CPEN 291

Project 1 Report

Contribution summary:

Sammy Brache 16.66%

Additional functionality, circuit design

Zhihang Zhang 16.66%

Motor controller, shift register, RPM measurement, Primary functionality 1 re-implementation

Amir Tootooni 16.66%

Shift register IO extension, protoboard design and soldering, mode & lever switch and ISR, Primary functionality 1 re-implementation

Robin Reyes 16.66%

Primary functionality 2, reflective sensor circuit design & soldering

Richard Tian 16.66%

Primary functionality 1, Primary functionality 2

Johnny Ma 16.66%

Primary functionality 1, Primary functionality 2

B. Introduction and motivations

This project report is to complement our team's version of the multi-functional robot. The main objective of the project was to study and practice using software and hardware to integrate a fundamental aspect of embedded programming, that being the use of external feedback to facilitate spatial awareness. Furthermore, our group in particular was interested in exploring the ability to implement precise movements for our robot. Many of the design considerations made during the project were used to find the best implementation of the robot, as well as to modularize the different components of both hardware and software in an attempt to minimize our time spent debugging.

C. Project Description (the body of the report)

Principle Functionality 1

To implement the `keepStraight()` function, we first tried hard measuring and recording the output speed values, given various PWM values. These values were then graphed as PWM versus speed to figure out the relative speed values for each motor. However, this approach did not work, as for different voltage sources the ratio between PWM and speed stayed the same, but the actual values would be different. Our second approach was to use a PID to fix the errors in the actual speed produced to align it to the wanted speed. We first tried using the PID library itself to no avail, as it took too long to speed up and the errors spiked really high at random points. After partially writing our own PID library, we realized that this approach may not work as well as we thought. Our third approach was to set both wheels to the same PWM, check the speed of each wheel, and then set the right wheel's speed to the left wheel's speed and vice versa by using the ratio of the output speed to the input PWM. This worked for fast speeds, but since we saw that the relation from PWM to speed was exponential from the first implementation, the RPM input for lower speeds would result in a much greater error for the output speed. Our final implementation was to align the magnets on both sides to the hall effect sensor and move the robot straight by counting the rotations and keeping the distance travelled on both wheels the same. And if one wheel was ahead of the other wheel, the robot would stop the wheel that is ahead and let the other wheel catch up. This proved very effective even at long distances. However since this affected 90 degree turns as 90 degree turns would cause the robots wheels to be misaligned.

We finally stuck with our third implementation after rigorous testing and decided it would be the best of both worlds. Decently straight lines and a working 90 degree turn. For 90 degree turns, we compared using a timed delay to turn 90 degrees versus keeping track of the distance on wheel travels while the other wheel stays still. From testing, the later method proved to be most effective.

For the `slowDown()` implementation, principal function 1 calls `slowDown()` whenever the sonar senses a distance of 50cm or less in front of it. This function repeatedly calls our above `keepStraight()` function with a linear decreasing PWM causing the robot to slow down at a consistent rate until the speed is either 0, or the robot is too close to the object in front. From here, we used the servo and sonar, in the same fashion as the previous lab, to check both sides and turn into the side with the largest distance. Turns are orchestrated as described above.

Principle Functionality 2

The goal of adding functionality for the robot to follow a line had a focus of precision and tolerance in mind. The main decision this part of the project relied on was the number and placement of optical sensors to use as feedback. The original design relied on four optical sensors, placed in a trapezoidal fashion. This original placement decision was made so the robot could readjust its trajectory whenever one of the sensors read that it was hovering over tape. The need of four sensors was to ensure that even the tightest turns could be read by the robot.

However, when reviewing the specifications of the assignment and upon discovering that the tracks our robot would need to navigate did not have tight turns. Thus we decided that using two sensors to guide the robot would be sufficient. Rather than having the robot change its trajectory when one of its sensors senses tape, in this new layout, the robot’s trajectory would change if a sensor sensed if it was no longer sensing tape. This way, since the sensors are consistently over the tape, the robot would be recalculating its proper trajectory more frequently, and overall, the robot would follow the track more closely. Using only two sensors also reduces its power consumption.

To conserve space on the master circuit, we decided to place the circuit of each sensor off the master circuit and only attach wires onto the master circuit that corresponded to the sensor’s source, ground, and signal pins. Each sensors’ circuit was soldered onto a mini perfboard and hot glued to the underside of the robot.

Software wise, the arduino would first decide on whether each sensor was reading the reflectiveness of the tape or the floor by determining if the `analogRead()` value of each sensor was above or below a certain threshold, which was measured by the robot before running on the track by reading the reflectiveness values of the tape and the floor around it. Based on the readings, the robot would then decide its direction by writing a PWM to the left or right motor. If both sensors read that they were both on the tape, the robot would go straight by setting both motors to the same PWM. If the right sensor read the tape and the left sensor read the floor, the robot would turn right by setting the PWM of the right motor to zero and the PWM of the left motor to non-zero, until the left sensor reads the tape again. Conversely so, turning left would follow a similar behaviour. If both sensors read the floor, the robot would stop by sending both motors a PWM of zero. We also implemented a counter that only stopped the robot if both sensors read the floor for a long enough duration to suppress the error a single reading would make or if the robot overshot a turn. Doing the initial reading of the floor and tape values to calculate a threshold ensures that our robot is not reliant on magic numbers and can be adaptable to any environment.

Additional functionality:

Our robot's additional functionality was the ability to write out words passed to it via a function called drawWord, as well as the addition of limit switches to detect collisions. The `drawWord(char *word)` function wrote out a word passed to it via its one parameter: `word`, a `char` array containing the word to write. The limit switches, when pushed, triggered an interrupt service routine (ISR) that moved forwards or backwards and turned left or right depending on which limit switch was triggered and the robot's surroundings at the time of the collision.

The helper functions used by `drawWord(char *word)` were `drawLine(int length, bool backtrack, float fraction)`, `rotate(int degrees)`, `liftPen()`, and `lowerPen()`. The `drawLine(int length, bool backtrack, float fraction)` function advanced the robot forwards a certain distance, and had three parameters. The parameters were: `length`, a `int` equal to the distance the robot should move in a straight line, `backtrack`, a `boolean` that was true if the robot should retrace the line it just drew and false if not, and `fraction`, a `float` that encoded what portion of the line the robot should retrace. For example, `drawLine(10, true, 0.5)` would cause the robot to move 10 units forward, then turn around and move to the halfway point of the line it just drew. If `backtrack` was true, rotate was called to turn the robot around 180 degrees, then `drawLine(length*fraction, false, 0.0)` to move the robot `length*fraction` units along the line just drawn.

The `rotate(int degrees)` function would simply rotate the robot a certain amount of degrees counterclockwise, its only parameter was the amount of degrees to rotate the robot. It was implemented by turning one wheel the number of rotations needed to turn the desired amount of degrees.

The `liftPen()` and `lowerPen()` functions were void parameter functions used to lift and lower the pen the robot used to draw.

The implementation of `drawWord(char *word)` was straight forward. If the word passed to it was greater than one letter long, then `drawWord(char *word)` would be called on each letter of the word. If the word passed to `drawWord(char *word)` was one character, then the function would enter a switch statement on that character. The switch statement had a case for each of the 26 letters of the alphabet, and numbers 0-9. Each case contained the calls to `drawLine(int length, bool backtrack, float fraction)`, `rotate(int degrees)`, `liftPen()` and `lowerPen()` that would result in the robot writing the correct letter. After a letter was drawn, the robot would draw a space, and rotate the robot to face "up the page". For example, below is the code to draw a letter L.

```
case 'l':
case 'L':
  lowerPen();
  drawLine(font_size, true, 1.0);
  rotate(90);
  lowerPen();
  drawLine(font_size/2, false, 0.0);
  liftPen();
  drawLine(space, false, 0.0);
  rotate(90);
return;
```

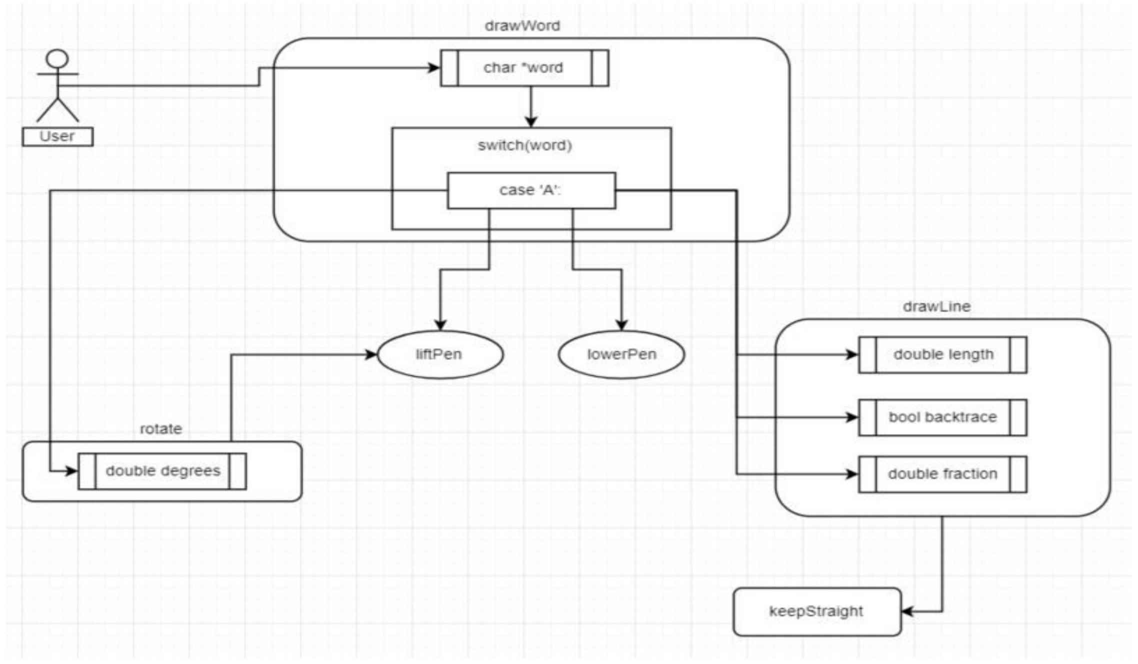
NOTE: The following characters and numbers were not implemented: {B, C, D, G, J, P, Q, R, S, U, Y, 2, 3, 4, 5, 6, 7, 8, 9}

During the demo, `drawWord(char *word)` was unable to complete its task. This can be attributed to the reliance of `drawWord(char *word)` on precise rotation to complete its task, the robot's inability to execute precise turns using our implementation. Our implementation allowed the robot to execute accurate, but not precise, 90 degree turns, so the "typeface" our robot would use to draw letters contained required only 90, -90 or 180 degree turns counter-clockwise. Even with the use of this typeface, our robot was unable to draw letters correctly because with each turn the robot would stray further from the path it should be following to draw a certain letter.

