ROB 6103：ADVANCED MECHATRONICS
Fall 2021

Project 1
Autonomous COVID Testing Kit Delivery Robot

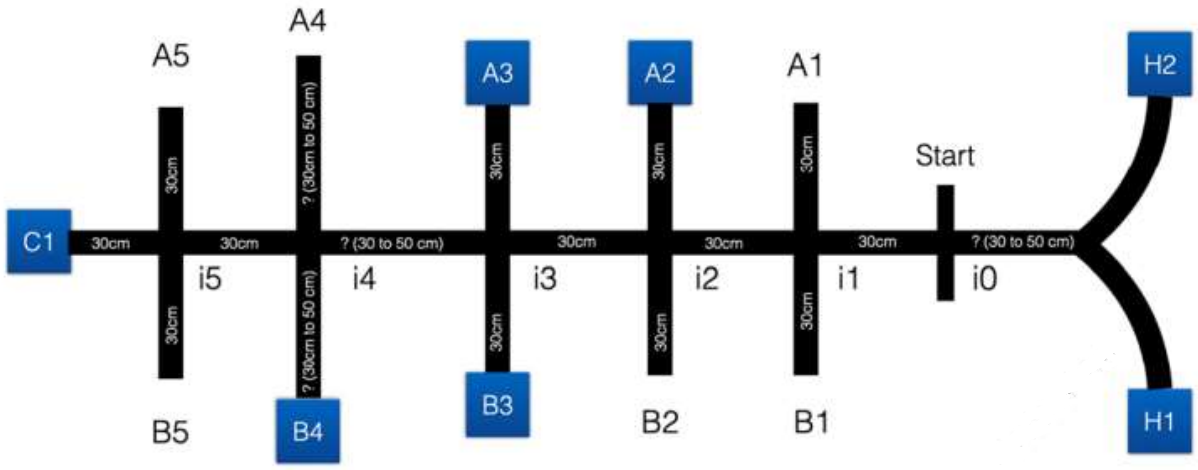
DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING
NEW YORK UNIVERSITY

Student Names: Xinzhou Jiang (xj2106)
Penmetsa Tejaswi Raju (pr2258)
Rishi Eswar Varma Vegesna (rv2210)

Instructor: Prof. Vikram Kapila

Table of Contents

OBJECTIVES



COVID pandemic has disrupted normal life. Given these unusual circumstances, there is a need for a contact less delivery system that can allow a COVID test kit provider to drop off the test kits at desired locations in a parking lot. You're tasked with the responsibility of designing a robot that will be able to perform a set of tasks to accomplish our goals.

Tasks:

1. The Robot should start from either H1 or H2
2. The Robot must follow the path indicated by black tape
3. The Robot must reach the Start position I0 and not look for objects around, this is the start Position.
4. Then it should continue following the black Tape and should detect an intersection.
5. When it reaches an intersection, it must STOP and look for the objects.
6. If the Robot detects the Objects, it must turn towards them and approach them and should STOP 5 cm apart before the object.

ELECTRONIC DESIGN

Various electronic components are used to achieve the given objectives. List of Electronic components used are as follows:

1. Parallax continuous rotation servo motors
2. QTR- 8RC Reflectance Sensor Array
3. HC-SR04 Ultrasonic Sensors
4. Arduino Uno Board
5. Power Supply Module
6. LED

1. Parallax Continuous Rotation Servo Motors

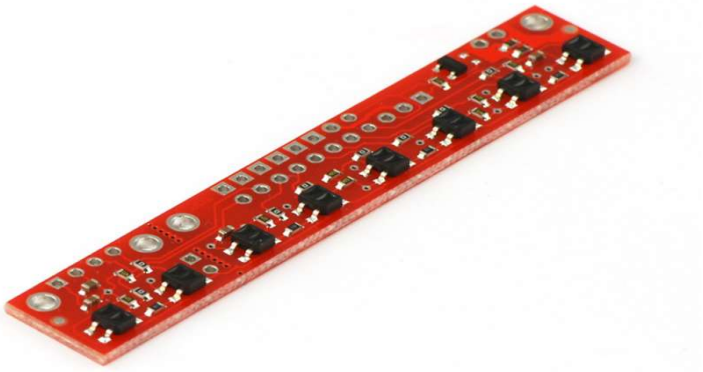
Continuous rotation servos are standard servos that have been modified to offer open-loop speed control instead of their usual closed-loop position control. The modification effectively turns them into 360-degree rotatable motors with integrated motor drivers in a compact, inexpensive package. A wheel can be added to the servo motor which can create a drive system for your robot that can be controlled using an RC signal or through a simple direct connection from a single microcontroller I/O line. The specifications of the Parallax Continuous Servo used are as follows:

a. 0 to 50 RPM with Pulse Width Modulation ramp up option
b. Weight of each servo is 42.5 grams
c. Power Requirements: 4 to 6 VDC and 15-200 mA according to the load the motors are on

