EGRE 364 – Microcomputer Systems

Final Project

Final Design Report

Lab Section: Thursday 1pm

Report Submitted on: December, 11 2015

Quan Ma & Jose Ramirez

PLEDGE: [illegible] [illegible]

"On my honor, I have neither given nor received unauthorized aid on this assignment"

Abstract:

The purpose of this report is to outline the motivation, goals, and criteria that was necessary for completing the EGRE 364 Final Design Project. This project consisted s of designing and implementing an embedded system, more specifically an autonomous robot that is capable of navigating through a maze, follow a black line drawn on a white surface, and draw a visually appealing image on a canvas all using the knowledge and techniques acquired throughout this course. The project deliverables, timelines, constraints, and how each feat was implemented are clearly outlined throughout this report. In understanding how all the elements will come together, this report also outlines the theoretical mechanical design, electrical design, software architecture, project management, and the final cost to build the system.

Introduction:

The project required a robot design that is capable of maze navigating, line following, and artwork. To do this our team needed to design and build an autonomous vehicle that is easy to implement and agile enough to perform these tasks and more.

Mechanical Design:

When considering the mechanical design of the robot, there are a few features we decided that should be taken into consideration and which in the end were implemented on the final design. The original design of the robot consisted of a frame that was purely made up of 2 levels of breadboards as seen in Figure 1 below.

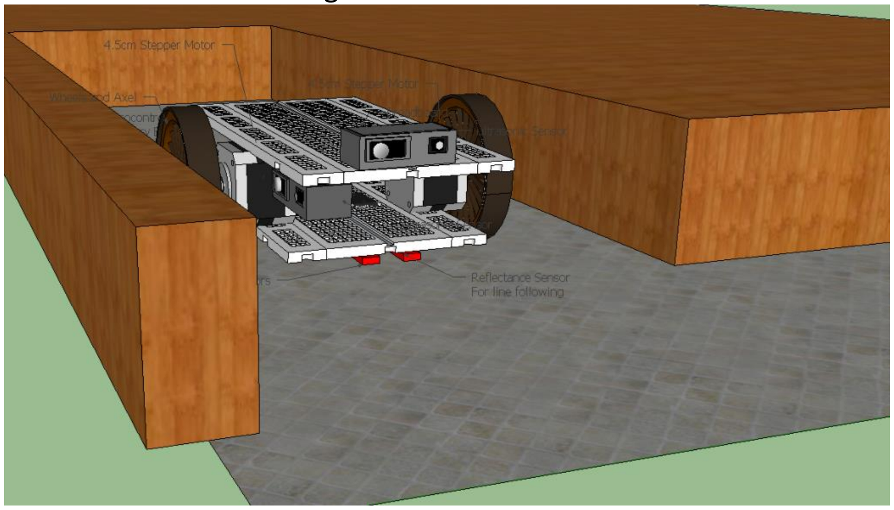
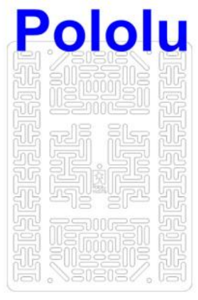


Figure 1: 3D render of design consideration for the robot , showing two levels of breadboard design where the motors sit on the bottom breadboard, the microcontroller and H-Bridges placed on the bottom breadboard and the distance sensors places as seen in the figure , and the Wheels placed in the center of the robot itself.

The original design was that the robot was to be 5"x 5" leaving plenty of room for the maze, we soon found that it was quite difficult translating that size to the physical design , so we constrained the design to 7"x 5" , which still worked and was the perfect size fort the maze completion. As seen in figure 1 above, the design that was thought out for the robot was not implementable due to the fact that the placement of the motors was going to be a challenge to place on a breadboard securely without any brackets screwed to the base. For this fact the whole idea for a breadboard as the base of the robot was trashed and we opt out to use a plastic base expansion plate with all the mounting holes necessary to allow for the design to have many possibilities. See Figure 2 Below.



Pololu RP5/Rover 5 expansion plate RRC07B (wide) transparent clear.

Figure 2: Chassis for the robot Rover 5 Expansion plate 6.8" x 5"

The expansion plate allowed for proper placement of the motors and sensors. The unlimited amount of holes allowed for an easier implementation of the mechanical design. Instead of the breadboard at the base, 4 breadboards where use as the structure of the robot rather than the base. Placing 2 breadboards horizontally and 2 more breadboards bridging the microcontroller on top , allowed for an easier design of the robot for all the different competitions. See Figure 3 below for a visual representation of how the final design of the robot looked like with the four breadboards as the structure of the robot.

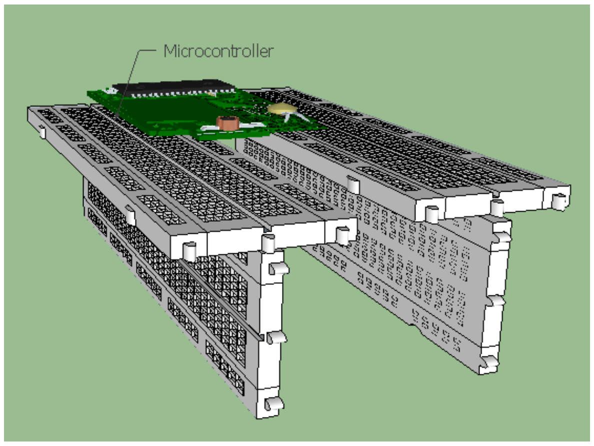


Figure 3: Structure frame for the design of the robot showing 4 breadboards with the micro controller bridge at the top, allowing for easy connections to the GPIO pins. The two breadboards on the bottom sides, allowed for an easier connection to the H-bridges and moto. This helped organize the wire clutter and allowed for an easier and manageable system that was easy to troubleshoot.

The motors where placed on the Chassis expansion plate using mounting brackets and 80 cm in diameter wheels where attached to the motors themselves using connection hubs that attached to the shaft of the motor. This allowed for an easy mounting of the wheels to the motors. Figure 4 below illustrate where the motors where position one the expansion plate.

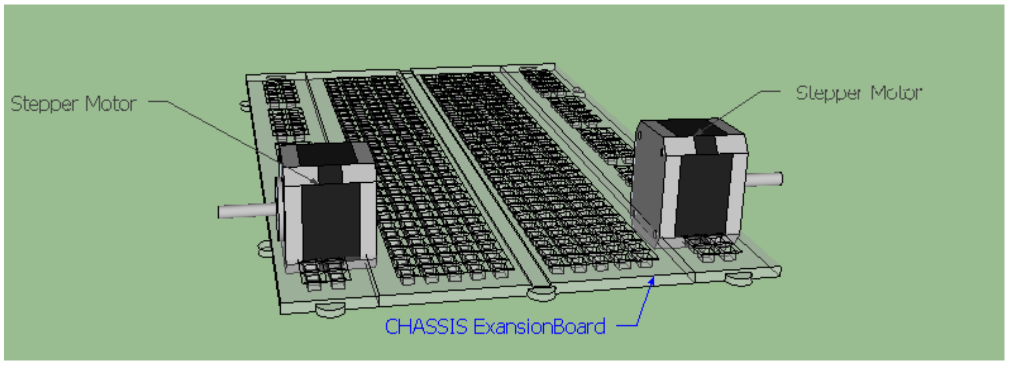
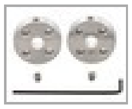
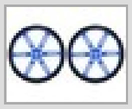


Figure 4: Placement of the motors on the expansion plate seen if figure 2 above the original design called for the motors to be placed towards the center of the base but it didn't allow enough space for the distance sensor and third wheel , so it was decided that they would be placed towards the back as seen in this figure.

The specific hub , motor brackets, and wheels used for this system can be seen I figure 5 below


Pololu Universal Aluminum Mounting Hub for 5mm Shaft, #4-40 Holes (2-Pack)


Pololu Wheel 80×10mm Pair - Blue

Pololu Stamped Aluminum L-Bracket for NEMA 14 Stepper Motors

Figure 5: Mounting hub, specific wheels used with the system, and motor brackets used to secure the motors to the base. These where essentially all mechanical parts of this system. Parts that consisted of the general structure and movement of the system.

To balance the robot, two caster wheels seen in figure 6 where attached at the front of the robot which allowed for balance and stability of the robot, these wheels allowed for the robot to have full omnidirectional freedom necessary for the robot to be able to perform in all the tasks.

Figure 6: Caster wheels used to stabilize the robot , in the original design consideration ,these caster wheels proved to be effective and where used in the final design.

For placement of sensors, it was figured that putting the reflective sensors as far as possible from your pivoting wheels, will give the microcontroller a better reaction time and reduce the chance of overshooting during the line following as well as the artwork trial. Since it allows more time to react. The distance between the sensors depended on the track itself, and in terms where places 1-3cm apart to detect whether there are discontinuities in the track. As for the distance sensors, placing one at a 45° angle at one edge of the chassis and the other facing the front allowed for a better sense of the maze. General placement of the sensors can be seen in Figure 7 below.

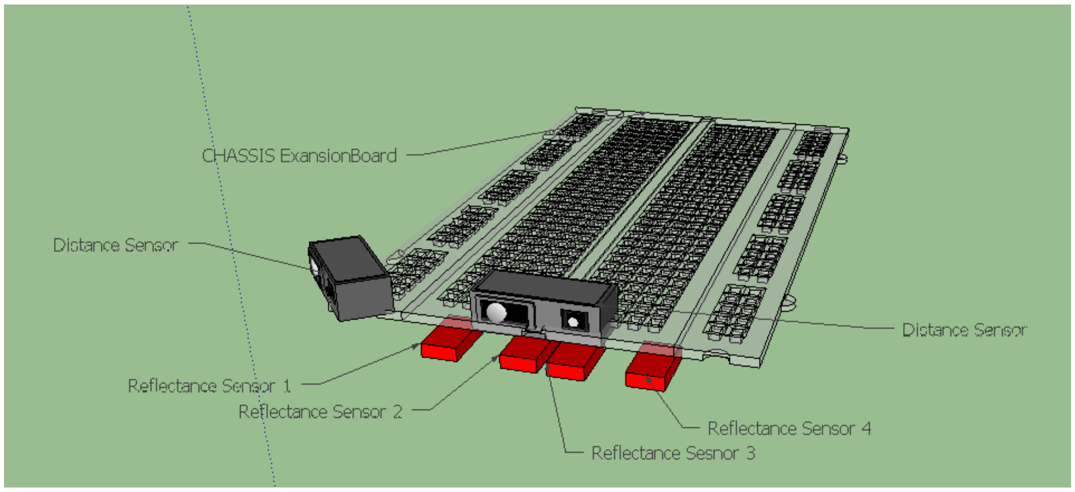


Figure 7: Placement of distance sensors and reflective sensors. The original design called for the distance sensors to be placed one facing completely to the left and one facing completely to the right. This was changed and one was placed at a 45° degree angle on one of the corners, and the other distance sensor facing the front. The reflective sensors were placed on the bottom of the robot as thought in the original designed, worked well with the implementation.

In order to have come up with the best placement of these sensors. The only way we could see it is just doing multiple trials and testing to gather enough data for their placement. This placement was found the best for the sensors to optimize the operation for the maze, line and artwork competitions.

The Distance sensors were mounted on the chassis Expansion board using the brackets seen in figure 8 below

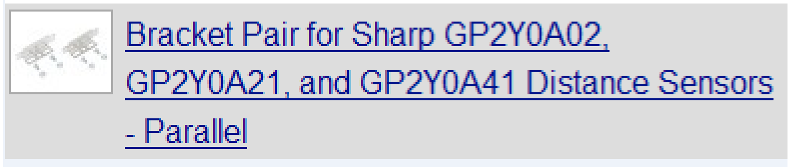

Bracket Pair for Sharp GP2Y0A02, GP2Y0A21, and GP2Y0A41 Distance Sensors - Parallel

Figure 8: Mounting brackets used for the distance sensors

Although not used in the line following and the Maze completion, the writing utensil was also a mechanical feat we had to overcome. In order to have an effective writing utensil we engineered a way to have a writing utensil perfectly centered on the robot to be effective in drawing accurate designs on a canvas. Figure A-C shows the mechanical system used for the writing utensil.

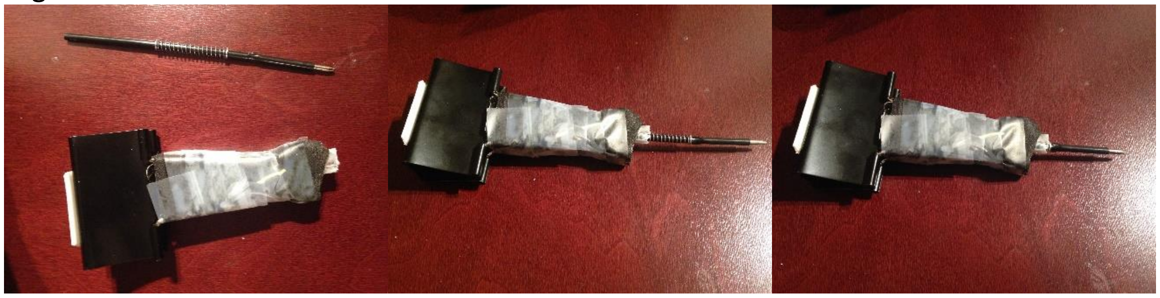


Figure A: Shows the writing utensil mechanical design the large case is a weight used to keep the pen touching the ground. The spring in the pen is used to keep the vibrations away from the writing as the robot is riding across the canvas. This system allows for an easy replacement of the pen if the ink was to run out.

Below Figure B shows how the writing utensil was placed at the completion. Placement of the pen was crucial it needed to be perfectly centered to the motors as it was originally in the design that was thought out for this system.

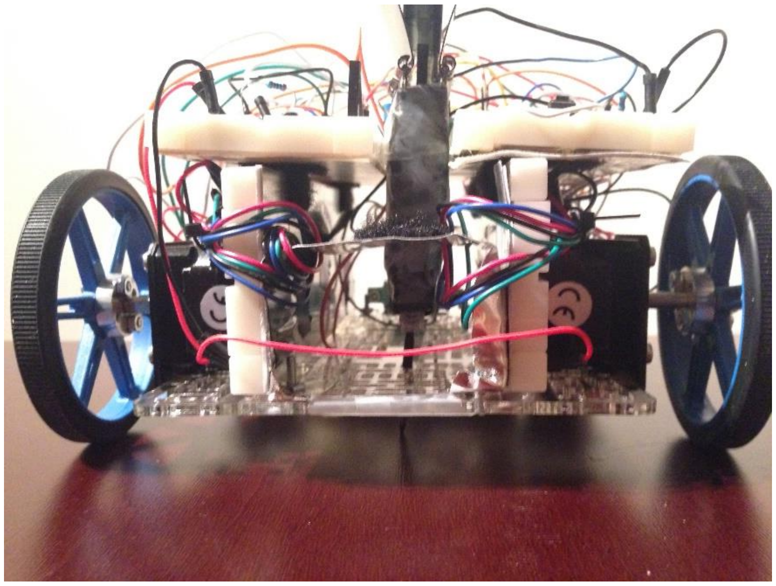


Figure B: How the pen was placed on the system, it can be seen is perfectly center between the motors as originally thought in the Design, the weight incasing fits well between the breadboards and the pen sits perfectly on the ground.

Figure C on the following page shows a closer look of the writing utensil and how it sits on the ground.

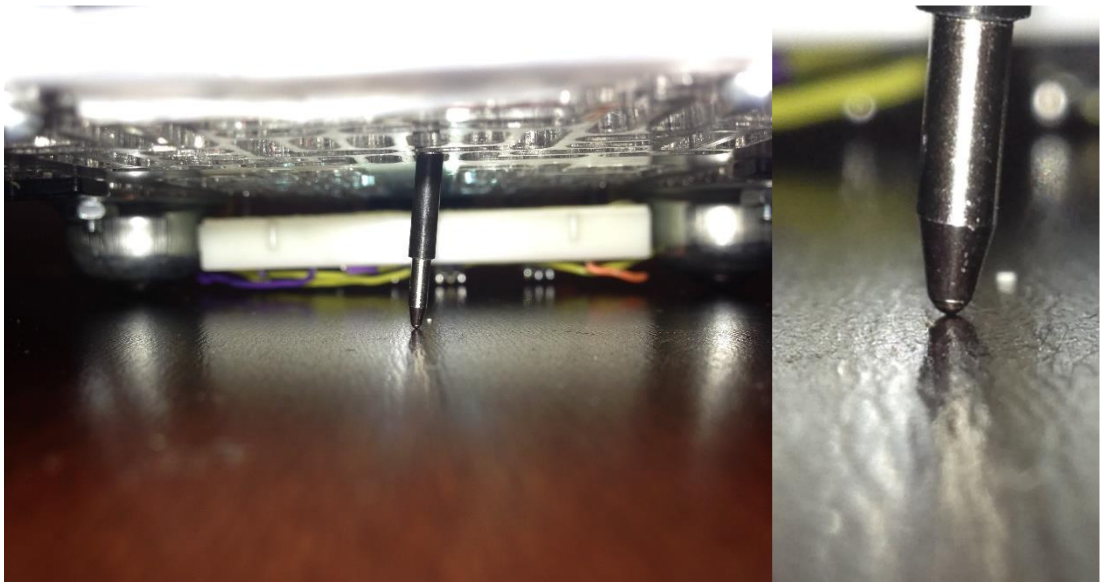


Figure C: Close up of the Ball point tip of the pen and how it perfectly sits on the ground with enough pressure to write with precision.

Electronic Design:

The Electrical Design was pretty intuitive, the schematic design was straight forward and it went hand in hand from what was original planned. The electrical design was a build off from the previous labs. See figure 9 below for the general layout of the electrical system. We took the time to draw a visual schematic that was easy to read and follow and which laid out all the components used to in the system to be controlled by the microcontroller.