The additional hardware components used for the additional functionality were a servo motor, a crayon, and limit switches. The crayon was attached to the servo motor, which was mounted on the bottom and in the middle of the robot's chassis. When `lowerPen()` or `liftPen()` was called, the servo would turn 30 degrees down or up respectively. It was determined through trial and error that 30 degrees was the perfect angle of rotation to ensure that when lowered, the crayon would push hard enough on the ground to produce visible markings, but not so hard as to affect the alignment or attachment of the crayon with respect to the servo.

The lever switches were added an extra feature for all the functionalities of the robot. With the circuit design enabling us to have any on the 6 input pins of the shift register to be interrupt enabled, thus we were able to add two lever switches with interrupts so that whenever they are pushed the robot will move back/forward while beeping the piezo buzzer and changing the led colors accordingly to signify if it has detected the interrupt. For the interrupt to function properly we needed the body of the lever functions in the main loop, thus the other function called in the loop should not have long delays in them, as it will cause a lag in the lever switch functionality.

The software system diagram can be found below in Figure 1, while the hardware system diagram can be found in Appendix F.



D. Test and evaluations

Overall Testing

Initially, all of our arduino files were separated at the beginning. Each team member would test their code in separate files or separate branches until the test cases worked. Then if the implementation worked well, the file would be copied into the “main” folder and merged into the “master” branch. This allowed everyone to work on their part without creating conflicts. Testing parts before integrating them was particularly important as some functionalities relied on the same procedure, for instance the robot going straight was a feature that was required for principle functionality 1 and additional functionality.

Principle Functionality 1

Given that we tried a multitude of different approaches to make the robot go straight, we tested them in different ways. For PID-based implementation, we plotted the RPM/speed measured by two hall effect sensors and gave frictions to one wheel by hands and checked if the speed of that wheel restored to the setpoint. For the distance-based implementation, we run `align()` and `keepAligned()` a few times and checked the alignment of the magnets. We also tested the ultrasonic sensor by changing the distance from the sensor to the object in front of it.

We encountered a few problems when implementing principle functionality 1. In terms of hardware, one problem is the long sample time of RPM. Initially we had only one magnet that triggers each hall effect sensor in one cycle; meaning RPM is measured every revolution of the wheels. As a result, adjusting PWMs to keep straight would be difficult because once the robot realizes it should speed up one wheel, a noticeable amount of error has already accumulated. To shorten the sample time, we flipped one of the magnets so that RPM can be measured every half revolution. We also had difficulty implementing `align()`. Initially magnets were aligned based on the level-sensitivity, which is problematic since the initial position of two magnets is somewhat random. Even though the implementation guarantees that the readings from hall effect sensors are the same after execution, two magnets are not necessarily aligned. We finally changed the implementation to be based on edge sensitivity. That means two motors will be turned on regardless of the initial position of magnets, and they are stopped after one falling edge, thus two magnets will be aligned. Another problems is the inconsistent readings from the ultrasonic sensor. We handled that by taking multiple measurements and calculating the average.

To test the overall code for principle functionality 1, we first tested the robot in the hallway without any obstacles to make sure it goes straight. Then we put one object on the way to test slowdown function and turning function. Finally we put multiple obstacles and checked if the robot reacts correctly to each of them.

Principle Functionality 2

Before the completion of the hardware component, the software was first evaluated to determine if it worked algorithmically. After the completion of the circuits, two sensors were held by hand to simulate the movement of the actual robot, and made to follow a path of tape. A mock program was then written to print to the serial monitor what values the optical sensors read and which direction to turn. Through this simple testing, we concluded the ideal sensor positioning, polling time, and "turning threshold" values for the actual robot.

After integrating the sensors onto the robot, we used a method of gradual escalation to test our robot's ability to follow tracks of increasing complexity. Our robot was able to follow the each track consistently, with the only problem being that the robot would randomly stop. We inferred that this was caused by a false random reading of both sensors reading the floor, and it was for this reason we made the design decision to implement a counter to only stop the robot if it reads a certain amount of consecutive stop readings.

Though it worked consistently well on the few tracks, when we changed a track to the floor instead of the counter we used, or when we changed the lighting of the track, the optical sensor thresholds for the black tape to its surroundings became embedded with errors and was inaccurate. We realized this was due to the preset "turning threshold" values being incompatible with the different environment. To fix this issue, we wrote code that caused the robot to sense the optical values of the tape, rotate and sense the optical values of its surroundings, and then find a more accurate threshold value by finding an optimal between value of the two. This way, the robot could more accurately tell if one optical sensor was off the tape and would not rely on preset numbers.

Additional Functionality

Testing the `drawWord()` function was straight forward, as the function can draw a physical word, which we can tell if it is right. Since we rigorously tested and debugged the `keepStraight()` and `rotate90()` functions for principle functionality 1, we knew that these functions worked and could implement them into the `drawWord()` function. The biggest issue facing this functionality was attaching a writing utensil to the servo. A marker was too long and had to be attached to the side of the robot, making it difficult to attach, unbalanced, and changed the center of rotations. Thus we decided on a smaller crayon connected to the bottom center of the robot to draw letters, however the markings it made were much lighter. This also had us change the way rotate works as the fixed point of rotation now had to be at the centre of the robot, rather than one of the wheels of the robot. The function was then scripted to draw "LIT" and "LOL" one after the other rotating and lifting the crayon as needed, we tested this on printing paper and made sure it partially works.

Circuit Design

Designing the circuit required us to implement the circuit for each component separately first and then attempt to integrate everything into one circuit. We designed separate circuits for the hall effect sensors, the LM35 & servo, reflective optical sensors, LCD and shift registers. We made fritzing for all the circuit and then built them. In the initial stages of our design, we used the mentioned circuits built as prototypes on breadboards to test the circuit design and our robot functionalities. Once we were confident in the design we integrated the separate circuits into one breadboard. The design for the shift register was particularly challenging as we required input pins and shift registers are not designed for input pins. Thus we designed a circuit using diodes (and optionally NPN transistors) to read inputs from lever switches. Initially for the input enabled shift register, we designed the input pins using NPN BJTs so that they can be used with any kind of sensor and not only switches. The plus side of having input pins on the shift register was that we could interrupt enable as many pins as we liked by setting the corresponding shift register pins to high and connect the common input to pin 2 or 3 of the arduino board. The initial design in included appendix C (second image). As we decided to only use 2 reflective sensors instead of 4, there were enough input pins of the arduino board itself so connect to sensors other than switches. Thus we changed the circuit design to what it is now.

We needed to write separate code to test if the extra IO pins made through the shift registers work correctly (code included in files "ShiftRegIpinTest.ino", "ShiftRegIOpinTest.ino" and "ShiftRegisterFinal. ino"), after thoroughly testing the pin-extension circuit we integrated it to the main breadboard fritzing. When we had the final circuit design we made code specific to that circuit to test it (code include in file "breadBoardTest.ino"). We used the multimeter to check the voltage on various pins on the shift register and the scope to the check the noise caused by the servos. In the initial design we noticed high levels of noise when turning the servo so we added capacitors to short the power and thus reducing the noise significantly. When we were convinced the circuit implementation is correct we re-configured the circuit to fit it on ½ sized protoboard and soldered the parts and tested the circuit again with same procedure as its breadboard counterpart. We used the unused shift register output pins and connect them to two LED's and a buzzer to signify when interrupts are run.

Finally we measured the power consumption of the added circuit, arduino board, motors and the overall robot power consumption. Our circuit design required 1.29 watts for arduino uno and protoboard on average. There were also some spikes in current when the servos were moving which resulted in inaccurate sonar readings; we turned the LCD backlight off to avoid that problem. The motors need a maximum power of 2.45 watts when run on max speed and an average of about 2 watts. Thus the robot overall consumed an average power 3.74 watts. We believe, with the given functionalities the power consumption of the circuits are the best they could be, but some ways to improve them was to use the 8574 and 8574A (TI and NXP) IO extenders which are specifically built for input and out and are considerably more efficient than a shift-register-based input circuit.

Robot Layout

One of the main challenges in this project was to find the correct layout for the hall effect sensors as the correct movement of the robot was dependent on them. At the beginning we were planning to attach the magnets to the wheels themselves and and the sensors to the outside of the shassy. Because of the susceptibility to damage this design introduced to the sensors, we decided to change the magnet alignment and connect the magnets to the inside shaft of the motor blocks. Thus the hall effect sensors were attached on top of the motor gearbox. The magnets were connected back to back in a SN-(shaft)-SN alignment so that a full rotation of the wheels will put a falling edge on the hall effect output only once. After testing this magnet layout we found that having a wheel-rotation-resolution of one is insufficient for the rotation and move straight operations of the robot. Therefore we changed the magnet layout to SN-SN-(shaft)-NS-NS using 4 magnets on each shaft. We had to use 2 magnets on each side as the opposing magnetic fields cancel each other out and hall effect readings became inconsistent. This new layout increased the wheel-rotation-resolution to half a rotation which made the rotations and straight movement much smoother. For further improvement we should have added more magnets to increase the resolution even more, however we were to far into the project to make any major changes to the robot.

The protoboard was designed to have the pins connected to the arduino board under it and pins connect to different sensors and parts above it so that it seats above the arduino and acts like an extension shield. This design opened up space on the robot so we can attach the LCD, servo and power bank all on top of the robot.

E. Conclusions and Reflections

This project forced us to handle objectives that relied on precision, unlike any project we have had to handle before. Because of this, we struggled early on in trying to invoke behaviour in our robot that should work in ideal conditions. We soon learned that proper behaviour of our robot should not rely on accuracy dependent code due to the nature of analog input and output. This is especially true as the cost of the hardware we used meant that cannot be trusted to give extremely accurate results. From this lesson, we adjusted our design decisions to account for well thought out hardware integration, and wrote programs that accounted for the error the hardware produced.

Another important lesson learned is that when dealing with hardware that interacts with the outside world, reliability and atomicity is of utmost importance. For example, using the hall effect and magnets to measure the speed the wheel rotates is inefficient and introduces error to measurements. Thus, using the rising and falling edges of the hall effect and magnets is a better way to keep track of the robots wheels.

Because of the robot being our first project in a six person team, we encountered hardships in coordinating our work. Many of us had a hard time reading the code of other members, and trying to integrate our code into the larger program was difficult because of this. After addressing this issue, each member made more of an effort to document their code properly, and our workflow became more efficient as a result.

Despite our struggle in organizing our code, our team of six was able to work as a cohesive team from start to finish. One of the best practices we observed that resulted in this was our process of discussing our plans first before implementation and evenly dividing up the work. This way, all members of the group knew what was being worked on and the progress each subgroup was making in their endeavors. This also made it so that each member knew what had to be worked on next based on what was currently being developed. While our work on the robot continued right up to the deadline, we also feel proud in that none of the time we used was wasted.

