CPEN 291 2016W2

Project 1 Progress Report

Lab section: L2A   Group #: G5   Group's Lab-Bench #'s: 5A & 5B

Name	Student ID#	Contribution
Yu Chen	11691152	1/6
Benjamin Hong	38307154	1/6
Yuxiang Huang	14605159	1/6
Yuhao Huang	55562152	1/6
Ziqiao Lin	10668168	1/6
Hanyu Wu	36434158	1/6

Contribution Summary:

Github Repo Set-up : Yu

Principle Functionality 1: Yu, Hanyu, Yuxiang

Principle Functionality 2: Yuxiang

Principle Functionality 3: Benjamin, Ziqiao, Yu

Arduino Circuitry (LCD & sensors) Setup: Yu, Hanyu, Yuxiang

Android App Development: Benjamin , Ziqiao

Raspberry Pi: Yuxiang, Yuhao

B) Introduction and Motivations

This project focus on working with 2WD Mobile Platform, Arduino, Raspberry PI, LCD, and sensors.

Motivations

- Become competent to handle the complexity of circuitry
- Become adept at task delegation, module design, and get accustomed to working in a larger team environment
- Learn usage of many more electronic components such as motors and especially the many sensors
- Experience even longer, extensive testing

Design Objectives

- Create and learn android programming with Java and Android Studios
    - Design an elegant Graphical User Interface
    - Be able to switch modes (manual, auto, follow line)
    - Make the app read values given from the Arduino
- Be able to live stream through Raspberry PI's camera and upload the live stream on the internet
- Implement all principal functionalities

C) Project Description

1) **Principle functionality 1, autonomous mode**
   Principle Function 1 - Autonomous

For principal function 1, we implemented movement and control functions, in order to move the robot autonomously. The movement functions include **moveForward**, **turnLeft**, **turnRight**, **decelerate**, and the control functions include **sonarRead**, **scanAround**, and **rotate**.

According to the requirement of basic function 1, the robot needs to first move forward at its maximum speed, and if the robot detects an object in front of it, it needs to gradually decelerate, and stop at a location close to that object. To implement this functionality, first the function moveForward() is used to make the robot move forward forever until it receives next movement instructions. A while loop is used to achieve this purpose. Meanwhile, function sonarRead() constantly reads the distance by the ultrasonic Sensor. Once an obstacle is detected in the way of the robot at a distance limit, the while loop would be broken, and the function decelerate() will be used to gradually decelerate the robot, and finally stop the car as close to the object as possible. The function decelerate() takes deceleration level as a parameter. We did several experiments to find the deceleration rate that can make the robot stop as close to the object as possible. After the robot has fully stopped, the function scanAround will rotate the servomotor; since the ultrasonic sensor is mounted on the servomotor, the servomotor will rotate together with the sensor, from 0 to 180 degrees. The ultrasonic sensor checks the distance of the obstacle at different directions, and then stores the values in an array. Since the robot wants to turn to the direction that has more space to move, the function gets the maximum value from the array and return the corresponding direction which will be used to decide next movement of the robot. There are only two choices: left and right, both 90 degrees, according to the lab requirement. After the robot has finished rotating, it then again goes forward, and repeats the loop described above.

Principle Function 1 - Moving Straight

In addition, we need to use Hall effect sensors and magnets to keep the car moving straight. One common approach to achieve this functionality is that we put 5 magnets on each wheel of the robot, and make the south pole facing the Hall effect sensors as close as possible. Then we connect the output pins of the sensors to 2 analog pins on Arduino. Since the Hall effect sensor outputs low when the south pole of a magnet is close to it, we can write a function to constantly check the value read by the analog pin, and record the intervals between two minimum values, for both sensors in the left and right. When the robot is moving straight, the intervals of two wheels should be the same; otherwise, one motor must be moving faster than the other. Normally, if the interval of the left motor is less than that of right motor (both motors turning forward), then the robot is moving right. If the difference in motor speed is detected, we will adjust the motors accordingly.

2) Principle functionality 2, Line Follow Mode

The principle functionality 2 is to make the robot follow a dark line on a light surface. In order to achieve this goal, we use movement and control functions. The movement functions are **turnLeftForward_slow, turnRightFoward_slow**, **turnLeftForward_fast**, **turnRightForward_fast**, **sharpTurnLeft**, **sharpTurnRigh**t; the control functions are **checkBreakpoint**, **lineFollowPrepare**, and **scanGround**. We integrate most of the functions listed above into the **lineFollow** function, and use **lineFollowPrepare** separately for calibration at the beginning. Like principle functionality 1, we use movement functions to control the motor speed, and use control functions to detect the dark track and control the movement of the robot in different scenarios.

We used four optical reflective sensors. An Adafruit solderable breadboard is mounted at the front of the robot to hold the sensors. The sensors are soldered onto the solderable breadboard vertically, with the downside at the same level, in order to minimize the reading error. The optical reflective sensors are connected to the breadboard and Arduino, using 5 jumper wires each; 4 wires are directly connected to the sensor, and one wire is connected to one Arduino analog pin. The sensors are connected in the way that the Arduino analog pins read lower values if the sensors are facing darker surfaces, and vice versa. Therefore, the robot is able to detect the relative location of the dark track to the robot, and adjust its movement accordingly. We considered many different situations the robot may encounter on the track, for example, straight line, small angle turn, large angle turn, sharp angle turn, discontinuity, cross lines, and stop of the line. We programmed the Arduino in the way that the robot reacts differently and properly in different scenarios, and this is the reason why 6 different movement functions are used. The main difference among the movement functions is that we set different left and right motor power (speed) in different functions. For example, we set left motor speed to 0, and right motor speed to 225 in sharpTurnLeft function, so that the robot is able to effectively turn left without further deviating from the track, when facing a sharp corner; but in the turnLeftForward_slow function, we set the left motor speed to 50 and right motor speed to 150, so that the robot can gradually turn left while still being able to move forward to keep the speed, when facing a small angle towards left on the track. In addition, we used the scanGround function inside the movement functions, so that the robot is always checking the position of the dark track while turning and moving; this enables the robot to make proper reactions as fast as possible.

Furthermore, in order to enable the robot to follow tracks properly (especially deals with cross lines and automatically stops at the end of track) with different contrast to the surface, we used **lineFollowPrepare** function to calibrate the sensor readings. The idea is that the program records the maximum and minimum readings from the optical reflective sensor, and calculate the average of the the two numbers as a middle value. The constantly compares the reading from the four sensors to the middle value: if all four readings are lower than the middle value, it means the robot has reached the end of the track and it stops; if all four readings are higher than the middle value, it means the robot has encountered a cross, and it is supposed to move forward.

3) Additional functionality 1, manual remote control through Android application

The code for the Android app was based off and was modified from the source code at https://github.com/Mayoogh/Arduino-Bluetooth-Basic. We first attempted to write the entire app from scratch, but we had trouble understanding and writing the Java code specifically regarding the Bluetooth part. Nevertheless, we still learned about the basics of how Bluetooth works in Android and Android studios to be able to modify the source code to our liking. We changed the GUI completely from the original code. We only used the original source code purely for the benefit of using the Bluetooth code.