2. QTR-8RC Reflectance Sensor Array

This sensor module has 8 IR LED/phototransistor pairs mounted on a 0.375" pitch, making it a great detector for a line-following robot. Pairs of LEDs are arranged in series to halve current consumption, and a MOSFET allows the LEDs to be turned off for additional sensing or power-savings options. Each sensor provides a separate digital I/O-measurable output**.** You can then read the reflectance by withdrawing that externally applied voltage on the OUT pin and timing how long it takes the output voltage to decay due to the integrated phototransistor. Shorter decay time is an indication of greater reflection. The specification of this IR sensor array is as follows:

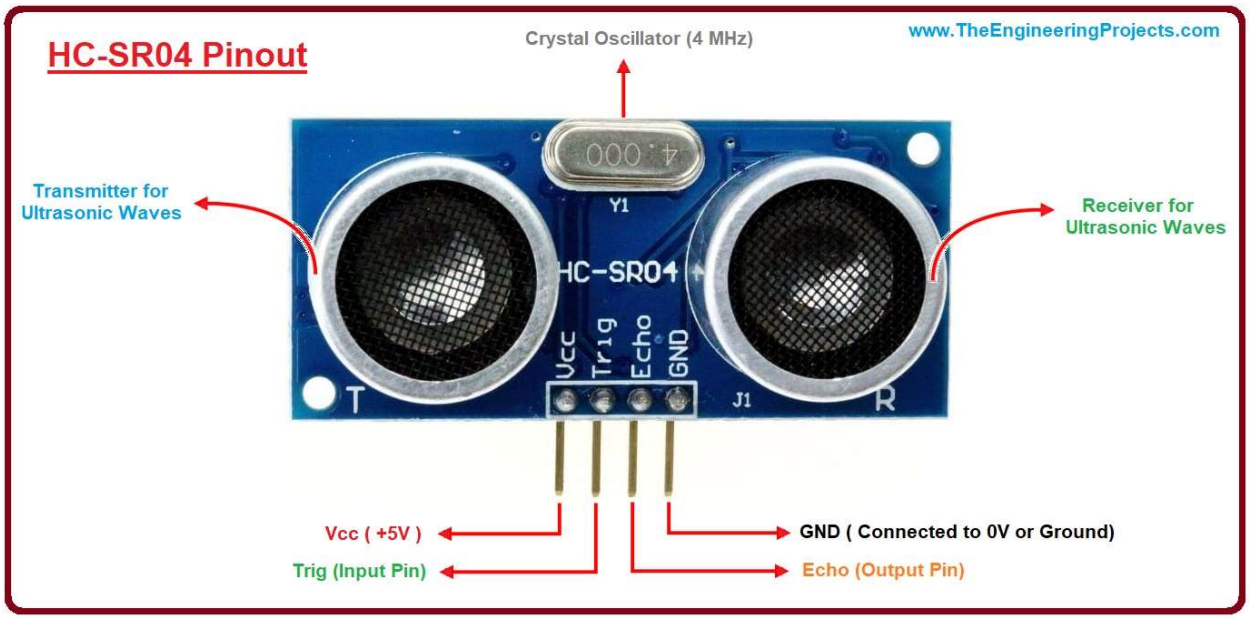
a. Operating Voltage: 3.3-5 V
b. Supply Voltage: 100 mA
c. Optimal Sensing Distance: 3mm



3. Ultrasonic Sensors

An ultrasonic sensor is an instrument that measures the distance to an object using ultrasonic sound waves. An ultrasonic sensor uses a transducer to send and receive ultrasonic pulses that relay back information about an object's proximity. For the project we are using the HC-SR04 ultrasonic sensor. It has two ultrasonic transducers. One of which acts as a transmitter which converts electrical signal into 40 KHz ultrasonic sound pulses. The receiver listens for the transmitted pulses that are reflected by the surface of object whose proximity is to be detected. After receiving the reflected pulses, it produces an output pulse whose Pulse out and pulse in time can be used to determine the proximity of the object. The specifications of the Ultrasonic Sensors used are as follows:

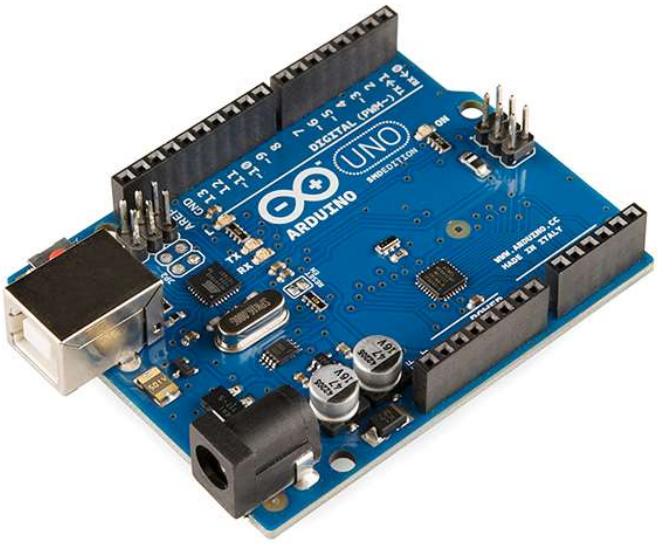
a. Operating Voltage: 5 VDC
b. Operating Current: 15 mA
c. Operating Frequency: 40 KHz
d. Max Range: 4 m
e. Minimum Range: 2 cm