F. References and bibliography

References

https://www.youtube.com/watch?v=ng2PVOePN68

https://www.youtube.com/watch?v=JEpWlTl95Tw

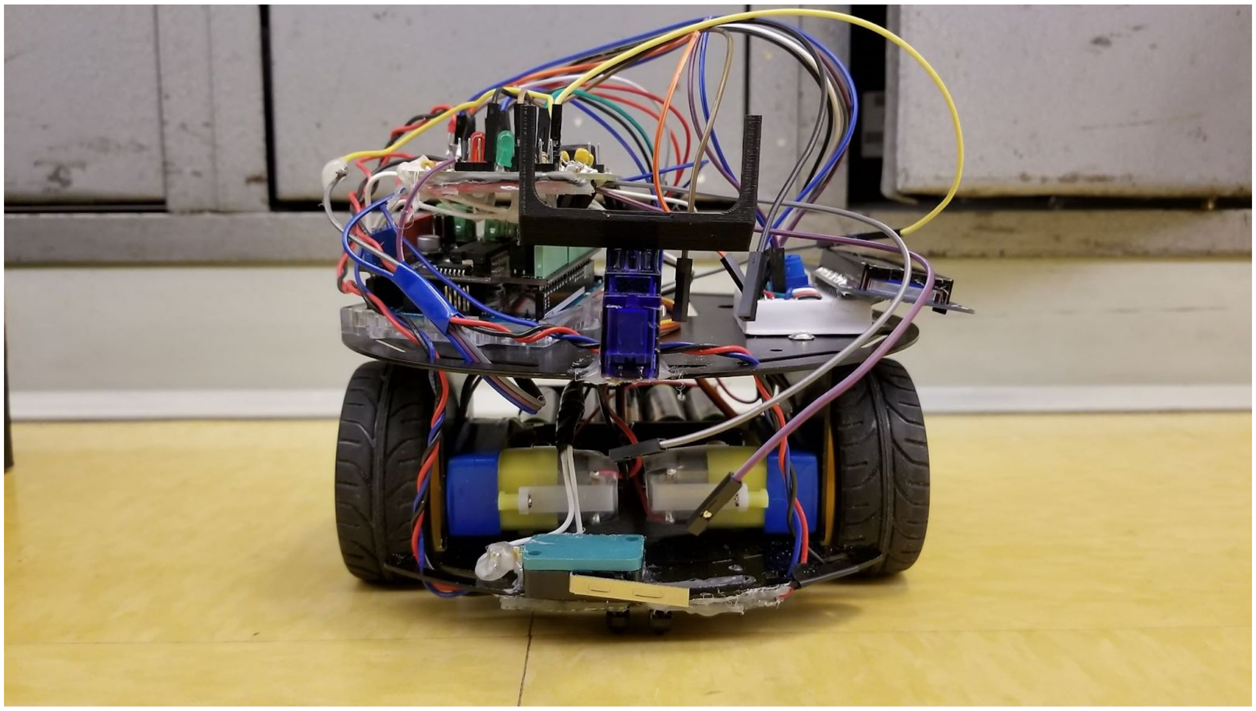
https://robotics.stackexchange.com/questions/167/what-are-good-strategies-for-tuning-pid-loops

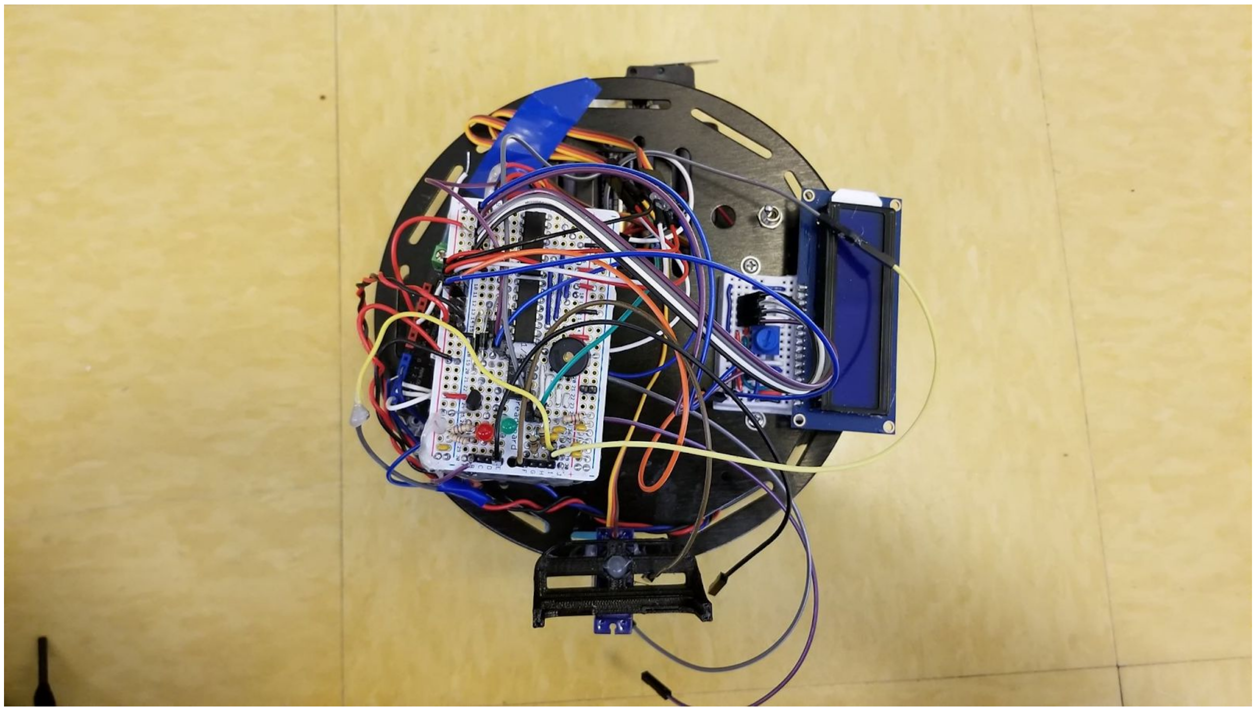
https://www.arduino.cc/en/Tutorial/ShiftOut

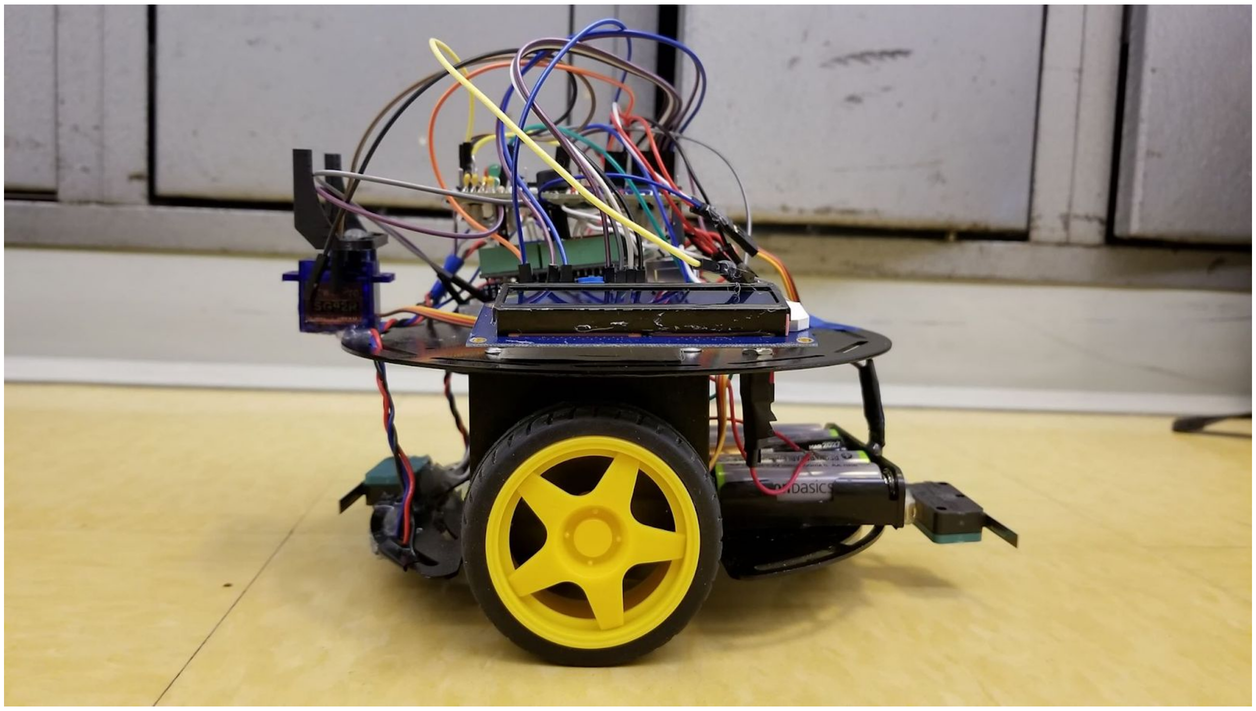
https://playground.arduino.cc/Code/PIDLibrary

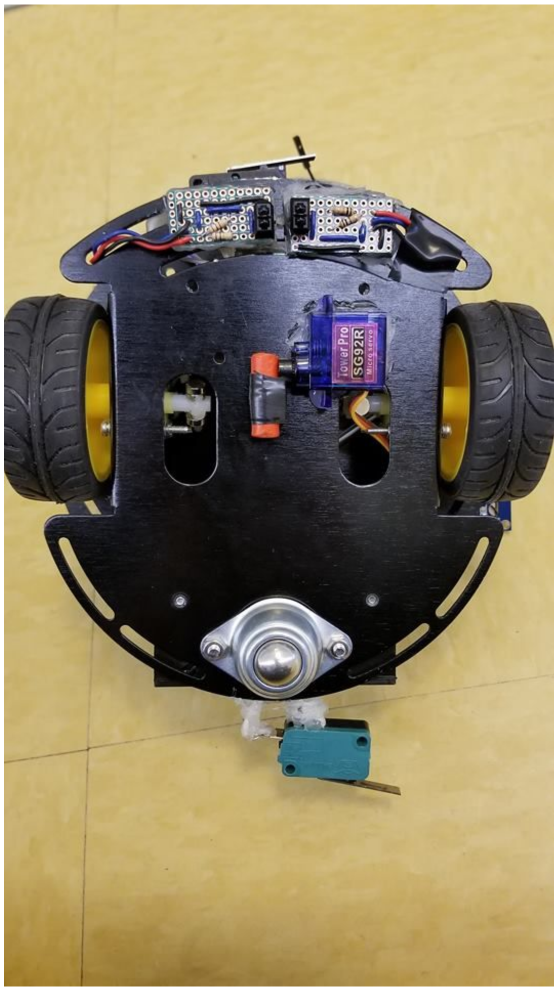
http://www.gammon.com.au/forum/bbshowpost.php?id=11518