App Main Page View

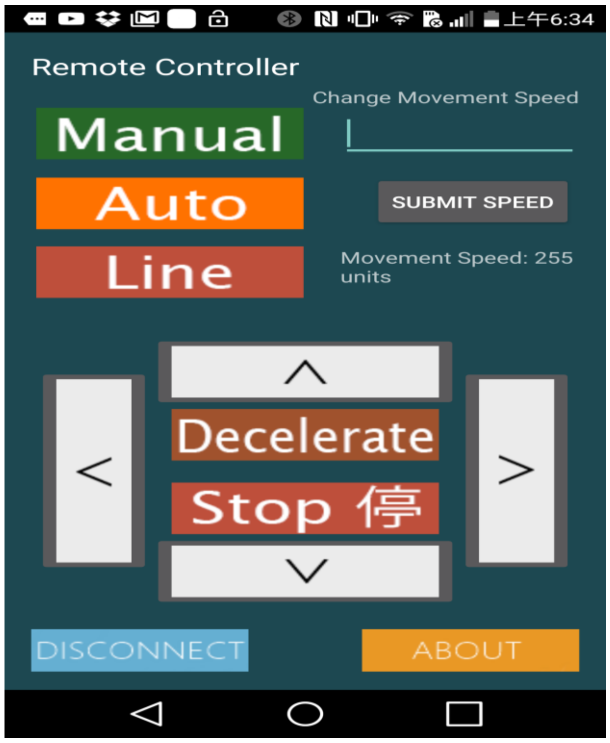


Figure 2. Android App GUI

Changes made to the original source code:

- All original buttons were removed (ie. the On and off button for the LEDs) except the Disconnect and About.
- 3 buttons for each modes and 5 motion buttons were added.
  - The buttons are of type ImageButton and each image were made with the use of Photoshop CS6.
- The About page was completely changed and is now about our group and our project
- Name of the app has been changed
- Icon of the app has been changed
- Splash screen has been changed

Mode Button Information

- Each button for manual mode, autonomous mode, follow-line mode.
- Press them to switch between modes.

Motion Button Information

- Hitting the left, up, down, right buttons activate and deactivates them.
  - Ex. Hit forward once and robot will move forward indefinitely until forward button is pressed again.

- Motion buttons only work when the mode is in remote control.
- When remote control button is pressed, all motion stops (if motion was happening as a result of different modes) and stays idle until either motions buttons are pressed or mode changes to a different mode
- When mode is not remote control, motion will begin immediately

All Button Permutation and Circumstances

1. If while motion is ongoing (ie. Button is pressed once) and a different motion button is pressed, then the ongoing motion will stop and will begin the ongoing motion.
2. If while motion is ongoing and the mode is changed to a different mode, then the ongoing motion will stop and begin its new mode's protocol.
3. If while mode is in remote control mode and the remote control button is pressed again, it resumes its previous motion. Examples:
    - If moving forward and remote control mode button is pressed again, then it keeps moving forward
    - If there is no motion and is idle and the remote control button is pressed, it stays idle

Remote Control function in Arduino

Our goal is using android app to control the car via bluetooth. We use Bluetooth module HC-05 and implement a function called read_instrucution in the main part. In the physical level, we use pin 9 for RX and pin 10 for TX. We import the SoftwareSerial library and set it up at the setup loop. In our main function, firstly we check whether there is data received via bluetooth and then we define a char type as the data we receive because the data received from Bluetooth is all char or string type. Then we use several if statement to check which instruction we get from app. We set forward functions as 'M', turn right as 'R', turn left as 'L' and sudden stop as 'S'. If we received data other than these, we will call method Decelerate_Stop to stop. All these methods have already explained above. In our Android app, each button will send corresponding data via bluetooth to arduino code.

4) Additional functionality 2, Raspberry PI and camera feed

Our original goal was to control the camera to record videos, and detect faces in the vision. The recorded video is uploaded to the Internet, so that people can see the real-time vision on the computer or smartphone. To achieve this goal, we installed a software called MJPEG-Streamer to our Raspberry Pi. This app can create a local host and upload the real time video streaming to the localhost. The localhost can be accessed by any device connected to the same network as Raspberry Pi's. In order to remotely access the video by laptops and phones, we use one laptop to open a wifi hotspot, and connect the Raspberry Pi to the hotspot. This guarantees the Raspberry Pi can share the same network with other devices. In addition, we planned to implement the face detection with OpenCV. First we figured out a way to execute commands on Raspberry Pi remotely with a laptop using ssh commands. Next we installed OpenCV on the Raspberry Pi. We did this by following the steps in an online tutorial which is included in the reference list. We implemented a program to detect faces in pictures. However, we could not implement face detection function in videos.

D) Tests & Evaluations and Specific Design/Implementation Decisions

- At the beginning of this project, after the robot is assembled, we write four basic movement functionalities, which allows the robot to move forward, turn and stop. We designed a testing program and let it move in a square pattern, and turn around and draw that square in reverse. So that we can make sure:
    - It goes in straight line: the two motor on both side has to turn with approximately same speed, the more accurate adjustments are done by magnets and hall effect sensors. We repeatedly adjust the power settings of two motors, is one side goes faster, we will lower the power on that side, by half interval search, we are able to find the proper adjustment to the motors.
    - When it turns, it can face a direction perpendicular to the previous direction. The left turn/right turn function controls the turing degree by changing the time both wheels turning in reversed direction. This involves a lot of testing and changing of the code. We let the robot turn around and time the time intervals in between, take the average of the turning 360 degree time and divide it by four, then we approximately get the time it needed to turn 90 degrees.
    - When it stops, it will decelerate and comes to a eventual stop, we will need to test how fast should it decelerate so that the stopping will have the expected behavior: neither stopping too early that it is too far from the obstacles, nor too late that it crash into the blocking object. For this part, we still use the half interval search, if too fast, then cut the steps in half, else if stops too late, we take the one at the half point of upper interval, and eventually find the satisfactory stopping deceleration.
- For testing principle functionality one. When we implement the sensors, whenever we have an unexpected behavior, meaning the sensor and the code logic does not perform the designed functionality. We will try to use the Serial port to printed the reading values on the screen, we follows the following procedure:
    - If the serial reading is reasonable, then it is very like that there is something wrong with the logic, we are not processing the readings properly, we will look into the part of code related to the sensor and the abnormality.
    - If the serial reading is not reasonable or have no readings:
        - First we remove the sensor, and build a testing circuit and program just to see if the sensor is working, in some cases, like optical sensors, which are fragile, it did breaks if not used properly.
        - If the sensor works correctly, we will look into the circuitry. We will be drawing fritzing sketches and refer to the datasheet of the corresponding component(s), we will look especially carefully for short circuiting or “Miss by one” case on the breadboard.
- For principle functionality one. The major and only physical output module we have is the LCD screen. To debug the LCD screen, first we tried to update the lcd among the code lines, but later we found this is a not professional practise. We put the lcd commands in a code block and send

values and format that we want it to display, and solves the problem.

- For principle functionality one, since the lcd will take up 6 data pins, we have problems with not having enough pins. We are trying to use two arduinos, one is the master sender, and the other is a slave receiver, connected through serial ports on arduino board RX and TX. We have problems with this serial communication, the master sender(with sensors and motors etc.) is sending data a lot faster than the speed humans can read, which limits the speed of the slave receiver(with the LCD). For the additional functionality, the bluetooth is using serial ports. For those reasons, we decided to do some rearranging of the sensors and wires so that there is eventually enough wires to perform the basic functionalities as well as the bluetooth remote control, and use LCD attached as output.
- For principle functionality two. When we were adding the components for principle functionality two, we burnt 3 optical reflective sensors. The diagnosis result is that we may have some short circuit on the breadboard, which triggered large current through the sensors, and as a result completely burnt them. The short circuit problem is a serious problem throughout this project, and this one caused the most serious damage. We have not found an effective way to solve this problem, except for being very careful when doing the wiring.
- For principle functionality two, the biggest challenge i enable the car to turn at sharp corners at the track. We thought about this problem and used 4 sensors at first, but we put the sensors very close to each other. This limited the detection range of the sensor. Therefore, finally, we used 4 sensors with each sensor has a 2 cm distance to the ones beside it. We decided to use a 2 cm distance, since the width of the track is 2 cm. This maximize the detection range of the sensors, and solved the problem.
- For testing additional functionality one, when testing the logic of the code, we use keyboard serial. Instead of actual sensor readings, we use certain keyboard values to control the movement of the robot, so that the debug will take less time. For example, if we want to see if sonar reading a distance below the limit we set will lead to the turning of the motor so that the sonar can scan around. We will do this:
    - Press key 1 to force the robot into autonomous mode
    - Press key to let the robot move forward until it is close enough to trigger the scanning
    - See if the scanning works
    - See distance get from the scanning on the Serial port and decide if it has made proper range detect and decisions to turn to the direction with larger spaces available.
- We had a circuit problem such that the bluetooth module kept turning on and off sporadically. We realized the module wasn’t connected to the 5V row/columns on the breadboard correctly. This was due to our messy circuitry and so we revised and improved the circuitry to make it look more neat and easier for looking at the circuit.
- For additional functionality one, with the android app that is used to control the robot in manual mode though bluetooth, first we did not put the bluetooth with the robot in the first point. Since you can not use serial monitor when using RX and TX, we moved bluetooth module to other pins temporarily. When we press arrows or stop on the app, we can see on the Serial monitor which signal is being received. This is used to debug the app design and decide the frequency to update the signal so that the robot will be much easier to control in this mode.