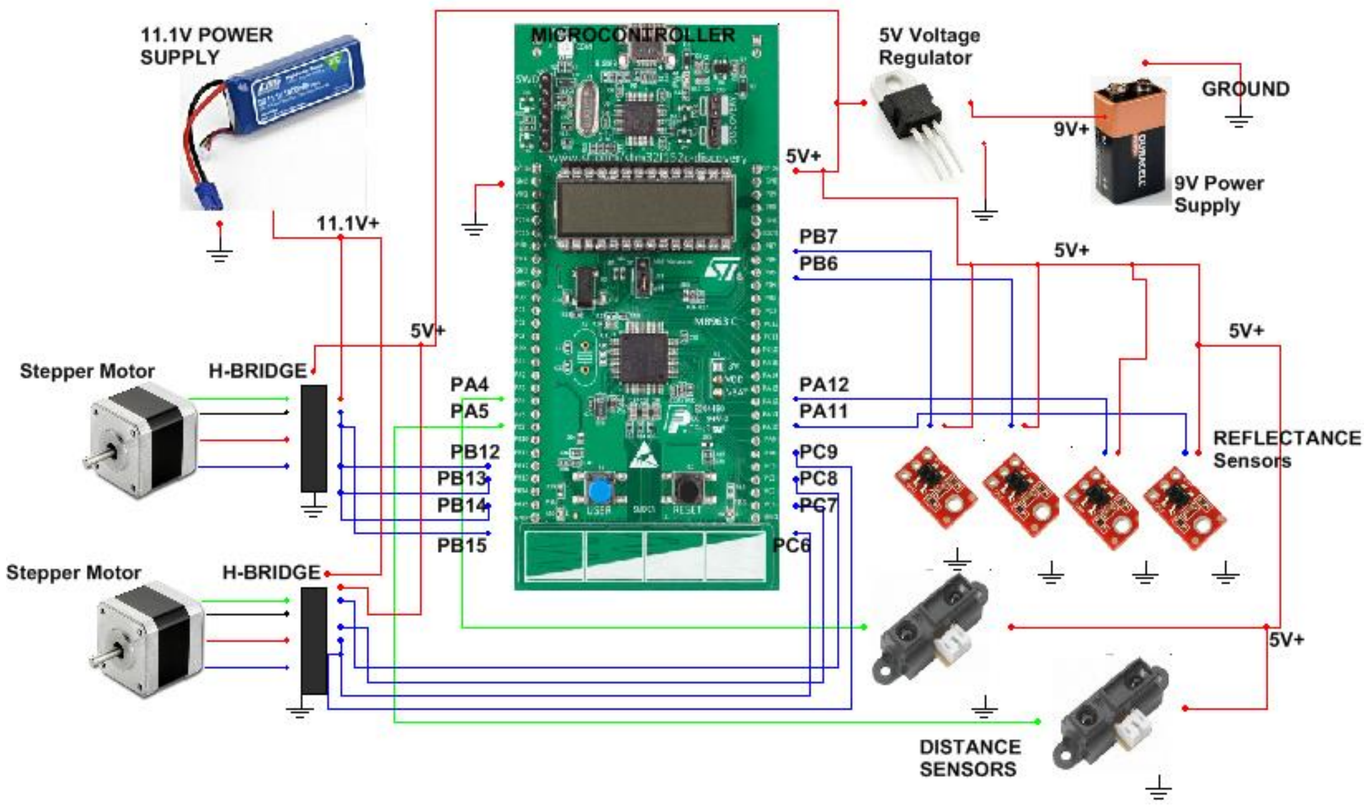


Figure 9: Physical schematic of the system , shows where each of the components was connected in relation to the Microcontroller. In the original design we may have chosen different GPOI pins to connect the sensors but the basic idea is kept the same.

Parts that will be used for electrical design:

#1207	stepper motor 2 pololu
#352	breadboard 2 pololu
#136	distance sensor 2 pololu
455-1126-ND	distance sensor connectors 2 digikey
#2459	reflectance sensor 4 pololu
#1704	jump wires 15 pololu
296-9911-5-ND	motor drive chip 2 digikey (H-Bridge)
497-3450-5-ND	5V regulator 1 digikey

(1) Battery 9v Duracell

(1) 11.1v , 1800mah LIPO Battery

In general we utilize at least 8 pins to drive both stepper motors and at least 4 pins for the reflectance sensors and 2 pins for the distance sensors. That gave us a total of 14 pins used. Carefully choosing the pins allowed us to still utilize the LCD for testing for the pins that were chosen for the distance and reflective sensors were free pins that were not used by the LCD. This was a smart move on our part for it allowed to easily trouble shoot on the spot if something went wrong.

Looking Back to figure 9 above, overall schematic of the system, we can see the H-Bridges. Below, Figure 10, we see a more in detail representation on how the connection of the H-Bridges were made.

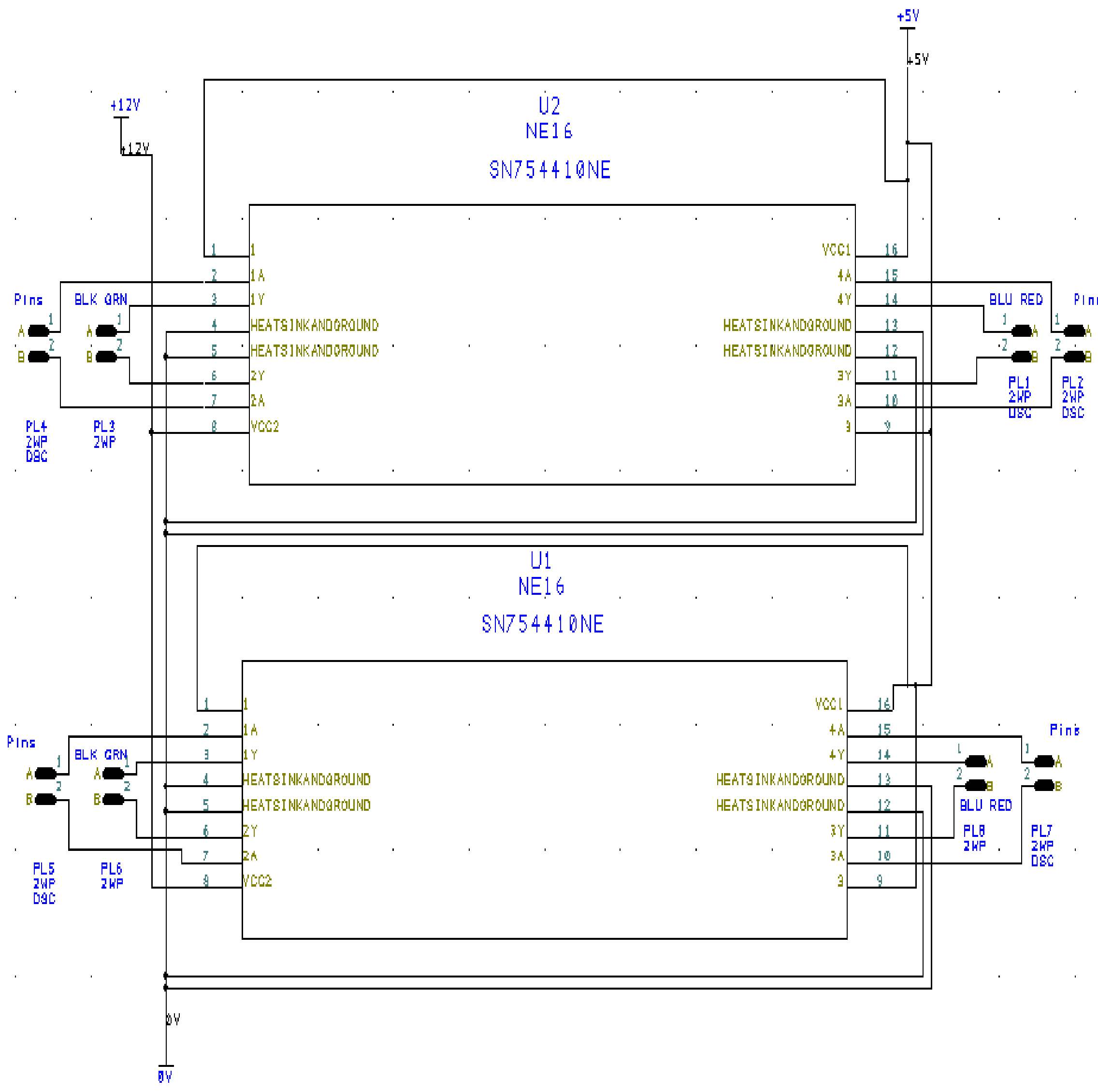


Figure 10: H-Bridge Connections schematic, this was important to get right for it was the H-Bridge which was the most important part on being able to drive the motor forward. The H-Bridges used for the system were specifically SN754410NE.

On the next page you will find the PCB Design for this Schematic, detailing the different connections

Below ,Figure 11, you will find the PCB design that was implemented for the system, which encompasses the H-Bridge Schematics found in figure 10.

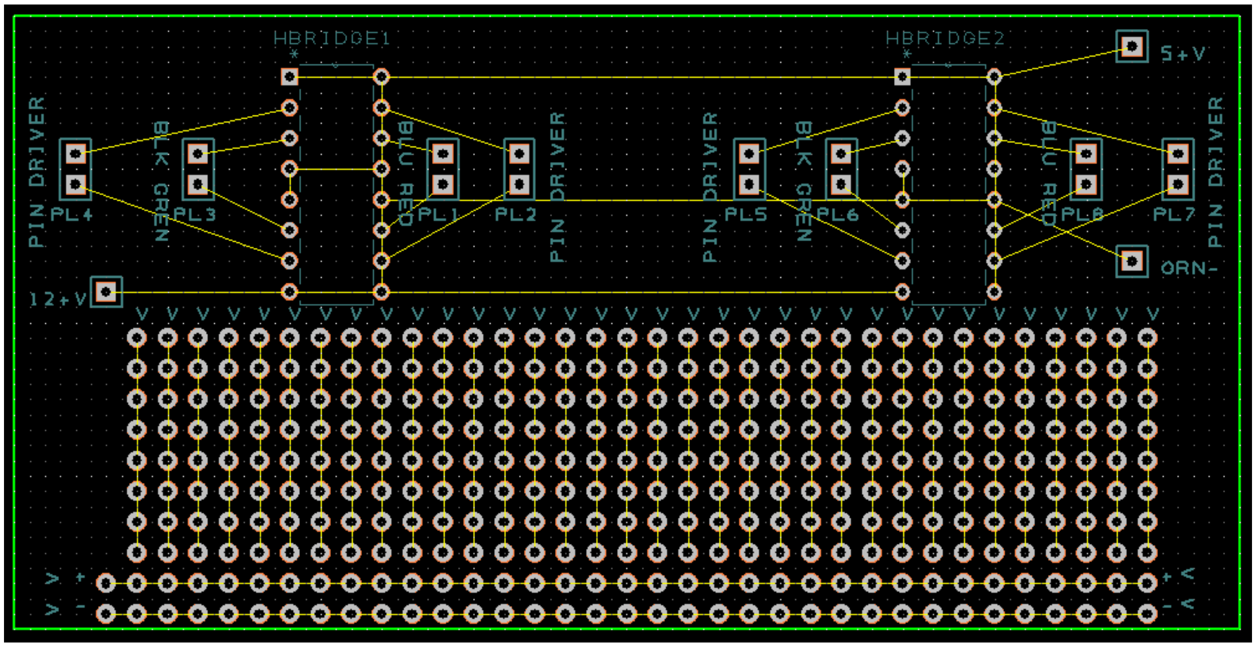


Figure 11: PCB Design for the system, nodes are labeled and can be seen clearly.

Although a PCB Design was implemented, our budget did not allow for a physical implementation and utilization in the system. Nevertheless, it may be used for future implementations for is a working design.

The final Look of the system can be seen in the images below, showing all the sensors placed , the motors , and H-Bridges including all the wires connected to the sensors from the Microcontroller. All that was laid out in the figures above Figure 2-11 can be seen in the final results below, Figure 12 and Figure 13 on the following pages.

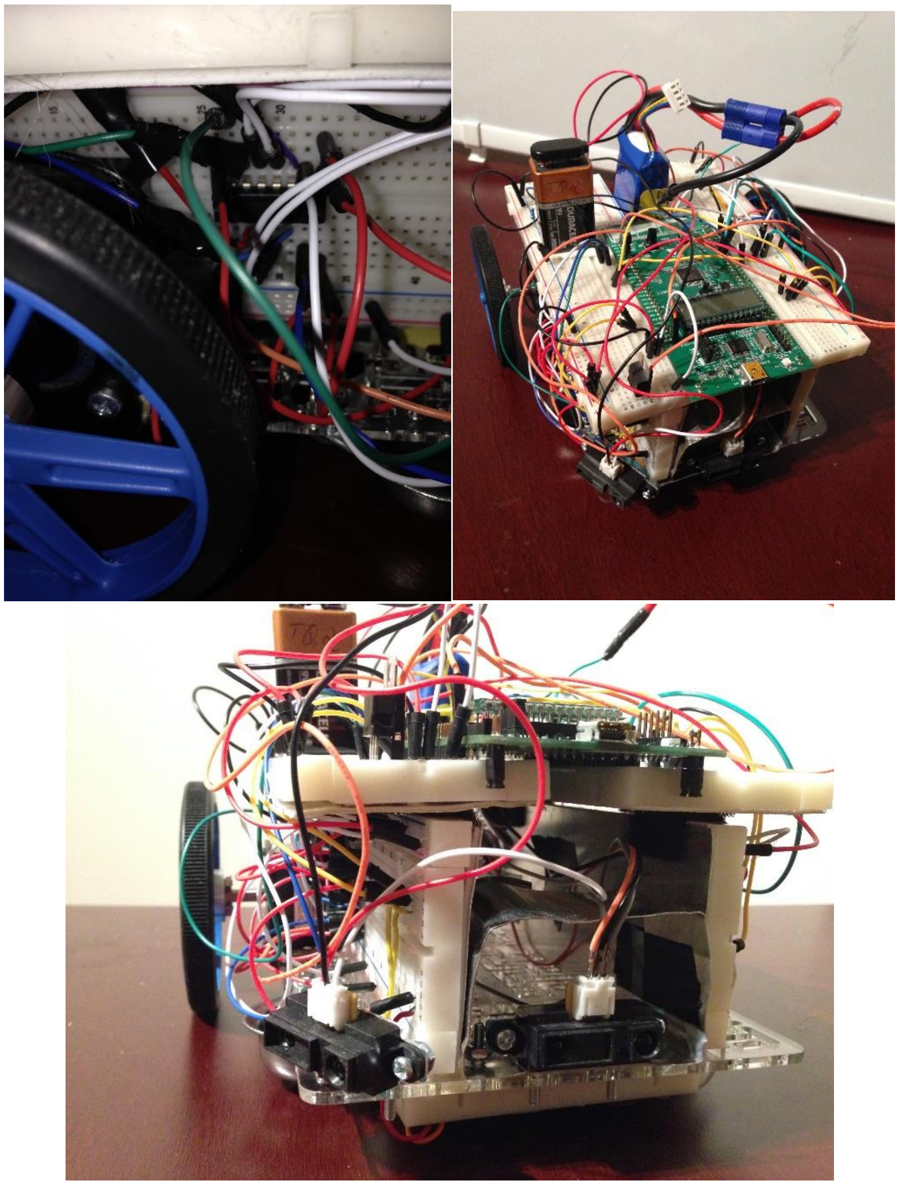


Figure 12: Starting From top Left , Is the H-Bridge chip on the side on top of the Side breadboard as discussed in the mechanical design , the image on the right and bottom show the top and front of the system, with the both distance sensors on the front one Placed at a 45° angle on the chassis.

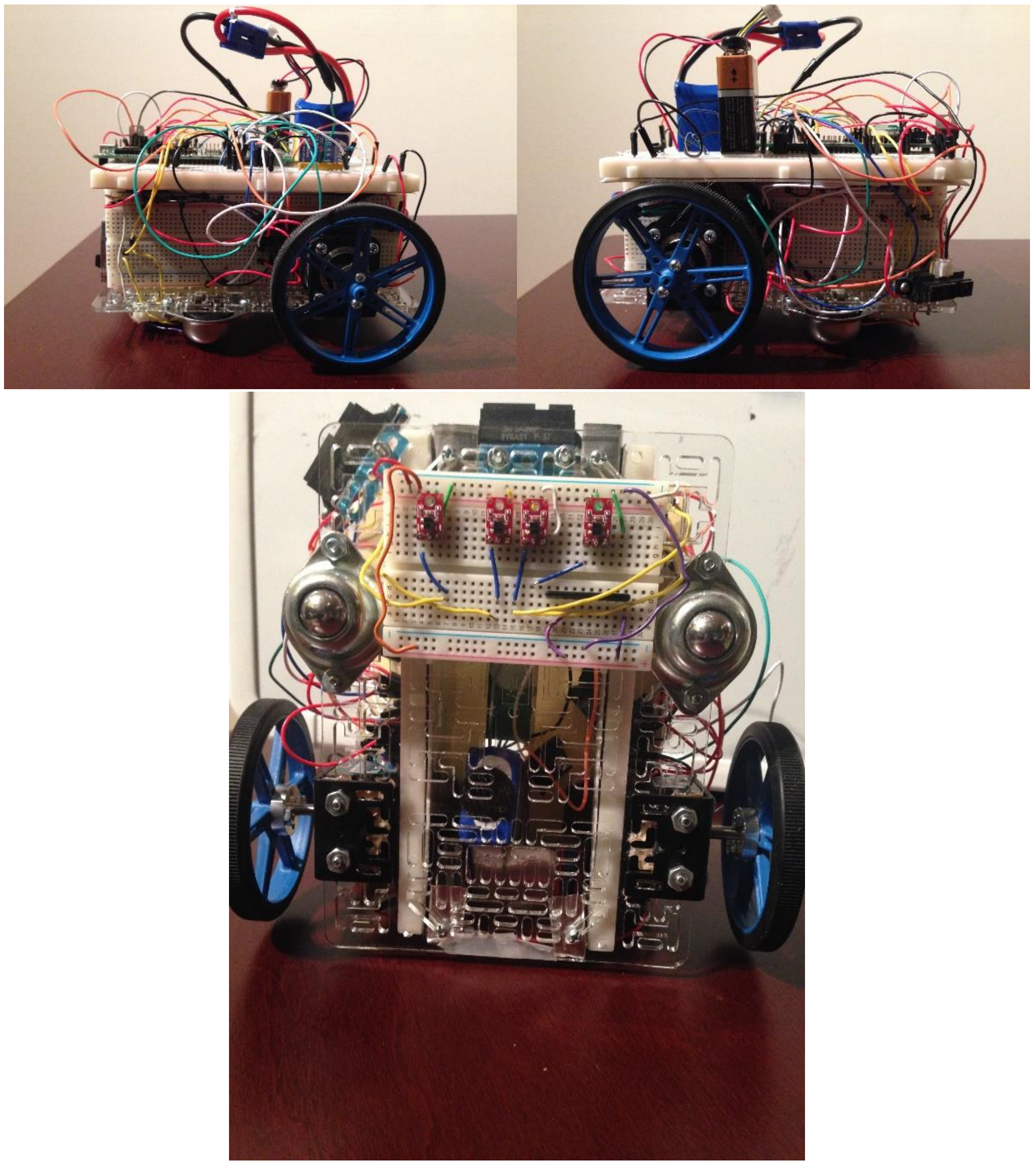


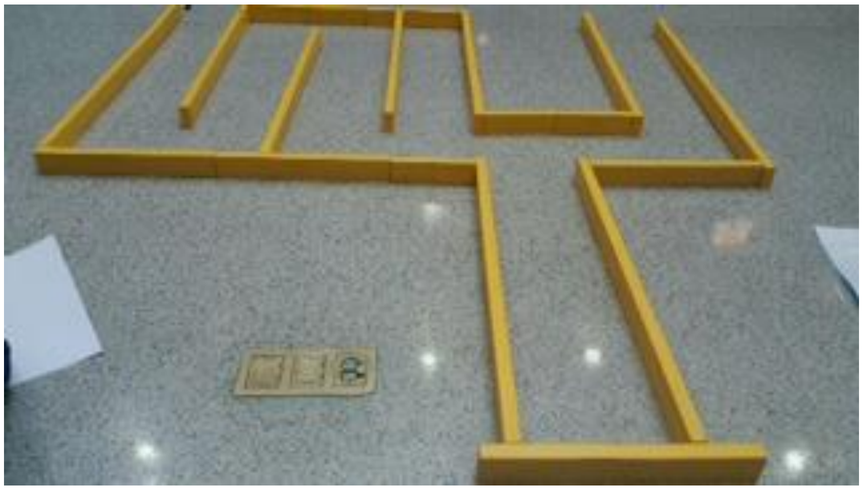
Figure 13: Images show the left and right of the system as well as the bottom where the reflectance sensors are placed. As can be seen on the bottom of the robot you find the two caster wheels used to stabilize the robot. On the top and left and right images you may have a view of the battery 9v used and the 11.1v battery used to power the system.

