ELEC 291 Section 20C

Project 1

Project #: 1	Lab Section: L2D	Group #: G18; Benches 4C/5C
Kenton Ma	32689151	16.66 %
Linda Munisi	22535158	16.66 %
Luvian Wang	16939134	16.66 %
Sirine Trigui	46443157	16.66 %
Yixin Zhao	49427140	16.66 %
Xi Chu	17353137	16.66 %

***Contribution Summary*:**

- **Kenton Ma**: Documented code relevant to the first principle function. Drafted speed control logic. Soldered the motor and battery switch.
- **Linda Munisi**: Worked on the additional functionality, mini servo, and Fritzing schematic.
- **Luvian Wang**: Programmed logic to allow the robot to turn left or right. Drafted logic for complete functionality (i.e. outlined how to combine all functionalities to work together).
- **Sirine Trigui**: Contributed in robot assembly and worked on optical sensors and mini servo for the second functionality.
- **Yixin Zhao**: Worked on Fritzing, mini servo and the logic to allow the robot to follow a line based on optical sensor data.
- **Xi Chu**: Programmed speed calibration to allow robot move in a straight manner. Coded the logic for additional function.

Introduction and Motivations

The objectives of this project is to implement a multi-functional robot. In this project, there are two principle functions. Both involve the robot react to the environment autonomously. Another objective is to learn the fundamental aspects of robotics using motors. Additionally, we are to use and advance our knowledge on Arduino, sensors, and physical computing. We are to use the provided robot kit to implement the principle functions. Along with the principle functions, we are to implement a third function. Our third function allows us to control the movement of the robot via infrared signals from a remote. Unlike other labs, we will be using an external power

supply to power our robot. Consequently, we must consider power consumption of all of our components.

PROJECT DESCRIPTION

First Functionality:

Here, we implemented several functions to allow the robot to operate autonomously when switched on. The robot must move at its maximum speed in a straight manner (until it detects an object in front of it), then gradually slow down once it gets within a certain distance range. After slowing down to a complete stop, the robot must then scan left and right and choose a direction that has the best/longest free space for movement. Next, it will turn 90 degrees according to the environment and continue moving until it detects another object within the set range.

Components Used and Assembling:

The components that we used to complete this functionality are as follows: Arduino Uno, 2A Motor Shield, a ultrasonic range finder (HC-SR04), a temperature sensor (LM35), two Hall Effect sensor (model US1881), a servo motor, five 1.5V AA batteries, two magnets, an LCD display, two DC Geared Motors, and the 2WD Mobile Platform.

First, we assembled the mobile platform using the provided manual that came with the package (Turtle 2WD Mobile Platform). Next, we drafted the main sequence of instructions for the first principle function. There are three main sub-functions that will be occurring in parallel. The first involves calibrating the speed to allow the robot to move as fast as possible in a straight manner. The second sub-function involves detecting objects with the HC-SR04. The third being direction detection using the servo motor in conjunction with the HC-SR04 and wheels in turning to desired direction. After having gone through these sub-functions in sequence, it will loop back and repeat.

Coding and Testing:

First we had the robot moving forward but it was not on a straight path. We wrote that uses the ultrasonic sensors first before fixing the straight line. In the code, we used the temperature sensor to get the surrounding temperature and then use the value obtained to calculate the distance measured by the ultrasonic sensor. For the detecting range, the robot slows down when it detects an object within a range of 50cm, and then when it gets 10cm form the object, it stops and measures the left and right distance. It then turns to the side with the larger distance.

After that, we proceeded on fixing the robot to go straight. While working on the speed calibration step, we found many challenges. There were many factors to consider such as the weight distribution of the robot, the physical variations in the wheels, and the amount of gain per duty cycle in each DC motor. We started this function by making sure that each major component works as it should. For example, we created a simple test code to make sure that the DC motors are operating properly. We created and re-used past programs to aid us in this step. Doing this will help us debug later in the future (when the robot is completely assembled).

After making sure that the two DC motors, HC-SR04, LM35, Hall Effect sensor, and servo motor are working independently, we started to implement the drafted instruction sequence mentioned above.

Our first attempt in implementing the speed calibration involved using the Hall Effect sensor and two magnets in conjunction with the two DC motors. At this point in the project, we are only concerned with getting the robot to go as straight as possible while going at max speed. We extended the Hall Effect's pins by soldering on wires, so that we can hang them next to a magnet that will be attached to the wheel. Alternatively, we could have connected them to small breadboards and hang them next to each wheel. We taking a step by step approach as to avoid bugs. The general structure of our implementation consisted of calculating the speed of each wheel by using the Hall Effect sensor data. We used Arduino's interrupt functions to initiate an interrupt service routine (ISR) whenever the south pole side of the magnets is in front of the Hall Effect sensor (see Fig. 1.0). Note that all variables associated with the ISR are declared volatile. When taking the speed measurements, we ignore the first measurement because we do not have a base for our time yet. Doing this will allow us to correctly get the initial speeds of each wheel. The ISR keeps track of the time it takes for one revolution.

```
// ISR that determines speed of right wheel
void wheelSpeedR()
{
  // ignore first reading to properly calculate speed
  if (ignoreFirstR < 1) {
    lastTimeR = millis();
    ignoreFirstR++;
    return;
  }

  currTimeR = millis();
  float timeDiff = (currTimeR - lastTimeR) / 1000.0;

  if (timeDiff < MIN_TIME_DIFF) {
    lastTimeR = currTimeR;
    return;
  }

  lastTimeR = currTimeR;
  currSpeedR = PI * WHEEL_DIAM / timeDiff;
```

Figure 1.0: Calculating Speed of Right Wheel

Additionally, it calculates the speed in cm/s by using the following formula:

$$Speed\,(cm/s) = (PI * wheel\ diameter) / time\ taken\ for\ one\ revolution$$

After calculating the speed for both wheels, we will compare them and calibrate according to which wheel is faster. The calibration is done by decreasing an adjustment variable by 1 and adding that to the current Pulse Width Modulation (PWM). In other words, the analog speed of the faster wheel will start decreasing (see Fig 1.1).

```
digitalWrite(M1, HIGH);
digitalWrite(M2, HIGH);
analogWrite(E1, MAX_SPEED + adjustR); // adjustR will be negative if need adjustments
analogWrite(E2, MAX_SPEED + adjustL); // adjustL will be negative if need adjustments
```

Figure 1.1: PWM and Speed Calibration.

Immediately, we found problems as the robot veered off to the left at the beginning of our tests. We suspected weight distribution to be a major factor as well as physical characteristics of the wheel because one wheel looked slightly deformed compared to the other wheel. As starting with a high speed did not seem to work. We tried a different approach. This time, we calibrate the wheels starting with a lower speed.

Starting with lower speeds, we noticed that the minimum analog speed required to get the robot to move was around 130. Instead of decrementing the adjustment variable, we tried incrementing it. An issue we found was that it was slow to adjust because we are only changing the PWM value by 1 each time. We tried doing increments of 5. It worked better, but the weight distribution seemed to be a big factor in this implementation also. Other factors to keep in mind is that PWM can go to a max of 255 and that our increment adjustments should not be too high. Incrementing the adjust variable by a higher factor may cause an overshoot in speed correction, thus the robot will veer to one side.

We switched to a different approach. This time, we start from a relatively high base speed, which we set as 220 (PWM value). This allows us room to adjust the slower wheel up to a value of 255. We perform the speed adjustments based on the speed data that we get from the Hall Effect sensors. If the left/right wheel is faster, then we speed up the right/left wheel by incrementing the corresponding adjustment variable. We have chosen a tolerance of 0.5 as the wheels need not be exact due to other environmental factors such as friction of floor and

traction of each wheel. The following function is called during each interrupt (see Fig. 1.2).

```
void calcSpeedAdjust() {
  if (abs(currSpeedL - currSpeedR) > 0.5) {
    if (currSpeedR > currSpeedL && (((analogSpeedOfRobot + adjustL+1) * 115)
      adjustL += 1;
    } else if (currSpeedL > currSpeedR && (((analogSpeedOfRobot + adjustL+1)
      adjustR += 1;
    }
  } else {
    //Serial.println("REACHED SAME SPEED");
    return;
  }
}
```

Figure 1.2: Speed Adjustment Algorithm

In conjunction with adjusting the adjustment variable, we also scaled the output signals. The reason being is because during our tests, one speed was faster than the other at max PWM. We took this ratio and found that the left wheel is 15% slower than the right wheel, so we scale the PWM speed by 15% for the left wheel. This is if the robot is at a distance range of >= 50 cm from an object. If the robot is <= 50 cm from an object the robot will start slowing down. When it slows down the wheels start to get disproportional again. To counteract this unwanted behaviour, we scale the right wheel down by a factor of 25% to keep the robot going in a straight motion (see Fig. 1.3).

```
if (distance > 50) {
  digitalWrite(M1, HIGH);
  analogWrite(E1, analogSpeedOfRobot + adjustR);
  digitalWrite(M2, HIGH);
  analogWrite(E2, ((analogSpeedOfRobot + adjustL) * 115) / 100);
}
else if (distance <= 50) {
  digitalWrite(M1, HIGH);
  analogWrite(E1, (analogSpeedOfRobot + adjustR) * 0.75);
  digitalWrite(M2, HIGH);
  analogWrite(E2, analogSpeedOfRobot + adjustL);
}
```

Figure 1.3: Motor Scaling

Alternatively, we should have used a basic proportional-integral-derivative (PID) controller type algorithm to facilitate the robot's movements. This would allow us to have more accurate adjustments to allow the robot to move at a faster speed while going as straight as possible.