- For additional functionality one, when we first wrote the bluetooth code of the Arduino code, we noticed there was a ~1 second delay between every command it receives. This was a problem as it made controlling the robot in manual much much harder. We realized that this delay was because in the Arduino code, we used BT.readString() rather than BT.read(). First function returns a string and the second returns a character. We suspect working with strings require more processing time. Thus, we used .read() to fix the delay.
- At the end of this project, after we tested each individual components works properly and the basic logic is correct. We started to put things together for combined testing.
  - We design the circuitry on fritzing and have different people building and checking the circuits.
  - System with multiple functionalities and high complexities are very hard to debug. When we observe unexpected behaviors, first we come up with assumptions which part could be wrong. If we can locate which sensor this problem is related to, we will have people testing if the sensor itself is working properly, which usually is, since we already tested it before we put it onto the board altogether. At the same time, we will have people look into the logic of the code related to this part. After testing with the sensor, we will focus testing about the codes. We will check again with the wiring to make sure there is no hazard.
  - If we can not locate the part of the system that potentially has problems, we will test from the highest level to the lowest level. We will first test if the mode choosing part is working, then test the modes separately. For instance, if autonomous mode is not working, we will check the decision module of the code. If the decision is reasonable, we will look into all the data sources. Until we reach the lowest level which is the sensor reading and basic movements.
- For powering the robot, we use batteries to power the motors, when deciding which method to power the arduino, we choosed power bank over batteries, so that the batteries will not wear out too fast.
- For additional functionality 2, one big problem we encountered in this part was remotely controlling the Raspberry Pi with a laptop. We first tried to set a hot spot on the laptop in order for the Pi to connect to the laptop.. However, this solution failed because the Mac laptop cannot share the internet from wifi to wifi, and it also does not have a port for an ethernet cable. Finally we came up with an idea that although we cannot connect ethernet cable to the laptop, we can still connect the Raspberry Pi to the ethernet through a router. So this problem is now solved.
- For additional functionality 2, the biggest problem we encountered was finding resources online. Since none of us has previous experience with python and networking, we spent a lot of time looking for online resources to implement low latency video streaming. This significantly improved our self learning ability.

E) Conclusions and Reflections

- The background research, discussion and planning were detailed, valid and helpful
- The task delegation is clear and effective, every member in the group made even contribution and we are able to advance the project progress within the areas that each of us is familiar or have experience in.
- The result of the design is satisfactory:
    - For the principal functionality I, the robot is able to move autonomously:
        - The building of the robot is successful, strong and ready for change
        - The switch control of mode is effective
        - The 3D printed Sonar sensor holder is very useful
        - The LCD is able to provide informations, sensors' circuitries are correct.
        - The robot can move forward almost in a straight line
        - The object detect and decision process is successful and reasonable
        - The sonar is able to scan around to provide data to the decision logic
        - Overall, the implementation of principal functionality meets expectation
    - For the principal functionality II, the robot is able to follow the line:
        - The robot's mode can be changed with a switch, besides the tilting power switch
        - The robot is able to follow the track and deal with curves and crossings with a high pace of movement
        - Robot is able to identify tracks with low contrast
        - Overall, the implementation of the principal functionality is complete and successful.
    - For the bluetooth functionality:
        - The mode can be changed from other mode to manual or from manual to autonomous.
        - The bluetooth connection is stable and the data flow is good.
        - The handling of the robot is friendly
        - The GUI design is clear and simple
        - The speed adjustment is useful and original
        - Overall, the manual mode successfully enhanced the usefulness and originality of our robot design,
    - For the photocell on raspberry pi 3:
        - The photocell works correctly and is able to stream videos through internet connection
        - The human face identification, even not 100% accurate, is original and a spectacular feature to have.
        - Due to the power supply reasons, we are unable to place the pi and photocell on the robot
        - Overall, the implementation is complete, but due to limitations that are not able to demonstrate. We will regard this development as a great experience and introduction that will be useful in Project 2

- Design tool Fritzing was useful and is ideal for a prior circuit planning and a visualization of a certain schematic design.
- The coding practice was helpful; we were able to enhance C and Java programming skills and become familiar with working with Arduino and Processing.
- Testing, evaluating and improving process has been lengthy and at some time, tedious, but certainly invaluable. It has been a core and great help to solving our problems.
- By building circuits with own hand, we were able to understand better regarding the electronic components and circuitry knowledge.
- Working together first time as a team of six is an invaluable experience for all of us, we got to know each other and begin to learn how to work as a bigger project team
- Version control tools such as Github is very useful, so that everyone's contribution is recorded and is helpful for us when working on a objective concurrently.
- Areas to be improved:
    - We should consider the weight of boards and components on the robot and make adjustment to the alignment of two motors as we add more things to it, so that the robot will constantly move forward in a straight line.
    - We should make better use of the branch feature of github
    - We could communicate more often so as to keep everybody informed of the details of the project progress.
    - We could come up with conventions when coding, such as common case for variables and even libraries for functions. We should address everyone use more professional coding practice: write specifications and comments while we write the code
- Overall, the successful complete of this robot project will not only become a milestone in our academic life as a engineering student, but also a unforgettable and invaluable experience along our life journey. The development requires enormous effort and energy, sometimes even bring us sorrow and disappointment. However, nothing will compare to the excitement when we see it operate as expected for the first time. What a great feeling of satisfaction and achievement! Time goes with the wind, but those tears of joy will be like crystal, glittering in our life.

F) Reference and bibliography

App Sources Code and Bluetooth implementation:

- https://create.arduino.cc/projecthub/user206876468/arduino-bluetooth-basic-tutorial-d8b737

Bluetooth:

- https://www.instructables.com/id/Connect-Arduino-Uno-to-Android-via-Bluetooth/
- https://www.instructables.com/id/Connect-Arduino-Uno-to-Android-via-Bluetooth/

Hall Effect Sensor:

- http://playground.arduino.cc/Code/HallEffect
- https://diyhacking.com/arduino-hall-effect-sensor-tutorial/
- http://www.hobbytronics.co.uk/arduino-tutorial11-hall-effect
- https://www.instructables.com/id/Hall-effect-sensor/

Hall Effect Sensor Using Interrupt:

- http://playground.arduino.cc/Main/ReadingRPM

Raspberry Pi Low Latency Video Streaming

- http://petrkout.com/electronics/low-latency-0-4-s-video-streaming-from-raspberry-pi-mjpeg-streamer-opencv/