Although it can be hard to follow the electrical wiring for the system, Figure 9, Figure 10, and Figure 11 have a good way of showing how the electrical system is implemented and all wires in the system seen in figures 12 and 13 can be seen serve a purpose.

<u>Software Architecture:</u>

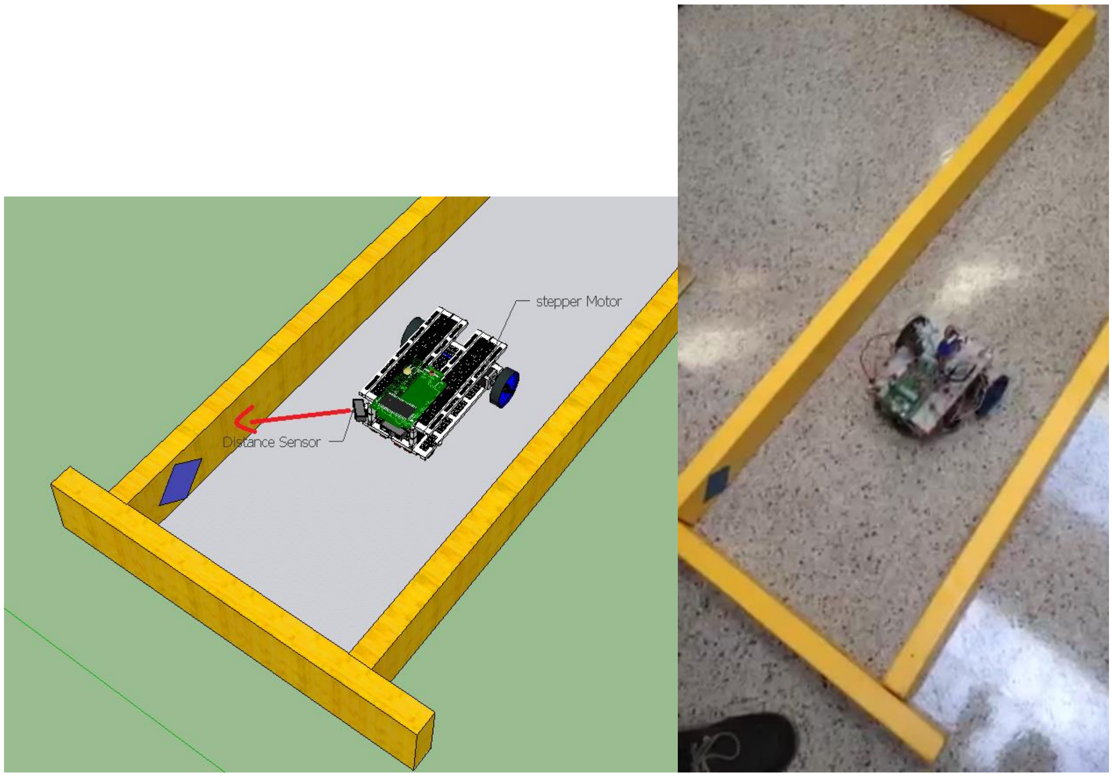
<u>Maze Algorithm:</u>

The Maze algorithm implemented followed what was originally designed in the consideration. For the maze algorithm we need to understand how a maze works and how a human could potentially be able to navigate the maze without opening one's eyes. With understanding in thinking how this may be accomplished. We came up with an algorithm solved the problem at hand and was successful. An algorithm which mimics a right hand on one side of the maze wall. Refer the following pages for an explanation of how the robot was able to navigate through the maze successfully. In the original design consideration for the algorithm it was considered to use both of the distance sensors for the algorithm. But it was later figured that using only one at a 45° angle was sufficient enough and made for the program to be far simpler. The Maze that was needed to be tackled can be seen in Figure 14 below.



<u>Figure 14:</u> Maze for the competition.

As the robot travel through the maze we had it look at the right side at all times using the sensor that was placed at the corner of the chassis at a 45° angle. then have it decide if it can turn left then turn **<u>left</u>**, if it not able to turn left than **<u>continue</u>** in a straight line , and if able to turn **<u>right</u>** then turn right. See the following page for how this was implemented at the most difficult part of maze, the dead end, seen towards the bottom right of the image in Figure 14 above as an example.



<u>Figure 15: Showing robot detecting the right wall with the Distance sensor at a distance that is between 10cm and 20cm. The the robot keeps just keeps moving forward. This is the case for all the Straight aways in the maze. Image on the left simply shows what where is the sensor point to.</u>

Code running at this point can be seen below: if the side sensor is greater than or equal to 10cm and less than or equal to 20cm then keep moving forward.

```
if( side_sensor >= 10  && side_sensor <= 20 )   // if is between 10 and 20cm away then
                                                    keep moving foward
{
   for(i=0; i<4; i++)
   {
     StepMotor_half_right(&motor_right); // half step stepper motor right
     Delay (motor_delay);
     StepMotor_half_left(&motor_left); // half step stepper motor left
     Delay (motor_delay);
   }
}
```

This code runs every straight away, the sensor constantly checks if this condition is true and continues to move forward straight.

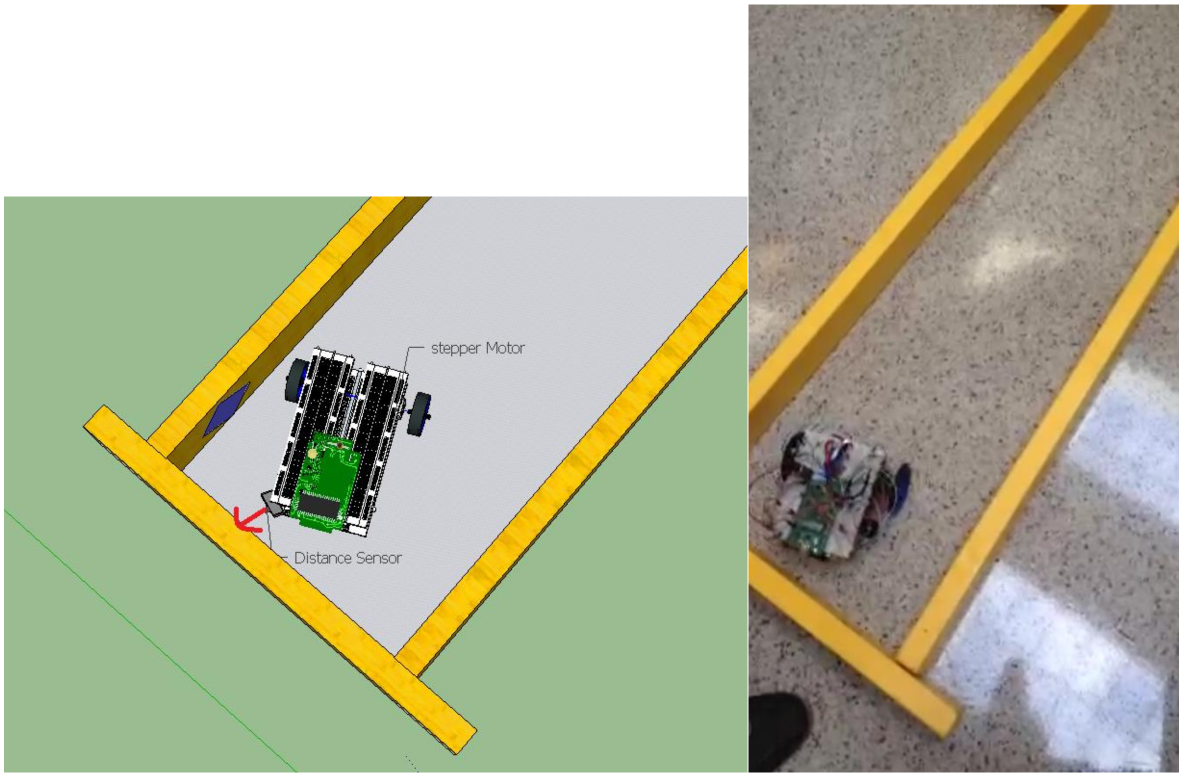


Figure 16: Showing robot detecting the front wall with the Distance sensor at a distance that is less than 10cm . The the robot in this condition turn left. This is the case for all the times the robot sees the front wall of the maze.

Code running at this point can be seen below: if the side sensor is less than 10cm turn left.

This insures that the robot keeps its distance from the right wall and doesn't crash.

```
else if ( side_sensor < 10)    // if it gets less than 10cm then shift a bit to theleft
{
   for(i=0; i<3; i++)
   {
       StepMotor_half_right(&motor_right); // half step stepper motor right
       Delay (motor_delay);
       StepMotor_half_left_reverse(&motor_left); // half stepper motor left reverse
        Delay (motor_delay);
    }
}
```

This code runs every time the sensor sees that is less than 10cm from the wall and just keeps turning left.

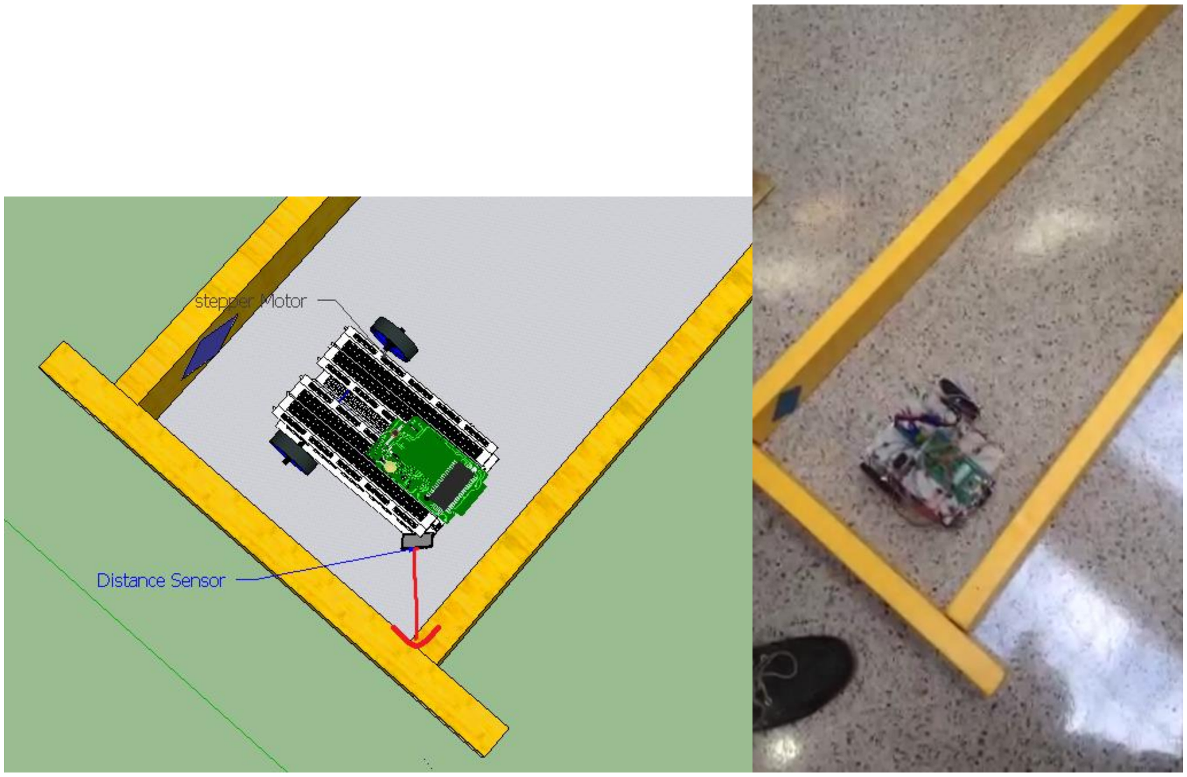


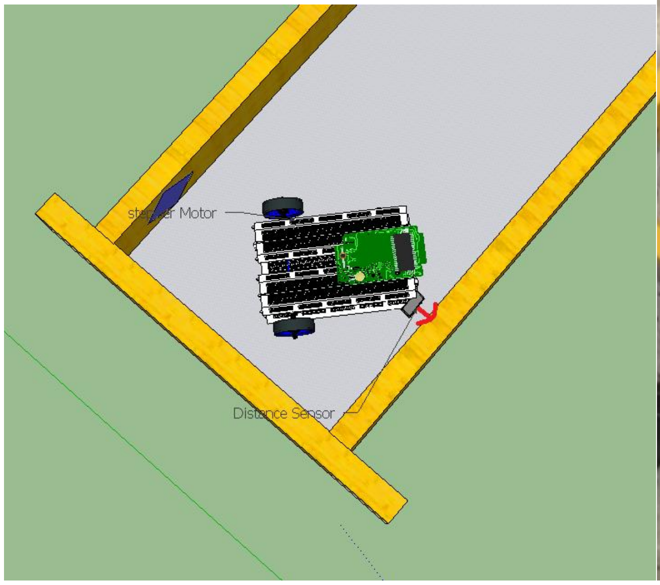
Figure 17: Showing robot detecting the front wall with the Distance sensor at a distance that is less than 10cm . The the robot in this condition turn left. This is the case for all the times the robot sees the front wall of the maze.

Code running at this point can be seen below: if the side sensor is less than 10cm turn left.

This insures that the robot keeps its distance from the right wall and doesn't crash.

```
else if ( side_sensor < 10)   // if it gets less than 10cm then shift a bit to theleft
{
   for(i=0; i<3; i++)
   {
       StepMotor_half_right(&motor_right); // half step stepper motor right
       Delay (motor_delay);
       StepMotor_half_left_reverse(&motor_left); // half stepper motor left reverse
        Delay (motor_delay);
    }
}
```

This code runs every time the sensor sees that is less than 10cm from the wall and just keeps turning left.



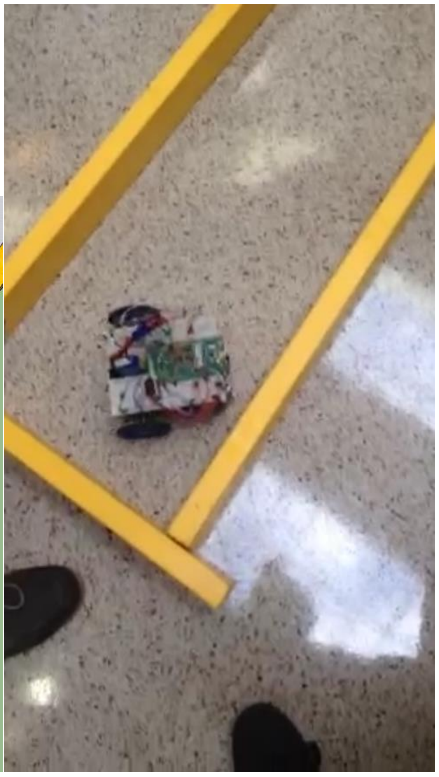


Figure 18: Showing robot detecting the front wall with the Distance sensor at a distance that is less than 10cm . The the robot in this condition turn left. This is the case for all the times the robot sees the front wall of the maze.

Code running at this point can be seen below: if the side sensor is less than 10cm turn left.

This insures that the robot keeps its distance from the right wall and doesn't crash.

```
else if ( side_sensor < 10)   // if it gets less than 10cm then shift a bit to theleft
{
   for(i=0; i<3; i++)
   {
       StepMotor_half_right(&motor_right); // half step stepper motor right
       Delay (motor_delay);
       StepMotor_half_left_reverse(&motor_left); // half stepper motor left reverse
        Delay (motor_delay);
    }
}
```

This code runs every time the sensor sees that is less than 10cm from the wall and just keeps turning left.

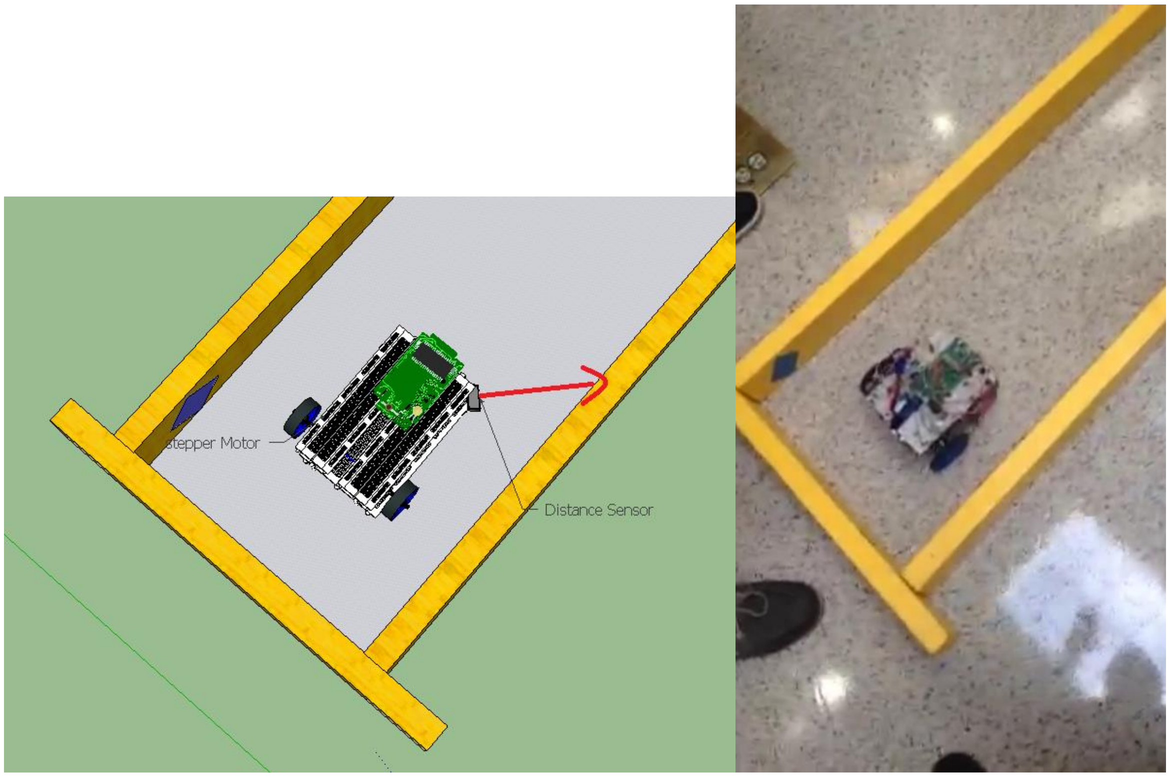


Figure 19: Showing robot detecting the right wall with the Distance sensor at a distance that is between 10cm and 20cm. The the robot keeps just keeps moving forward. This is the case for all the Straight aways in the maze.