Now that we have a working algorithm to facilitate a straight moving robot, we started to design and code the logic to enable the robot to gradually slow down and come to a complete stop when encountering an object in front of it. Here, we will incorporate the HC-SR04, LM35, LCD

and Servo motor. Using the HC-SR04, we will be able to calculate the distance of an object relative to the robot. In this function, the LCD will be used to display distance of an object (if any) relative to the robot as well as the speed of the robot in cm/s. The LM35 will be used to increase the accuracy in our distance calculation. The Servo motor will be used to allow the ultrasonic sensor to scan left and right once the robot has come to a complete stop. The logic that will be looped goes as follows: calculate distance, check if distance is within a set threshold and make any adjustments to speed based on environmental input, scan left and right if robot has reached stopping distance and turn 90 degrees towards side with more space.

To calculate distance, we used the code that we had previously coded in a different lab as our base. To make the code more modular, we made the distance calculation into a function called *calcDist()*.

After calculating the distance, we will compare that to our threshold. The robot should start slowing down once it has detected a distance value of less than 50 cm and stop completely once it reaches a distance of less than or equal to 10 cm. This is done through using the built-in Arduino functions *map()* and *constrain()* the mapping will allow a gradual speed reduction and constraint will help keep the robot going at top speed if it is detecting a distance greater than or equal to 50 cm. After coming to a complete stop, a flag will be set to enable the scan logic. The flag that keeps track of the robot's state (i.e. stop or keep going) is named *stopped*. A value of 1 signifies that the robot has stopped and a value of 0 means keep going.

Once the robot has stopped and the flag is set, we start to execute the scan logic. The scan logic we have decided to make as a separate function to increase modularity in our code (named *getDirection()*). The *getDirection()* function controls the servo in conjunction with the *calcDist()* function to retrieve the left distance and right distance. Here, we used the servo library to enable us to control the servo. After retrieving the left and right distances, we make a comparison and determine whether to turn left (value of 1) or right (value of 2). This value is returned. This information is used to enable either the *turnLeft()* or *turnRight()* based on the return value of *getDirection()*. The left and right turn functions are implemented by performing a PWM on the respective pins of each wheel and delayed until a 90 degree is reached. This 90 degree turn is based on the delay. This delay was determined through repeated testing. An alternative method should be used as this is not the best solution. Perhaps using a sensor that can determine the robot's orientation will enable us to do this (e.g. a compass). After doing so, we reset the *stopped* flag and loop back again.

We used the LCD library to enable us to print information regarding the distance that it calculates and speed of the robot. The printed speed is the speed of the slower wheel. We could have chosen to display the speed of the right wheel, but since the speed of the two wheels should be relatively the same, we arbitrarily chose to display the speed of the left wheel. This information is stored variable *speedOfRobot*. The distance will be printed along-side the speed and is stored in the variable *distance*. The units are also shown on the display. The LCD

will also notify the user when it has stopped and started to determine the next direction to take. When it is turning left or right, it will display “Turning left/right”.

Second Functionality:

In this functionality, we implemented the functions to allow the robot follows a darker line comparing to the ground.

Components and Assembly:

The components that we used to complete this functionality are as follows: Arduino, 2A Motor Shield, three optical sensors (TCRT5000), five 1.5V AA batteries, an LCD display, two DC Geared Motors, and the 2WD Mobile Platform.

When switching the robot on, in addition to the components mentioned above, we had to add reflective optical sensors (TCRT5000) that allows the robot to guide itself along a path which is much darker than the background.

We started with connecting the circuit using one optical sensor. Since the optical sensor’s datasheet does not have any information about how the device should be connected, we followed some videos to get a correct circuit. Therefore, two pins of the optical sensor are directly connected to ground while the third pin is connected to an Arduino digital pin through a resistor and the last is connected to the 5v pin through a resistor as well. It is clearer in our Fritzing.

Coding and Testing:

Facing the ground, we first printed the values that we were getting from the analog pin on the serial monitor. We used white and black colors to get an idea about the corresponding values for different colors. The values were between 0 and 1023. Starting from 40, the sensor detects the black color. Therefore, whenever the analog pin is reading an integer between 0-40 we consider it is detecting white whereas it is reading 40-1023 it is detecting dark colors (in this case black). Since the black path does include some curves, we had to add another sensor to be able to detect a curve.

At the beginning, we put two sensors far apart from each other on the breadboard. The logic used here is to get the difference between the readings of the two sensors. When the difference is greater than 100, we compare two values from the left and right sensor, then determining which one has the greater value; the robot turns to the direction that sensor stands for. For example, if the difference is greater than 100, and the left sensor has a greater value than the right one, we consider the left one is on a darker color and the robot needs to follow it. So the robot will turn left for 300 milliseconds, and then it stops. On the other hand, if the difference is smaller than 100, we consider it as an error and ignore it; the robot keeps following the current direction. In this situation, we had default delay in turning function which made the turning neither efficient nor accurate. Also, it went off the line sometimes unless we set the speed very low.

```
digitalWrite(M1, HIGH);
digitalWrite(M2, HIGH);
// enable only the right wheel
analogWrite(E1, 255);
analogWrite(E2, 0);
delay(200);
// disable both wheels
analogWrite(E1, 0);
analogWrite(E2, 0);
```

In order to solve those problems, we added another third sensor. The logic has two main conditions: the middle sensor is on black tape or not. In the first condition, we check the value of the other two sensors. If they are different, the robot goes to the direction of the sensor that has the higher value; if all of them are on black, the robot goes straight but with a small value; or, the left and the right ones are on white tape, the robot moves with a higher speed. Under the other condition, the logic is the same when the values of left and the right sensors are different. Otherwise, when all three are on the white tape, the robot goes straight with a higher speed, if the left and right ones are on black then moves with a lower speed.

After several testing, we believed our logic is good. Since we didn't want it to turn 90 degrees, we used *go()* to turn in a specific turning speed. Also, we have different speed when all are on black or the middle one is on white when the left and right sensors are on white. The reason is that we want it to go slow enough for it to find the right direction. This function required the robot to detect a line, therefore the sensors had to be facing the ground. We put the mini breadboard with the sensors connected to it on the front platform of the robot facing downwards so the optical sensors can detect the black line.

Therefore, we encoded our robot as the following:

- Whenever the middle sensor is on the black tape and the side ones on the ground, we are keeping the robot to move forward
- If in any case, the middle sensor is not facing the black line, and one of the side sensors are then the robot has to follow the direction of the sensor that is on the black tape.
- If the three sensors are not on the black path, then the robot is simply moving forward.

For this functionality, we really encountered some issues with the hardware(optical sensors). The position of those sensors makes a big difference in whether the robot is going to follow the path or not. For example, positioning three of the in the same line , the robot will definitely go out of the track.When we did our demo, we had to put them in a zigzag position.Even in this position, the robot will follow the path perfectly when it started as clockwise while it went out of the path when it was tested counterclockwise.

Additional functionality:

Our additional functionality involves using the remote to control the robot's direction, to switch between functions and also to choose a specific speed of the robot.
We are using a remote to switch between function A, function B and function C. Function C can be used to control the robot as follows:

- If the user presses the up arrow button, the robot will move forward.
- If the user presses the down arrow button, the robot will move backward.
- If the user presses the left arrow button, the robot will turn/rotate left.
- If the user presses the right arrow button, the robot will turn right.
- If the user presses the middle button, the robot will stop and promote the user to select a speed level among fast, medium and slow speed using up and down button. After user select a speed level, press power button to let the robot to move again.
- If the user presses the power button, current running function will stop and the program will reset.

Components and Assembly:
The components that we used to complete this functionality are as follows: an IR remote control, IR Receiver and the components from the previous functions.

Coding and testing:
For this function, first we connected the receiver to the circuit. Following the datasheet for the receiver, we connected one pin of the receiver to ground, the other one to 5V and the other pin is connected to a digital pin of the Arduino. Here, we had to be careful since the IR receiver can use only a specific pin that is digital pin 11 of the Arduino.

The A, B, C buttons on the remote are used to switch between functions. Function A is the same first function as above where the robot moves in a straight line and then stops when it detects an object within a certain distance range. The second function is also the same as above where the robot follows the black line by using the detection from the optical sensors attached to the robot. For function C, we can control the moving direction of the robot by using the up, down, right and left buttons on the remote. We can also choose a speed level (high, medium, slow) by using middle button.