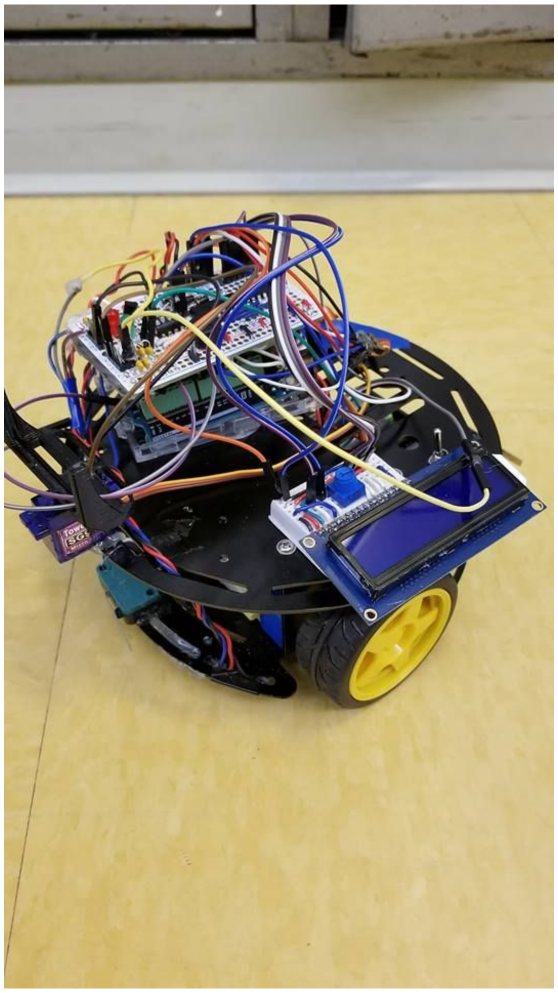
Appendix A – Robot pictures











Appendix B - Code

```
#include <Servo.h>
#include <math.h>

// shift registers pins
#define REG_INPUT_LEVER1 11 // front lever
#define REG_INPUT_LEVER2 10 // back lever

#define SW1 12
#define SW2 13
#define SW3 14
#define SW4 15

#define GREEN_LED 8
#define RED_LED 9

#define BUZZER 0
```

```
#define RS 3
#define E 2
#define D4 4
#define D5 5
#define D6 6
#define D7 7

/**********************************************
   MASTER PIN ASSIGNMENTS
 **********************************************/

#define LEFT_HALL_EFFECT 18 //a4
#define RIGHT_HALL_EFFECT 17 //a3
#define LM35 A5
#define COMMON_INPUT_PIN 2
#define TRIG 9
#define ECHO 3
#define CLOCK_PIN 8
#define SERVO 10
#define SERVO_PEN 11
#define LATCH_PIN 12
#define DATA_PIN 13

/**********************************************
   Variables to determine the functionality of the robot
 **********************************************/
unsigned char mode; // 1 for pf1 - 2 for pf2 - 3 for af
unsigned char mask = 0;
#define PF1 1
#define PF2 2
#define AF 3

/**********************************************
   GLOBAL VARIABLES FOR THE ULTRASONIC SENSOR AND SERVO
 **********************************************/
Servo myservo;
float rightDistance;
float leftDistance;
float currentDistance;
unsigned long interval = 0;
bool not_drawn = true;
Servo pen_servo;

/**********************************************
   CONSTANTS FOR THE MOTOR SHIELD
 **********************************************/
```

```
const float radius = 3.25 / 2; // assuming the radius is the same for two
wheels
const int numOfMotors = 2;

// distance of an object in front of the robot that indicates
// when the robot should stop and slow down
const int TOO_CLOSE = 30;
const int SLOW_DOWN = 50; // 50cm + circumference
//individual motors of the motor shield
// using struct to store infor about two motors and control them
// get help from:
https://www.dfrobot.com/wiki/index.php/2A_Motor_Shield_For_Arduino_Twin)_(SKU:
DRI0017)

boolean flag = true;

typedef struct
{
  const int enPin;
  const int directionPin;
  const int sensorPin;

  float RPM;
  float Speed;

  const int timeout; //  in millis

  int prev_Reading;
  int rotationCounter;
  unsigned long prev_Millis;
  const unsigned long debounce_Millis;
} MotorContrl; // define this struct as a new type called MotorControl

int E1Pin = 5;
int M1Pin = 4;
int E2Pin = 6;
int M2Pin = 7;

// initialize two motors
// E1 and M1 are right wheel, E2 and M2 are left wheel

MotorContrl Motors[] = {
  {E1Pin, M1Pin, RIGHT_HALL_EFFECT, 0, 0, 5000, 1, 0, 0, 30},
  {E2Pin, M2Pin, LEFT_HALL_EFFECT, 0, 0, 5000, 1, 0, 0, 30},
}; // create an array of MotorControl objects and initialize them 0

const int LEFT_FORWARD = LOW;
const int LEFT_BACKWARD = HIGH;
const int RIGHT_FORWARD = HIGH;
const int RIGHT_BACKWARD = LOW;
const unsigned char RIGHT_MOTOR = 0;
const unsigned char LEFT_MOTOR = 1;

// unsigned long bitPattern = 1<<17;
unsigned long bitPattern = 0x0c00; //pins 11&10 are the lever sensor and are
interupt inabled
unsigned char numOfBits = 16;

unsigned char sw1;
unsigned char sw2;
unsigned char sw3;
unsigned char sw4;
unsigned char interrupt = 0;

int font_size = 2;
int space = 1;
bool pen_up;
int prev_length;

void setup()
{
  Serial.begin(9600);

  //setup the servo motor, lm35, and the ultrasonic sensor

  pinMode(LM35, INPUT);    // Set LM35 to input
  pinMode(ECHO, INPUT);    // Set ECHO to input
  pinMode(TRIG, OUTPUT);   // Set TRIG to output
  pinMode(SERVO, OUTPUT); // Set SERVO to output
  pinMode(SERVO_PEN, OUTPUT); // Set SERVO_PEN to output

  pinMode(RIGHT_HALL_EFFECT, INPUT);
  pinMode(LEFT_HALL_EFFECT, INPUT);

  //attach servo to pin D9 and st its start position to 90
  myservo.attach(SERVO);
  myservo.write(80); //needs to be called twice
  delay(330);
  myservo.write(90);
  delay(330);
  pen_servo.attach(SERVO_PEN);
  pen_servo.write(0);
  delay(330);
  pinMode(M1Pin, OUTPUT);
  pinMode(M2Pin, OUTPUT);

  //LCD & Shift Register initialization

  pinMode(LATCH_PIN, OUTPUT);
  pinMode(CLOCK_PIN, OUTPUT);
  pinMode(DATA_PIN, OUTPUT);
  pinMode(COMMON_INPUT_PIN, INPUT);

  sw1 = digitalShiftRead(SW1);
  sw2 = digitalShiftRead(SW2);
  sw3 = digitalShiftRead(SW3);
  sw4 = digitalShiftRead(SW4);

  clearBits();
  if (sw4) // only attach interrupt if sw4 is on
    attachInterrupt(digitalPinToInterrupt(COMMON_INPUT_PIN), doISR, RISING);
  mode = setMode();

  // setup lcd and display relevant info
  setupLCD();
  blinkCurser(false);
  setCurserLocation(0, 0);
  writeLCDString("CPEN 291");
  setCurserLocation(1, 0);
  writeLCDString("Team G19");
  delay(1500);
  clearDisplay();
  setCurserLocation(0, 0);
  writeLCDString("Align Wheels");
  setCurserLocation(1, 0);
  writeLCDString("QB! GO! GO!");
  align();
  clearDisplay();
  setCurserLocation(0, 0);
  writeLCDString("Alignment Done!");
   for (int i = 5; i > 0; i--) {
    setCurserLocation(1, 0);
    String s = "Start in " + String(i) + " secs";
    writeLCDString(s);
    writeLCD(i); delay(1000);
  }
}

void loop()
{
  if (interrupt)
```

```
{
  mask = 0;
  clearDisplay();
  setCurserLocation(0, 0);
  writeLCDString("Lever Pushed!");
  digitalShiftWrite(RED_LED, LOW);
  digitalShiftWrite(GREEN_LED, HIGH);
  if (interrupt == 1) //front
  {
    setCurserLocation(1, 0);
    writeLCDString("Move Back!");
    for (int i = 0; i < 4; i++)
    {
      keepAligned(170, RIGHT_BACKWARD, LEFT_BACKWARD);
      buzz(BUZZER, 120);
    }
    keepAligned(170, RIGHT_FORWARD, LEFT_BACKWARD);

  }
  else //back
  {
    setCurserLocation(1, 0);
    writeLCDString("Move Forward!");
    for (int i = 0; i < 4; i++)
    {
      keepAligned(170, RIGHT_FORWARD, LEFT_FORWARD);
      buzz(BUZZER, 120);
    }
    keepAligned(170, RIGHT_FORWARD, LEFT_BACKWARD);
  }
  digitalShiftWrite(GREEN_LED, LOW);
  interrupt = 0;
}
else if (mode == PF1 && !interrupt)
{
  if (!mask)
  {
    clearDisplay();
    setCurserLocation(0, 0);
    writeLCDString("Function 1");
    mask = 1;
  }
  principal_function1();
}
else if (mode == PF2 && !interrupt)
{
  if (!mask)
```

```
    {
      clearDisplay();
      setCurserLocation(0, 0);
      writeLCDString("Function 2");
      mask = 1;
    }
    principal_function2();
  }
  else if (mode == AF && !interrupt && not_drawn)
  {
    if (!mask)
    {
      clearDisplay();
      setCurserLocation(1, 0);
      writeLCDString("Additional Function");
      mask = 1;
    }
    drawWord("L");
    drawWord("I");
    drawWord("T");
    not_drawn = false;
  }

  else if (!mask && !interrupt)
  {
    clearDisplay();
    setCurserLocation(0, 0);
    writeLCDString("ERROR: Set Mode");
    setCurserLocation(1, 0);
    writeLCDString("Switch modes & Restart");
    delay(1500);
    mask = 1;
  }
}

/****************************************************************************
****************************************************************************
**************************
   FUNCTIONS TO CHANGE MODE AND ISR

****************************************************************************
****************************************************************************
*************************/
unsigned char setMode()
{
  if (sw1)
    return PF1;
  if (sw2)
    return PF2;
  if (sw3)
    return AF;
  return 0;
}

//interuppt service routine
void doISR()
{
  int lever1 = digitalShiftRead(REG_INPUT_LEVER1);
  int lever2 = digitalShiftRead(REG_INPUT_LEVER2);
  if (lever1)
  {
    interrupt = 1;
    digitalShiftWrite(RED_LED, HIGH);
    digitalShiftWrite(GREEN_LED, LOW);
    keepAligned(170, RIGHT_BACKWARD, LEFT_BACKWARD);

  }
  else if (lever2)
  {
    interrupt = 2;
    digitalShiftWrite(RED_LED, LOW);
    digitalShiftWrite(GREEN_LED, HIGH);
    keepAligned(170, RIGHT_FORWARD, LEFT_FORWARD);
  }
}

/*******************************************************************************
*******************************************************************************
***************************
   LIBRARY FUNCTIONS FOR THE MOTOR

*******************************************************************************
*******************************************************************************
**************************/
//ALIGNS THE TWO WHEELS SO THAT THE HALL EFFECT SENSORS ARE IN THE SAME
POSITION
void align()
{
  int lastRh = digitalRead(RIGHT_HALL_EFFECT);
  int lastLh = digitalRead(LEFT_HALL_EFFECT);
  int rh;
  int lh;
  setMotorDirection(RIGHT_MOTOR, RIGHT_FORWARD);
  setMotorDirection(LEFT_MOTOR, LEFT_FORWARD);
  bool falling_left = false;
  bool falling_right = false;

  do
  {
    analogWrite(Motors[RIGHT_MOTOR].enPin, 90);
    analogWrite(Motors[LEFT_MOTOR].enPin, 90);

    rh = digitalRead(RIGHT_HALL_EFFECT);
    lh = digitalRead(LEFT_HALL_EFFECT);

    if ( (rh == LOW && lastRh == HIGH) || falling_right ) {
      analogWrite(Motors[RIGHT_MOTOR].enPin, 0);
      falling_right = true;
    }

    if ( (lh == LOW && lastLh == HIGH) || falling_left) {
      analogWrite(Motors[LEFT_MOTOR].enPin, 0);
      falling_left = true;
    }
    lastRh = rh;
    lastLh = lh;
  } while (!falling_left || !falling_right); // at least one of them is not
aligned

}/**
   @params:
          radius of the wheel in cm
   @return:
          speed in cm/sec
*/
float rpmToSpeed(float rpm, float r)
{
  return rpm * 2 * M_PI * r / 60;
}

void initializeMotor()
{
  for (int i = 0; i < numOfMotors; i++)
  {
    digitalWrite(Motors[i].enPin, LOW);
    pinMode(Motors[i].directionPin, OUTPUT);
    pinMode(Motors[i].enPin, OUTPUT);
  }
}

void resetRPMAndSpeed(int motorNum)
```

```
{
  Motors[motorNum].RPM = 0;
  Motors[motorNum].Speed = 0;
  Motors[motorNum].prev_Reading = 1;
  Motors[motorNum].rotationCounter = 0;
  Motors[motorNum].prev_Millis = 0;
}

/**void
   @params:
          motorNum, a number represents the motor whose speed is going to be
set
          0 is the right motor, and 1 is the left one.

          direction, either FORWARD or BACKWARD
*/
void setMotorDirection(int motorNum, int directionM)
{
  digitalWrite(Motors[motorNum].directionPin, directionM);
}

/**
   keeps the wheels aligned takes the direction on l and r motors
*/
unsigned long keepAligned(int PWM, int rd, int ld)
{
  int lastRh = digitalRead(RIGHT_HALL_EFFECT);
  int lastLh = digitalRead(LEFT_HALL_EFFECT);
  int rh;
  int lh;
  setMotorDirection(RIGHT_MOTOR, rd);
  setMotorDirection(LEFT_MOTOR, ld);
  bool falling_left = false;
  bool falling_right = false;

  unsigned long period = millis();
  do
  {
    analogWrite(Motors[RIGHT_MOTOR].enPin, PWM);
    analogWrite(Motors[LEFT_MOTOR].enPin, PWM);

    rh = digitalRead(RIGHT_HALL_EFFECT);
    lh = digitalRead(LEFT_HALL_EFFECT);

    if ( (rh == LOW && lastRh == HIGH) || falling_right ) {
      analogWrite(Motors[RIGHT_MOTOR].enPin, 0);
      falling_right = true;
    }

    if ( (lh == LOW && lastLh == HIGH) || falling_left) {
      analogWrite(Motors[LEFT_MOTOR].enPin, 0);
      falling_left = true;
    }

    lastRh = rh;
    lastLh = lh;

  } while (!falling_left || !falling_right); // at least one of them is not
aligned

  period = millis() - period;
  return period;
}

/*****************************************************************************
*****************************************************************************
***************************
   PRINCIPAL FUNCTION 1

*****************************************************************************
*****************************************************************************
**************************/
void principal_function1()
{
  //Measure Distance
  currentDistance = readDistance(1); // read once
  clearDisplay();
  setCurserLocation(0, 0);
  writeLCDString(String(currentDistance, 2));

  if (currentDistance > SLOW_DOWN)
    interval = keepAligned(255, RIGHT_FORWARD, LEFT_FORWARD);
  else if (currentDistance > TOO_CLOSE)
    interval = keepAligned(currentDistance * 4 + 50, RIGHT_FORWARD,
LEFT_FORWARD);
  else
  {
    align();
    readBothSides();

    if (leftDistance < rightDistance)
      keepAligned(160, RIGHT_BACKWARD, LEFT_FORWARD);

    else
```

```
      keepAligned(160, RIGHT_FORWARD, LEFT_BACKWARD);
    align();
  }
  setCurserLocation(1, 0);
  writeLCDString(String((60000.0 / (2 * interval)), 2));
}

/*****************************************************************************
*****************************************************************************
***************************
   PRINCIPAL FUNCTION 2

*****************************************************************************
*****************************************************************************
**************************/
int const TAPE = 0, FLOOR = 1;
int LEFT_OPTIC_BLACK_THRESH,  RIGHT_OPTIC_BLACK_THRESH;
boolean first = true;
int const LEFT_OPTIC_SENSOR = 1,  RIGHT_OPTIC_SENSOR = 2;

int stop_counter = 0;
int line_speed = 100;

void principal_function2(){
  if(first == true){
    delay(5000);
    clearDisplay();
    writeLCDString("SCANNING");
    int RIGHT_OPTIC_TAPE = analogRead(RIGHT_OPTIC_SENSOR); //read large value
when on tape
    int LEFT_OPTIC_TAPE = analogRead(LEFT_OPTIC_SENSOR);
    analogWrite(Motors[LEFT_MOTOR].enPin, 90);
    delay(2000);
    analogWrite(Motors[LEFT_MOTOR].enPin, 0);

    /*
     * More precise implementation that checks a range of values instead of
just on and off the tape
    //turn right until right sensor is on floor but left sensor is on tape
    setMotorDirection(RIGHT_MOTOR, BACKWARD);
    while(fabs(analogRead(RIGHT_OPTIC_SENSOR)-RIGHT_OPTIC_MAX) < 300){
      analogWrite(Motors[RIGHT_MOTOR].enPin,90);
    }
    analogWrite(Motors[RIGHT_MOTOR].enPin,0);
    setMotorDirection(RIGHT_MOTOR, FORWARD);
    int RIGHT_OPTIC_MIN = analogRead(RIGHT_OPTIC_SENSOR);
    if(analogRead(LEFT_OPTIC_SENSOR) < LEFT_OPTIC_MAX)
```

```
      LEFT_OPTIC_MAX = analogRead(LEFT_OPTIC_SENSOR);
    RIGHT_OPTIC_BLACK_THRESH = (RIGHT_OPTIC_MAX + RIGHT_OPTIC_MIN)/2;
    while(analogRead(RIGHT_OPTIC_SENSOR) < RIGHT_OPTIC_BLACK_THRESH){
      analogWrite(Motors[RIGHT_MOTOR].enPin,90); //keep turning right until