Code running at this point can be seen below: if the side sensor is greater than or equal to 10cm and less than or equal to 20cm then keep moving forward.

```
if( side_sensor >= 10  && side_sensor <= 20 )   // if is between 10 and 20cm away then
                                                   keep moving foward
{
   for(i=0; i<4; i++)
  {
    StepMotor_half_right(&motor_right); // half step stepper motor right
    Delay (motor_delay);
    StepMotor_half_left(&motor_left); // half step stepper motor left
    Delay (motor_delay);
  }
}
```

This code runs every straight away, the sensor constantly checks if this condition is true and continues to move forward straight.

So the above figures demonstrates how the robot was able to manage the dead end and continually make a left turn , but in the scenario where the robot had to a right turn the algorithm was slightly more involved. As can be seen on the following page.

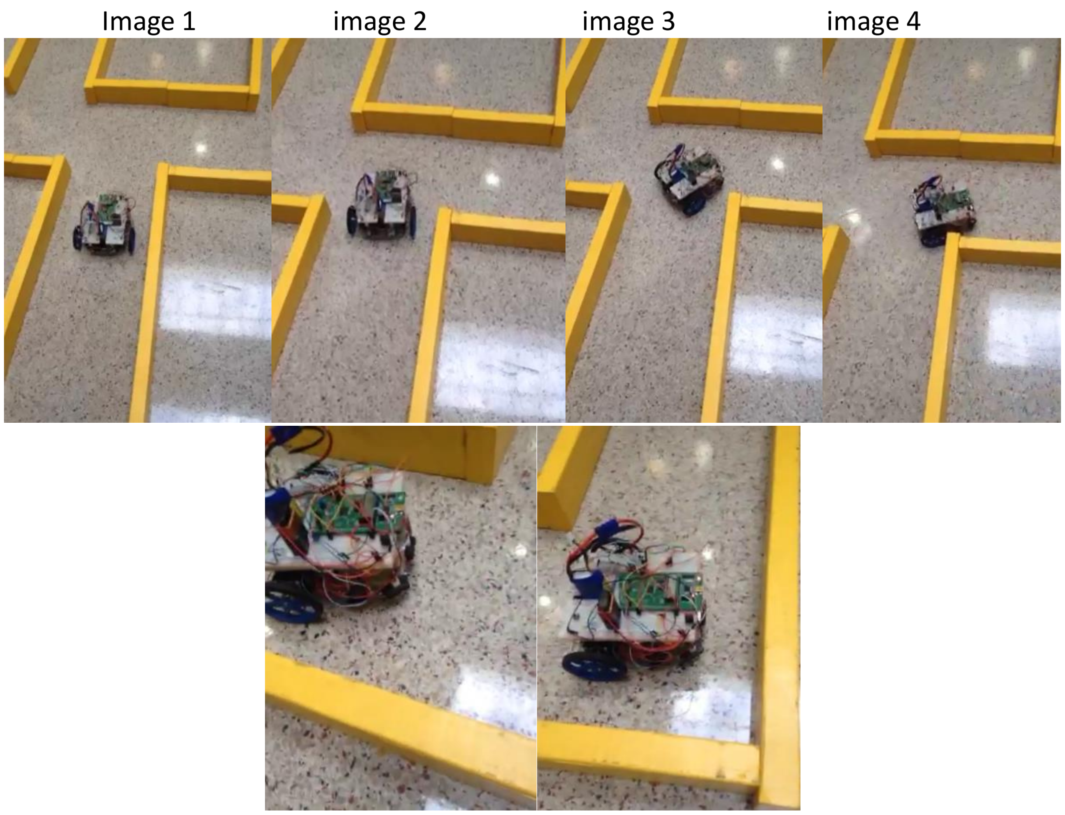


Figure 20: Making a Right turn Break down, image top left to bottom right.

Starting from the image 1 in figure 20, the robot runs the code that was seen in figure 15 or 19. But as the robot peaks over the corner, image 2, with the distance sensor the following code takes over and the robot begins to react to the right turn imediately.

```
else if ( side_sensor >= 30)   // if it become greater than 30cm then perform the turn
                                right sequence

{
    for(i=0; i< 400; i++)
       {
          StepMotor_half_right(&motor_right); // half step stepper motor right
          Delay (motor_delay);
          StepMotor_half_left(&motor_left); // half step stepper motor left
          Delay (motor_delay);
       }
    for(i=0; i<150; i++)
       {
          StepMotor_half_right_reverse(&motor_right); // half right reverse
          Delay (motor_delay);
          StepMotor_half_left(&motor_left); // half step stepper motor left
          Delay (motor_delay);
        }
}
```

As soon as it detects that the distance sensor reads more than 30cm the code runs the motors for about 2 full steps and then turns the robot right. This is to ensure that the robot is overshooting the corner and does not end up hitting it.

As the robot finish it sequence of turning it checks the codition once more and it becomes true therefore in image 3,4 ,figure 20, it runs through the sequence again because the sensor will read greater than 30cm. Once it finishes with the sequence the robot will then run to this condition Image 5, figure 20. Where the distance sensor reads greater than 20cm and less than 30cm. In this case the robot will simply turn right to find the right wall again using the following code sequence.

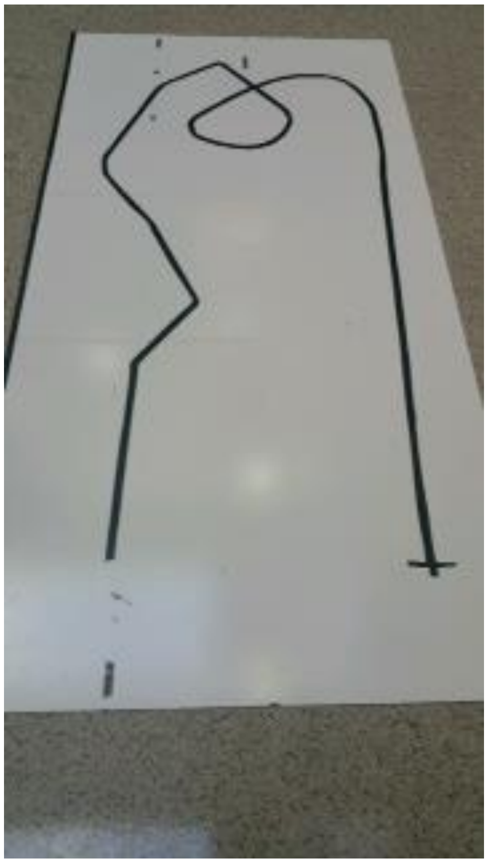
```
else if ( side_sensor > 20 && side_sensor < 30)    // if is greater
than 20 cm and less than 30cm adjust yourself to move right
 {
    for(i=0; i<30; i++)
     {
     StepMotor_half_right_reverse(&motor_right); // half step stepper
                                                 motor right reverse
      Delay (motor_delay);
      StepMotor_half_left(&motor_left); // half step stepper motorleft
      Delay (motor_delay);
     }
}
```

Once the robot find the right wall again , it will be the same as the figures 15-19. For moving forward and making a left turn image 6 , figure 20.

(See full code implementaion for the Maze algorith fully comminted in the Attached Project files and in Appendix)

<u>**Line Following:**</u>

Using what was learned from lab and we implement the reflectance sensors to help in the aid of the line follower. The original design consideration worked well for the final design implementation and the idea worked to allow the robot to follow a black line successfully. With line following we have to different cases each of the reflective sensors see either black or white , in simpler terms this may just be represented as true or false , '1' or a '0' respectively. The track that was needed to be tackled can be seen below figure 21.



<u>**Figure 21: Line Following Track Used in the competition and which the robot completed successfully.**</u>

For line following we decided to go with 4 sensors to read the line on the floor. These four sensors can be denoted as four binary digits **1111**: sensor 1, sensor 2, sensor 3 and sensor 4 respectively. There are just a few rules that the robot had to take in order to be a successful in following the line. As the robot is following the line, it will continue as long as any of the sensors detect a line, that is 16 different possible ways it can see the line. If the robot for any reason leaves view of the line that is **0000**, then it baked up slightly, <u>this was slightly different in the design consideration for in the design consideration the robot was suppose to continuously turn</u> until <u>it saw the line and it will search within that area to see if it can get back on track with gathering 1 or more sensors seeing the line.</u>

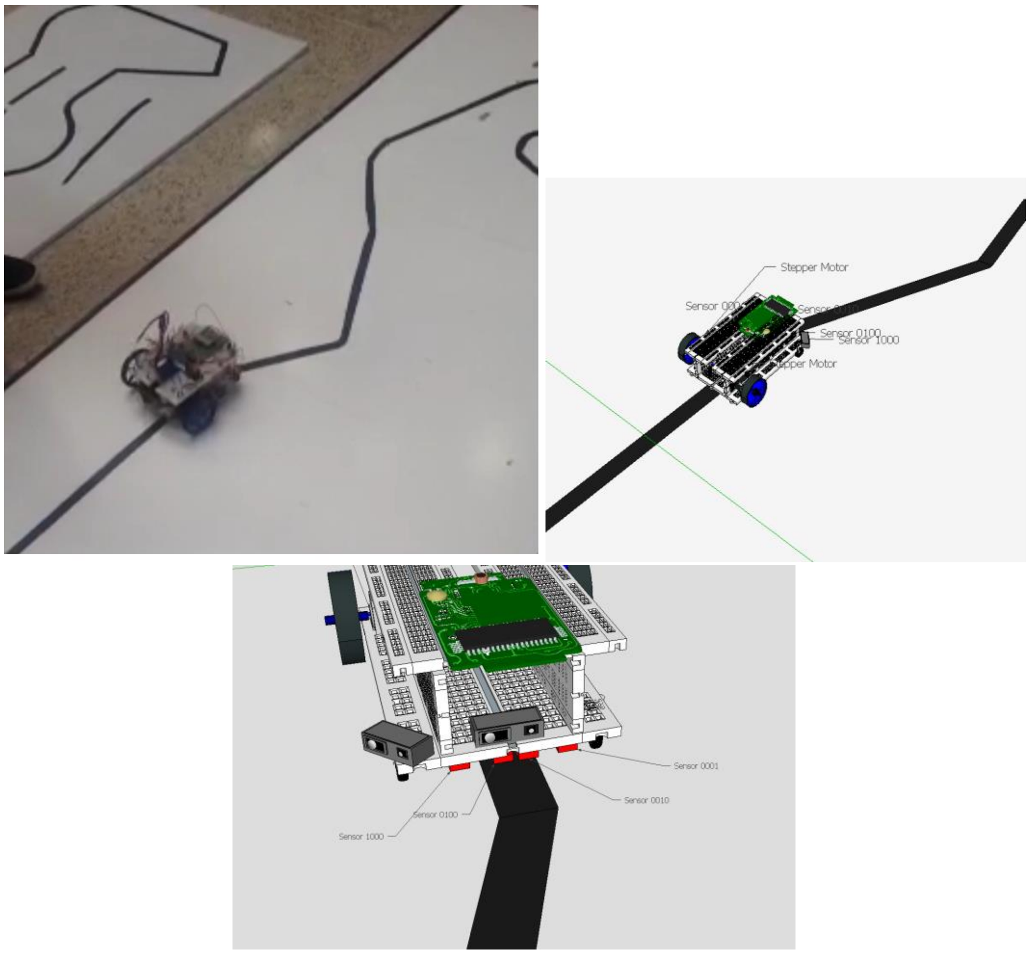


Figure 22: Scenario for line following robot, all three images represent the same scenario at different angle, Sensors are labeled in correspondence to the code. And the function used for the sensors that can be seen in the Appendix header file for the Reflectance sensors.

With the line follower the robot, the robot makes a decision every step it takes to insure it stays on the line. The algorithm was simple initiate the sensors, return the result, make a decision, and move the motors. The following pages will outline how this was achieved.

Image 1

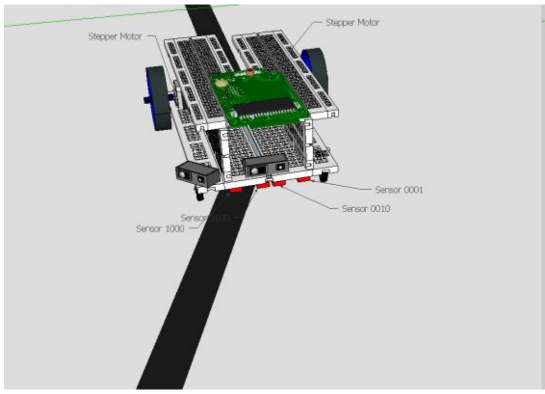
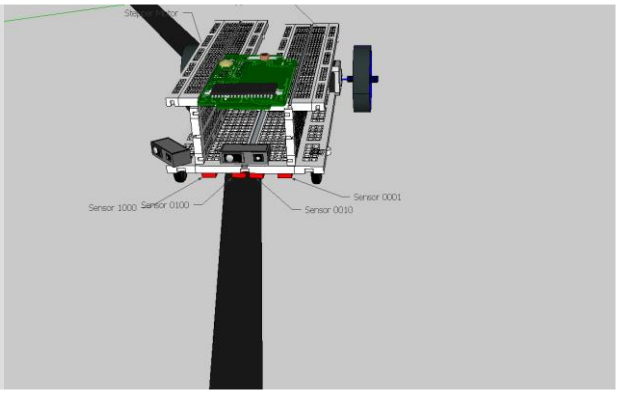


image 2



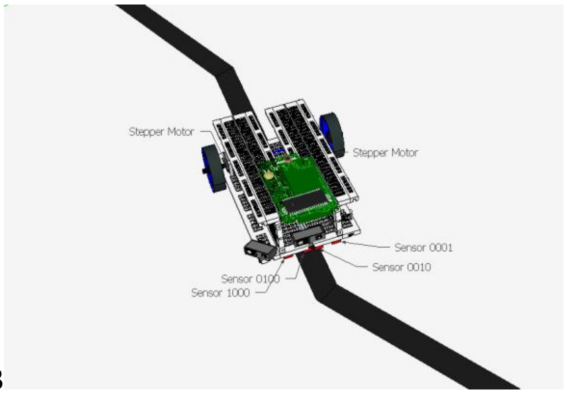


Image 3

Figure 23: Robot moving through the line and making decisions every step of the way Image 1 shows the robot moving forward stopping, analyzing the situation, and in figure 2 it corrects itself and continues to follow the line in image 3.

In order to achieve what can be seen in Figure 23, it can be broken down into 3 logical steps. As the sensor read the conditions set in front of them the value is stored in a byte with each bit representing the status of one particular sensor. If a bit is 1, then the corresponding sensor is over the line and if the bit is 0, then it is not on the line. The following examples will make it clear.

In image one in figure 23, If the value of the sensors is '**0110'**, or sensor 1=0 , sensor 2=1 , sensor 3=1, and sensor 4=0 respectively then the robot sees the line and move forward. In the original design consideration the value of that was returned was a decimal number and the values would have been added up to the corresponding decimal number condition. What was decided to do was have the sensor represent a hexadecimal value **0x0000** or **0x0101** for example. See the code on the following page on how this is calculated in the program.

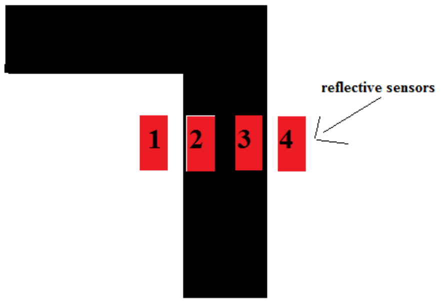
Code for determining what to do based on the sensors. This code is placed inside a while statement and continuously runs. The following code only shows a snip of the cases , there are 16 cases total, this code shows two but the other 14 cases are the basic same idea. After it does one case the program loops again and it performs the corresponding case based on the condition from the *final_move* below. In the image 1 as the robot moved forward the hexadecimal representation that may be seen from the sensors can be **0x0001**. This runs the corresponding case 0x0001, which triggers a sequence for the motor to move slightly to the right and them move forward slightly.

```
final_move =0;    // final move variable clearing to move to the next
             move
       move[0] = sensor_1000();  // sensor 1
       move[1] = sensor_0100();  //sensor 2
       move[2] = sensor_0010();  // sensor 3
       move[3] = sensor_0001();  // sensor 4
  // this is the driver of the system , the robot reacts based on the
// hexadecimal number that comes in for final_move
final_move = move[0]| move[1]| move[2] | move[3];    // check which condition is met
                                               based on the hexadecimal combination

      switch(final_move)    // This Checks the final move to see what
                               the motors need to do
         {                 // based on the combination for final move
                               Hexadecimal number it does the case corresponding
           case 0x0000:     // the motor turns the amount of loops in
                                each for loop so some cases might go
                                  longer than others
              for( i=0; i<30; i++)      // depending on the condition of the robot.
              {
               StepMotor_half_left_reverse(&motor_left);
               Delay(motor_delay);
               StepMotor_half_right_reverse(&motor_right);
               Delay(motor_delay);
              }
              check = 1;
              break;

          case 0x0001  :   // in this case the robot responds accordingly to 0x0001
              for( i=0; i<30; i++)
              {
               StepMotor_half_left(&motor_left);
               Delay (motor_delay);
               StepMotor_half_right_reverse(&motor_right);
               Delay (motor_delay);
              }
              for( i=0; i<50; i++)
              {
               StepMotor_half_left(&motor_left);
               Delay (motor_delay);
                StepMotor_half_right(&motor_right);
               Delay (motor_delay);
              }
```

In image 1 , figure 23, the sensor reads **0x0001** , during this case the robot will see the line in its far right and it terms will correct itself and turn right. This can be true for the case in which sees the sensors **'1100'** or **'0011'** or even **'1000'** and **'0001'**. If the most extreme cases **'1000'** are done or **'0001'** is implemented that means that is a need for a turn and the robot will turn accordingly. That means the speed and direction of both motors are adjusted.



<u>Figure 24:</u> Top to bottom view of what the sensors see this may be represented as '0110', in which case the robot stays moving forward In image 3 of figure 23.

A closer look of one of the functions for the reflectance sensors can be seen below and how it returns the correct hexadecimal value based on the reading of the each particular sensor themselves. The code below is for the sensor 0x1000 on the robot, it is controlled by pin PA11. First the pin is on , then it turns off , the reflectance sensor reads the value, a counter is set, and based on the value of the counter the value is returned to the main function as 0x1000 or 0x0000, respectively.

```
uint32_t sensor_1000()
  {
       int8_t count;
       //First reflective sensor (PA11)
       GPIOA->MODER &= ~(0x03 << (2*11));
       GPIOA->MODER |= (0x01 << (2*11));
       GPIOA->ODR = 0x1 << 11;    // turn on the pin then delay it
       Delay(50);

       count = 0;
       GPIOA->MODER &= ~(0x03 << (2*11));

       while (GPIOA->IDR & 0x800)
              count++;
       Delay(50);
       if (count > 10){
              return 0x1000; // value returned to use in final move
       }
       else{
              return 0x0000; // value returned to use in final move
       }
    }
```

Artwork:

For the Artwork objective, taken what we learned from the previous objectives ie, maze and line following, respectively, it was understood that for the artwork part of the project, it all comes down to fine tune motor control. This was exactly what was considered in the original design. Although in the original design, it was thought that a small motor could be incorporated to control the pen, we sought this to be completely unnecessary, and placing the pen dead center of the two stepper motors sufficed.