In terms of the code, we started with downloading the library and sample code and testing it on our Arduino board. After making sure the sample code works, we added the direction control and function switch logic into this sample code. The direction control logic worked fine, but we first had trouble changing state from function C to the other functions. Then, we found that we need to add signal decoded logic in each of the three functions so that it would not stuck in process. In addition to the remote control, we add a function to select the moving speed level of the robot. After we choose function C by using the remote, we could press the middle button to start choosing the speed level. There are three speed levels: high speed, medium speed and low speed. So in function C, we implemented two states: after we press middle button, we are in speed selecting state; or after pressing power button, we go back to direction control state.

Originally, we planned to use middle button to switch back to direction control state, but we encountered problem during testing, so we used power button instead.

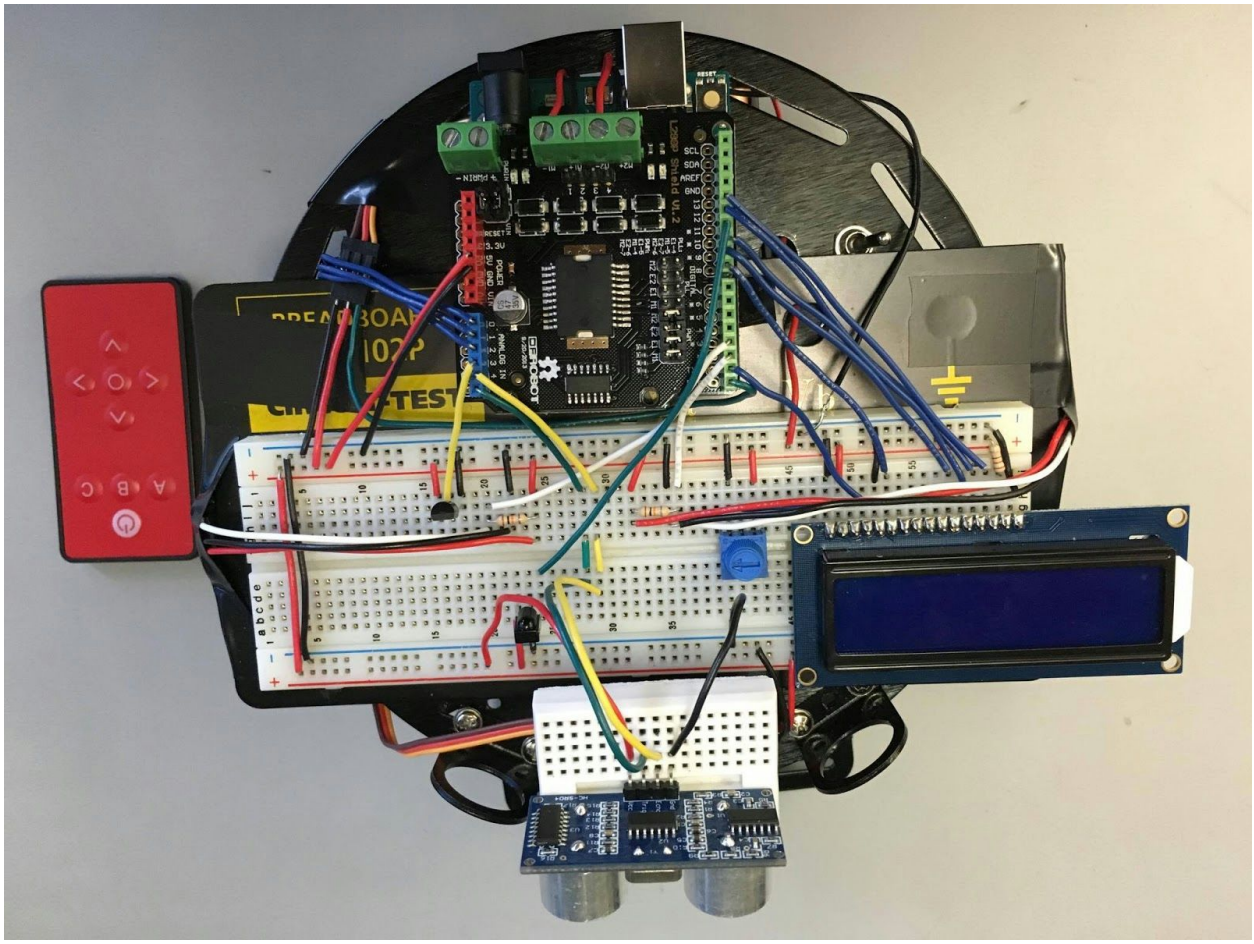
In order to make the robot go straight, we added the speed adjustment logic in function A to this part. Since the actual velocity of the two wheels was different, we applied a factor of 1.15 to the left wheel to make them move in the same speed. However, we then found that this ratio only worked for the maximum speed, and for the medium and slow speed, the robot still did not go straight, so we keep testing it and found that the ratio is 1.25 for the medium speed and 1.35 for the slow speed.