Face Detection Using OpenCV

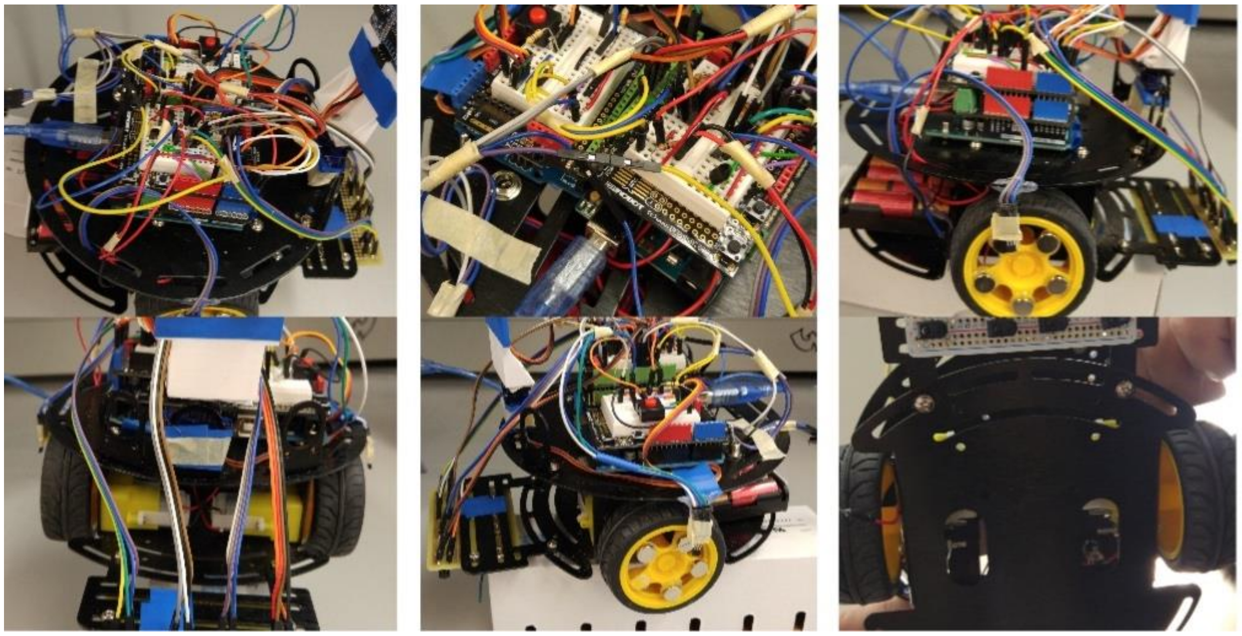
- http://opencv.org/downloads.html
- http://www.cs.princeton.edu/courses/archive/fall08/cos429/CourseMaterials/Precept1/facedetect.pdf

Arduino Line Follower Robot

- http://www.instructables.com/id/Line-following-Robot-with-Arduino/

Appendix A – Robot pictures

Include pictures of your robot here. The pictures should clearly show the robot as a whole, and also the location/installation of the sensors, circuitry, etc. as clearly as possible.