For qualification the task was to draw a simple square that was bigger than 9” by 9”, and smaller than 15” by 15”. Ideally a square that is approximately 12”. For the competition the task was to draw a visually appealing image, it was decided that a balloon was the best choice because everyone like balloons. Figure 25 below shows the breakdown of the basic components used to achieve the drawing feat.

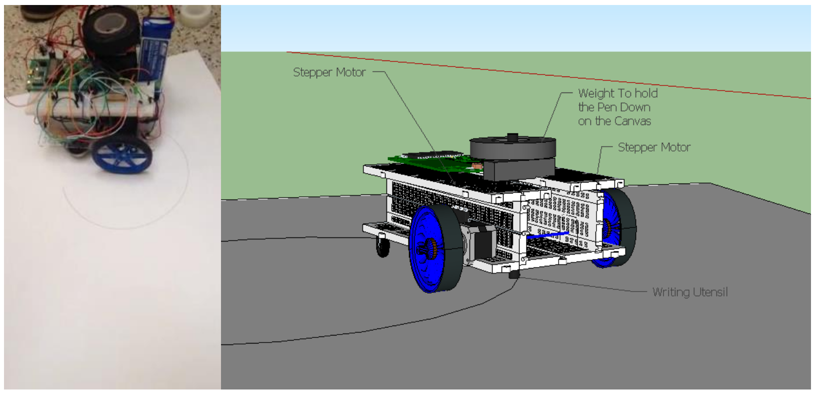


Figure 25: Shows the actual robot drawing a circle on the right and the 3D render of the robot outlining the different components used to make the drawing happen.

In order to achieve a circle the task was very easy and the implementation took no time at all. Using what was learned to control the robot in the line and maze competition the task of drawing the desired image we wanted was quite feasible. Below shows the code for the robot drawing a circle. This was implemented inside a while statement.

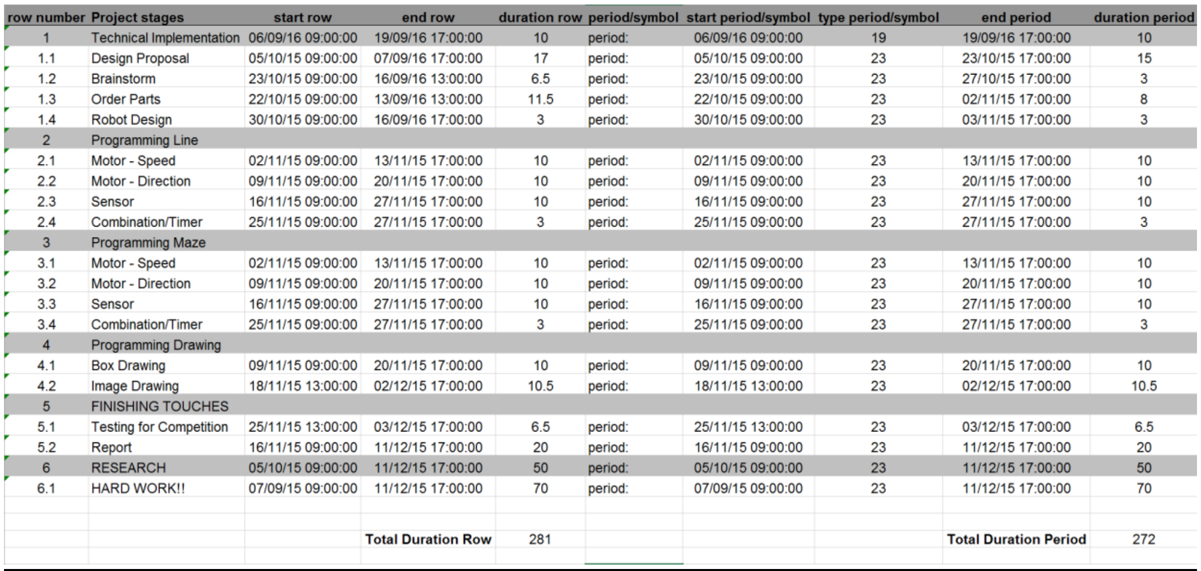
```
for(i=0; i<2190; i++)
{
   StepMotor_half_right(&motor_right); // BIG PERFECT CIRCLE
   Delay (motor_delay_draw);
}
```

Although the implantation for the drawing was simple, the build up to achieve such simplicity was not easy, motor function timing , and coordination with the motors and mechanical placement of the utensil made it challenging. But as far as the software implementation in main. It was fairly straight forward. The rest follows the same logic. Different for loop timing and motor direction.

(See appendix for al source code for all the different competitions, including header files used to implement each of programs used.

Project Management:

The project was managed really well, the design consideration and the actual implantation when hand in hand. No problems at all and the competitions were a success . the following gant chart shows the schedule that was followed.



row number	Project stages	start row	end row	duration row	period/symbol	start period/symbol	type period/symbol	end period	duration period
1	Technical Implementation	06/09/16 09:00:00	19/09/16 17:00:00	10	period:	06/09/16 09:00:00	19	19/09/16 17:00:00	10
1.1	Design Proposal	05/10/15 09:00:00	07/09/16 17:00:00	17	period:	05/10/15 09:00:00	23	23/10/15 17:00:00	15
1.2	Brainstorm	23/10/15 09:00:00	16/09/16 13:00:00	6.5	period:	23/10/15 09:00:00	23	27/10/15 17:00:00	3
1.3	Order Parts	22/10/15 09:00:00	13/09/16 13:00:00	11.5	period:	22/10/15 09:00:00	23	02/11/15 17:00:00	8
1.4	Robot Design	30/10/15 09:00:00	16/09/16 17:00:00	3	period:	30/10/15 09:00:00	23	03/11/15 17:00:00	3
2	Programming Line								
2.1	Motor - Speed	02/11/15 09:00:00	13/11/15 17:00:00	10	period:	02/11/15 09:00:00	23	13/11/15 17:00:00	10
2.2	Motor - Direction	09/11/15 09:00:00	20/11/15 17:00:00	10	period:	09/11/15 09:00:00	23	20/11/15 17:00:00	10
2.3	Sensor	16/11/15 09:00:00	27/11/15 17:00:00	10	period:	16/11/15 09:00:00	23	27/11/15 17:00:00	10
2.4	Combination/Timer	25/11/15 09:00:00	27/11/15 17:00:00	3	period:	25/11/15 09:00:00	23	27/11/15 17:00:00	3
3	Programming Maze								
3.1	Motor - Speed	02/11/15 09:00:00	13/11/15 17:00:00	10	period:	02/11/15 09:00:00	23	13/11/15 17:00:00	10
3.2	Motor - Direction	09/11/15 09:00:00	20/11/15 17:00:00	10	period:	09/11/15 09:00:00	23	20/11/15 17:00:00	10
3.3	Sensor	16/11/15 09:00:00	27/11/15 17:00:00	10	period:	16/11/15 09:00:00	23	27/11/15 17:00:00	10
3.4	Combination/Timer	25/11/15 09:00:00	27/11/15 17:00:00	3	period:	25/11/15 09:00:00	23	27/11/15 17:00:00	3
4	Programming Drawing								
4.1	Box Drawing	09/11/15 09:00:00	20/11/15 17:00:00	10	period:	09/11/15 09:00:00	23	20/11/15 17:00:00	10
4.2	Image Drawing	18/11/15 13:00:00	02/12/15 17:00:00	10.5	period:	18/11/15 13:00:00	23	02/12/15 17:00:00	10.5
5	FINISHING TOUCHES								
5.1	Testing for Competition	25/11/15 13:00:00	03/12/15 17:00:00	6.5	period:	25/11/15 13:00:00	23	03/12/15 17:00:00	6.5
5.2	Report	16/11/15 09:00:00	11/12/15 17:00:00	20	period:	16/11/15 09:00:00	23	11/12/15 17:00:00	20
6	RESEARCH	05/10/15 09:00:00	11/12/15 17:00:00	50	period:	05/10/15 09:00:00	23	11/12/15 17:00:00	50
6.1	HARD WORK!!	07/09/15 09:00:00	11/12/15 17:00:00	70	period:	07/09/15 09:00:00	23	11/12/15 17:00:00	70
			Total Duration Row	281				Total Duration Period	272

Figure 26: Task and the Amount of time it took

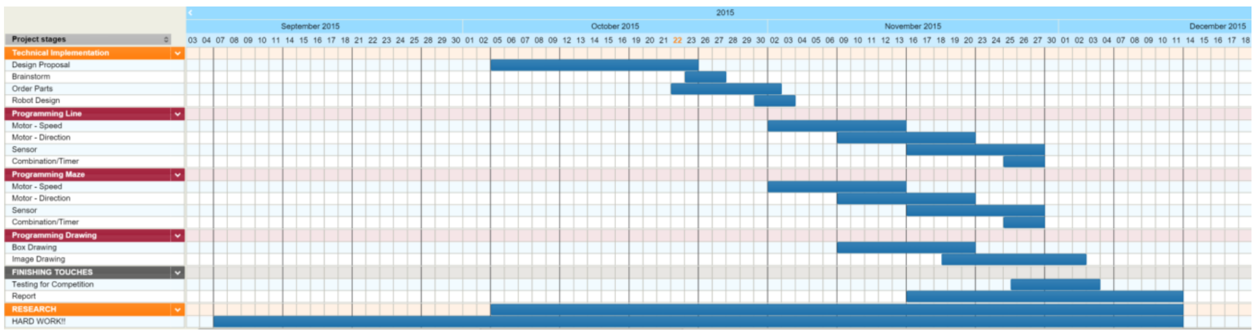
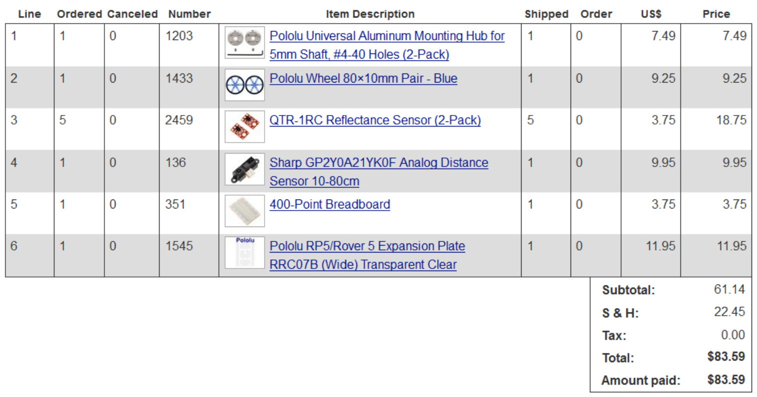


Figure 27: Gant Chart showing Work distribution

All members of the group contributed equally and the project was managed successfully with very little upsetting moments. It was **hard work** all the way.

<u>Cost Analysis: The cost was almost where we wanted to be , we did go a little above budget but that's okay for it was a succesfull build.</u>

Table 1: Total cost 1

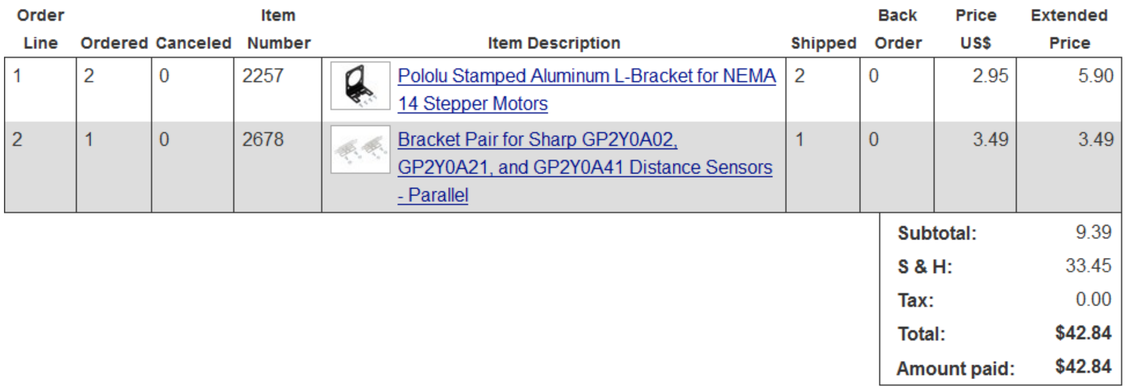


Line	Ordered	Canceled	Number	Item Description	Shipped	Order	US$	Price
1	1	0	1203	Pololu Universal Aluminum Mounting Hub for 5mm Shaft, #4-40 Holes (2-Pack)	1	0	7.49	7.49
2	1	0	1433	Pololu Wheel 80×10mm Pair - Blue	1	0	9.25	9.25
3	5	0	2459	QTR-1RC Reflectance Sensor (2-Pack)	5	0	3.75	18.75
4	1	0	136	Sharp GP2Y0A21YK0F Analog Distance Sensor 10-80cm	1	0	9.95	9.95
5	1	0	351	400-Point Breadboard	1	0	3.75	3.75
6	1	0	1545	Pololu RP5/Rover 5 Expansion Plate RRC07B (Wide) Transparent Clear	1	0	11.95	11.95

Subtotal:	61.14
S & H:	22.45
Tax:	0.00
Total:	$83.59
Amount paid:	$83.59

E-Flite 1800mAh 3S 11.1V 30C LiPo, 13AWG EC3
$32.99

Table 2: Cost for Additional Parts gathered.



Order Line	Ordered	Canceled	Item Number	Item Description	Shipped	Back Order	Price US$	Extended Price
1	2	0	2257	Pololu Stamped Aluminum L-Bracket for NEMA 14 Stepper Motors	2	0	2.95	5.90
2	1	0	2678	Bracket Pair for Sharp GP2Y0A02, GP2Y0A21, and GP2Y0A41 Distance Sensors - Parallel	1	0	3.49	3.49

Subtotal:	9.39
S & H:	33.45
Tax:	0.00
Total:	$42.84
Amount paid:	$42.84

Table 3: Cost for additional parts gathered.

Table 1 , Table 2, and **Table 3** in the previous page displays the cost for the Final Project. The stepper motor mount was needed to mount the stepper motor. This was the best approach rather than gluing the stepper motor into the board which is inconsistent and may lead to complications to the alignment that will hinder the speed/precision of the race for the competition. Total cost for the build came to be around close to **<u>$240</u>** . This is due to additional accessories bought such as batteries, LIPO charger, LEDS, Wires, Glue, Scissors, Electrical tape, regular tape, resistors, breadboards, and Microcontrollers because we burnt a few.

<u>Conclusion:</u>

Our team decided to tackle this project with the right attitude in order to obtain a successful build for this final project. All possible approaches were considered and we arrived to a final design which allowed for success in the completion. This mostly consists of determining what constraints would be involved when designing this embedded system in a manner that would satisfy the final project requirements.

Some comments about the project may be that it was way to expensive to build and for broke college students like us, this was just detrimental on our wallets. Another problem that we encountered was the fact that we had limited parts to use. We should be provided with spare parts just in case some of them broke. Over all it was hard work all the way and we had fun building and programing our robot.

APPENDIX A : Source Code used.

Header Files Used throughout all the programs: .h files

System Clock .h file:

```
//*****************************************************
// includes system clock
//*****************************************************

#include "stm32l1xx.h"
#include "math.h"

volatile uint32_t msTicks;    ///counts 1ms timeTicks
uint32_t SystemCoreClock    = 2097000;
uint32_t val=0;

/*------------------------------------------------------------------------
  SysTick_Handler
 //------------------------------------------------------------------------*/
void SysTick_Handler(void) {
  msTicks++;
}

/*--------------------------------------------------------------------------
 // delays number of tick Systicks (happens every 1 ms)
 //--------------------------------------------------------------------------*/
void Delay (uint32_t dlyTicks) {
  uint32_t curTicks;

  curTicks = msTicks;
  while ((msTicks - curTicks) < dlyTicks);
}
```