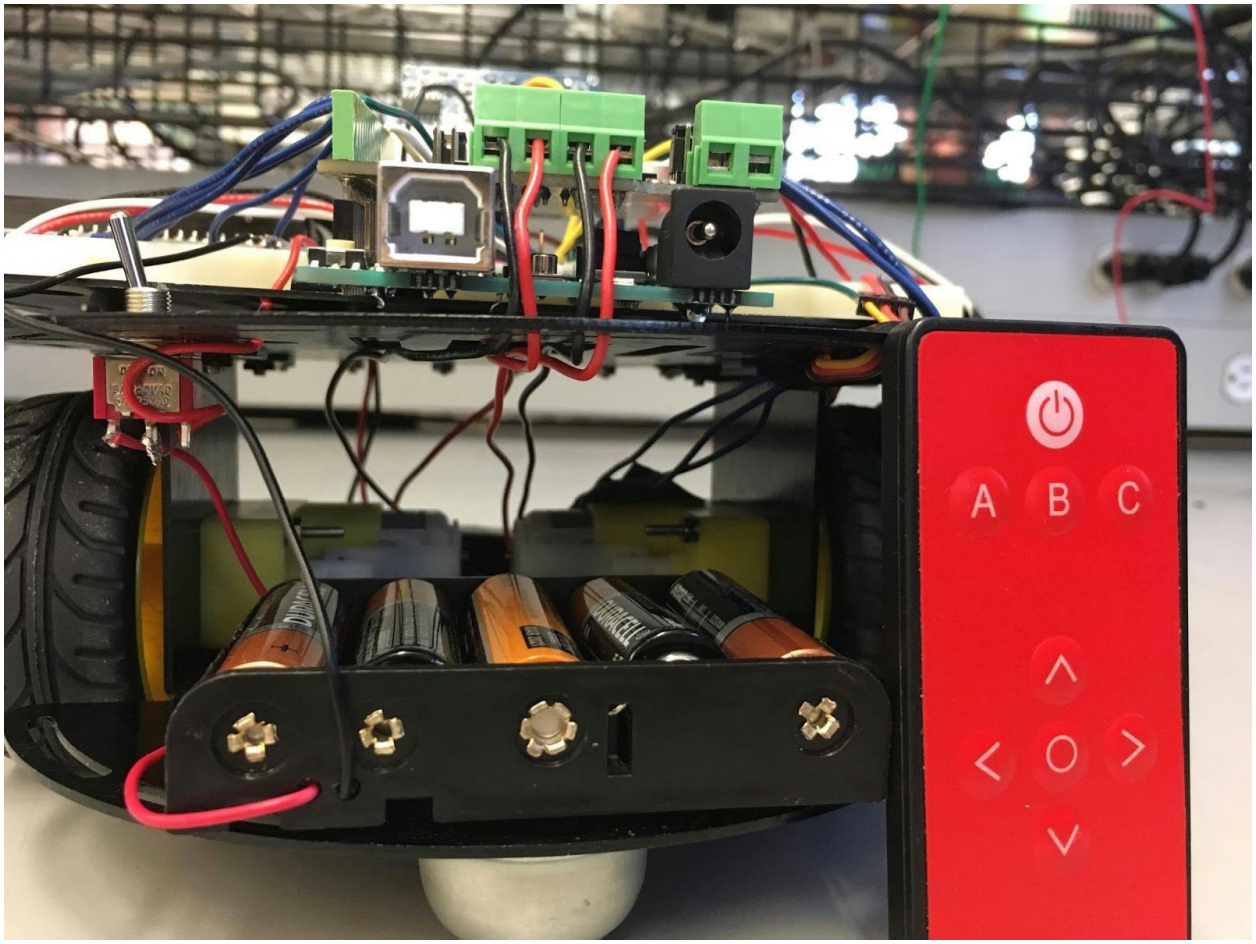
References and Bibliography

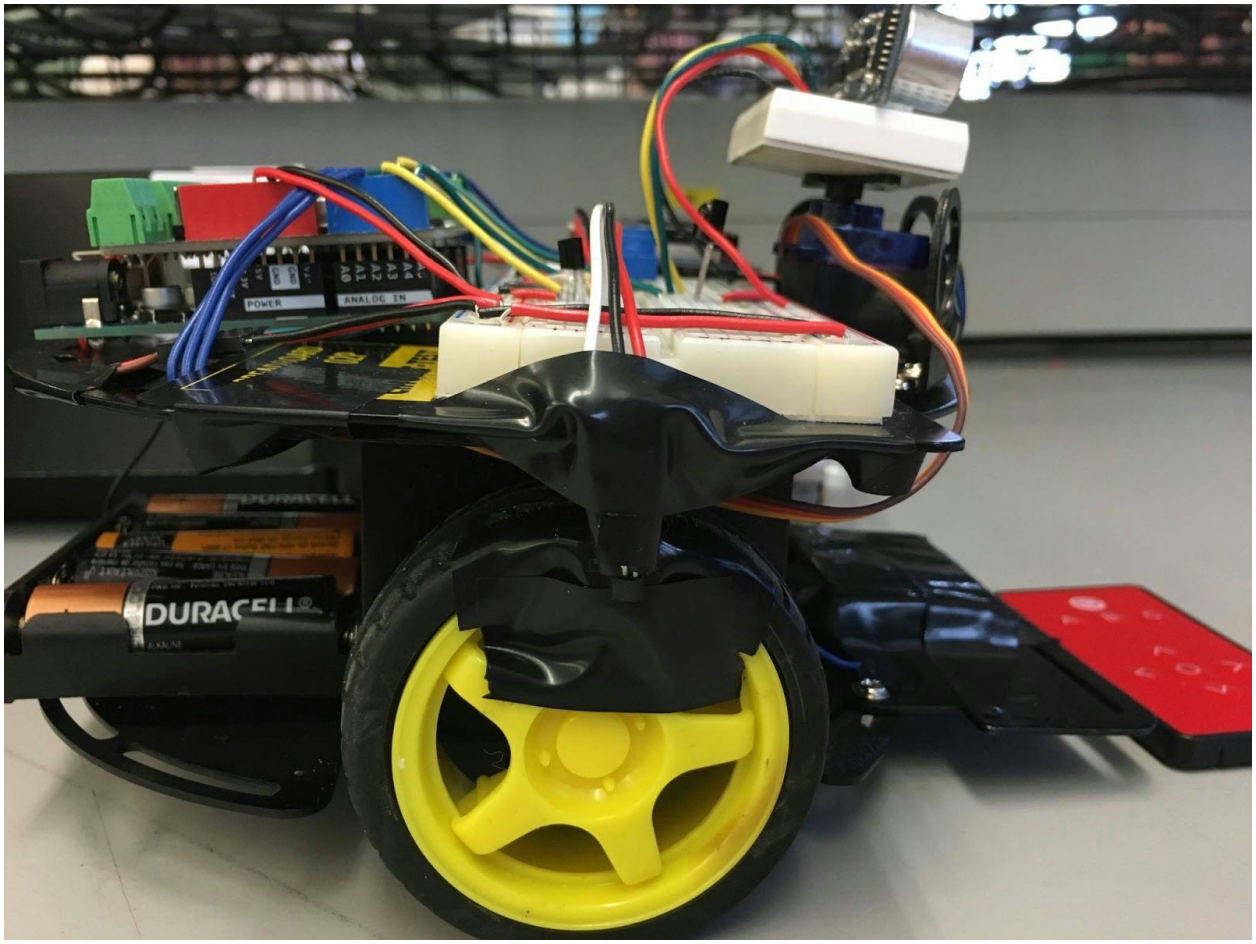
Data constraints for analog speed: https://www.arduino.cc/en/Reference/Constrain
Data mapping for analog speed: https://www.arduino.cc/en/Reference/Map
Interrupts for RPM: https://www.arduino.cc/en/Reference/AttachInterrupt
Wheel diameter: http://www.dfrobot.com/index.php?route=product/product&path=46&product_id=352#.VsYrFyArK00
The TRCT5000 for fritzing breadboard schematic: https://github.com/robertoostenveld/fritzing/blob/master/TCRT5000%20sensor%20module.fzpz
Connecting the tcrt5000 to Arduino Uno: https://www.youtube.com/watch?v=TySSW2EfNRM
Motor Shield wiki: http://www.dfrobot.com/wiki/index.php?title=Arduino_Motor_Shield_(L298N)_(SKU:DRI0009)
Motor Shield schematic: ArduinoL298ShieldSch.pdf
Mobile Platform Assembly Manual: 2WDTurtleAssemblyManual.pdf
Motor Shield's controller datasheet: L298N_datasheet.pdf
Arduino Reference: http://arduino.cc/en/Reference/HomePage
Temperature sensor IC datasheet: LM35_datasheet.pdf
HC-SR04 datasheet 1: HC_SR04_1.pdf
HC-SR04 datasheet 2: HC-SR04_Manual.pdf
Reflective Optical Sensor datasheet: tcrt5000_datasheet.pdf
Hall Effect Sensor datasheet: HallEffect_datasheet.pdf
IR Control Kit Hookup Guide: https://learn.sparkfun.com/tutorials/ir-control-kit-hookup-guide?_ga=1.217730774.1325785113.1441747493