Appendix B - Code

```
#include<Servo.h>

#include <LiquidCrystal.h>

#include <SoftwareSerial.h>

Servo servo;                    //Create an object myServo

SoftwareSerial BT(9, 10);

// initialize the LCD library with the numbers of the interface pins


//robot motor pin configuration

//Arduino PWM Speed Control： E1 is right motor; E2 is left motor

const int E1 = 5;

const int M1 = 4;

const int E2 = 6;

const int M2 = 7;


//pin configurations

const int TRIG_PIN = 12;

const int ECHO_PIN = 11;

const int SERVO_PIN = 13;

const int LM_PIN = A0;      //Pin for LM-35 temperature sensor

const int SWITCH_PIN = 8;        //Pin for pull up Switch

const int MAIN_SWITCH_PIN = A5;   // main switch pin configuration


//Declaration of variables for basic function 2

//Pins for 2 optical reflective sensors

const int ORsensorpin_1 = A1;

const int ORsensorpin_2 = A2;

const int ORsensorpin_3 = A3;

const int ORsensorpin_4 = A4;



//define some constants here

const int DISTANCE_LIMIT = 15;

const int MAX_SPEED = 255;
const int DECELERATION = 45;

const int LINE_FOLLOW_SPEED = 190;

const int TURNING_SPEED_NORM = 120;

const int TURNING_SPEED_DIFF = 80;

const int TURNING_SPEED_MAX = 225;




//define some global variables here


char instr='A';

int dis;

int main_switch = 0;

int temperature;

int lcd_distance;

int flag=1;

int velocity=255;

//declaration of variables for hall effect sensor

volatile byte half_revolutions;

unsigned int rpm;

unsigned long timeold;


float speedLeft;

float speedRight;

long normalPower=200;

long newPower=normalPower;




int midVal = 0;

int minSensorVal = 1023;

int maxSensorVal = 0;


int TempORsensorVal_2;

int ORsensorVal_1;
int ORsensorVal_2;

int ORsensorVal_3;

int ORsensorVal_4;


void setup()

{

  pinMode(M1, OUTPUT);        //DC motor

  pinMode(M2, OUTPUT);


  pinMode(TRIG_PIN, OUTPUT);    //Sonar

  pinMode(ECHO_PIN, INPUT);


  pinMode(SWITCH_PIN, INPUT_PULLUP);    //Switches

  pinMode(MAIN_SWITCH_PIN,INPUT);

  Serial.begin(9600);

  servo.attach(SERVO_PIN);

  BT.begin(9600);

}


void loop()

{

  main_switch = digitalRead(MAIN_SWITCH_PIN);

  if(main_switch == HIGH){

   /* choose the mode based on the pushed button */

   chooseMode(flag);

   //readSwitch();

   update_flag(instr);

  /* update the flag based on the signal received */


  /* after successfully change the mode, make it stop and wait for 1 sec */

   Serial.print("Successfully changed the mode ");

   Stop();

   delay(1000);

  }

}
```

```
/*
 * Mode 1: Basic functionality 1
 * The robot will move at its maximum speed forward until it detects an object in front of it at
some distance limit
 * When the robot detects an object, it will decelerate and rotate the sensors to find the
direction that has larger space
 * Then the robot turn left or right towards that direction.
 */


void mode1(){


  Serial.println("In mode 1");      //For Debug use
  moveForward(velocity);


  do {
    //LCD output instrctions
    /*lcd.setCursor(0,0);
    lcd.print("temp is ");
    lcd.print(readTmpLM());
    lcd.setCursor(0,1);
    lcd.print("dis is ");
    lcd.setCursor(7,1);
    lcd.print(readSonar());*/


    Serial.println("moving forward");
   // update_keyboard();
    update_instr();
    if(digitalRead(MAIN_SWITCH_PIN)==LOW){
      return;
    }
    if( flag!=1 ||instr =='F' || instr =='C') {
      return;   //check the interrupt flag
    }


    dis=readSonar();
    //detect range, if less than limit, decelerate

    if (dis < DISTANCE_LIMIT  && dis !=-1) {

     decelerate(DECELERATION);

     Serial.println("break out the loop!!!!");

     Stop();

     break;

    }

     dis=readSonar();

  }while((dis >= DISTANCE_LIMIT || dis ==-1 )&& digitalRead(MAIN_SWITCH_PIN)==HIGH);   //check if
anychanges after one loop

    //update_keyboard();

    update_instr();

   // this code block is used to monitor switches

   if(digitalRead(MAIN_SWITCH_PIN)==LOW){

    return;

   }

    if(flag!=1 || instr =='F' || instr =='C' ) {

      return; //return to loop method

    }

    Stop();

  int scanVal=scanAround(2);

  delay(500);

  rotate(scanVal);

}




//basic functionality 2

void mode2(){

  lineFollowPrepare();

  //delay(1000);

  while(flag==2){                //indicates line following

    //Serial.println("In mode 2");    //for debug use

   // update_keyboard();

    update_instr();
    if(digitalRead(MAIN_SWITCH_PIN)==LOW){
      return;
    }
    else if(flag!=2 || instr=='A' || instr=='C') {
      return;
    }
    lineFollow();
  }

}



//additional functionality: Remote control mode
//controlled by bluetooth
void mode3(){
  while(flag==3 ){
    Serial.println("In mode 3");
    read_instruction();
    //read_keyboard();
    if(digitalRead(MAIN_SWITCH_PIN)==LOW){
      return;
    }
    else if(flag!=3 || instr =='A' || instr =='F') {
      return;
    }
  }
}

/**
 * Choose the mode based on the signal it received
 * @param flag: the mode we want to choose
 * if it is '1', then change to mode1;
 * if it is '2', then change to mode2;
 * if it is '3', then change to mode3
 */
void chooseMode(int flag){
  switch(flag){
    case 1:mode1();
    break;
    case 2:mode2();
    break;
    case 3:mode3();
    break;
    default :
    mode1();
  }
}

char readSwitch(){
  if( digitalRead(SWITCH_PIN)==LOW){
    return '2';
   }
   else if (digitalRead(SWITCH_PIN)==HIGH) {
    return '1';
   }
}

void update_flag(char instr){
  if(instr=='A'){
        flag=1;
      }
  else if (instr=='F'){
        flag=2;
  }
  else if (instr=='C'){
        flag=3;
  }
}
```

```
/*
 * The function is to move robot forward at the given power
 * and the robot will not stop moving forward until it receives a new instruction,such as Stop()
 * @param power to control the speed
 * the higher the power is, the fast the robot moves
 * the power is range from [0,255]
 *
 *
 */
void moveForward(long power) {
  digitalWrite(M1, HIGH);
  digitalWrite(M2, HIGH);
  analogWrite(E1, power);
  analogWrite(E2, power);
}

void moveBackward(long power) {
  digitalWrite(M1, LOW);
  digitalWrite(M2, LOW);
  analogWrite(E1, power);
  analogWrite(E2, power);
}

/**
 * The function is to stop the robot immedietly
 */
void Stop() {
  digitalWrite(M1, HIGH);
  digitalWrite(M2, HIGH);
  analogWrite(E1, 0);
  analogWrite(E2, 0);
  newPower=normalPower;
}
```

```
/**
 * make the robot stop gradully based on the deceleraion it is given
 * @param deceler: the decelertion level
 * if it is small, it takes more time for robot to stop
 */
void decelerate(int DECELERATION) {
  for (int i = 255; i >= 0; i = i - DECELERATION) {
    digitalWrite(M1, HIGH);
    digitalWrite(M2, HIGH);
    analogWrite(E1, i);
    analogWrite(E2, i);
    delay(5);
  }
}

void turnRight(long Time) {
  Serial.println("turn Right");
  unsigned long oldtime;
  oldtime = millis();

  while (millis() - oldtime <= Time) {
    digitalWrite(M1, LOW);
    digitalWrite(M2, HIGH);
    analogWrite(E1, velocity);
    analogWrite(E2, velocity);
  }
}

void turnLeft(long Time) {
  Serial.println("turn left");
  unsigned long oldtime;
  oldtime = millis();

  while (millis() - oldtime <= Time) {
    digitalWrite(M1, HIGH);
    digitalWrite(M2, LOW);
    analogWrite(E1, velocity);
    analogWrite(E2, velocity);
  }
}

/**
 * scan the surrounding from 0 to 180 degrees
 * @param parts:divide 180 degrees into several parts
 *  the bigger the parts is, the sensor will stop more times to record values
 * @return the angle the robot need to turn
 *  the angle should between [0,180]
 */
 int scanAround(int parts){
  int vals[parts];
  int maxDegree=0;
  int maxAngle=0;
  servo.write(0); //turn the servo to 0 degree
  for(int i=0;i<=parts;i++){
    servo.write(i*180/parts);
    delay(500);
    vals[i]=readSonarAvg();
    Serial.print("Distance at:");   //for debug use
    Serial.println(i);
    Serial.println(vals[i]);

    if (vals[i]>=maxDegree){
      maxAngle=i;
      maxDegree=vals[i];
    }
  }
 servo.write(90);  //make the sensor facing forward again
 return maxAngle;
}
```

```
/**
 * rotate the robot based on the given degree
 * if 0<=degree<90, turn right
 * if 90<degree<=180, turn left
 * if degree=90, do nothing
 */
void rotate(int degree){
  if( degree==0){
    turnRight(250);
  }
  else if ( degree==2){
    turnLeft(250); //TODO Chanve the value to make it turn 90 degree
  }
}


 /**
  *   Function: Control the sonar to send a pulse, and measure the duration from the echo, 
calculate the distance as per temperature
  *  @return: a long that indicates the distance it gets, in cm, from 0 to 200, -1 if the range 
is not reasonable.
  */
long readSonar(){
    long duration, distance;
    digitalWrite(TRIG_PIN, HIGH);           //set trigger pin to HIGH
    delayMicroseconds(1000);
    digitalWrite(TRIG_PIN, LOW);            //set trigger pin to LOW
    duration = pulseIn(ECHO_PIN, HIGH);     //read echo pin


    long temp=readTmpLM();                 //read temperature
    long sound_speed=331.5 + (0.6 * temp);                    //calculate the sound speed at the 
point
    distance = (duration * sound_speed * 0.0001)/2;         //compute distance from duration of 