Motors Implementation .h files:

```
// *****************************HEADER FILE FOR MOTOR
// contains constants and Functions for Full step and half step for both motors
// HALF and Full step Right motor
// HALF and Full step LEFT motor
//*******************************************************************
//----------------------------------------------------------------
// Define Constants
#include "stm32l1xx.h"
#include "math.h"
//#include "system_clock.h"


#define F 1
#define R 2
#define O 0

//----------------------------------------------------------------

typedef struct _Motor
{
       unsigned long totalSteps;
       unsigned short step;
} Motor;
```

```
//----------------------------------------------------------------
// Prototypes
void coil_m1 (uint8_t dir, uint8_t p1, uint8_t p2);
void StepMotor_half_left(Motor *);  //HALF Step with LEFT   motor PB15,14,13,12
void StepMotor_full_left(Motor *);  //FULL Step with LEFT   motor PB15,14,13,12
void StepMotor_full_left_reverse (Motor *);

void StepMotor_half_right(Motor *); //HALF Step with RIGHT motor PC6,7,8,9
void StepMotor_full_right(Motor *); //FULL Step with RIGHT motor PC6,7,8,9
void StepMotor_full_right_reverse (Motor *);
/*----------------------------------------------------------------------------
  initialize LED Pins
 *----------------------------------------------------------------------------*/
void Motor_Init (void) {


       //this initializes the buttons and gpio pins outputs to pb4 to pb11 , buttons inputs pb12
and pb13
//  RCC->AHBENR |=  (1UL <<  1);                  /* Enable GPIOA,GPIOB,GPIOC, clock         */
       RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOCEN;

  GPIOB->MODER   &=  ~(0xFF000000); // PB15-12 for Stepper Motor LEFT     //GOOD
       GPIOB->MODER   |=   (0x55000000); // PB15-12 for Stepper Motor LEFT    //GOOD

       GPIOB->OTYPER  &=  ~(0xFF000000);   //clear
  GPIOB->OSPEEDR &=  ~(0xFF000000);   //clear

       GPIOC->MODER   &=  ~(0x000FF000); // PC 6-9 for Stepper Motor RIGHT    // GOOD
       GPIOC->MODER   |=   (0x00055000); // PC6-9 for Stepper Motor RIGHT   // GOOD

       GPIOC->OTYPER  &=  ~(0x000FF000);  //clear
  GPIOC->OSPEEDR &=  ~(0x000FF000);  //clear
}
/*----------------------------------------------------------------------------
  Function that outputs high for requested pin
 *----------------------------------------------------------------------------*/
void Pin_High (uint8_t num)
{
       if( num == 15 || num == 14 || num == 13 || num == 12 )
       GPIOB->BSRRL = (1<<num);
       else
       GPIOC->BSRRL = (1<<num);
}
/*----------------------------------------------------------------------------
  Function that outputs low for requested pin
 *----------------------------------------------------------------------------*/
void Pin_Low (uint8_t num)
{
       if( num == 15 || num == 14 || num == 13 || num == 12 )
       GPIOB->BSRRH = (1<<num);
       else
       GPIOC->BSRRH = (1<<num);
}

// Motor States GENERIC WorksWith all PINS
void coil_m1 (uint8_t dir,uint8_t p1, uint8_t p2){


       if(dir == F) {
              Pin_High(p1);
              Pin_Low(p2);
       };
       if(dir == R){
              Pin_Low(p1);
              Pin_High(p2);
       }
       if(dir == O) {
              Pin_Low(p1);
              Pin_Low(p2);
       }
       return;
```

```
}



//-------------------MOTOR CASES FULL/HALF STEPS -------------------------

// half step eight cases only wokrs with pins
void StepMotor_half_left (Motor *motor1){  //PB 15,14,13,12 WORKING HALF LEFT

        //4 // 5 // 6 //7

        switch (motor1->step){//15 //14 //13 //12
                case 0:
                        coil_m1(F,15,14);
                        coil_m1(O,13,12);
                        break;
                case 1:
                        coil_m1(F,15,14);
                        coil_m1(F,13,12);
                        break;
                case 2:
                        coil_m1(O,15,14);
                        coil_m1(F,13,12);
                        break;
                case 3:
                        coil_m1(R,15,14);
                        coil_m1(F,13,12);
                        break;
                case 4:
                        coil_m1(R,15,14);
                        coil_m1(O,13,12);
                        break;
                case 5:
                        coil_m1(R,15,14);
                        coil_m1(R,13,12);
                        break;
                case 6:
                        coil_m1(O,15,14);
                        coil_m1(R,13,13);
                        break;
                case 7:
                        coil_m1(F,15,14);
                        coil_m1(R,13,12);
                        break;
        }

        motor1->totalSteps++;
        motor1->step++;
        if (motor1->step == 8){
                motor1->step = 0;
        }
        return;

}
void StepMotor_half_left_reverse (Motor *motor1){  //PB 15,14,13,12 WORKING HALF LEFT

        //4 // 5 // 6 //7

        switch (motor1->step){//15 //14 //13 //12
                case 7:
                        coil_m1(F,15,14);
                        coil_m1(O,13,12);
                        break;
                case 6:
                        coil_m1(F,15,14);
                        coil_m1(F,13,12);
                        break;
                case 5:
                        coil_m1(O,15,14);
```

```
				coil_m1(F,13,12);
				break;
			case 4:
				coil_m1(R,15,14);
				coil_m1(F,13,12);
				break;
			case 3:
				coil_m1(R,15,14);
				coil_m1(O,13,12);
				break;
			case 2:
				coil_m1(R,15,14);
				coil_m1(R,13,12);
				break;
			case 1:
				coil_m1(O,15,14);
				coil_m1(R,13,13);
				break;
			case 0:
				coil_m1(F,15,14);
				coil_m1(R,13,12);
				break;
		}

		motor1->totalSteps++;
		motor1->step++;
		if (motor1->step == 8){
			motor1->step = 0;
		}
		return;

}

void StepMotor_full_left (Motor *motor1){  //PB 15,14,13,12 WORKING FULL LEFT


		switch (motor1->step)
			{
			case 0:
				coil_m1(F,15,14);
				coil_m1(O,13,12);
				break;
			case 1:
				coil_m1(O,15,14);
				coil_m1(F,13,12);
				break;
			case 2:
				coil_m1(R,15,14);
				coil_m1(O,13,12);
				break;
			case 3:
				coil_m1(O,15,14);
				coil_m1(R,13,12);
				break;
		 }

		motor1->totalSteps++;
		motor1->step++;
		if (motor1->step == 4){
			motor1->step = 0;
		}
		return;
}

void StepMotor_full_left_reverse (Motor *motor1){  //PB 15,14,13,12 WORKING FULL LEFT


		switch (motor1->step)
			{
			case 3:
				coil_m1(F,15,14);
```

```
                        coil_m1(O,13,12);
                        break;
                case 2:
                        coil_m1(O,15,14);
                        coil_m1(F,13,12);
                        break;
                case 1:
                        coil_m1(R,15,14);
                        coil_m1(O,13,12);
                        break;
                case 0:
                        coil_m1(O,15,14);
                        coil_m1(R,13,12);
                        break;
         }

        motor1->totalSteps++;
        motor1->step++;
        if (motor1->step == 4){
                motor1->step = 0;
        }
        return;
}

void StepMotor_half_right (Motor *motor1){ // PC 6,7,8,9 WORKING HALF RIGHT
                        //4 // 5 // 6 //
                       //6 // 7 // 8 // 9
        switch (motor1->step){//15 //14 //13 //12

                case 7:
                        coil_m1(F,6,7);
                        coil_m1(O,8,9);
                        break;
                case 6:
                        coil_m1(F,6,7);
                        coil_m1(F,8,9);
                        break;
                case 5:
                        coil_m1(O,6,7);
                        coil_m1(F,8,9);
                        break;
                case 4:
                        coil_m1(R,6,7);
                        coil_m1(F,8,9);
                        break;
                case 3:
                        coil_m1(R,6,7);
                        coil_m1(O,8,9);
                        break;
                case 2:
                        coil_m1(R,6,7);
                        coil_m1(R,8,9);
                        break;
                case 1:
                        coil_m1(O,6,7);
                        coil_m1(R,8,9);
                        break;
                case 0:
                        coil_m1(F,6,7);
                        coil_m1(R,8,9);
                        break;
        }

        motor1->totalSteps++;
        motor1->step++;
        if (motor1->step == 8){
                motor1->step = 0;
        }
        return;
```

```
void StepMotor_half_right_reverse (Motor *motor1){ // PC 6,7,8,9 WORKING HALF RIGHT
                                          //4 // 5 // 6 //7
                                       //6 // 7 // 8 // 9
        switch (motor1->step){//15 //14 //13 //12

                case 0:
                        coil_m1(F,6,7);
                        coil_m1(O,8,9);
                        break;
                case 1:
                        coil_m1(F,6,7);
                        coil_m1(F,8,9);
                        break;
                case 2:
                        coil_m1(O,6,7);
                        coil_m1(F,8,9);
                        break;
                case 3:
                        coil_m1(R,6,7);
                        coil_m1(F,8,9);
                        break;
                case 4:
                        coil_m1(R,6,7);
                        coil_m1(O,8,9);
                        break;
                case 5:
                        coil_m1(R,6,7);
                        coil_m1(R,8,9);
                        break;
                case 6:
                        coil_m1(O,6,7);
                        coil_m1(R,8,9);
                        break;
                case 7:
                        coil_m1(F,6,7);
                        coil_m1(R,8,9);
                        break;
        }

        motor1->totalSteps++;
        motor1->step++;
        if (motor1->step == 8){
                motor1->step = 0;
        }
        return;

}

void StepMotor_full_right (Motor *motor1){   // PC 6,7,8,9 WORKING FULL RIGHT
        switch (motor1->step)
                {
                case 3:
                        coil_m1(F,6,7);
                        coil_m1(O,8,9);
                        break;
                case 2:
                        coil_m1(O,6,7);
                        coil_m1(F,8,9);
                        break;
                case 1:
                        coil_m1(R,6,7);
                        coil_m1(O,8,9);
                        break;
                case 0:
                        coil_m1(O,6,7);
                        coil_m1(R,8,9);
                        break;
         }

        motor1->totalSteps++;
        motor1->step++;
        if (motor1->step == 4){
                motor1->step = 0;
        }
        return;
}

void StepMotor_full_right_reverse (Motor *motor1){   // PC 6,7,8,9 WORKING FULL RIGHT

        switch (motor1->step)
                {
                case 0:
                        coil_m1(F,6,7);
                        coil_m1(O,8,9);
                        break;
                case 1:
                        coil_m1(O,6,7);
                        coil_m1(F,8,9);
                        break;
                case 2:
                        coil_m1(R,6,7);
                        coil_m1(O,8,9);
                        break;
                case 3:
                        coil_m1(O,6,7);
                        coil_m1(R,8,9);
                        break;
         }

        motor1->totalSteps++;
        motor1->step++;
        if (motor1->step == 4){
                motor1->step = 0;
        }
        return;
}
```