the left sensor is on the tape again
    }
    analogWrite(Motors[RIGHT_MOTOR].enPin,0);
    //turn left until left sensor is on floor but right sensor is on tape
    setMotorDirection(LEFT_MOTOR, BACKWARD);
    while(fabs(analogRead(LEFT_OPTIC_SENSOR)-LEFT_OPTIC_MAX) < 300){
      analogWrite(Motors[LEFT_MOTOR].enPin,90);
    }
    analogWrite(Motors[LEFT_MOTOR].enPin,0);
    setMotorDirection(LEFT_MOTOR, FORWARD);
    int LEFT_OPTIC_MIN = analogRead(LEFT_OPTIC_SENSOR);
    if(analogRead(RIGHT_OPTIC_SENSOR) < RIGHT_OPTIC_MAX)
      RIGHT_OPTIC_MAX = analogRead(RIGHT_OPTIC_SENSOR);
    LEFT_OPTIC_BLACK_THRESH = (LEFT_OPTIC_MAX + LEFT_OPTIC_MIN)/2;
    while(analogRead(LEFT_OPTIC_SENSOR) < LEFT_OPTIC_BLACK_THRESH){
      analogWrite(Motors[LEFT_MOTOR].enPin,90);
    }
    analogWrite(Motors[LEFT_MOTOR].enPin,0);
    */

    int RIGHT_OPTIC_FLOOR = analogRead(RIGHT_OPTIC_SENSOR);
    int LEFT_OPTIC_FLOOR = analogRead(LEFT_OPTIC_SENSOR);

    LEFT_OPTIC_BLACK_THRESH = (LEFT_OPTIC_TAPE + LEFT_OPTIC_FLOOR)/2;
    RIGHT_OPTIC_BLACK_THRESH = (RIGHT_OPTIC_TAPE + RIGHT_OPTIC_FLOOR)/2;
    setMotorDirection(LEFT_MOTOR, LEFT_BACKWARD);
    while(analogRead(RIGHT_OPTIC_SENSOR) < RIGHT_OPTIC_BLACK_THRESH ||
analogRead(LEFT_OPTIC_SENSOR) < LEFT_OPTIC_BLACK_THRESH){
//analogRead(LEFT_OPTIC_SENSOR) < LEFT_OPTIC_BLACK_THRESH
      analogWrite(Motors[LEFT_MOTOR].enPin,90); //keep turning right until the
left sensor is on the tape again
    }
    analogWrite(Motors[LEFT_MOTOR].enPin,0);
    setMotorDirection(LEFT_MOTOR, LEFT_FORWARD);

    first = false;
    clearDisplay();
    setCurserLocation(0, 0);
    writeLCDString("RIGHT T:" + (String)RIGHT_OPTIC_BLACK_THRESH);
    setCurserLocation(1, 0);
    writeLCDString("LEFT T:" + (String)LEFT_OPTIC_BLACK_THRESH);
    delay(5000);
  }
  int left_optic_read = (analogRead(LEFT_OPTIC_SENSOR) > 
LEFT_OPTIC_BLACK_THRESH) ? TAPE : FLOOR;
  int right_optic_read = (analogRead(RIGHT_OPTIC_SENSOR) > 
RIGHT_OPTIC_BLACK_THRESH) ? TAPE : FLOOR;

  if(left_optic_read == TAPE && right_optic_read == TAPE){
    clearDisplay();
    writeLCDString("STRAIGHT");
    stop_counter = 0;
    analogWrite(Motors[LEFT_MOTOR].enPin,line_speed);
    analogWrite(Motors[RIGHT_MOTOR].enPin,line_speed);
  }
  else if(left_optic_read == FLOOR && right_optic_read == TAPE){
    clearDisplay();
    writeLCDString("RIGHT");
    stop_counter = 0;
    analogWrite(Motors[RIGHT_MOTOR].enPin,0);
    analogWrite(Motors[LEFT_MOTOR].enPin,line_speed*9/10);
  }
  else if(left_optic_read == TAPE && right_optic_read == FLOOR){
    clearDisplay();
    writeLCDString("LEFT");
    stop_counter = 0;
    analogWrite(Motors[LEFT_MOTOR].enPin,0);
    analogWrite(Motors[RIGHT_MOTOR].enPin,line_speed*9/10);
  }
  else{
    stop_counter++;
    if(stop_counter > 80){
      clearDisplay();
      writeLCDString("STOP");
      analogWrite(Motors[RIGHT_MOTOR].enPin,0);
      analogWrite(Motors[LEFT_MOTOR].enPin,0);
    }
    delay(10);
  }
}

// buzzes the said pin for the given period in miliseconds
void buzz(int pin, int miliseconds)
{
  for (int i = 0; i < miliseconds; i += 2)
  {
    digitalShiftWrite(pin, LOW);
    delay(1);
    digitalShiftWrite(pin, HIGH);
    delay(1);
  }
  digitalShiftWrite(pin, LOW);
}

void readBothSides()
{
  myservo.write(90);
  rotate90Deg(0);
  delay(500);
  leftDistance = readDistance(20);
  clearDisplay();
  setCurserLocation(0, 0);
  writeLCDString(String(leftDistance, 2));
  delay(100);
  rotate90Deg(1);
  delay(800);
  rightDistance = readDistance(20);
  writeLCDString(String(rightDistance, 2));
  delay(100);
  myservo.write(90);
  delay(800);
}

void rotate90Deg(int directionM)
{
  int des = (directionM == 0) ? 180 : 0;
  myservo.write(des);
  delay(2000);
}

/*******************************************************************************
*******************************************************************************
**************************************
   SHIFT REGISTER FUNCTIONS

*******************************************************************************
*******************************************************************************
*************************************/
//set all pins on shift register to 0
void clearBits()
{
  digitalWrite(LATCH_PIN, LOW);
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, 0x00);
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, 0x00);
  digitalWrite(LATCH_PIN, HIGH);
}
```

```
// read pin from pin needs to be from 12-15 otherwise code will be compromised
int digitalShiftRead(int pin)
{
  unsigned long temp = bitPattern;
  temp = temp & 0x00ff;
  temp = temp | (1 << pin);

  // set the LATCH_PIN to low potential, before sending data
  digitalWrite(LATCH_PIN, LOW);

  // the original data (temp part of bitpattern)
  // assuming that this bit is the only bit set to 1 among all bits used as
inputs
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, ((temp & 0xff00) >> 8)); //send
upper 8 bits
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, (temp & 0x00ff));        //send
lower 8 bits

  // set the LATCH_PIN to high potential, after sending data
  digitalWrite(LATCH_PIN, HIGH);

  delay(10);                                  // delay for debouncing and for
the shift register to update
  int result = digitalRead(COMMON_INPUT_PIN); //read common input if high the
input pin is high

  // restore the old bit pattern need to do so to keep the interrupt on level
sensors
  digitalWrite(LATCH_PIN, LOW);
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, ((bitPattern & 0xff00) >> 8));
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, (bitPattern & 0x00ff));
  digitalWrite(LATCH_PIN, HIGH);

  delay(5); //wait for shift register to update

  return result;
}

// display a number on the digital segment display
void digitalShiftWrite(int pin, int value)
{

  // it would be something like 00010000 or 11101111
  unsigned long bits = (value == HIGH) ? (1 << pin) : ~(1 << pin) | 0xfc00;
//long doesnt sign extend
```

```
  // update bit pattern
  // OR makes that bit 1 and keeps other bits; AND makes that bit 0 and keeps
other bits
  bitPattern = (value == HIGH) ? (bits | bitPattern) : (bits & bitPattern);

  // set the LATCH_PIN to low potential, before sending data
  digitalWrite(LATCH_PIN, LOW);

  // the original data (bit pattern)
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, ((bitPattern & 0xff00) >> 8));
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, (bitPattern & 0x00ff));

  // set the LATCH_PIN to high potential, after sending data
  digitalWrite(LATCH_PIN, HIGH);
}

/*****************************************************************************
******************************************************************************
**************************************
   LIBRARY FUNCTIONS FOR THE LCD

******************************************************************************
******************************************************************************
*************************************/

const int LCDPins[] = {D4, D5, D6, D7};

//got help from https://stackoverflow.com/questions/36274902
void displayNumber(int num)
{
  char number[sizeof(int) * 4 + 1]; // plus 1 for \0
  sprintf(number, "%d", num);
  writeLCDString(number);
}

void shiftCurser(bool right)
{                              //true to right shift false for left shift
  digitalShiftWrite(RS, LOW); // instructions
  if (right)
  {
    writeInstruction(0x14);
  }
  else
  {
    writeInstruction(0x10);
  }
}
```

```
void shiftDisplay(bool right)
{                              //true to right shift false for left shift
  digitalShiftWrite(RS, LOW); // instructions
  if (right)
  {
    writeInstruction(0x1C);
  }
  else
  {
    writeInstruction(0x18);
  }
}

void curserHome()
{                              //Returns the cursor to the home position
(Address 0). Returns display to its original state if it was shifted.
  digitalShiftWrite(RS, LOW); // instructions
  writeInstruction(0x02);
  delay(2);
}

void setCurserLocation(int row, int col)
{
  int address = (row == 1) ? 0x40 : 0x00;
  address += col;              //the address value is this
  address += 0x80;             //but D7 needs to be high so 1AAA AAAA
  digitalShiftWrite(RS, LOW); // instructions
  writeInstruction(address);
}

void blinkCurser(bool doesBlink)
{
  digitalShiftWrite(RS, LOW); // instructions
  if (doesBlink)
    writeInstruction(0x0F);
  else
    writeInstruction(0x0E);
}

void writeLCDString(String s)
{
  int i = 0;
  while (s[i] != '\0')
  {
    writeLCD(s[i]);
    i++;
  }
}

void writeLCD(char character)
{
  digitalShiftWrite(RS, HIGH); // there are all data
  int value = (int)character;
  writeInstruction(value);
}

void setupLCD()
{
  digitalShiftWrite(RS, LOW); // there are all instructions

  delay(40); //wait for Vcc to rice to 5v

  writeLCDPins(0x3); //FUNCTION SET 8bits

  delay(5); //delay >4.1ms

  writeLCDPins(0x3);      //FUNCTION SET 8bits
  delayMicroseconds(105); //delay >100us

  writeLCDPins(0x3); //FUNCTION SET 8bits

  writeLCDPins(0x2); //FUNCTION SET 4bits

  writeInstruction(0x28); // N=HIGH as we have 2 lines and F=0 font is 5*8

  writeInstruction(0x08); // DisplayOFF

  writeInstruction(0x01); // DisplayClear
  delay(2);

  writeInstruction(0x06); // Increment counter + no Shift
}

void clearDisplay()
{
  digitalShiftWrite(RS, LOW);
  writeInstruction(0x01);
  delay(2);
}
void writeInstruction(int value)
{
  writeLCDPins((value & 0xF0) >> 4); //upper 4 bits;
  writeLCDPins(value & 0x0F);        //lower 4 bits
  delayMicroseconds(40);
}
void writeLCDPins(int val)
{
  for (int i = 0; i < 4; i++)
    digitalShiftWrite(LCDPins[i], (val & (1 << i)) == (1 << i));
  pulseE();
}

void pulseE(void)
{
  digitalShiftWrite(E, LOW);
  delayMicroseconds(1); //wait 150ns
  digitalShiftWrite(E, HIGH);
  delayMicroseconds(1); //wait 150ns
  digitalShiftWrite(E, LOW);
  delayMicroseconds(10); //wait 150ns
}

float readDistance(int num)
{
  // datasheet:
https://www.robotshop.com/media/files/pdf2/hc-sr04-ultrasonic-range-finder-dat
asheet.pdf
  float speedOfSound;
  int numOfMeasurements = num;
  float usPerMeter;
  float temp;
  float distance;

  float sum = 0;
  for (int i = 0; i < numOfMeasurements; i++)
  {
    //calculate temp from LM35 reading
    temp = analogRead(LM35) * 500.0 / 1024; //read LM35 and use constant from
https://create.arduino.cc/projecthub/TheGadgetBoy/making-lcd-thermometer-with-
arduino-and-lm35-36-c058f0
                                                  //  float temp = 23;
    //determine constants for calculating distance
    speedOfSound = 331.5 + (0.6 * temp);
    usPerMeter = 10000.0 / speedOfSound;

    // Send >10us pulse to TRIG to enable URF
    digitalWrite(TRIG, LOW);
    delayMicroseconds(2); // make sure trig is actually low
    digitalWrite(TRIG, HIGH);
    delayMicroseconds(12);
    digitalWrite(TRIG, LOW);

    //calculate distance from pulse width in us returned by pulseIn()
    distance = pulseIn(ECHO, HIGH) / (2 * usPerMeter);

    if (distance > 400)
      distance = 400;
    else if (distance < 5)
      distance = 5;

    sum+= distance;

    if(numOfMeasurements != 1)
      delay(60); // cycle period recommended by the datasheet
  }

  distance = sum / numOfMeasurements;
  if (distance > 400)
    distance = 400.0;
  else if (distance < 5)
    distance = 5.0;

  return distance;
}

/*****************************************************************************
******************************************************************************
***************************
   ADDITIONAL FUNCTION

******************************************************************************
******************************************************************************
**************************/
void drawWord(char *word){
  int i = 0;
  if(strlen(word) > 1){
    for(int i = 0; i < strlen(word); i++){
      drawWord(word[i]);
    }
    return;
  }

  switch(word[i]){

    case 'a':
    case 'A':
//      //int A_hypotenuse = font_size/cos(15);
//      rotate(345);
//      lowerPen();
//      drawLine(A_hypotenuse, true, 1/2);
//      rotate(105);
//      lowerPen();
//      drawLine(A_hypotenuse*cos(75), false, 0.0);
//      rotate(105);
//      lowerPen();
//      drawLine(A_hypotenuse/2, true, 1.0);
//      lowerPen();
//      drawLine(A_hypotenuse/2, false, 0.0);
//      rotate(75);
//      drawLine(space, false, 0.0);
//      rotate(90);
    return;

    case 'b':
    case 'B':
    return;

    case 'c':
    case 'C':
    return;

    case 'd':
    case 'D':
    return;

    case 'e':
    case 'E':
      lowerPen();
      drawLine(font_size, false, 0.0);
      rotate(270);
      lowerPen();
      drawLine(font_size/2, true, 1.0);
      rotate(90);
      drawLine(font_size/2, false, 0.0);
      rotate(90);
      lowerPen();
      drawLine(font_size/3, true, 1.0);
      rotate(90);
      drawLine(font_size/2, false, 0.0);
      rotate(90);
      drawLine(font_size/2, false, 0.0);
      liftPen();
      drawLine(space, false, 0.0);
      rotate(90);
    return;

    case 'f':
    case 'F':
      lowerPen();
      drawLine(font_size, false, 0.0);
      rotate(270);
      lowerPen();
      drawLine(font_size/2, true, 1.0);
      rotate(90);
      drawLine(font_size/2, false, 0.0);
      rotate(90);
      lowerPen();
      drawLine(font_size/3, true, 1.0);
      rotate(90);
      drawLine(font_size/2, false, 0.0);
      rotate(90);
      drawLine(space + font_size/2, false, 0.0);
      rotate(90);
    return;

    case 'g':
    case 'G'://////////////////////////////////////////////////////////
    return;

    case 'h':
    case 'H':
      lowerPen();
      drawLine(font_size, true, 1/2);
      rotate(90);
      lowerPen();
      drawLine(font_size/2, false, 0.0);
      rotate(90);
      lowerPen();
      drawLine(font_size/2, true, 1.0);
      lowerPen();
      drawLine(font_size/2, false, 0.0);
      rotate(90);
      drawLine(space, false, 0.0);
      rotate(90);
    return;

    case '1':
    case 'i':
    case 'I':
      lowerPen();
      drawLine(font_size, true, 1.0);
```