echo Pin
   // Serial.println(duration);
    Serial.println(distance);
   // delay(50); //Warning: make delay 50ms
    if (distance >= 200 || distance <= 0){   //deciding whether or not distance is reasonable
```

```
        return(-1);                         //if not, return -1

    }

    else{

        return(distance);

    }

  }


/**

 * Function: Read from analog pin the LM35 temperature sensor is connected to, and map the voltage to temperature in degree celsius

 * @Param: None

 * @Return: Float value indicates the temperature reading from LM35, nan if the sensor is not ready

*/

float readTmpLM(){

  float readVoltage = analogRead(LM_PIN);

  delay(10);

  readVoltage = analogRead(LM_PIN);         //Read twice and discard the first value so that the reading is not interfered by noise

  float tmp = map(readVoltage, 0, 225, 0, 100);

  temperature=tmp;


  return tmp;

}


int readSonarAvg(){

  int a=readSonar();

  delay(5);

  int b=readSonar();

  delay(5);

  int c=readSonar();

  delay(5);

  int d=readSonar();

  return (a+b+c+d)/4;

}
```

```
/*

 * Checks for Bluetooth if there is available data and if there is, update instruction to the new
instruction received from Bluetooth

 */

void update_instr() {

  if(BT.available()) {

     String data = BT.readString();


      if( data.charAt(0) >= (int)'0' && data.charAt(0) <= (int) '9' ){

        velocity = speed_map(data.charAt(0) - '0');

        //Serial.print("Changed speed to: ");

        //Serial.println(velocity);

        BT.print(convert_int_for_android(velocity));  //let the app know

      }else{

        instr = data.charAt(0);

        Serial.print("keyboard command received:    ");

      Serial.println(instr);

      }

    }

}


/*

 * When sending values to the app from Arduino, the android socket for some reason forgets the
first digit

 * if the number is 2 or more digits. So this function account for this

 */

int convert_int_for_android(int number) {

  if(number < 10) {

    return number;

  } else if(number <100) {

    return number+100;

  } else if(number < 1000) {

    return number+1000;

  }

  //Android app is programmed to only enter numbers between 0 and 255, so don't need to handle
bigger values
```

```
}



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

/*

 * This method is to use Bluetooth to control the car. We will receive the char type

 * input from android app via bluetooth, which we named it as "instruction". Then we use

 * several if statments to determine the command that we receive. 'R' means turn right. 'L'

 * means turn left. 'M' means move forward. And 'S' means sudden stop. 'D' means deacceleration.

 * Besides, we also have '1','2' and '3' represents 3 different models, Auto represents '1',

 * '2' means Tracing model, '3' means Manual mode. If we receive the signal except these, it will
return error

 */

void read_instruction(){

    char instruction;


    if( BT.available()){

      // if text arrived in from BT serial


      instruction = BT.read();

      Serial.println(instruction);


      if( instruction >= (int)'0' && instruction <= (int) '9' ) {

        velocity = speed_map(instruction - '0');

        BT.print(convert_int_for_android(velocity));  //let the app know

        //Serial.print("Updated velocity to: ");

        //Serial.println(velocity);

        return;

      }


      if(instruction == 'R' ){             //turn right instruction

        turnRight(20);                  // I think the time interval for this is quite long

        Serial.println("Turn Right");

      }else if(instruction == 'L' ){      //turn left instruction

        turnLeft(20);
        Serial.println("Turn Left");
      }else if(instruction == 'M' ){      //move forward instruction
        moveForward(velocity);
        Serial.println("Move Forward");
      }
      else if(instruction == 'B' ){      //move forward instruction
        moveBackward(velocity);
        Serial.println("Move Backward");
      }
      else if(instruction == 'D' ){      //move forward instruction
        decelerate(DECELERATION);
        Serial.println("Decelerate");
      }
      else if(instruction == 'S' ){      //sudden stop insturction
        Stop();
        Serial.println("Sudden Stop");
      }
      else if(instruction == 'A'){     //switch mode to automatical driving
         instr='A';
      }
      else if(instruction == 'F'){     //switch mode to automatical driving
         instr='F';
      }
      else if(instruction == 'C'){     //switch mode to automatical driving
         instr='C';
      }
      else{                            //instruction do not exist
         Serial.println("Wrong Instruction");   // print out wrong instruction
      }
    }
}

/*
 * This function maps 0-9, which are the speed levels received from the app via Bluetooth, to their corresponding speeds
 * We do this because reading a single digit from the Arduino via Bluetooth is faster than
receiving more than one digit numbers.

 */

int speed_map(int integer) {

  switch(integer){

    case 9: velocity = 255;

    break;

    case 8: velocity = 210;

    break;

    case 7:velocity = 180;

    break;

    case 6:velocity = 160;

    break;

    case 5:velocity = 140;

    break;

    case 4:velocity = 110;

    break;

    case 3:velocity = 80;

    break;

    case 2:velocity = 60;

    break;

    case 1:velocity = 35;

    break;

    case 0:velocity = 0;

  }

}


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

/*

 * THIS FUNCTION IS FOR TESTING USE

 * TO READ THE SERIAL INPUT FROM KEYBOARD

 * IT CAN BE DELETED AFTER THE TESTS

 */

void read_keyboard(){

    char instruction;
```

```
if( Serial.available()){
  // if text arrived in from serial

  String s= Serial.readString();
  instruction=s[0];

  Serial.println(instruction);
  if(instruction == 'R' || instruction == 'r'){            //turn right instruction
    turnRight(750);                  // I think the time interval for this is quite long
    Serial.println("Turn Right");
  }else if(instruction == 'L' || instruction == 'l'){      //turn left instruction
    turnLeft(750);
    Serial.println("Turn Left");
  }else if(instruction == 'M' || instruction == 'm'){      //move forward instruction
    moveForward(255);
    Serial.println("Move Forward");
  }
  else if(instruction == 'D' || instruction == 'd'){      //move forward instruction
    decelerate(DECELERATION);
    Serial.println("Decelerate");
  }
  else if(instruction == 'S' || instruction == 's'){      //sudden stop insturction
    Stop();
    Serial.println("Sudden Stop");
  }
  else if(instruction == '1'){     //switch mode to automatical driving
     instr='1';
  }
  else if(instruction == '2'){     //switch mode to automatical driving
     instr='2';
  }
  else if(instruction == '3'){     //switch mode to automatical driving
     instr='3';
  }
```

```
      else{                            //instruction do not exist

         Serial.println("Wrong Instruction");

      }

    }

}



//Additional moving functions for basic function 2

/* This function is used to make the car turn left (while moving forward) at a relatively small angle

 * parameter: int power, int Time

 * return: void

 */

void turnLeftForward_Slow(int power, long Time) {

  unsigned long oldtime;

  oldtime = millis();


  //Chnage the speed of the motor and keep it for Time (ms)

  while (millis() - oldtime <= Time) {

    digitalWrite(M1, HIGH);

    digitalWrite(M2, HIGH);

    analogWrite(E1, power);

    analogWrite(E2, power - TURNING_SPEED_DIFF);


    Serial.println("Left");


  //ScanGround to find the right track; if the robot deviates too much from the track, break the while loop

  scanGround();


  //If the right sensors(1, 2) start to give higher values, then it means the car is moving too left,

  //so turnleft function should break and stop the robot from moving left

    if( (ORsensorVal_1 + ORsensorVal_2)/2 >= ((ORsensorVal_3 + ORsensorVal_4)/2 ) &&

        !(ORsensorVal_1 < midVal - 50 &&  ORsensorVal_2 < midVal - 50 &&

          ORsensorVal_3 < midVal - 50  && ORsensorVal_4 < midVal - 50 ) ){
```

```
      break;

    }

  }

}

/*This function is used in function 2 to make the car turn left (while moving forward) at a relativel=y small angle*/

void turnLeftForward_Fast(int power, long Time) {

  unsigned long oldtime;

  oldtime = millis();


  //Chnage the speed of the motor and keep it for Time (ms)

  while (millis() - oldtime <= Time) {

    digitalWrite(M1, HIGH);

    digitalWrite(M2, HIGH);

    analogWrite(E1, power);

    analogWrite(E2, 0);


    Serial.println("FasterLeft");


  //ScanGround to find the right track; if the robot deviates too much from the track, break the while loop

  scanGround();


  //If the right sensors(1, 2) start to give higher values, then it means the car is moving too left,

  //so turnleft function should break and stop the robot from moving left

    if( (ORsensorVal_1 + ORsensorVal_2)/2 >= ((ORsensorVal_3 + ORsensorVal_4)/2 ) &&