LCD Header File .h File:

```
#define bool _Bool

void LCD_Clock_Init(void);
void LCD_PIN_Init(void);
void LCD_Configure(void);
void reflective_sensors(void);

//void LCD_Display_String2(void); //white

/*  =========================================================================
                                LCD MAPPING
    =========================================================================
                   A
     _  ----------
COL |_| |\   |J  /|
       F| H  |  K |B
     _  |  \ | /  |
COL |_| --G-- --M--
        |   /| \  |
       E|  Q |  N |C
     _  | /  |P  \|
DP  |_| -----------
                   D
*/


/* Constant table for cap characters 'A' --> 'Z' */
const uint16_t CapLetterMap[26] = {
        /* A      B      C      D      E      F      G      H      I  */
        0xFE00,0x6714,0x1d00,0x4714,0x9d00,0x9c00,0x3f00,0xfa00,0x0014,
        /* J      K      L      M      N      O      P      Q      R  */
        0x5300,0x9841,0x1900,0x5a48,0x5a09,0x5f00,0xFC00,0x5F01,0xFC01,
        /* S      T      U      V      W      X      Y      Z  */
        0xAF00,0x0414,0x5b00,0x18c0,0x5a81,0x00c9,0x0058,0x05c0
};

/* Constant table for number '0' --> '9' */

const uint16_t NumberMap[10] = {
        /* 0      1      2      3      4      5      6      7      8      9  */
        0x5F00,0x4200,0xF500,0x6700,0xEa00,0xAF00,0xBF00,0x04600,0xFF00,0xEF00
};



volatile uint32_t msTicks;                      /* counts 1ms timeTicks       */
uint32_t SystemCoreClock    = 2097000;

/*----------------------------------------------------------------------------
  SysTick_Handler
 *----------------------------------------------------------------------------*/
void SysTick_Handler(void) {
  msTicks++;
}

/*----------------------------------------------------------------------------
  delays number of tick Systicks (happens every 1 ms)
*----------------------------------------------------------------------------*/
void Delay (uint32_t dlyTicks) {
  uint32_t curTicks;

  curTicks = msTicks;
 while ((msTicks - curTicks) < dlyTicks);
}


// Converts an ascii char to the a LCD digit
static void LCD_Conv_Char_Seg(uint8_t* c, bool point, bool column, uint8_t* digit) {
  uint16_t ch = 0 ;
  uint8_t i,j;

  switch (*c) {
    case ' ' :
      ch = 0x00;
      break;

    case '0':
    case '1':
    case '2':
    case '3':
    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
    case '9':
      ch = NumberMap[*c-0x30];
      break;

    default:
      /* The character c is one letter in upper case*/
      if ( (*c < 0x5b) && (*c > 0x40) ) {
        ch = CapLetterMap[*c - 'A'];
      }
      /* The character c is one letter in lower case*/
      if ( (*c <0x7b) && ( *c> 0x60) ) {
        ch = CapLetterMap[*c - 'a'];
      }
      break;
  }

  /* Set the digital point can be displayed if the point is on */
  if (point) {
    ch |= 0x0002;
  }

  /* Set the "COL" segment in the character that can be displayed if the column is on */
  if (column) {
    ch |= 0x0020;
  }

  for (i = 12,j=0; j<4; i-=4,j++) {
    digit[j] = (ch >> i) & 0x0f; //To isolate the less signifiant dibit
  }
}

// This function is to display a given ASCII character on a specified position
// Input:
// ch: the ASCII value of the character to be display
// colon: The colon boolean flag indicates whether the colon is display.
// position: The LCD can display six decimal digits, thus the position is between 1 and 6.
void LCD_WriteChar(uint8_t* ch, bool point, bool colon, uint8_t position,uint8_t cm) {
  uint8_t digit[4];     /* Digit frame buffer */

  // Convert displayed character in segment in array digit
  LCD_Conv_Char_Seg(ch, point, colon, digit);


  while ((LCD->SR & LCD_SR_UDR) != 0); // Wait for Update Display Request Bit

    LCD->RAM[4] &= 0x2000; // clear bar 3
               LCD->RAM[6] &= 0x2000; // clear bar 2
               LCD->RAM[4] &= 0x8000; // clear bar 1
               LCD->RAM[6] &= 0x8000; // bar 0
       if (cm < 30)
       {
               LCD->RAM[6] |= 0x8000; // bar 0
       }
  if (cm <21)
       {
               LCD->RAM[4] |= 0x8000; // bar 1
       }
       if (cm < 13)
       {
               LCD->RAM[6] |= 0x2000; // bar 2
       }
       if (cm < 5)
       {
               LCD->RAM[4] |= 0x2000; // bar 3
       }


  switch (position) {
    /* Position 1 on LCD (Digit 1)*/
    case 1:
      LCD->RAM[0] &= 0xcffffffc;
      LCD->RAM[2] &= 0xcffffffc;
      LCD->RAM[4] &= 0xcffffffc;
      LCD->RAM[6] &= 0xcffffffc;

      LCD->RAM[0] |= ((digit[0]& 0x0c) << 26 ) | (digit[0]& 0x03) ; // 1G 1B 1M 1E
      LCD->RAM[2] |= ((digit[1]& 0x0c) << 26 ) | (digit[1]& 0x03) ; // 1F 1A 1C 1D
      LCD->RAM[4] |= ((digit[2]& 0x0c) << 26 ) | (digit[2]& 0x03) ; // 1Q 1K 1Col 1P
      LCD->RAM[6] |= ((digit[3]& 0x0c) << 26 ) | (digit[3]& 0x03) ; // 1H 1J 1DP 1N

      break;

    /* Position 2 on LCD (Digit 2)*/
    case 2:
      LCD->RAM[0] &= 0xf3ffff03;
      LCD->RAM[2] &= 0xf3ffff03;
      LCD->RAM[4] &= 0xf3ffff03;
      LCD->RAM[6] &= 0xf3ffff03;
      LCD->RAM[0] |= ((digit[0]& 0x0c) << 24 )|((digit[0]& 0x02) << 6 )|((digit[0]& 0x01) << 2 )
; // 2G 2B 2M 2E
      LCD->RAM[2] |= ((digit[1]& 0x0c) << 24 )|((digit[1]& 0x02) << 6 )|((digit[1]& 0x01) << 2 )
; // 2F 2A 2C 2D
      LCD->RAM[4] |= ((digit[2]& 0x0c) << 24 )|((digit[2]& 0x02) << 6 )|((digit[2]& 0x01) << 2 )
; // 2Q 2K 2Col 2P
      LCD->RAM[6] |= ((digit[3]& 0x0c) << 24 )|((digit[3]& 0x02) << 6 )|((digit[3]& 0x01) << 2 )
; // 2H 2J 2DP 2N

      break;

    /* Position 3 on LCD (Digit 3)*/
    case 3:
      LCD->RAM[0] &= 0xfcfffcff;
      LCD->RAM[2] &= 0xfcfffcff;
      LCD->RAM[4] &= 0xfcfffcff;
      LCD->RAM[6] &= 0xfcfffcff;

      LCD->RAM[0] |= ((digit[0]& 0x0c) << 22 ) | ((digit[0]& 0x03) << 8 ) ; // 3G 3B 3M 3E
      LCD->RAM[2] |= ((digit[1]& 0x0c) << 22 ) | ((digit[1]& 0x03) << 8 ) ; // 3F 3A 3C 3D
      LCD->RAM[4] |= ((digit[2]& 0x0c) << 22 ) | ((digit[2]& 0x03) << 8 ) ; // 3Q 3K 3Col 3P
      LCD->RAM[6] |= ((digit[3]& 0x0c) << 22 ) | ((digit[3]& 0x03) << 8 ) ; // 3H 3J 3DP 3N

      break;

    /* Position 4 on LCD (Digit 4)*/
    case 4:
      LCD->RAM[0] &= 0xffcff3ff;
      LCD->RAM[2] &= 0xffcff3ff;
      LCD->RAM[4] &= 0xffcff3ff;
      LCD->RAM[6] &= 0xffcff3ff;

      LCD->RAM[0] |= ((digit[0]& 0x0c) << 18 ) | ((digit[0]& 0x03) << 10 ) ; // 4G 4B 4M 4E
      LCD->RAM[2] |= ((digit[1]& 0x0c) << 18 ) | ((digit[1]& 0x03) << 10 ) ; // 4F 4A 4C 4D
      LCD->RAM[4] |= ((digit[2]& 0x0c) << 18 ) | ((digit[2]& 0x03) << 10 ) ; // 4Q 4K 4Col 4P
      LCD->RAM[6] |= ((digit[3]& 0x0c) << 18 ) | ((digit[3]& 0x03) << 10 ) ; // 4H 4J 4DP 4N

      break;

    /* Position 5 on LCD (Digit 5)*/
    case 5:
      LCD->RAM[0] &= 0xfff3cfff;
      LCD->RAM[2] &= 0xfff3cfff;
      LCD->RAM[4] &= 0xfff3efff;
      LCD->RAM[6] &= 0xfff3efff;

      LCD->RAM[0] |= ((digit[0]& 0x0c) << 16 ) | ((digit[0]& 0x03) << 12 ) ; // 5G 5B 5M 5E
      LCD->RAM[2] |= ((digit[1]& 0x0c) << 16 ) | ((digit[1]& 0x03) << 12 ) ; // 5F 5A 5C 5D
      LCD->RAM[4] |= ((digit[2]& 0x0c) << 16 ) | ((digit[2]& 0x01) << 12 ) ; // 5Q 5K   5P
      LCD->RAM[6] |= ((digit[3]& 0x0c) << 16 ) | ((digit[3]& 0x01) << 12 ) ; // 5H 5J   5N

      break;

    /* Position 6 on LCD (Digit 6)*/
    case 6:
      LCD->RAM[0] &= 0xfffc3fff;
      LCD->RAM[2] &= 0xfffc3fff;
      LCD->RAM[4] &= 0xfffc3fff;
      LCD->RAM[6] &= 0xfffc3fff;

      LCD->RAM[0] |= ((digit[0]& 0x04) << 15 ) | ((digit[0]& 0x08) << 13 ) | ((digit[0]& 0x03) <<
14 ) ; // 6B 6G 6M 6E
      LCD->RAM[2] |= ((digit[1]& 0x04) << 15 ) | ((digit[1]& 0x08) << 13 ) | ((digit[1]& 0x03) <<
14 ) ; // 6A 6F 6C 6D
      LCD->RAM[4] |= ((digit[2]& 0x04) << 15 ) | ((digit[2]& 0x08) << 13 ) | ((digit[2]& 0x01) <<
14 ) ; // 6K 6Q    6P
      LCD->RAM[6] |= ((digit[3]& 0x04) << 15 ) | ((digit[3]& 0x08) << 13 ) | ((digit[3]& 0x01) <<
14 ) ; // 6J 6H   6N

      break;
```

```
     default:
      break;
  }

  LCD->SR |= LCD_SR_UDR;
  while ((LCD->SR & LCD_SR_UDD) == 0);
}

void LCD_PIN_Init(void){
        RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOCEN;

        GPIOA->MODER &= ~(0x0C03F00FC);
        GPIOA->MODER |= 0x802A00A8;

        GPIOA->AFR[0] &= ~(0x0FFF0);
        GPIOA->AFR[0] |=       0x0BBB0;
        GPIOA->AFR[1] &= ~(0x0F0000FFF);
        GPIOA->AFR[1] |=       0x0B0000BBB;

        GPIOB->MODER &= ~(0x0FFFF0FC0);
        GPIOB->MODER |= 0x0AAAA0A80;

        GPIOB->AFR[0] &= ~(0x0FFF000);
        GPIOB->AFR[0] |=       0x0BBB000;
        GPIOB->AFR[1] &= ~(0x0FFFFFFFF);
        GPIOB->AFR[1] |=       0x0BBBBBBBB;

        GPIOC->MODER &= ~(0x0FFF0FF);
        GPIOC->MODER |= 0x0AAA0AA;
        GPIOC->AFR[0] &= ~(0x0FF00FFFF);
        GPIOC->AFR[0] |=       0x0BB00BBBB;
        GPIOC->AFR[1] &= ~(0x0FFFF);
        GPIOC->AFR[1] |=       0x0BBBB;
}

void LCD_Configure(void) {
        LCD->CR &= ~LCD_CR_BIAS;
        LCD->CR |= (2UL<<5);
        LCD->CR &= ~LCD_CR_DUTY;
        LCD->CR |= (3UL<<2);
        LCD->FCR &= ~LCD_FCR_CC;
        LCD->FCR |= (4UL<<10);
        LCD->FCR &= ~LCD_FCR_PON;
        LCD->FCR |=LCD_FCR_PON_2;
        LCD->CR  |= LCD_CR_MUX_SEG;
        LCD->CR &= ~LCD_CR_VSEL;

        //LCD->FCR &= LCD_FCR_PS;
        //LCD->FCR |= 0x2<<22;

        //LCD->FCR &= ~LCD_FCR_BLINK;
        //LCD->FCR |= 0x3<<16;
        //LCD->FCR &= ~LCD_FCR_BLINKF;
        //LCD->FCR |= 0x7<<13;

        while (!(LCD->SR & LCD_SR_FCRSR)) ;
        LCD->CR |= LCD_CR_LCDEN;
        while (!(LCD->CR & LCD_CR_LCDEN));
        while (!(LCD->SR & LCD_SR_RDY)) ;

}



void LCD_Clock_Init(void){
        // Note from STM32L Reference Manual:
        // After reset, the RTC Registers (RTC registers and RTC backup registers) are protected
        // against possible stray write accesses. To enable access to the RTC Registers, proceed
as
```

```
        // follows:
        // 1. Enable the power interface clock by setting the PWREN bits in the RCC_APB1ENR
register.
        // 2. Set the DBP bit in the PWR_CR register (see Section 4.4.1).
        // 3. Select the RTC clock source through RTCSEL[1:0] bits in RCC_CSR register.
        // 4. Enable the RTC clock by programming the RTCEN bit in the RCC_CSR register.
        RCC->APB1ENR |= RCC_APB1ENR_PWREN;    // Power interface clock enable
        PWR->CR      |= PWR_CR_DBP;       // Disable Backup Domain write protection
        RCC->CSR                |= RCC_CSR_RTCSEL_LSI;// LSI oscillator clock used as RTC clock
        //LSI clock varies due to frequency dispersion
        //RCC->CSR              |= RCC_CSR_RTCSEL_LSE;// LSE oscillator clock used as RTC clock
        RCC->CSR                |= RCC_CSR_RTCEN;                    // RTC clock enable

        /* Disable the write protection for RTC registers */
        RTC->WPR = 0xCA;                // RTC write protection register (WPR)
        RTC->WPR = 0x53;// Write "0xCA" and "0x53" to unlock the write protection

        // Wait until MSI clock ready
        while((RCC->CR & RCC_CR_MSIRDY) == 0); // MSI Ready Flag is set by hardware

        /* Enable comparator clock LCD */
        RCC->APB1ENR |= RCC_APB1ENR_LCDEN;

        /* Enable SYSCFG */
        RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

        RCC->CSR |= RCC_CSR_LSION;
        while( (RCC->CSR & RCC_CSR_LSIRDY) == 0 );

        /* Select LSI as LCD Clock Source */
        RCC->CSR &= ~RCC_CSR_RTCSEL_LSI;
        RCC->CSR |= RCC_CSR_RTCSEL_LSI;    // LSI oscillator clock used as RTC and LCD clock
        RCC->CSR |= RCC_CSR_RTCEN;
}
void myLCD6(int color){  //postion  6
            while ((LCD->SR & LCD_SR_UDR) != 0); // Wait for Update Display Request Bit
        switch(color){
          case 1: //B
                        LCD->RAM[0] &= 0xfffc3fff;
      LCD->RAM[2] &= 0xfffc3fff;
      LCD->RAM[4] &= 0xfffc35ff;
      LCD->RAM[6] &= 0xfffc35ff;

      LCD->RAM[0] |= (0x00038000) ; // 6B 6G 6M
      LCD->RAM[2] |= (0x0002C000) ; // 6A 6C 6D
      LCD->RAM[4] |= (0x00004000) ; // 6P
      LCD->RAM[6] |= (0x00020000) ; // 6J

                  break;

               case 2:  //W
            LCD->RAM[0] &= 0xfffc3fff;
      LCD->RAM[2] &= 0xfffc3fff;
      LCD->RAM[4] &= 0xfffc35ff;
      LCD->RAM[6] &= 0xfffc35ff;

      LCD->RAM[0] |= (0x00024000) ; // 6B 6E
      LCD->RAM[2] |= (0x00018000) ; // 6F 6C
      LCD->RAM[4] |= (0x00010000) ; // 6Q
      LCD->RAM[6] |= (0x00004000) ; // 6N
        //Delay(500);
               break;
               default:
                       break;
        }
         LCD->SR |= LCD_SR_UDR;
  while ((LCD->SR & LCD_SR_UDD) == 0);
}
void myLCD5(int color){  //position 5
            while ((LCD->SR & LCD_SR_UDR) != 0); // Wait for Update Display Request Bit
        switch(color){
         case 1: //B
      LCD->RAM[0] &= 0xfff3cfff;
      LCD->RAM[2] &= 0xfff3cfff;
      LCD->RAM[4] &= 0xfff3e5ff;
      LCD->RAM[6] &= 0xfff3e5ff;

      LCD->RAM[0] |= (0x000C2000) ; // 5B 5G 5M
      LCD->RAM[2] |= (0x00043000) ; // 5A 5C 5D
      LCD->RAM[4] |= (0x00001000) ; // 5P
      LCD->RAM[6] |= (0x00040000) ; // 5J

                  break;

               case 2:  //W
      LCD->RAM[0] &= 0xfff3cfff;
      LCD->RAM[2] &= 0xfff3cfff;
      LCD->RAM[4] &= 0xfff3e5ff;
      LCD->RAM[6] &= 0xfff3e5ff;

      LCD->RAM[0] |= (0x00041000) ; // 5B 6E
      LCD->RAM[2] |= (0x00082000) ; // 5F 5C
      LCD->RAM[4] |= (0x00080000) ; // 5Q
      LCD->RAM[6] |= (0x00001000) ; // 5N
        //Delay(500);
               break;
               default:
                       break;
        }
         LCD->SR |= LCD_SR_UDR;
  while ((LCD->SR & LCD_SR_UDD) == 0);
}

void myLCD4(int color){  //position 4
         while ((LCD->SR & LCD_SR_UDR) != 0); // Wait for Update Display Request Bit
        switch(color){
         case 1: //B
      LCD->RAM[0] &= 0xffcff3ff;
      LCD->RAM[2] &= 0xffcff3ff;
      LCD->RAM[4] &= 0xffcff5ff;
      LCD->RAM[6] &= 0xffcff5ff;

      LCD->RAM[0] |= (0x00300800) ; // 4G 4B 4M
      LCD->RAM[2] |= (0x00100C00) ; // 4A 4C 4D
      LCD->RAM[4] |= (0x00000400) ; // 4P
      LCD->RAM[6] |= (0x00100000) ; // 4J

                  break;

               case 2:  //W
      LCD->RAM[0] &= 0xffcff3ff;
      LCD->RAM[2] &= 0xffcff3ff;
      LCD->RAM[4] &= 0xffcff5ff;
      LCD->RAM[6] &= 0xffcff5ff;

      LCD->RAM[0] |= (0x00100400) ; // 4B 4E
      LCD->RAM[2] |= (0x00200800) ; // 4F 4C
      LCD->RAM[4] |= (0x00200000) ; // 4Q
      LCD->RAM[6] |= (0x00000400) ; // 4N
        //Delay(500);
               break;
               default:
                       break;
        }
         LCD->SR |= LCD_SR_UDR;
  while ((LCD->SR & LCD_SR_UDD) == 0);
}
void myLCD3(int color){  //position 3
         while ((LCD->SR & LCD_SR_UDR) != 0); // Wait for Update Display Request Bit
        switch(color){
         case 1:                        //B
      LCD->RAM[0] &= 0xfcfffcff;
      LCD->RAM[2] &= 0xfcfffcff;
      LCD->RAM[4] &= 0xfcfff5ff;
      LCD->RAM[6] &= 0xfcfff5ff;

      LCD->RAM[0] |= (0x03000200) ; // 3G 3B 3M
      LCD->RAM[2] |= (0x01000300) ; // 3A 3C 3D
      LCD->RAM[4] |= (0x00000100) ; // 3P
      LCD->RAM[6] |= (0x01000000) ; // 3J
                  break;
               case 2:  //W
      LCD->RAM[0] &= 0xfcfffcff;
      LCD->RAM[2] &= 0xfcfffcff;
      LCD->RAM[4] &= 0xfcfff5ff;
      LCD->RAM[6] &= 0xfcfff5ff;

      LCD->RAM[0] |= (0x01000100) ; // 3B 3E
      LCD->RAM[2] |= (0x02000200) ; // 3F 3C
      LCD->RAM[4] |= (0x02000000) ; // 3Q
      LCD->RAM[6] |= (0x00000100) ; // 3N
               //Delay(500);
               break;
               default:
                       break;
        }
         LCD->SR |= LCD_SR_UDR;
  while ((LCD->SR & LCD_SR_UDD) == 0);
}

void LCD_Display(uint16_t data){  //SIDE SENSOR
 uint16_t cm=0;
        uint8_t num[2], distance;
        uint8_t i;
        uint8_t count;

  cm =(4096 * 1000)/((300 * data) - 13 * 4096);  // good formula for side sensor (0A41SK)
        //cm -= 4;
        if (cm > 30)
                cm = 30;
        if (cm < 4)
                cm =4;

        num[0] = cm/10;
        num[1] = cm%10;

        for(i = 0; i<2; i++)
        {
                distance = num[i] + 48;
                LCD_WriteChar(&distance,0,0,i+1,cm);
        }
}
void LCD_Display_2(uint16_t data){   //FRONT SENSOR
 uint16_t cm=0;
        uint8_t num[2], distance;
        uint8_t i;
        uint8_t count;

        cm =(4096 * 1000)/((155 * data) - 13 * 4096);   // good formula for front sesnor (2Y0A21)
        //cm -= 4;
        if (cm > 30)
                cm = 30;
        if (cm < 4)
                cm =4;
        num[0] = cm/10;
        num[1] = cm%10;

        for(i = 0; i<2; i++)
        {
                distance = num[i] + 48;
                LCD_WriteChar(&distance,0,0,i+4,cm);
        }
}
```

APPENDIX B: Header Files for Distance Sensors and Reflectance Sensors

Distance Sensor .h File:

```
void distance(void);
void distance();
        RCC->AHBENR    |=RCC_AHBENR_GPIOAEN; //Enable GPIOA for ultrasonic
//Set PA 4 as analog input
// BY DEFINITION PA 4 HAS ANALOG INPUT SIGNAL CONNECTED TO CHANNEL 4
// ADC_IN4
        GPIOA->MODER           &= ~(0x03 << (2*4));
        GPIOA->MODER           |= (0x03 << (2*4)); // 11 is analog

         // Set PA 5
        GPIOA->MODER           &= ~(0x03 << (2*5));
        GPIOA->MODER           |= (0x03 << (2*5));

        //I/O at 2MHz low speed 01
        GPIOA->OSPEEDR         &= ~(0x03 << (2*4)); //bit clear
        GPIOA->OSPEEDR         |= (0x01 << (2*4)); // or 01 on PA4

        GPIOA->OSPEEDR         &= ~(0x03 << (2*5)); //bit clear
        GPIOA->OSPEEDR         |= (0x01 << (2*5)); // or 01 on PA5

        //output push pull state
        GPIOA->OTYPER          &= ~(1<<4); //Reset state 0 on PA4
        GPIOA->OTYPER          &= ~(1<<5); //Reset state 0 on PA5

        //Pull up Pull down Register
        GPIOA->PUPDR           &= ~(0x03 << (2*4)); //clears for no pupdr
        GPIOA->PUPDR           &= ~(0x03 << (2*5)); //clears for no pupdr
//--------------------------------
       RCC->CR |= RCC_CR_HSION;
        while ((RCC->CR & RCC_CR_HSIRDY)==0);
        RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
        ADC1->CR1 &= ~(ADC_CR1_RES);  //reset
        ADC1->CR1 |= ADC_CR1_EOCIE;   //end of conversion
        ADC1->CR1 |= (ADC_CR1_SCAN);  //scan
         ADC1->SQR1 &= ~ADC_SQR1_L;    // clears 00000 for 1 conversion
        ADC1->SQR5 |= (4&ADC_SQR5_SQ1);                       // 1st conversion in regular sequence
        ADC1->SMPR3 &= ~ADC_SMPR3_SMP4;
        ADC1->SMPR3  |= ADC_SMPR3_SMP4;               // 111: 384 cycles for channel 4
        ADC1->CR2       &= ~(ADC_CR2_DELS);           //clear delay
        ADC1->CR2       |= (ADC_CR2_DELS_0);          //no delay
        ADC1->CR2 |= ADC_CR2_ADON;                    //AD converter on
        ADC1->CR2      &= ~(ADC_CR2_EXTEN);           //trigger on rising edge bit clear
        ADC1->CR2   |= (0x03)<<28;                    //trigger on rising edge

        ADC1->CR2 |= ADC_CR2_SWSTART;                         //start of conversion
}

uint16_t sensor_side(uint16_t data){  //SIDE SENSOR
 uint16_t cm=0;
        uint8_t num[2], distance;
        uint8_t i;
        uint8_t count;

  cm =(4096 * 1000)/((155 * data) - 13 * 4096);  // good formula for side sensor (0A41SK)
        //cm -= 4;
        if (cm > 40)   // constrain the results to be between 4 and 40
               cm = 40;
        if (cm < 4)
               cm =4;

        num[0] = cm/10;    // lcd purpose
        num[1] = cm%10;

        return cm;
}
```

```
uint16_t sensor_front(uint16_t data){    //FRONT SENSOR
 uint16_t cm=0;
        uint8_t num[2], distance;
        uint8_t i;
        uint8_t count;

        cm =(4096 * 1000)/((155 * data) - 13 * 4096);    // good formula for front sesnor (2Y0A21)
        //cm -= 4;
        if (cm > 30)
                cm = 30;
        if (cm < 4)
                cm =4;

        num[0] = cm/10;
        num[1] = cm%10;

        return cm;
}
```

REFLECTANCE SENSORS .h File:

```
//*****************************************************
// HEader file including reflectance sensor implementation
// Each of the functions returns a hexadecimal
// which represents the position of the actual reflectance sensor
// from left to right.
// Returned from each funtion is a hex value 0x1000,0x0100,0x0010, 0x0001
//*****************************************************
#include <stdbool.h>  // importated library used for booleans
uint32_t sensor_1000()
{
          int8_t count;
          uint8_t color[2];

                color[0] = 'B';
                color[1] = 'W';
                //First reflective sensor (PA11)
                GPIOA->MODER &= ~(0x03 << (2*11));
                GPIOA->MODER |= (0x01 << (2*11));
                GPIOA->ODR = 0x1 << 11;     // turn on the pin  then delay it
                Delay(50);

                count = 0;
                GPIOA->MODER &= ~(0x03 << (2*11));

                while (GPIOA->IDR & 0x800)
                        count++;
                Delay(50);
                if (count > 10){

                        return 0x1000; // value returned to use in final move
                }
                else{

                        return 0x0000; // value returned to use in final move
                }
}
uint32_t sensor_0100()
{
          int8_t count;
          uint8_t color[2];

                color[0] = 'B';
                color[1] = 'W';
                //Second reflective sensor (PA12)
                GPIOA->MODER &= ~(0x03 << (2*12));
                GPIOA->MODER |= (0x01 << (2*12));
                GPIOA->ODR = 0x1 << 12;      // turn on the pin  then delay it
                Delay(50);
```

```
		count = 0;
		GPIOA->MODER &= ~(0x03 << (2*12));

		while (GPIOA->IDR & 0x1000)
			count++;
		Delay(50);
		if (count > 10)
		{
			return 0x0100; // value returned to use in final move
		}
		Else
		{
			return 0x0000; // value returned to use in final move
		}
}
uint32_t sensor_0010()
{
      int8_t count;
         uint8_t color[2];

		color[0] = 'B';
		color[1] = 'W';
		//Third reflective sensor (PB6)
		GPIOB->MODER &= ~(0x03 << (2*6));
		GPIOB->MODER |= (0x01 << (2*6));
		GPIOB->ODR = 0x1 << 6;        // turn on the pin  then delay it
		Delay(50);

		count = 0;
		GPIOB->MODER &= ~(0x03 << (2*6));

		while (GPIOB->IDR & 0x40)
			count++;
		Delay(50);
		if (count > 10)
		{
		   return 0x0010; // value returned to use in final move
		}
		Else
		{
		   return 0x0000; // value returned to use in final move
		}
}
uint32_t sensor_0001()
{
         int8_t count;
         uint8_t color[2];
		color[0] = 'B';
		color[1] = 'W';
			       //Fourth reflective sensor (PB7)
		GPIOB->MODER &= ~(0x03 << (2*7));
		GPIOB->MODER |= (0x01 << (2*7));
		GPIOB->ODR = 0x1 << 7;     // turn on the pin  then delay it
		Delay(50);

		count = 0;
		GPIOB->MODER &= ~(0x03 << (2*7));

		while (GPIOB->IDR & 0x80)
			count++;
		Delay(50);
		if (count > 10)
		{
			return 0x0001;  // value returned to use in final move
		}
		Else
		{
			return 0x0000;  // value returned to use in final move
		}
}
```