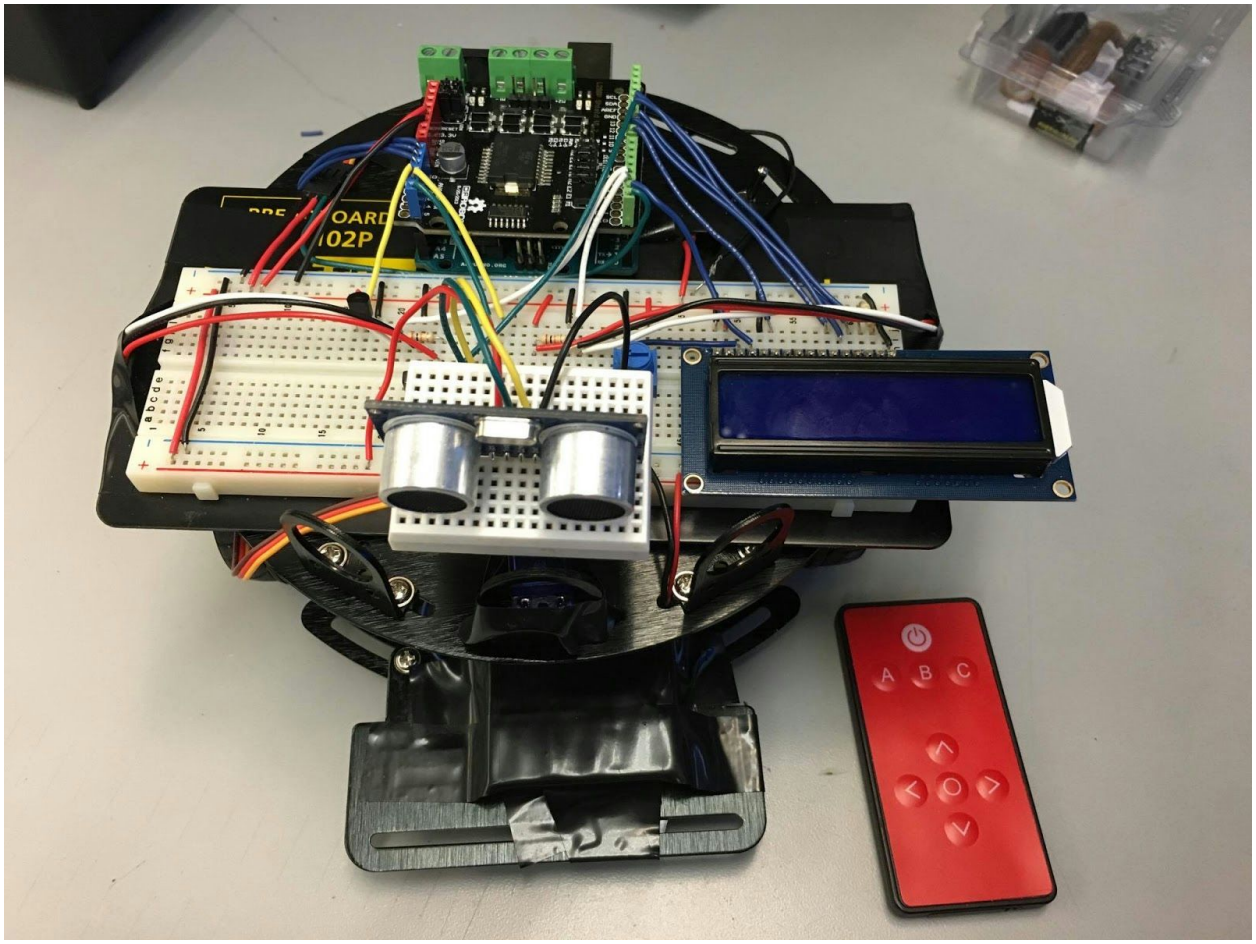
Appendix A – Robot pictures

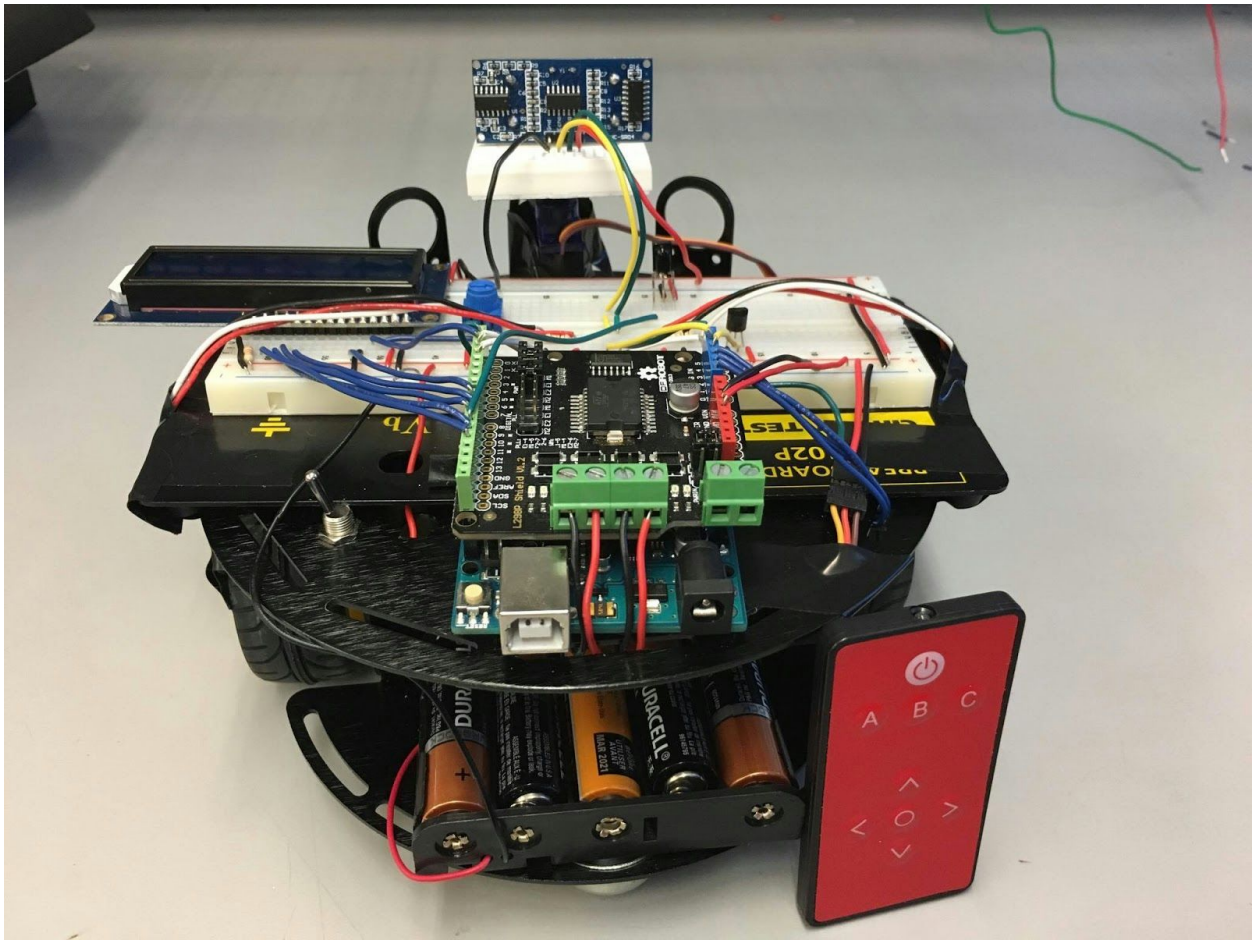
This appendix contains the pictures of the robot circuitry, sensors, all the components used and the remote.

