process of coding made the robot run much smoother through the course and also helped the team stay organized and more effectively troubleshoot problems within their code. The main lesson that the team learned was the effective use of trial and error. Everything on paper may sound easy, but once an attempt is started, things can go wrong and that's where trial and error comes in to play. It allows one to quickly identify the problem and find the solution.

Appendix A:

Below is the entire code of the robot.

```
#include <Servo.h>
#include <WemosInit.h>
#define rightpin D8
#define leftpin D9

Servo right, left;
int leftSensor = ultrasonicPing(D7,D6);
int frontSensor = ultrasonicPing(D2,D3);
int rightSensor = ultrasonicPing(D5,D4);
int state = 0;

/*
 * time = milli(); is a function that reads the time
 * currentTime = milli() - time;
 */

//right is right wheel
//left is left wheel
//D2 and D3 for the front sensor
//D6 and D7 for side sensor

void setup()
{
  right.attach(rightpin);
  left.attach(leftpin);

  Serial.begin(115200);

  //right sensor
  pinMode(D4,INPUT);//D4 is echo
  pinMode(D5,OUTPUT);//D5 is trig

  //front sensor
  pinMode(D2,OUTPUT);//D2 is trig
  pinMode(D3,INPUT);//D3 is echo

  //left sensor
```

```
  pinMode(D7, OUTPUT);//D7 is trig
  pinMode(D6, INPUT);//D6 is echo

  //stops the motors
  right.write(90);
  left.write(90);
  delay(11000);
}

void leftTurn(int time){
    right.write(90);
    left.write(90);//stops motors again
    left.write(102);
    right.write(83);
    delay(time);
}

void rightTurn(int time){
   right.write(90);//right
   left.write(80);//left
   delay(time);
}

void followWallLeftForward(int distance){
int leftSensor = ultrasonicPing(D7,D6);
  right.write(80);
  left.write(81);

  if(leftSensor <= distance){
    right.write(85);
    delay(10);
  }

  if(leftSensor >= distance){
    left.write(83);
    delay(10);
  }
}

void followWallRightForward(int distance){

int rightSensor = ultrasonicPing(D5,D4);
```

```
  right.write(80);
  left.write(81);

  if(rightSensor <= distance){
    left.write(83);
    delay(10);
  }

  if(rightSensor >= distance){
    right.write(85);
    delay(10);
  }
}

void loop() {
int leftSensor = ultrasonicPing(D7,D6);
int frontSensor = ultrasonicPing(D2,D3);
int rightSensor = ultrasonicPing(D5,D4);
delay(100);

    //Serial.println(frontSensor);

//follows the wall, starts at position 1
  if (state == 0){
  followWallRightForward(540);
  if (frontSensor <= 700){state = 1;}
  }

  else if (state == 1) {
    leftTurn(1000);
    right.write(90);
    left.write(90);
    delay(500);
    state = 2;
  }
  // travels towards target C
  else if (state == 2){
    followWallRightForward(1100);
    if(frontSensor <= 2400 ) { //was 2300
      right.write(90);
      left.write(90);
      state = 3;
      }
```

```
  }

  else if (state == 3){
    rightTurn(1000);
    state = 4;
  }

//hits  C and B
  else if (state == 4){
    followWallLeftForward(2350);
    delay(700);
    right.write(105);
    left.write(107);
    delay(2500);

   /*right.write(107);
    left.write(105);
    delay(1300);*/

    right.write(90);
    left.write(90);
    delay(500);
    state = 6;
  }
  //state 6 to 8 brings the robot between target C and target D.
  else if (state == 6) {
   followWallLeftForward(2350);
   if (frontSensor <= 760)
    state = 7;
  }

  else if (state == 7){
    rightTurn(1000);
    right.write(90);
    left.write(90);
    delay(500);
    state = 8;
  }

   else if (state == 8){
    followWallLeftForward(1100);
    if (frontSensor <= 2000)
      state = 9;
```

```
  }

//hits D and A
  else if (state == 9) {
    leftTurn(700);
    right.write(80);
    left.write(81);
    delay(600);

    right.write(90);
    left.write(90);
    delay(500);

    right.write(105);
    left.write(107);
    delay(2500);

    right.write(90);
    left.write(90);
    delay(500);
    state = 10;
  }
  // returns to starting location
  else if (state == 10) {
    rightTurn(900);
    followWallRightForward(630);
    delay(700);

      right.write(90);
      left.write(90);
      delay(15000);
  }

}
```