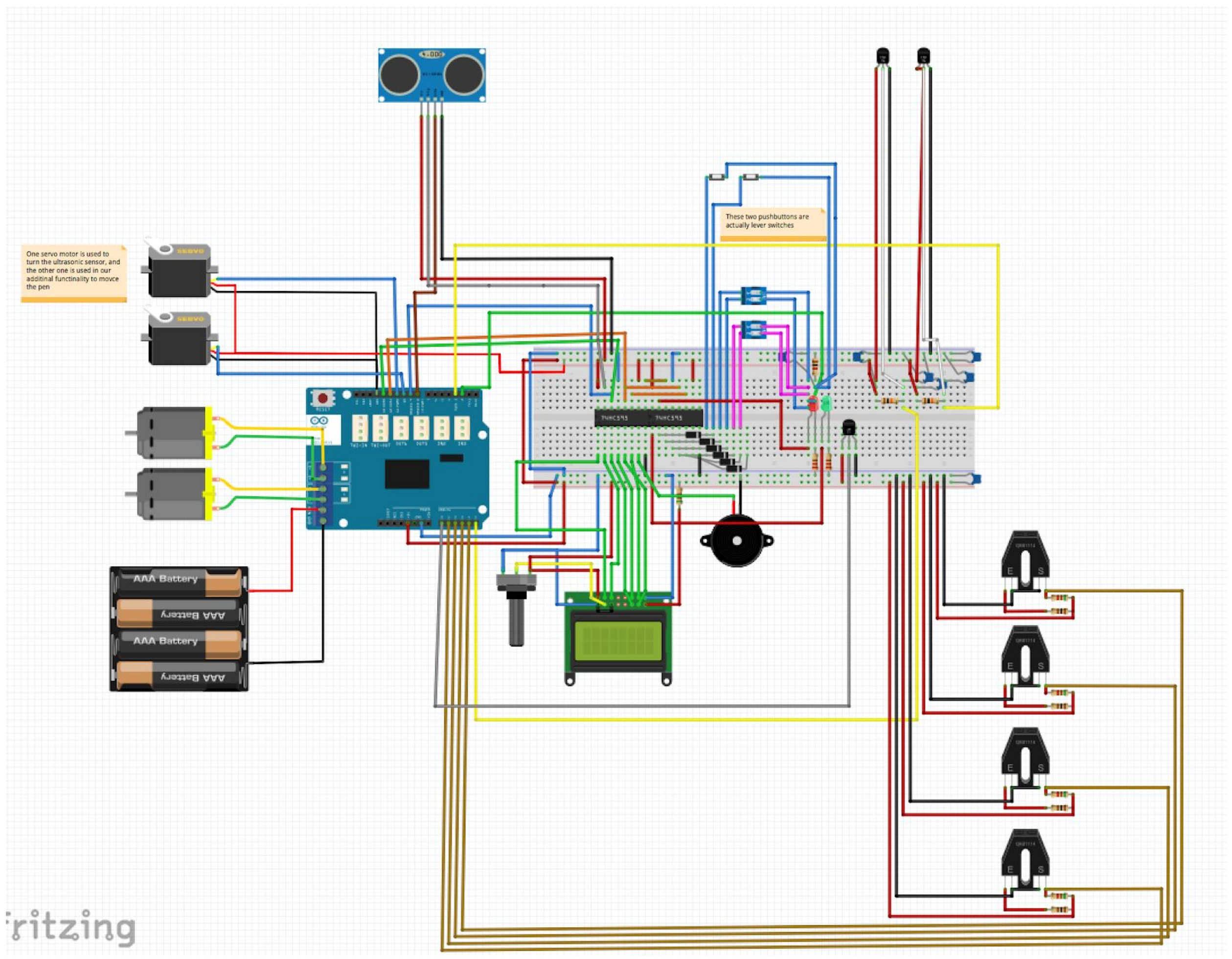
```
        rotate(-90);
        drawLine(space, false, 0.0);
        rotate(90);
      return;

      case 'j':
      case 'J':
      return;

      case 'k':
      case 'K':
        lowerPen();
        drawLine(font_size, true, 1/2);
        rotate(335);
        lowerPen();
        drawLine(font_size/(2*cos(25)), true, 1.0);
        rotate(50);
        lowerPen();
        drawLine(font_size/(2*cos(25)), false, 0.0);
        rotate(65);
        drawLine(space, false, 0.0);
        rotate(90);
      return;

      case 'l':
      case 'L':
        lowerPen();
        drawLine(font_size, true, 1.0);
        rotate(-90);
        lowerPen();
        drawLine(font_size/2, false, 0.0);
        liftPen();
        drawLine(space, false, 0.0);
        rotate(90);
      return;

      case 'm':
      case 'M':
        lowerPen();
        drawLine(font_size, false, 0.0);
        rotate(195);
        lowerPen();
        drawLine(font_size/cos(15), false, 0.0);
        rotate(150);
        lowerPen();
        drawLine(font_size/cos(15), false, 0.0);
        rotate(195);
        lowerPen();
        drawLine(font_size, false, 0.0);
        rotate(90);
        drawLine(space, false, 0.0);
        rotate(90);
      return;

      case 'n':
      case 'N':
        lowerPen();
        drawLine(font_size, false, 0.0);
        rotate(195);
        lowerPen();
        drawLine(font_size/cos(15), false, 0.0);
        rotate(165);
        lowerPen();
        drawLine(font_size, true, 1.0);
        rotate(90);
        drawLine(space, false, 0.0);
        rotate(90);
      return;

      case '0':
      case 'o':
      case 'O':
        lowerPen();
        drawLine(font_size, false, 0.0);
        rotate(-90);
        lowerPen();
        drawLine(font_size/2, false, 0.0);
        rotate(-90);
        lowerPen();
        drawLine(font_size, false, 0.0);
        rotate(-90);
        lowerPen();
        drawLine(font_size/2, true, 1.0);
        drawLine(space, false, 0.0);
        rotate(90);
      return;

      case 'p':
      case 'P':
      return;

      case 'q':
      case 'Q':
      return;
```

```
case 'r':
case 'R':
return;

case 's':
case 'S':
return;

case 't':
case 'T':
  drawLine(font_size, false, 0.0);
  rotate(-90);
  lowerPen();
  drawLine(font_size/2, false, 0.0);
  keepAligned(255, RIGHT_BACKWARD, LEFT_BACKWARD);
  align();
  rotate(-90);
  lowerPen();
  drawLine(font_size, false, 0.0);
  rotate(90);
  drawLine(space, false, 0.0);
  rotate(90);
return;

case 'u':
case 'U':////////////////////////////
return;

case 'v':
case 'V':
  drawLine(font_size, false, 0.0);
  rotate(195);
  lowerPen();
  drawLine(font_size/cos(15), false, 0.0);
  rotate(150);
  lowerPen();
  drawLine(font_size/cos(15), false, 0.0);
  rotate(195);
  drawLine(font_size, false, 0.0);
  rotate(90);
  drawLine(space, false, 0.0);
  rotate(90);
return;

case 'w':
case 'W':
        drawLine(font_size, false, 0.0);
        rotate(195);
        lowerPen();
        drawLine(font_size/cos(15), false, 0.0);
        rotate(150);
        lowerPen();
        drawLine(font_size/(3*cos(15)),false, 0.0);
        rotate(210);
        lowerPen();
        drawLine(font_size/(3*cos(15)),false, 0.0);
        rotate(150);
        lowerPen();
        drawLine(font_size/cos(15), false, 0.0);
        rotate(195);
        drawLine(font_size, false, 0.0);
        rotate(90);
        drawLine(space, false, 0.0);
        rotate(90);
      return;

      case 'x':
      case 'X':
        rotate(340);
        lowerPen();
        drawLine(font_size/cos(20), false, 0.0);
        rotate(110);
        drawLine((cos(70)*font_size)/cos(20), false, 0.0);
        rotate(110);
        lowerPen();
        drawLine(font_size/cos(20), false, 0.0);
        rotate(70);
        drawLine(space, false, 0.0);
        rotate(90);
      return;

      case 'y':
      case 'Y':
      return;

      case 'z':
      case 'Z':
//        drawLine(font_size, false, 0.0);
//        rotate(270);
//        drawLine,(font_size/2, false, 0.0);
//        rotate(200);
//        drawLine(font_size/(2*cos(20)));
      return;
```