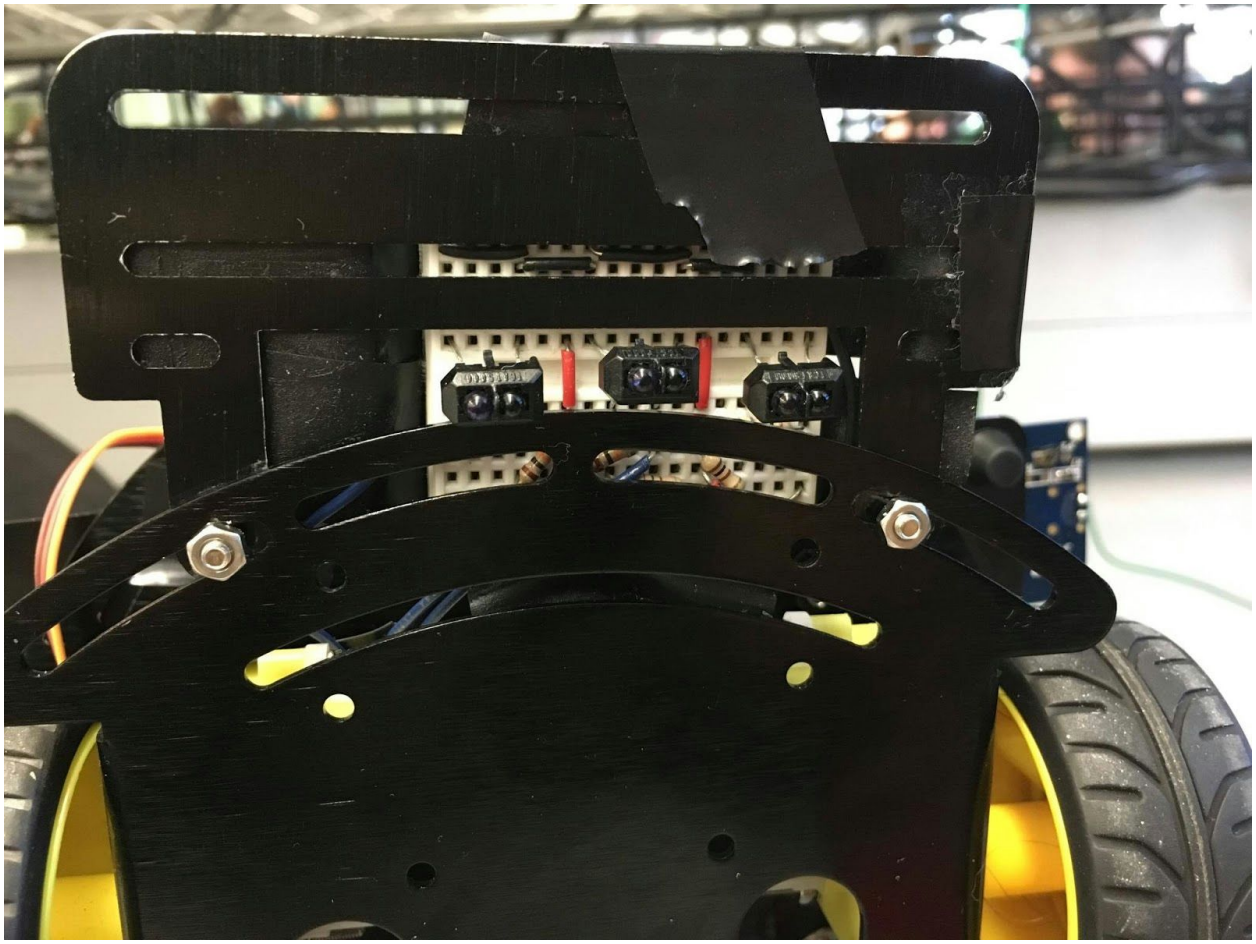



A B C


POWER
ANALOG IN
DURACELL


A B C


A B C



Appendix B - Code

```
// libraries
#include <Servo.h>
#include <LiquidCrystal.h>
#include <IRremote.h>

// initializations
LiquidCrystal lcd(1, 8, 9, 10, 12, 13); // CHECK BEFORE USE
Servo myServo;

//remote setting
int RECV_PIN = 11;
IRrecv irrecv(RECV_PIN);
decode_results results;
//remote commands
#define POWER 0x10EFD827
#define A 0x10EFF807
#define B 0x10EF7887
#define C 0x10EF58A7
#define UP 0x10EFA05F
#define DOWN 0x10EF00FF
#define Left 0x10EF10EF
#define RIGHT 0x10EF807F
#define STOP 0x10EF20DF
#define FAST 220
#define MEDIUM 180
#define SLOW 120

// HC-SR04 pins
#define TRIG_PIN A5
#define ECHO_PIN A4
#define TEMP_PIN A3
// Servo pin
#define SERVO 0
// motor pins
#define E1 5 // pin used for right motor
#define M1 4 // pin used for right motor
#define E2 6 // pin used for left motor
#define M2 7 // pin used for left motor
// hall-effect sensor pins
#define L_HALL_EFF 2 // pin used for hall effect sensor for left wheel
#define R_HALL_EFF 3 // pin used for hall effect sensor for the right wheel

// constants
#define SAMPLE_SIZE 8 // used in drowing noise from LM35
#define WHEEL_DIAM 6.5 // measured in cm
#define LEFT 1 // used in identifying next direction the robot will turn
```

```
#define SERVO_OFFSET 0 // offset used for servo to position the HC-SR04
#define MIN_TIME_DIFF 0.3347598729 // the minimum time in seconds for max speed of 61 cm/s
#define BASE_SPEED 220
#define MAX_SPEED 255

// global variables declarations
float duration = 0.0; // duration of pulse from HC-SR04
float distance = 0.0; // the calculated distance of an object relative to robot
int tempReading = 0; // temperature detected by LM35; used in improving distance calculations
int analogSpeedOfRobot = 0; // the analog equivalent of the robot's speed
float speedOfRobot = 0.0; // the actual speed of the robot in cm/s
int stopped = 0; // a flag used in identifying if the robot has due to an object in front of it
int servoFlag = 0; // a flag used in preventing the servo from scanning while it is trying to compute the next direction
float maxDistR = 0.0; // the maximum distance found from scanning to the right
float maxDistL = 0.0; // the maximum distance found from scanning to the left
int nxtDirection = 0; // the next direction that the robot should take; 1 for left and 2 for right
int adjustR = 0; // used to adjust speed of right wheel
int adjustL = 0; // used to adjust speed of left wheel
volatile float currTimeL; // used in calculating speed of left wheel
volatile float lastTimeL; // used in calculating speed of left wheel
volatile float currSpeedL; // current speed of left wheel
volatile float currTimeR; // used in calculating speed of right wheel
volatile float lastTimeR; // used in calculating speed of right wheel
volatile float currSpeedR; // current speed of right wheel
volatile int ignoreFirstR = 0; // flag to ignore first ISR of right wheel
volatile int ignoreFirstL = 0; // flag to ignore first ISR of left wheel

uint16_t lastCode = 0; // This keeps track of the last code RX'd

int A_flag = 0;
int B_flag = 0;
int C_flag = 0;
volatile int speed_flag = 0;
int speedLevel[3] = {SLOW, MEDIUM, FAST};
int index = 0;
volatile int stop_flag = 1;
#define sV 200 // velocity of going straight
#define tV 210  //velocity of turning
#define bV 100 // velocity of going forward when all on black


void setup(){
  // pin setup for HC-SR04
  pinMode(TRIG_PIN, OUTPUT); //pin for the trig pin of the ultrasonic sensor
  pinMode(ECHO_PIN, INPUT);  //the pin for the echo pin of the ultrasonic sensor
  pinMode(TEMP_PIN, INPUT);  //the pin for the temperature sensor
  // pin setup for motor
  pinMode(M1, OUTPUT); // right motor
```

```
  pinMode(M2, OUTPUT); // left motor

  irrecv.enableIRIn(); // Start the receiver

  // setup the LCD screen to a 2x16 (rows x cols) display
  lcd.begin(16, 2); //lcd setup

  lcd.setCursor(0, 0);
  lcd.print("Choose");
  lcd.setCursor(0, 1);
  lcd.print("a function");

  // setup servo
  myServo.attach(SERVO);

  // setup interrupts for speed calculation on each wheel of the robot
  attachInterrupt(digitalPinToInterrupt(R_HALL_EFF), wheelSpeedR, FALLING);
  attachInterrupt(digitalPinToInterrupt(L_HALL_EFF), wheelSpeedL, FALLING);
}

void loop() {
  if (irrecv.decode(&results))
  {
            /* read the RX'd IR into a 16-bit variable: */
            uint16_t resultCode = (results.value & 0xFFFF);

            /* The remote will continue to spit out 0xFFFFFFFF if a
            button is held down. If we get 0xFFFFFFF, let's just
            assume the previously pressed button is being held down */
            if (resultCode == 0xFFFF)
            resultCode = lastCode;
            else
            lastCode = resultCode;

            switch (resultCode){
            case POWER:
             analogWrite(E1, 0);
            analogWrite(E2, 0);
            A_flag = 0;
            B_flag = 0;
            C_flag = 0;
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Choose a Function ");
            break;
            case A:  //FOR FUNCTION ONE
             A_flag = 1;
            B_flag = 0;
```

```
        C_flag = 0;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Function A");
        break;
        case B:   //FOR THE SECOND FUNCTION
         A_flag = 0;
        B_flag = 1;
        C_flag = 0;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Function B");
        break;
        case C:
        analogWrite(E1, 0);
        analogWrite(E2, 0);
        A_flag = 0;
        B_flag = 0;
        C_flag = 1;
        stop_flag = 1;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Function C");
        break;
        default:
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Choose a Function ");
        break;
        }
        irrecv.resume();
}

//-------------reset variables in functionA ----------------------
distance = 0.0;
analogSpeedOfRobot = 0; // the analog equivalent of the robot's speed
speedOfRobot = 0.0; // the actual speed of the robot in cm/s
stopped = 0; // a flag used in identifying if the robot has due to an object in front of it
servoFlag = 0; // a flag used in preventing the servo from scanning while it is trying to compute the next direction
maxDistR = 0.0; // the maximum distance found from scanning to the right
maxDistL = 0.0; // the maximum distance found from scanning to the left
nxtDirection = 0; // the next direction that the robot should take; 1 for left and 2 for right
adjustR = 0; // used to adjust speed of right wheel
adjustL = 0; // used to adjust speed of left wheel
ignoreFirstR = 0; // flag to ignore first ISR of right wheel
ignoreFirstL = 0; // flag to ignore first ISR of left wheel
//-----------------------------------------------------------------
while (A_flag == 1) {
```

```
        func_A();
  }
  while (B_flag == 1) {
    func_B();        }
  //-------------reset variables in functionC ----------------------
  distance = 0.0;
  analogSpeedOfRobot = 0; // the analog equivalent of the robot's speed
  speedOfRobot = 0.0; // the actual speed of the robot in cm/s
  stopped = 0; // a flag used in identifying if the robot has due to an object in front of it
  servoFlag = 0; // a flag used in preventing the servo from scanning while it is trying to compute the next direction
  maxDistR = 0.0; // the maximum distance found from scanning to the right
  maxDistL = 0.0; // the maximum distance found from scanning to the left
  nxtDirection = 0; // the next direction that the robot should take; 1 for left and 2 for right
  adjustR = 0; // used to adjust speed of right wheel
  adjustL = 0; // used to adjust speed of left wheel
  ignoreFirstR = 0; // flag to ignore first ISR of right wheel
  ignoreFirstL = 0; // flag to ignore first ISR of left wheel
  //-----------------------------------------------------------------
  while (C_flag == 1) {
        func_C();
  }
}
```

//First functionality: Robot going with max speed, slowing down at 50cm range and stopping before hitting an obstacle ahead ,;

```
void func_A() {
  if (irrecv.decode(&results))   {
        switch (results.value)      {
        case POWER:
        analogWrite(E1, 0);
        analogWrite(E2, 0);
        A_flag = 0;
        B_flag = 0;
        C_flag = 0;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Choose a Function ");
        return;
        break;
        case A:  //FOR FUNCTION ONE
        A_flag = 1;
        B_flag = 0;
        C_flag = 0;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Function A");
        break;
        case B:
        A_flag = 0;
```

```
        B_flag = 1;
        C_flag = 0;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Function B");
        return;
        break;
        case C:
        analogWrite(E1, 0);
        analogWrite(E2, 0);
        A_flag = 0;
        B_flag = 0;
        C_flag = 1;
        stop_flag = 1;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Function C");
        return;
        break;
        default:
        A_flag = 1;
        break;
        }
        irrecv.resume();
}

-------------------------------------------------
// calc distance
distance = calcDist();

// LCD print
if (currSpeedL < currSpeedR) {
        speedOfRobot = currSpeedL;
} else {
        speedOfRobot = currSpeedR;
}

if (distance <= 10 && !stopped ) {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Determining ");
        lcd.setCursor(0, 1);
        lcd.print("direction.");
        stopped = 1;
        stopWheels();
}

// speed control logic when approaching an object
```

```
  if (!stopped) {
        analogSpeedOfRobot = map(constrain(distance, 10, 50), 10, 50, 120 , BASE_SPEED); // NOTE: may need
to change mapping when switching over to batteries
        if (distance > 50) {
        digitalWrite(M1, HIGH);
        analogWrite(E1, analogSpeedOfRobot + adjustR);
        digitalWrite(M2, HIGH);
        analogWrite(E2, ((analogSpeedOfRobot + adjustL) * 115) / 100);
        }
        else if (distance <= 50) {
        digitalWrite(M1, HIGH);
        analogWrite(E1, (analogSpeedOfRobot + adjustR) * 0.75);
        digitalWrite(M2, HIGH);
        analogWrite(E2, analogSpeedOfRobot + adjustL);
        }
        lcd.setCursor(0, 0);
        lcd.print("Speed(cm/s):");
        lcd.setCursor(12, 0);
        lcd.print(speedOfRobot);
        lcd.setCursor(0, 1);
        lcd.print("Dist(cm):");
        lcd.setCursor(9, 1);
        lcd.print(distance);
        lcd.setCursor(14, 1);
        lcd.print("  ");
  }

  // scan logic for next direction i.e left or right if stopped == 1
  if (stopped) {
        nxtDirection = getDirection();
        if (nxtDirection == LEFT) {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Turning left.");
        turnLeft();
        lcd.clear();
        } else {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Turning right.");
        turnRight();
        lcd.clear();
        }     }
}
```