        !(ORsensorVal_1 < midVal - 50 &&  ORsensorVal_2 < midVal - 50 &&

          ORsensorVal_3 < midVal - 50  && ORsensorVal_4 < midVal - 50 ) ){

      break;

    }

  }

}


/*This function is used in function 2 to make the car turn right (while mnoving forward) at a relatively small angle*/

void turnRightForward_Slow(int power, long Time) {
  unsigned long oldtime;

  oldtime = millis();


  //Change the speed of the motor and keep it for Time (ms)

  while (millis() - oldtime <= Time) {

    digitalWrite(M1, HIGH);

    digitalWrite(M2, HIGH);

    analogWrite(E1, power - TURNING_SPEED_DIFF);

    analogWrite(E2, power);


  //ScanGround to find the right track; if the robot deviates too much from the track, break the while loop

  scanGround();


  //If the left sensors(3, 4) start to give higher values, then it means the car is moving too right,

  //so turnright function should break and stop the robot from moving right

    if( (ORsensorVal_1 + ORsensorVal_2)/2 <= ((ORsensorVal_3 + ORsensorVal_4)/2) &&

         !(ORsensorVal_1 < midVal - 50 &&  ORsensorVal_2 < midVal - 50 &&

          ORsensorVal_3 < midVal - 50  && ORsensorVal_4 < midVal - 50 ) ){

      break;

    }

  }

}


/*This function is used in function 2 to make the car turn right (while moving forward) at a relatively big angle*/

void turnRightForward_Fast(int power, long Time) {

  unsigned long oldtime;

  oldtime = millis();


  //Change the speed of the motor and keep it for Time (ms)

  while (millis() - oldtime <= Time) {

    digitalWrite(M1, HIGH);

    digitalWrite(M2, HIGH);

    analogWrite(E1, 0);
    analogWrite(E2, power);


  //ScanGround to find the right track; if the robot deviates too much from the track, break the
while loop

  scanGround();


  //If the left sensors(3, 4) start to give higher values, then it means the car is moving too
right,

  //so turnright function should break and stop the robot from moving right

    if( (ORsensorVal_1 + ORsensorVal_2)/2 <= ((ORsensorVal_3 + ORsensorVal_4)/2) &&

         !(ORsensorVal_1 < midVal - 50 &&  ORsensorVal_2 < midVal - 50 &&

          ORsensorVal_3 < midVal - 50  && ORsensorVal_4 < midVal - 50 ) ){

      break;

    }

  }

}


/*This function is used in function 2 to make the robot turn left at a sharp corner*/

void sharpTurnLeft(int power, long Time) {

  unsigned long oldtime;

  oldtime = millis();


  //Change the speed of the motor and keep it for Time (ms)

  while (millis() - oldtime <= Time) {

    digitalWrite(M1, HIGH);

    digitalWrite(M2, LOW);

    analogWrite(E1, power);

    analogWrite(E2, power);


  //ScanGround to find the right track; if the robot deviates too much from the track, break the
while loop

  scanGround();


  //If the right sensors(1, 2) start to give higher values, then it means the car is moving too
left,

  //so turnright function should break and stop the robot from moving left

    if( (ORsensorVal_1 + ORsensorVal_2)/2 >= ((ORsensorVal_3 + ORsensorVal_4)/2) &&
         !(ORsensorVal_1 < midVal - 50 &&  ORsensorVal_2 < midVal - 50 &&
          ORsensorVal_3 < midVal - 50  && ORsensorVal_4 < midVal + 50 ) ){
      break;
    }
  }
}

/*This function is used in function 2 to make the robot turn right at a sharp corner*/
void sharpTurnRight(int power, long Time) {
  unsigned long oldtime;
  oldtime = millis();

  //Change the speed of the motor and keep it for Time (ms)
  while (millis() - oldtime <= Time) {
    digitalWrite(M1, LOW);
    digitalWrite(M2, HIGH);
    analogWrite(E1, power);
    analogWrite(E2, power);

  //ScanGround to find the right track; if the robot deviates too much from the track, break the
while loop
  scanGround();

  //If the left sensors(1, 2) start to give higher values, then it means the car is moving too
right,
  //so turnright function should break and stop the robot from moving right
    if( (ORsensorVal_1 + ORsensorVal_2)/2 >= ((ORsensorVal_3 + ORsensorVal_4)/2) &&
         !(ORsensorVal_1 < midVal - 50 &&  ORsensorVal_2 < midVal - 50 &&
          ORsensorVal_3 < midVal - 50  && ORsensorVal_4 < midVal - 50 ) ){
      break;
    }
  }
}

/*This function is used for basic function 2 to check if there is a breakpoint
 *on the track. If so, return true; if not, return false. If return true, the car
 *is supposed to continue moving forward.

 */

boolean checkBreakpoint(long Time) {

  unsigned long oldtime;

  oldtime = millis();


  while(millis() - oldtime < Time){

    moveForward(LINE_FOLLOW_SPEED);


    //ScanGround to find the right track; if the robot detects discontinuity on the track, break
the while loop

    //and return

    scanGround();


    if(!(ORsensorVal_1 < midVal - 50 &&  ORsensorVal_2 < midVal - 50 &&

     ORsensorVal_3 < midVal - 50  && ORsensorVal_4 < midVal - 50) ){

      return true;

    }

  }


  return false;

}




/*

 * THIS FUNCTION IS FOR TESTING USE

 * TO READ THE SERIAL INPUT FROM KEYBOARD

 * IT CAN BE DELETED AFTER THE TESTS

 */

void update_keyboard() {

     if(Serial.available()){

      String s= Serial.readString();

      instr=s[0];

      Serial.print("keyboard command received:    ");

      Serial.println(instr);
     }
}



/*
 * The function is to move robot forward at the given power
 * and the robot will not stop moving forward until it receives a new instruction,such as Stop()
 * @param power to control the speed
 * the higher the power is, the fast the robot moves
 * the power is range from [0,255]
 *
 *
 */
void adjustMovSpeed(long leftPower, long rightPower) {
  digitalWrite(M1, HIGH);
  digitalWrite(M2, HIGH);
  analogWrite(E1, leftPower);
  analogWrite(E2, rightPower);
}

//Used for hall effect sensor
void compareRpm(){

  if(speedLeft>speedRight){
    adjustMovSpeed(newPower-5,normalPower);
  }
  else if(speedLeft<speedRight){
    adjustMovSpeed(newPower+5,normalPower);
  }
}

char lineFollow(){
  int turningTime = 1000;

  //Scan the ground using the four optical reflective sensors to find the track.
  scanGround();


  //If all the values are lower than midVal, it means the robot is on the surface
  if(!(ORsensorVal_1 < midVal - 50 &&  ORsensorVal_2 < midVal - 50 &&
     ORsensorVal_3 < midVal - 50  && ORsensorVal_4 < midVal - 50 )){


    //If the difference between left and right side sensors are lower than 200, go forward
    if( ((ORsensorVal_1 + ORsensorVal_2)/2 <= ((ORsensorVal_3 + ORsensorVal_4)/2 + 200)) &&
        ((ORsensorVal_1 + ORsensorVal_2)/2 >= ((ORsensorVal_3 + ORsensorVal_4)/2 - 200)) ){
          moveForward(LINE_FOLLOW_SPEED);
    }else if(ORsensorVal_1 < midVal && ORsensorVal_2 < midVal &&
       ORsensorVal_3 < midVal && ORsensorVal_4 < midVal){
        moveForward(LINE_FOLLOW_SPEED);
    }


  //Sharp turn
    if(ORsensorVal_1 > midVal + 100 &&  ORsensorVal_2 > midVal + 100 && ORsensorVal_3 > midVal +
100 && ORsensorVal_4 < midVal - 100 ){
      sharpTurnRight(160, turningTime);
    }else if(ORsensorVal_1 < midVal - 100 && ORsensorVal_2 > midVal + 100 &&  ORsensorVal_3 >
midVal + 100 && ORsensorVal_4 > midVal + 100){
      sharpTurnLeft(160, turningTime);
    }


  //if right sensors(1, 2) read higher value, then the robot should turn right
    if( (ORsensorVal_1 + ORsensorVal_2/2) > ((ORsensorVal_3 + ORsensorVal_4)/2 + 500) ){
      Serial.println("In turningright");
      turnRightForward_Fast(TURNING_SPEED_MAX, turningTime);
    }else if( (ORsensorVal_1 + ORsensorVal_2)/2 > ((ORsensorVal_3 + ORsensorVal_4)/2 + 200) ){
      turnRightForward_Slow(TURNING_SPEED_NORM, turningTime);
    }


  //if left sensors(3, 4) read higher value, then the robot should turn left
    if( (ORsensorVal_1 + ORsensorVal_2)/2 < ((ORsensorVal_3 + ORsensorVal_4)/2 - 500) ){
      Serial.println("In turningleft");
      turnLeftForward_Fast(TURNING_SPEED_MAX, turningTime);
    }else if( (ORsensorVal_1 + ORsensorVal_2)/2 < ((ORsensorVal_3 + ORsensorVal_4)/2 - 200)){
      turnLeftForward_Slow(TURNING_SPEED_NORM, turningTime);
    }

  }else{
    Serial.println("In Stop");

    Stop();

  }
}

/*This function is used in function 2 to prepare and calibrate for line following function*/
void lineFollowPrepare(){
  int refVal = 0;
  int count = 0;
  int i = 0;

  //Calibrating the sensors, finding max and min reflectance values, and calculate the
  //average reflectance values among the surface and the line; the average value (midVal)
  //will be used to make the decision to stop the car later.
  //Use a nested for loop to get 10 sets of values to make the calibration more accurate.
  for(count = 0; count < 10; count++){
    for(i = 15; i < 20; i++){
      refVal = analogRead(i);
      if(refVal > maxSensorVal){
        maxSensorVal = refVal;
      }

      if(refVal < minSensorVal){
        minSensorVal = refVal;
      }

      delay(1);
    }
  }

  midVal = (maxSensorVal + minSensorVal)/2;
}

/* This function is used in function 2 to scan the ground and get the reading from the
 * 4 optical reflective sensors
 */
void scanGround(){
  ORsensorVal_1 = analogRead(ORsensorpin_1);
  ORsensorVal_2 = analogRead(ORsensorpin_2);
  ORsensorVal_3 = analogRead(ORsensorpin_3);
  ORsensorVal_4 = analogRead(ORsensorpin_4);
}
```