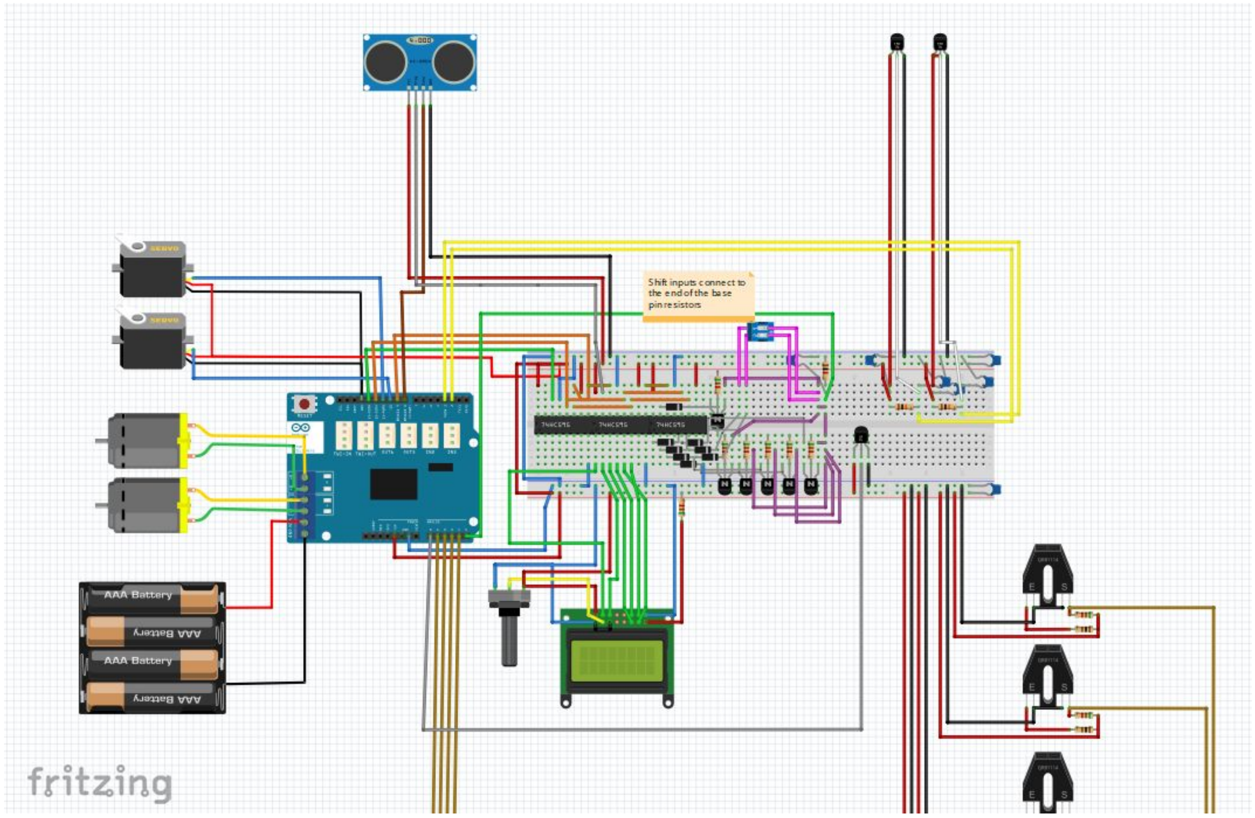
```
    }
  }
    void rotate(int degrees){
      int num = degrees/90;
      if(!pen_up)
        liftPen();

      //rotate "degrees" degrees counter-clockwise
      if(num >= 0){
        for(int i = 1; i <= num; i++){
          keepAligned(160, RIGHT_FORWARD, LEFT_BACKWARD);
          align();
        }
      }
      else {
        num = -num;
        for(int i = 1; i <= num; i++){
          keepAligned(160, RIGHT_BACKWARD, LEFT_FORWARD);
          align();
        }
      }
      delay(500);
    }

    //length is how long of a line to draw
    //backtrack is whether or not the robot will return to a point along the
  line it just drew
    //fraction is how far along the line the robot just drew to backtrack, 0.0
  <= fraction <= 1.0
    void drawLine(int length, bool backtrack, float fraction){
        for(int i = 1; i <= length; i++){
          keepAligned(255, RIGHT_FORWARD, LEFT_FORWARD);
          align();
        }

      prev_length = length;
      delay(500);

      if(backtrack){
        if(!pen_up)
          liftPen();
        for(int i = 1; i <= length; i++){
          keepAligned(255, RIGHT_BACKWARD, LEFT_BACKWARD);
          align();
        }
  //        rotate(180);
```

```
//      drawLine(prev_length*fraction, false, 0.0);
    }
  }
//
//  void backtrack(float fraction){
//      if(!pen_up)
//        liftPen();
//      rotate(180);
//      drawLine(prev_length*fraction, false, 0.0);
//  }

  void liftPen(){
    pen_servo.write(20);
    pen_up = true;
  }

  void lowerPen(){
    pen_servo.write(0);
    pen_up = false;
  }
```