//Second functionality: Robot should follow a black line

```
void func_B() {
  if (irrecv.decode(&results))
  {
```

```
switch (results.value)    {
case POWER:
//pressing the power to enable the  user to choose a function
analogWrite(E1, 0);
analogWrite(E2, 0);
A_flag = 0;
B_flag = 0;
C_flag = 0;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Choose a Function ");
return;
break;

//just in case if the user wants to switch back to function A
case A:  //FOR FUNCTION ONE
A_flag = 1;
B_flag = 0;
C_flag = 0;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Function A");
return;
break;

case B:
A_flag = 0;
B_flag = 1; // enable flag for function B
C_flag = 0;
lcd.clear();
lcd.setCursor(0, 0); //set the cursor of the LCD so that it prints Function B from the beginning
lcd.print("Function B");
break;

case C:
analogWrite(E1, 0);
analogWrite(E2, 0);
A_flag = 0;
B_flag = 0;
C_flag = 1;
stop_flag = 1;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Function C");
return;
break;
default:
B_flag = 1;
```

```
        break;
        }
        irrecv.resume();
}
digitalWrite(M1, HIGH);
digitalWrite(M2, HIGH);
int R = analogRead(A2) ;//right sensor
int L = analogRead(A0) ;// left sensor
int Mid = analogRead(A1) ; // middle sensor
// middle one on black
if (Mid > 100) {
        // left and right ones on white
        if (L < 100 && R < 100)
        go(sV, sV);
        // all on black
        else if (L > 100 && R > 100)
        go(bV, bV);
        // left on black turn until it is on white
        else if (L > R) {
        if (L > 100)
        go(tV, 0);
        else
        go(sV, sV);
        }
        // right on black turn until it is on white
        else if ( R > L) {
        if (R > 100)
        go(0, tV);
        else
        go(sV, sV);
        }
}

// middle one on white
else {
        // all on white
        if (L < 100 && R < 100)
        go(bV, bV);
        //left and right ones on black
        else if (L > 100 && R > 100)
        go(bV, bV);
        // left on black turn until it is on white
        else if (L > R) {
        if ( L > 100)
        go(tV, 0 );
        else
        go(sV, sV);
        }
```

```
            // right on black turn until it is on white
            else if (L < R) {
            if (R > 100)
            go(0, tV);
            else
            go(sV, sV);
            }
  }
}
// third functionality: Remote and controlling the speed
void func_C() {

  if (irrecv.decode(&results))
  {
            /* read the RX'd IR into a 16-bit variable: */
            uint16_t resultCode = (results.value & 0xFFFF);

            /* The remote will continue to spit out 0xFFFFFFFF if a
            button is held down. If we get 0xFFFFFFF, let's just
            assume the previously pressed button is being held down */
                        switch (resultCode)
            {
            case POWER:
            analogWrite(E1, 0);
            analogWrite(E2, 0);
            adjustR = 0;
            adjustL = 0;
            A_flag = 0;
            B_flag = 0;
            C_flag = 0;
            stop_flag = 1;
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Choose a Function ");
            break;

            case A:  //FOR FUNCTION ONE
            analogWrite(E1, 0);
            analogWrite(E2, 0);
            adjustR = 0;
            adjustL = 0;
            A_flag = 1;
            B_flag = 0;
            C_flag = 0;
            stop_flag = 1;
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Function A");
```

```
break;

case B:   //FOR THE SECOND FUNCTION
analogWrite(E1, 0);
analogWrite(E2, 0);
A_flag = 0;
B_flag = 1;
C_flag = 0;
adjustR = 0;
adjustL = 0;
stop_flag = 1;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Function B");
  break;

case C: //STOP IT AND MAKE THE USER CONTROL IT
digitalWrite(M1, HIGH);
digitalWrite(M2, HIGH);
A_flag = 0;
B_flag = 0;
C_flag = 1;
adjustR = 0;
adjustL = 0;
stop_flag = 1;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Function C");
break;

case UP:   //MOVE IT FOWARD
adjustR = 0;
adjustL = 0;
digitalWrite(M1, HIGH);
digitalWrite(M2, HIGH);
analogWrite(E1, speedLevel[index] + adjustR);
if (index == 2)
analogWrite(E2, (speedLevel[index] + adjustL) * 1.15);
else if (index == 1)
analogWrite(E2, (speedLevel[index] + adjustL) * 1.3);
else if (index == 0)
analogWrite(E2, (speedLevel[index] + adjustL) * 1.35);

C_flag = 1;
stop_flag = 1;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Go straight");
```

```
break;

case DOWN:  //REVERSE IT
adjustR = 0;
adjustL = 0;
digitalWrite(M1, LOW);
digitalWrite(M2, LOW);
analogWrite(E1, speedLevel[index] + adjustR);

if (index == 2)
analogWrite(E2, (speedLevel[index] + adjustL) * 1.15);
else if (index == 1)
analogWrite(E2, (speedLevel[index] + adjustL) * 1.35);
else if (index == 0)
analogWrite(E2, (speedLevel[index] + adjustL) * 1.45);

C_flag = 1;
stop_flag = 1;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Go Back");
break;

case RIGHT: //TURN IT RIGHT
digitalWrite(M1, HIGH);
digitalWrite(M2, HIGH);
analogWrite(E1, 0);
analogWrite(E2, 255);
adjustR = 0;
adjustL = 0;
C_flag = 1;
stop_flag = 1;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Turning Right");
break;

case Left:  //TURN IT LEFT
digitalWrite(M1, HIGH);
digitalWrite(M2, HIGH);
analogWrite(E1, 255);
analogWrite(E2, 0);
adjustR = 0;
adjustL = 0;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Turning Left");
C_flag = 1;
```

```
            stop_flag = 1;
            break;

            case STOP://stop and chose a speed level
            analogWrite(E1, 0);
            analogWrite(E2, 0);
            adjustR = 0;
            adjustL = 0;
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Stop");
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("choose the");
            lcd.setCursor(0, 1);
            lcd.print("speed level");
            speed_flag = 1;

            switchSpeed();//function for choosing speed level

            break;

            default:
            C_flag = 1;
            break;
            }

            irrecv.resume();
  }
}

void switchSpeed() {
  while (speed_flag && stop_flag) {
            if (irrecv.decode(&results))
            {
            /* read the RX'd IR into a 16-bit variable: */
            uint16_t resultCode = (results.value & 0xFFFF);
            switch (resultCode)
            {
            case POWER://reset button
            lcd.clear();
            lcd.print("Go!");
            A_flag = 0;
            B_flag = 0;
            C_flag = 1;
            speed_flag = 0;
            stop_flag = 0;
            break;
```

```
case UP:// change to a higher speed
A_flag = 0;
B_flag = 0;
C_flag = 1;
speed_flag = 1;
stop_flag = 1;
//the SpeedLevel array is {LOW, MEDIUM, HIGH}
if (index == 1) {//if the speed is MEDIUM
index = 2; // goes to HIGH
}
else if (index == 0) {//if the speed is LOW
index = 1;//goes to MEDIUM
}
else index = index;
lcd.clear();
lcd.setCursor(0, 0);
if (index == 0)
lcd.print("SLOW");
else if (index == 1)
lcd.print("MEDIUM");
else if (index == 2)
lcd.print("FAST");

break;

case DOWN://change to a lower speed
A_flag = 0;
B_flag = 0;
C_flag = 1;
speed_flag = 1;
stop_flag = 1;
if (index == 1) {//if the speed is MEDIUM
index = 0;// goes down to LOW
}
else if (index == 2) {//if  the speed is HIGH
index = 1;//goes to MEDIUM
}
else index = index; //if the speed is LOW, do nothing

lcd.clear();
lcd.setCursor(0, 0);
if (index == 0)
lcd.print("SLOW");
else if (index == 1)
lcd.print("MEDIUM");
else if (index == 2)
lcd.print("FAST");
```