Appendix C - Fritzing

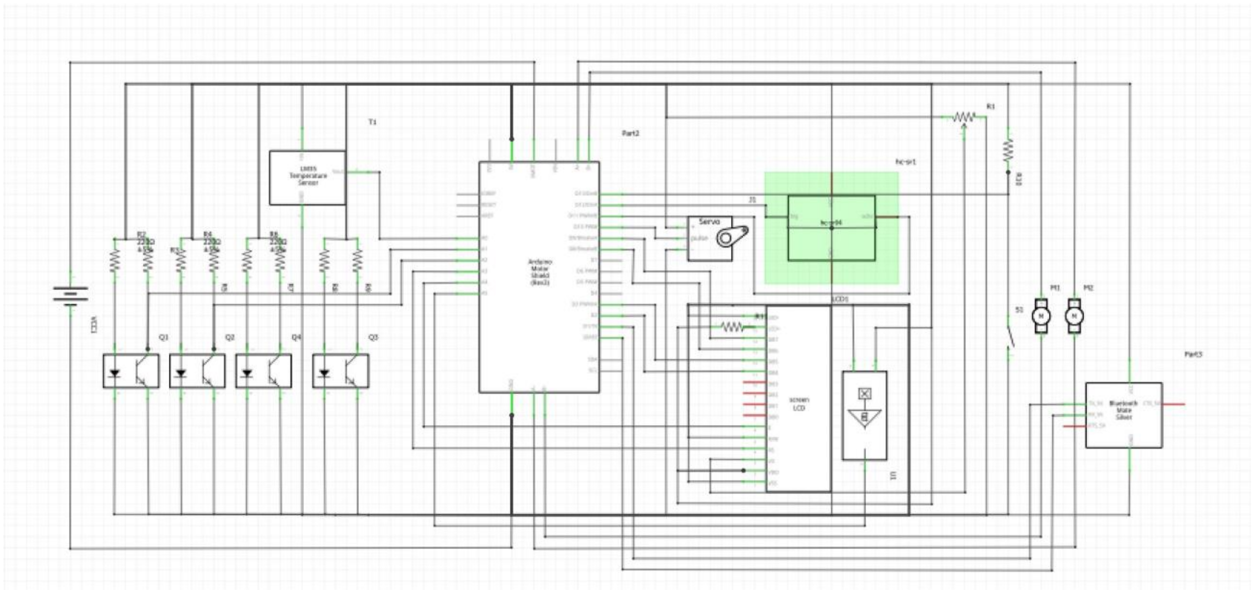


Figure 4. Schematic view of the circuit connection

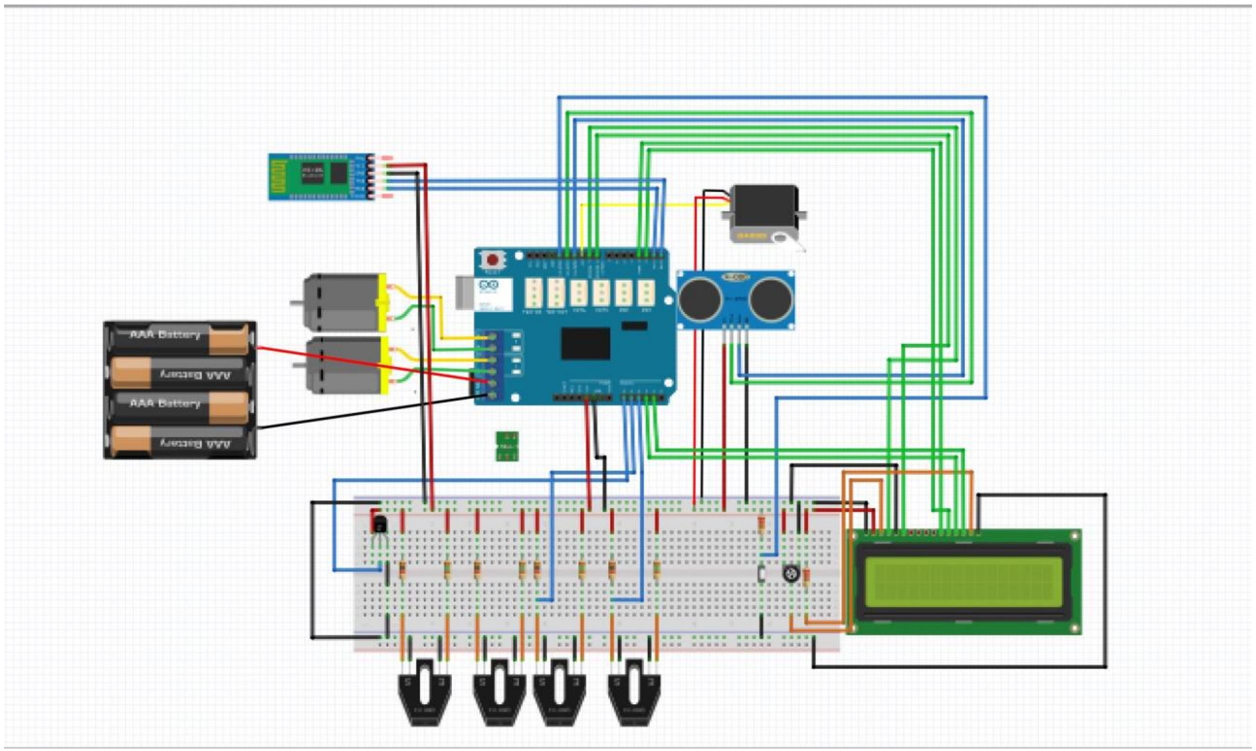


Figure 5. Breadboard view of the circuit connection

Appendix D - GitHub

- Most team members commit their codes and files to github, and we use git branch to separate people's work so that everyone can work on their own part without interfering with others' work.
- Yu is responsible for organizing the repositories and checking the codes before they can merge into master branch.
- We also used github to address issues and communicate with team members. We leave notes on README.md and comments on the codes to mention issues and errors, and notify the person to make some changes.

Appendix E – Component list

- HC-SR04: Ultrasonic Range Finder Sensor Module (Quantity: 1)
- Arduino UNO Rev3 Board(Quantity: 2)
- Power Bank 5V 1A 2500mAh(Quantity: 1)
- DFRobot Arduino Prototyping Shield (with small breadboard) (Quantity: 1)
- 1.5m USB Cable Type A to B(Quantity: 2)
- Standard LCD 16x2 + 10K POT + header(Quantity: 1)
- On-Off Power Button / Pushbutton Toggle Switch (Quantity: 1)
- Hall effect sensor - US5881LUA(Quantity: 2)
- Perma-Proto Half-sized Breadboard PCB [perf-board prototyping board](Quantity: 1)
- 2WD Mobile Platform (Turtle robot platform)(Quantity: 1)
    - geared DC motors(Quantity: 2)
    - battery holder(Quantity: 1)
    - wheel(Quantity: 2)
    - ball caster(Quantity: 1)
    - toggle switch(Quantity: 1)
    - sensor mounts(Quantity: 3)
    - chassis/frame(Quantity: 1)
    - Batteries(Quantity: 5)
- 2A Motor Shield for Arduino(Quantity: 1)
- Micro servo motor(Quantity: 1)
- Raspberry Pi 3 with 16G MicroSD(Quantity: 1)
- Round Magnets(Quantity: 10)
- Resistors(Quantity: 9)
    - 150 ohms ¼ watt 5%(Quantity: 4)
    - 1k ohms ¼ watt 5%(Quantity: 5)
- TRIMMER 10K OHM 0.5W TH [variable resistor/POT](Quantity: 1)
- LM-35 Temperature sensor(Quantity: 1)
- HC-05/HC-06 Bluetooth module(Quantity: 1)

Appendix F - Other

This is the solidworks drawing of an ultrasonic sensor holder. We 3-D printed this component and mounted it to the servo motor, at the front of our robot.

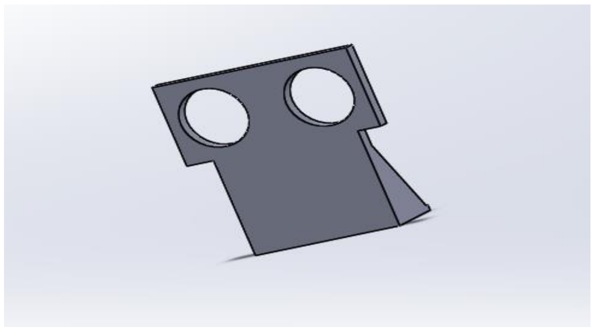


Figure 6. Solidworks Drawing of the ultrasonic sensor holder