Appendix C - Fritzing


One servo motor is used to turn the ultrasonic sensor, and the other one is used in our additinal functinality to movce the pen
These two pushbuttons are actually lever switches
AAA Battery
AAA Battery
AAA Battery
AAA Battery
74HC595
74HC595
fritzing

Initial Fritzing Design



Appendix D - GitHub

Every group member has reasonably and equally contributed to the Project 1 github repository.

Appendix E – Complete Component list

HC-SR04: Ultrasonic Range Finder Sensor Module (Quantity: 1)

GREEN LED (Quantity: 1)

RED LED (Quantity: 1)

2WD Mobile Platform (Quantity: 1)

Hall effect sensor (Quantity: 2)

Perma-Proto Half-sized Breadboard PCB (Quantity: 1)

IC 8-BIT SHIFT REGISTER 16-DIP (Quantity: 2)

Piezo Buzzer - PS1240 (Quantity: 1)

CAP CER 0.1UF 50V 10% RADIAL (Quantity: 7)

1N4148-TP RECTIFIER SILICON .15A 75V DO-35 (Quantity: 6)

Standard LCD 16x2 + 10K POT + header (Quantity: 1)

 Micro servo motor (Quantity: 2)

1k ohm Resistor (Quantity: 3)

220 ohm Resistor (Quantity: 2)

10k ohm Resistor (Quantity: 2)

100 ohm Resistor (Quantity: 2)

Reflective Optical Sensor (TCRT5000L) (Quantity: 2)

2A Motor Shield for Arduino (Quantity: 1)

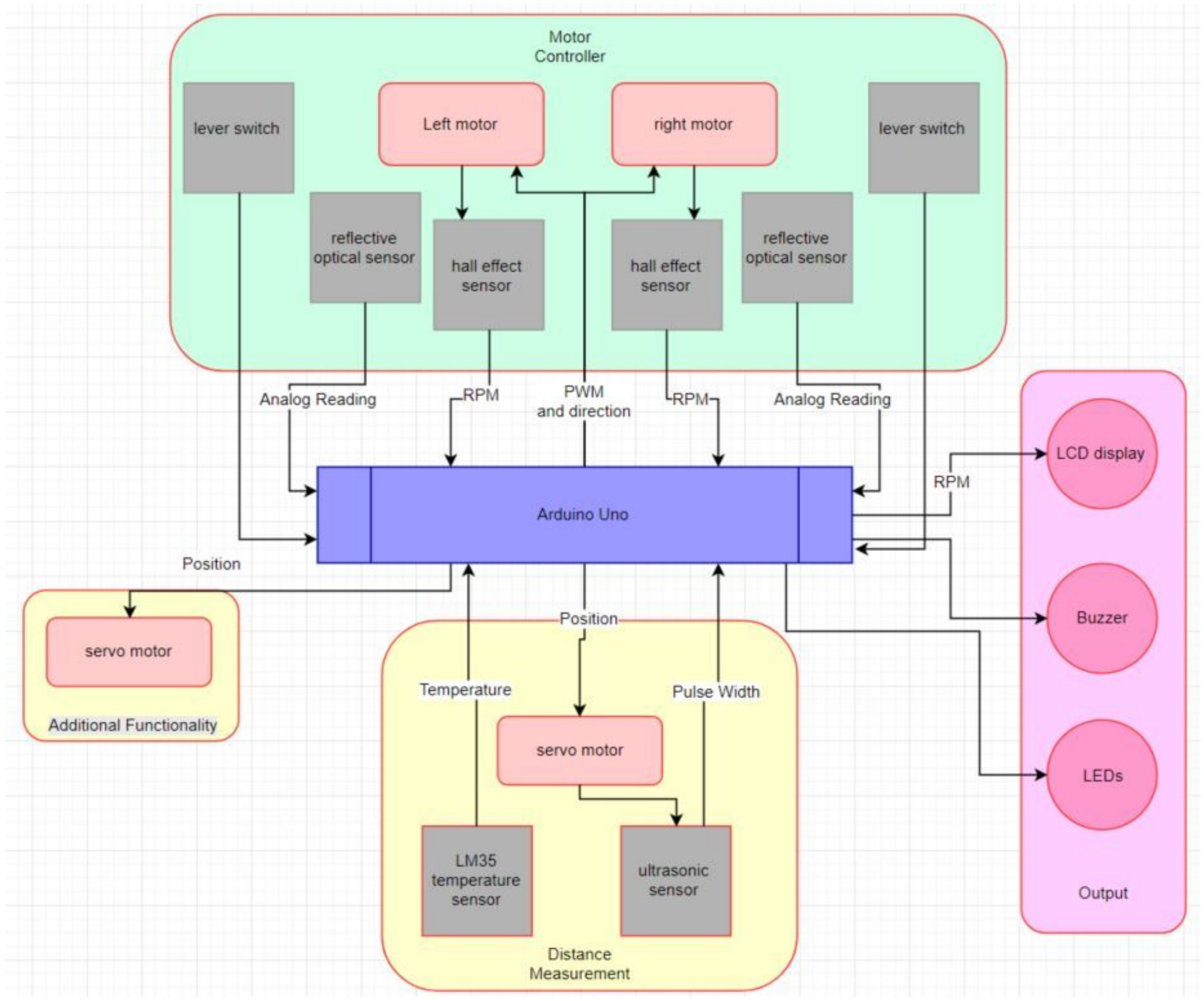
ALCO Dip Toggle Switch (DIP, 4 switches) (Quantity: 1)

Arduino UNO Rev3 Board (Quantity: 1)

Micro Switch w/Lever (Quantity: 2)

(Earlier design) 2N3904 NPN transistor (Quantity: 6)

Appendix F - System Block Diagram


Motor Controller
lever switch
Left motor
right motor
lever switch
reflective optical sensor
hall effect sensor
hall effect sensor
reflective optical sensor
Analog Reading
RPM
PWM and direction
RPM
Analog Reading
Arduino Uno
RPM
LCD display
Buzzer
LEDs
Output
Position
servo motor
Additional Functionality
Position
Temperature
Pulse Width
servo motor
LM35 temperature sensor
ultrasonic sensor
Distance Measurement