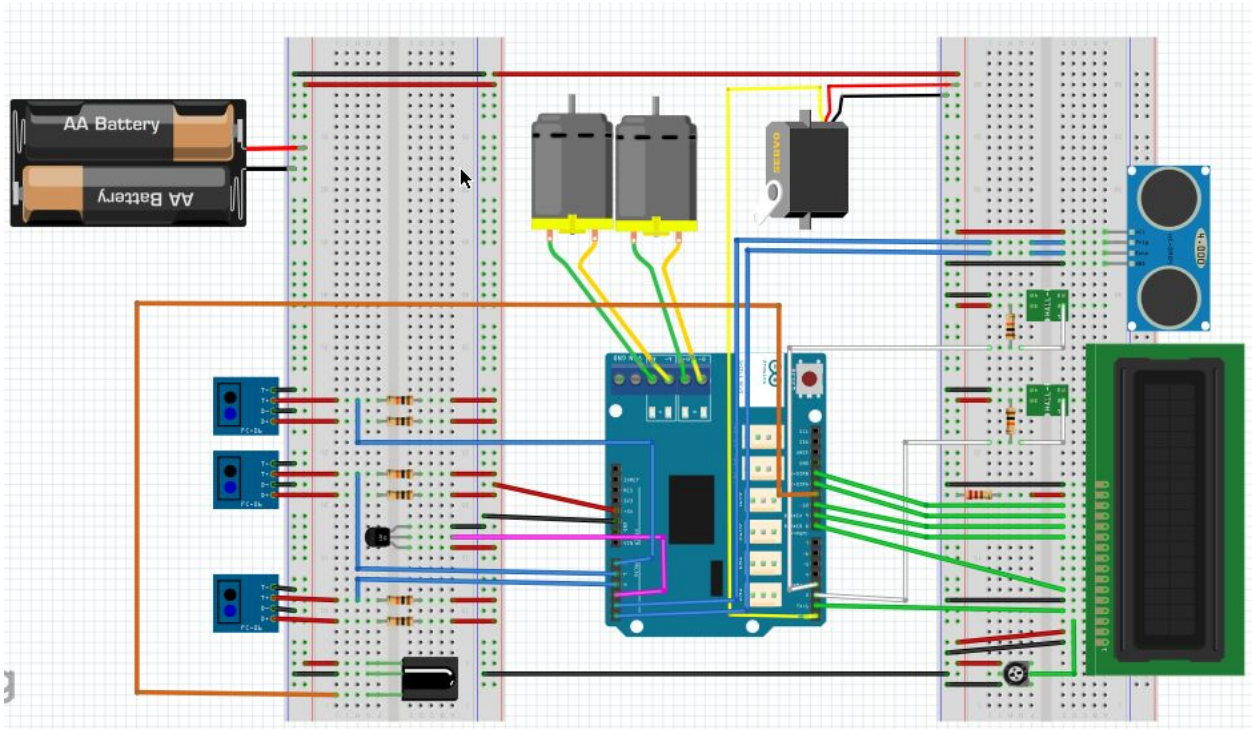
```
            break;

            default:
            speed_flag = 1;
            stop_flag = 1;
            C_flag = 1;
            break;
            }

            irrecv.resume();
            }
  }
}

float calcDist() {
  // to reduce noise, we take several sample readings and take their average
  tempReading = 0;
  for (int i = 0; i < SAMPLE_SIZE; i++) {
            tempReading = tempReading + analogRead(TEMP_PIN);
            delay(30);
  }

  // get average of temperature samples
  tempReading = tempReading >> 3; // shift right by 3 to divide by 8 (our sample size)

  // initialize the trigger pin by doing a digitalWrite() to LOW for 50ms
  digitalWrite(TRIG_PIN, LOW);
  delay(50);
  // start the trigger sequence by performing a digitalWrite() of HIGH for 10ms
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  // set to LOW on the same pin
  digitalWrite(TRIG_PIN, LOW);

  // calculate temperature from LM35
  float temperatureC = tempReading * (5.0 / 1024.0) * 100;
  // calculate the effective speed of sound
  float speedOfSound = (331.5 + (0.6 * temperatureC)) * 29 / 343;
  // obtain the echo pulse duration (in microseconds)
  duration = pulseIn(ECHO_PIN, HIGH);
  // get a more accurate distance calcSpeed using speedOfSound of sound
  distance = duration / (2 * speedOfSound);

  return distance;
}
  int getDirection() {
  int pos;
```

```
  stopWheels();
  if (stopped == 1 && servoFlag == 0) {
        // rotate servo right
        for (pos = 90 - SERVO_OFFSET; pos >= 0; pos -= 1) {
        myServo.write(pos);
        }
        maxDistR = calcDist(); // get distance on the right
        delay(1000); // wait for 1.5 seconds

        // rotate servo to left
        for (pos = 0; pos <= 180 - SERVO_OFFSET; pos += 1) {
        myServo.write(pos);
        }
        maxDistL = calcDist(); // get distance on left
        delay(1000); // wait for 1.5 seconds

        // rotate servo to middle
        for (pos = 180 - SERVO_OFFSET; pos >= 90 - SERVO_OFFSET; pos -= 1) {
        myServo.write(pos);
        }
  }

  if (maxDistR <= maxDistL) {
        return 1; // turn LEFT
  }
  else {
        return 2; // turn RIGHT
  }
  servoFlag = 1;
}

void turnLeft() {
  digitalWrite(M1, HIGH);
  digitalWrite(M2, LOW);
  // enable only the right wheel
  analogWrite(E1, 255);
  analogWrite(E2, 255);
  delay(580);
  // disable both wheels
  analogWrite(E1, 0);
  analogWrite(E2, 0);
  delay(100);
  adjustL = 0;
  adjustR = 0;
  stopped = 0;
  servoFlag = 0;
  lcd.clear();
}
```

```
void turnRight() {
  digitalWrite(M1, LOW);
  digitalWrite(M2, HIGH);
  // enable only the left wheel
  analogWrite(E1, 255);
  analogWrite(E2, 255);
  delay(550);
  // disable both wheels
  analogWrite(E1, 0);
  analogWrite(E2, 0);
  delay(100);
  adjustL = 0; // resetting the values back to zero
  adjustR = 0; // resetting the values back to zero

  stopped = 0;
  servoFlag = 0;
  lcd.clear();
}

void wheelSpeedL()    {
  if (ignoreFirstL < 1) {
            lastTimeL = millis();
            ignoreFirstL++;
            return;
  }

  currTimeL = millis();
  float timeDiff = (currTimeL - lastTimeL) / 1000.0;

  if (timeDiff < MIN_TIME_DIFF) {
            lastTimeL = currTimeL;
            return;
  }

  lastTimeL = currTimeL;
  currSpeedL = PI * WHEEL_DIAM / timeDiff;

  if (ignoreFirstR) {
            calcSpeedAdjust();
  }
}

void wheelSpeedR()   {
  if (ignoreFirstR < 1) {
            lastTimeR = millis();
            ignoreFirstR++;
            return;
```

```
  }

  currTimeR = millis();
  float timeDiff = (currTimeR - lastTimeR) / 1000.0;

  if (timeDiff < MIN_TIME_DIFF) {
        lastTimeR = currTimeR;
        return;
  }

  lastTimeR = currTimeR;
  currSpeedR = PI * WHEEL_DIAM / timeDiff;

  if (ignoreFirstL) {
        calcSpeedAdjust();
  }
}

void calcSpeedAdjust() {
  if (abs(currSpeedL - currSpeedR) > 0.5) {
        if (currSpeedR > currSpeedL && (((analogSpeedOfRobot + adjustL + 1) * 115) / 100) < MAX_SPEED
&& analogSpeedOfRobot + adjustR + 1 < MAX_SPEED)
        adjustL += 1;
        else if (currSpeedL > currSpeedR && (((analogSpeedOfRobot + adjustL + 1) * 115) / 100) <
MAX_SPEED && analogSpeedOfRobot + adjustR + 1 < MAX_SPEED)
        adjustR += 1;
  } else
        return;
}

void stopWheels() {
  digitalWrite(M1, HIGH);
  digitalWrite(M2, HIGH);
  analogWrite(E1, 0); // PWM control with speed adjustments
  analogWrite(E2, 0); // PWM control with speed adjustments
}

void go(int v1, int v2) {
  analogWrite(E1, v1); // PWM control the right wheel velocity
  analogWrite(E2, v2); // PWM control the left wheel velocity
}
```

Appendix C - Fritzing



Appendix D

HOW TO USE THE REMOTE CONTROL TO SELECT SPEED:

- MAKE SURE YOU ARE IN FUNCTION C BY CLICKING THE “C” BUTTON ON THE REMOTE
- TO SELECT THE SPEED, CLICK THE MIDDLE BUTTON (O) ON THE REMOTE
- USE THE UP BUTTON TO SET THE WHEELS TO A HIGHER SPEED AND THE DOWN BUTTON TO SET THEM TO A LOWER SPEED
- THE SPEEDS ARE IN A RANGE OF LOW, MEDIUM AND FAST (YOU WILL SEE THE DISPLAY ON THE LCD)
- THEN CLICK ON THE RESET BUTTON TO LOCK IN THE SPEED AND MAKE THE ROBOT MOVE IN THE SPEED YOU CHOSE (THE LCD WILL PRINT “GO!”)
- WHEN IN GO, YOU CAN USE THE ARROWS TO CONTROL THE DIRECTION OF THE ROBOT
- TO EXIT AND GET BACK TO OTHER FUNCTIONS (FUNCTION A AND B), CLICK THE RESET BUTTON ONCE