APPEDIX C : All .C files Used

Draw Bot .c File:

```
//**********************************************
// Draw bot for art drawing implemenation
// consist of for loops which have fine tuned motor control
// motors contained in a specified header file
//******************************************************
#include "stm32l1xx.h"
#include "math.h"
#include "motor_header.h"
#include "system_clock.h"


int main(void)
{
        uint16_t i=0,j=0;
        uint16_t motor_delay=5;
        uint16_t motor_delay_draw=10;

       //variables used to make it easier to change the amount of
       // corner lenght , line length , or geometry of the shape being drawn
        uint16_t straight=300;
        uint16_t straight_less = 200;
        uint16_t r_big = 100;
        uint16_t r_little = 50;
        uint16_t turn=220;
        uint16_t less_90 =200;
        uint16_t more_90 =300;

       Motor motor_right; //RIGHT MOTOR
       Motor motor_left; //LEFT MOTOR


       if (SysTick_Config(SystemCoreClock / 1000)) { // SysTick 1 msec interrupts
   while (1);                                    // Capture error
 }

       Motor_Init();  // initilozing the PB AND PC PINS FOR MOTOR
       motor_right.step = 0;        // right motor
       motor_right.totalSteps = 0;  //right motor

       motor_left.step = 0;        // left motor
       motor_left.totalSteps = 0;  //left motor

//*******************************************************************
//** By changing the timings you can draw any shape one desires
// and adding more for loops
// as it stand it draws a ballon
//************************************************************
        while (1)
        {
                 for(i=0; i<2190; i++)
                              {
                              StepMotor_half_right(&motor_right); // BIG PERFECT CIRCLE
                              Delay (motor_delay_draw);
                              }
                              for(i=0; i<turn; i++)
                              {
                              StepMotor_half_right_reverse(&motor_right);   //90 after circle
                              Delay (motor_delay);
                              StepMotor_half_left(&motor_left);
                              Delay (motor_delay);
                              }
                              for(i=0; i<straight_less; i++)
                              {
```

```
StepMotor_half_right(&motor_right);  //straight A
Delay (motor_delay);
StepMotor_half_left(&motor_left);
Delay (motor_delay);
}
for(i=0; i<turn; i++)
{
StepMotor_half_right_reverse(&motor_right);   // 90 turn at A
Delay (motor_delay);
StepMotor_half_left(&motor_left);
Delay (motor_delay);
}
for(i=0; i<straight_less; i++)
{
StepMotor_half_right(&motor_right);   // a - b
Delay (motor_delay);
StepMotor_half_left(&motor_left);
Delay (motor_delay);
}
for(i=0; i<less_90; i++)
{
StepMotor_half_right_reverse(&motor_right);   // a - b
Delay (motor_delay);
StepMotor_half_left(&motor_left);
Delay (motor_delay);
}
for(i=0; i<straight_less; i++)
{
StepMotor_half_right_reverse(&motor_right);   // b- c
Delay (motor_delay);
StepMotor_half_left(&motor_left);
Delay (motor_delay);
}
for(i=0; i<less_90; i++)
{
StepMotor_half_right_reverse(&motor_right);    //   1
Delay (motor_delay);
StepMotor_half_left(&motor_left);
Delay (motor_delay);
}
for(i=0; i<r_little; i++)
{
StepMotor_half_right(&motor_right);   // r little
Delay (motor_delay);
StepMotor_half_left(&motor_left);
Delay (motor_delay);
}
for(i=0; i<more_90; i++)
{
StepMotor_half_right(&motor_right);    //   2
Delay (motor_delay);
StepMotor_half_left_reverse(&motor_left);
Delay (motor_delay);
}
for(i=0; i<r_big; i++)
{
StepMotor_half_right(&motor_right);   // r big
Delay (motor_delay);
StepMotor_half_left(&motor_left);
Delay (motor_delay);
}
for(i=0; i<more_90; i++)
{
StepMotor_half_right_reverse(&motor_right);    //  3
Delay (motor_delay);
StepMotor_half_left(&motor_left);
Delay (motor_delay);
}
 for(i=0; i<r_big; i++)
{
StepMotor_half_right(&motor_right);   // r big
```

```
                                Delay (motor_delay);
                                StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                                }
                                for(i=0; i<more_90; i++)
                                {
                                StepMotor_half_right(&motor_right);    //   4
                                Delay (motor_delay);
                                StepMotor_half_left_reverse(&motor_left);
                                Delay (motor_delay);
                                }
                                for(i=0; i<more_90; i++)
                                {
                                StepMotor_half_right_reverse(&motor_right);    //   5
                                Delay (motor_delay);
                                StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                                }
                               for( i=0; i<r_little; i++)
                                {
                                StepMotor_half_right(&motor_right);   // r big
                                Delay (motor_delay);
                                StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                                }
                                for(i=0; i<less_90; i++)
                                {
                                StepMotor_half_right_reverse(&motor_right);    //   5
                                Delay (motor_delay);
                                StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                                }
                                for(i=0; i<straight_less; i++)
                                {
                                StepMotor_half_right(&motor_right);    //   5
                                Delay (motor_delay);
                                StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                                }
                                for(i=0; i<turn; i++)
                                {
                                StepMotor_half_right_reverse(&motor_right);    // 90
                                Delay (motor_delay);
                                StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                                }
                                for(i=0; i<straight; i++)
                                {
                                StepMotor_half_right(&motor_right);    //  5
                                Delay (motor_delay);
                                StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                                }
                                for(i=0; i<turn; i++)
                                {
                                StepMotor_half_right_reverse(&motor_right);    //  straight90
                                Delay (motor_delay);
                                StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                                }
                                for(i=0; i<straight_less; i++)
                                {
                                StepMotor_half_right(&motor_right);    // straight  5
                                Delay (motor_delay);
                                StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                                }

                                break;
        } //end main
```

Line Follower .c File:

```
//******************** Line Following Robot ******************************
// Contains LCD and Reflectance sensor implementation
//*****************************************************************************

#include "stm32l1xx.h"
#include "math.h"
#include <stdbool.h>
#include "motor_header.h"
//#include "LCD_header.h"   //***************** LCD Testing uncomment if want to use
#include "system_clock.h"
#include "reflector_sensors_header.h"

//void LCD_Display_Result(void);

int main(void) {
	uint32_t  final_move = 0x0000;  // hexadeximal number used to determine each case
	uint32_t move[4];

  static int check =0;   // checking which case statement is met
  int i=0;
	Motor motor_right; //RIGHT MOTOR
	Motor motor_left; //LEFT MOTOR
	//Delays for speed testing
	int motor_delay = 5;
	int led_delay =2;

	//LCD_Clock_Init();   //***************** LCD Testing uncomment if want to use
	 if (SysTick_Config(SystemCoreClock / 1000)) { /* SysTick 1 msec interrupts  */
while (1);                                  /* Capture error              */
  }
	Motor_Init();  // initilozing the PB AND PC PINS FOR MOTOR
	motor_right.step = 0;        // right motor
	motor_right.totalSteps = 0;  //right motor

	motor_left.step = 0;        // left motor
	motor_left.totalSteps = 0;  //left motor
//	LCD_PIN_Init();    //***************** LCD Testing uncomment if want to use
//	LCD_Configure();   //***************** LCD Testing uncomment if want to use
 while(1)
 {
	final_move =0;   // final move variable clearing to move to the next move
	move[0] = sensor_1000();  // sensor 1
	move[1] = sensor_0100();  //sensor 2
	move[2] = sensor_0010();  // sensor 3
	move[3] = sensor_0001();  // sensor 4

	//this is for led motor testing
	  GPIOA->MODER &= ~(0xF0);  //pa2 and 3
	  GPIOA->MODER |= 0x50;

	 // this is the driver of the system , the robot reacts based on the
	 // hexadecimal number that comes in for final_move
final_move = move[0]| move[1]| move[2] | move[3];    // check wich condition is met based on the
hexadecimal combination
  switch(final_move)    // This Checks the final move to see what the motors need to do
		{       // based on the comination for final move Hexadecimal number it does the
			      case corresponding
	case 0x0000:       // the motor turns the amount of loops in each for loop
			      so some cases might go longer than others
	 for( i=0; i<30; i++)      // depending on the condition of the robot.
	{
	 StepMotor_half_left_reverse(&motor_left);
	 Delay(motor_delay);
	StepMotor_half_right_reverse(&motor_right);
	 Delay(motor_delay);
	}
		check = 1;
	break;
```

```
case 0x0001  :
               for( i=0; i<30; i++)
               {
                StepMotor_half_left(&motor_left);
                Delay (motor_delay);
                StepMotor_half_right_reverse(&motor_right);
                Delay (motor_delay);
               }
               for( i=0; i<50; i++)
               {
                StepMotor_half_left(&motor_left);
                Delay (motor_delay);
                StepMotor_half_right(&motor_right);
                Delay (motor_delay);
               }
                       check = 2;
               break;

case 0x0010  :
               for( i=0; i<20; i++)
               {
                StepMotor_half_left_reverse(&motor_left);
                Delay (motor_delay);
                StepMotor_half_right(&motor_right);
                Delay (motor_delay);
               }
               for( i=0; i<50; i++)
               {
                StepMotor_half_left(&motor_left);
                Delay (motor_delay);
                StepMotor_half_right(&motor_right);
                Delay (motor_delay);
               }
                       check = 3;
               break;

case 0x0011  :
               for( i=0; i<50; i++)
               {
                       StepMotor_half_left(&motor_left);
                       Delay (motor_delay);
                       StepMotor_half_right_reverse(&motor_right);
                       Delay (motor_delay);
               }
               for( i=0; i<60; i++)
               {
                       StepMotor_half_left(&motor_left);
                       Delay (motor_delay);
                       StepMotor_half_right(&motor_right);
                       Delay (motor_delay);
               }
                       check = 4;
               break;

case 0x0100 :
               for( i=0; i<20; i++)
               {
                StepMotor_half_right(&motor_right);
                Delay(motor_delay);
               StepMotor_half_left_reverse(&motor_left);
                Delay(motor_delay);
               }
               for( i=0; i<50; i++)
               {
                StepMotor_half_right(&motor_right);
                Delay(motor_delay);
                StepMotor_half_left(&motor_left);
                Delay(motor_delay);
               }
```

```
        GPIOA->ODR = 0x1 << 2;
        Delay(2);
                check = 5;
        break;

case 0x0101  :
        for( i=0; i<30; i++)
        {
         StepMotor_half_right(&motor_right);
         Delay(motor_delay);
         StepMotor_half_left(&motor_left);
         Delay(motor_delay);
        }
        GPIOA->ODR = 0x1 << 3;
        Delay(2);

        check = 6;
        break;

 case 0x0110  :
        for( i=0; i<110; i++)
        {
         StepMotor_half_left(&motor_left);
         Delay(motor_delay);
         StepMotor_half_right(&motor_right);
         Delay (motor_delay);
        }
                check = 7;
        break;

  case 0x0111  :

for( i=0; i<50; i++)
 {
        StepMotor_half_left(&motor_left);
         Delay (motor_delay);
         StepMotor_half_right_reverse(&motor_right);
         Delay (motor_delay);
  }
 for( i=0; i<100; i++)
 {
        StepMotor_half_left(&motor_left);
        Delay (motor_delay);
        StepMotor_half_right(&motor_right);
        Delay (motor_delay);
  }

        GPIOA->ODR = 0x1 << 3;
        Delay(2);
                check = 8;
        break;

case 0x1000:

for( i=0; i<30; i++)
{
   StepMotor_half_right(&motor_right);
   Delay (motor_delay);
   StepMotor_half_left_reverse(&motor_left);
   Delay (motor_delay);
}
for( i=0; i<90; i++)
{
   StepMotor_half_right(&motor_right);
   Delay (motor_delay);
   StepMotor_half_left(&motor_left);
   Delay (motor_delay);
}
GPIOA->ODR = 0x1 << 2;
Delay(2);
check = 10;
```

```
                                    break;

                case 0x1001:
                for( i=0; i<30; i++)
                  {
                        StepMotor_half_right(&motor_right);
                        Delay (motor_delay);
                        StepMotor_half_left(&motor_left);
                       Delay (motor_delay);
                  }
                        GPIOA->ODR = 0x1 << 2;
                        Delay(2);
                               check = 10;
                        break;

                case 0x1010  :
                 for( i=0; i<30; i++)
                        {
                               StepMotor_half_right_reverse(&motor_right);
                               Delay (motor_delay);
                              StepMotor_half_left(&motor_left);
                                Delay (motor_delay);
                         }
                        for( i=0; i<50; i++)
                        {
                                StepMotor_half_right(&motor_right);
                                 Delay (motor_delay);
                                 StepMotor_half_left(&motor_left);
                                 Delay (motor_delay);
                  }
                        GPIOA->ODR = 0x1 << 2;
                        Delay(2);
                               check = 11;
                        break;

            case 0x1011  :
                 for( i=0; i<20; i++)
                {
                        StepMotor_half_right(&motor_right);
                        Delay (motor_delay);
                        StepMotor_half_left_reverse(&motor_left);
                        Delay (motor_delay);
                  }
                for( i=0; i<50; i++)
                 {
                   StepMotor_half_right(&motor_right);
                   Delay (motor_delay);
                   StepMotor_half_left(&motor_left);
                    Delay (motor_delay);
                  }
                        GPIOA->ODR = 0x1 << 2;
                        Delay(2);
                               check = 12;
                        break;

        case 0x1100  :
                        for( i=0; i<30; i++)
                       {
                             StepMotor_half_right(&motor_right);
                       Delay (motor_delay);
                        StepMotor_half_left_reverse(&motor_left);
                       Delay (motor_delay);
                        }
                      for( i=0; i<50; i++)
                        {
                        StepMotor_half_right(&motor_right);
                         Delay (motor_delay);
                         StepMotor_half_left(&motor_left);
                         Delay (motor_delay);
                         }
```

```
                                        GPIOA->ODR = 0x1 << 2;
                                        Delay(2);
                                        check = 13;
                                        break;

                                case 0x1101  :
                                         for( i=0; i<30; i++)
                                        {
                                         StepMotor_half_right(&motor_right);
                                         Delay (motor_delay);
                                                StepMotor_half_left_reverse(&motor_left);

                                         Delay (motor_delay);
                                        }
                                        for( i=0; i<50; i++)
                                        {
                                         StepMotor_half_right(&motor_right);
                                         Delay (motor_delay);
                                                StepMotor_half_left(&motor_left);
                                         Delay (motor_delay);
                                        }

                                        GPIOA->ODR = 0x1 << 2;
                                        Delay(2);
                                                check = 14;
                                        break;

                                        case 0x1110  :
                                        for( i=0; i<40; i++)
                                        {
                                         StepMotor_half_right(&motor_right);
                                         Delay (motor_delay);
                                        StepMotor_half_left_reverse(&motor_left);
                                         Delay (motor_delay);
                                        }
                                        for( i=0; i<50; i++)
                                        {
                                         StepMotor_half_right(&motor_right);
                                         Delay (motor_delay);
                                        StepMotor_half_left(&motor_left);
                                         Delay (motor_delay);
                                        }
                                        GPIOA->ODR = 0x1 << 2;
                                        Delay(2);
                                                check = 15;
                                        break;

                                        case 0x1111  :
                                        for( i=0; i<50; i++)
                                        {
                                         StepMotor_half_right(&motor_right);
                                         Delay (motor_delay);
                                         StepMotor_half_left(&motor_left);
                                         Delay (motor_delay);
                                        }
                                                check = 16;
                                        break;
                    }
                    i=0;
  }
}
```

Maze Follower .c file:

```
#include "stm32l1xx.h"
#include "math.h"
//#include "LCDmainHeader.h"   ////**************Uncomment to use the LCD *****************LCD
#include "distanceSensors.h"
#include "motor_header.h"
#include "system_clock.h"


int main(void)
{  uint16_t result2,result, result1,counter=0;  //variables needed for conversions
   uint16_t front_sensor =0,side_sensor=0;      // seperating front and side sensor
   uint16_t i=0,j=0;
   uint16_t motor_delay=5;   // motor delay defines the speed of the robot

  Motor motor_right; //RIGHT MOTOR
  Motor motor_left; //LEFT MOTOR

//              LCD_Clock_Init();    //*****************Uncomment to use the LCD**********************LCD
                if (SysTick_Config(SystemCoreClock / 1000)) { // SysTick 1 msec interrupts
   while (1);                              // Capture error
 }

  //LCD_PIN_Init();    //****************Uncoment To use the LCD*************************LCD
  //LCD_Configure();   //****************UNComment To use the LCD************************LCD
  distance();
  Motor_Init();  // initilozing the PB AND PC PINS FOR MOTOR
  motor_right.step = 0;       // right motor
  motor_right.totalSteps = 0;  //right motor

  motor_left.step = 0;       // left motor
  motor_left.totalSteps = 0;  //left motor

 while (1)
              {
     ADC1->CR2  |= ADC_CR2_SWSTART;  //ADC Conversion

     if(!(ADC1->SR & ADC_SR_EOC))
                {
                  result1 = ADC1->DR;
                 //LCD_Display_2(result1);   //***************uncomment to view LCD********************LCD
                 front_sensor=          sensor_front(result1);
                }
                if((ADC1->SR & ADC_SR_EOC))
                {
                 result2 = ADC1->DR;
                 //LCD_Display(result2);    //***************uncomment to view LCD*********************LCD
                 side_sensor= sensor_side(result2);
                }

                // below are the conditions in which the robot will face moving throught the maze
                if( side_sensor >= 10  && side_sensor <= 20 )   // if is between 10 and 20cm away then keep moving foward
                            {
                              for(i=0; i<4; i++)
                             {
                                        StepMotor_half_right(&motor_right); // half step stepper motor right
                                        Delay (motor_delay);
                                        StepMotor_half_left(&motor_left); // half step stepper motor left
                                        Delay (motor_delay);
                             }
                            }
```

```
         else if ( side_sensor < 10)   // if it gets less than 10cm then shift a bit to the left
                 {
                         for(i=0; i<3; i++)
                                 {
                                 StepMotor_half_right(&motor_right); // half step stepper motor right
                                 Delay (motor_delay);
                                 StepMotor_half_left_reverse(&motor_left); // half step stepper motor left
                                 Delay (motor_delay);
                                 }
                 }
         else if ( side_sensor > 20 && side_sensor < 30)   // if is greater than 20 cm and less than 30cm adjust yourself to move right
                 {
                         for(i=0; i<30; i++)
                                 {
                                 StepMotor_half_right_reverse(&motor_right); // half step stepper motor right reverse
                                 Delay (motor_delay);
                                 StepMotor_half_left(&motor_left); // half step stepper motor left
                                 Delay (motor_delay);
                                 }
                 }
         else if ( side_sensor >= 30)   // if it become greater than 30 then perform the turn right sequence
                 {
                         for(i=0; i< 400; i++)
                                 {
                                 StepMotor_half_right(&motor_right); // half step stepper motor right
                                 Delay (motor_delay);
                                 StepMotor_half_left(&motor_left); // half step stepper motor left
                                 Delay (motor_delay);
                                 }
                                 for(i=0; i<150; i++)
                                 {
                                 StepMotor_half_right_reverse(&motor_right); // half step stepper motor right reverse
                                 Delay (motor_delay);
                                 StepMotor_half_left(&motor_left); // half step stepper motor left
                                 Delay (motor_delay);
                                 }
                  }
         }// end of while loop
} //end main
```