4. Arduino UNO

Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button. The specification of an Arduino UNO Microcontroller board is:

a. Operating Voltage: 5 V
b. Input Voltage: 7-12 V
c. DC Current per I/O Pin: 20 mA
d. Microprocessor: ATMEGA 328P



5. Power Supply Module

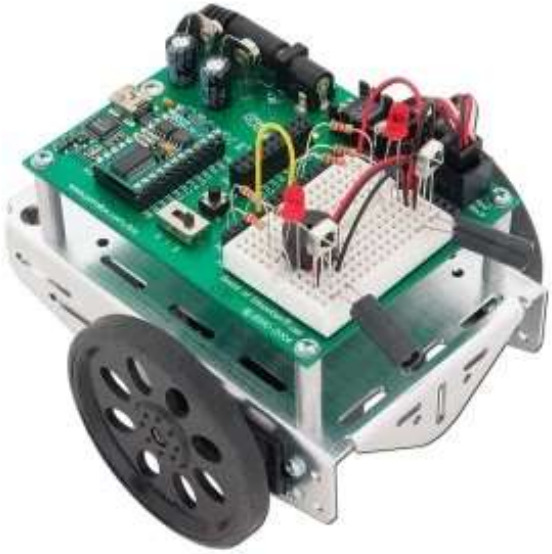
A Power Supply Module is a Voltage regulator is a system which is designed to maintain a certain volage rating be it a Step-Down Voltage or a Step Up. The voltage regulator we are using in our project is a 9V to 5V regulator that we are using to reduce the voltage supply from a 9 Volt battery to 5 V so as to supply the ultrasonic sensors with the required power that it needs. A voltage regulator is being used to power them as we found out that ultrasonic sensors were giving wrong readings when they were being powered simultaneously so we connected the servos and the ultrasonics differently just so that we are able to get consistent results.

6. Light Emitting Diode

A LED or light emitting diode is a semiconductor light source that emits light when current flows through it. We are using an LED as an indicator when we detect an intersection and also when we detect an object.

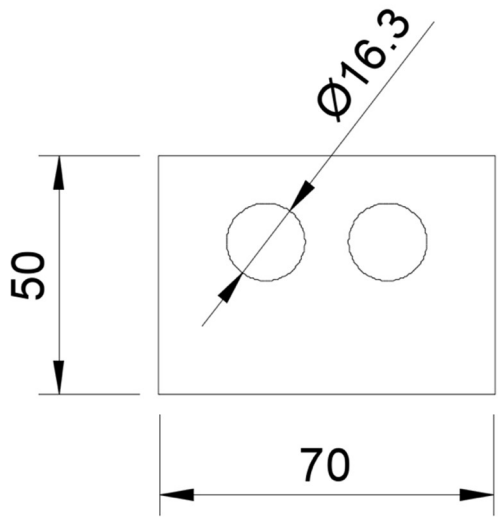
7. Boe-Bot Chassis

Boe-Bot Chassis is a aluminum made chassis that which is both durable and light weight. It also has numerous holes to fit in parts for both robot drive system and also the control systems for them. It also has slots to fit in the Continuous Servo motors which can be used for the motion of the robot.

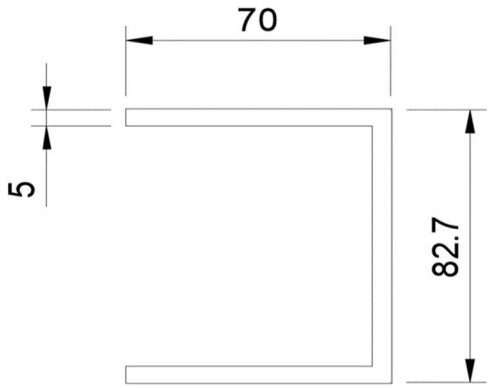


MECHANICAL DESIGN

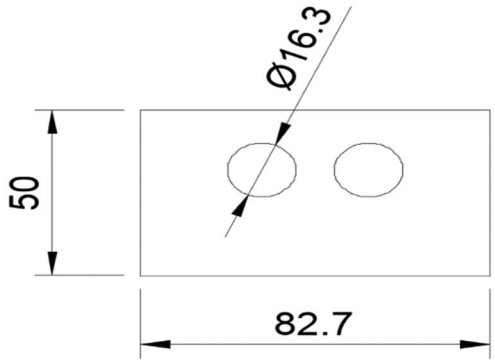
As the project uses 3 ultrasonic sensors to detect the presence and proximity of the object at the intersections a custom-built sensor mast was 3D printed to hold the ultrasonic sensors on the boe-bot chassis. The mast was printed in a way that it helps to holds the ultrasonic sensors at optimum height and angle to detect the 20 cm objects that are placed at the intersections. The CAD model design was designed in a way keeping in mind the limitations and specifications of the ultrasonic sensors. The CAD model diagrams of the Sensor mast are as follows:



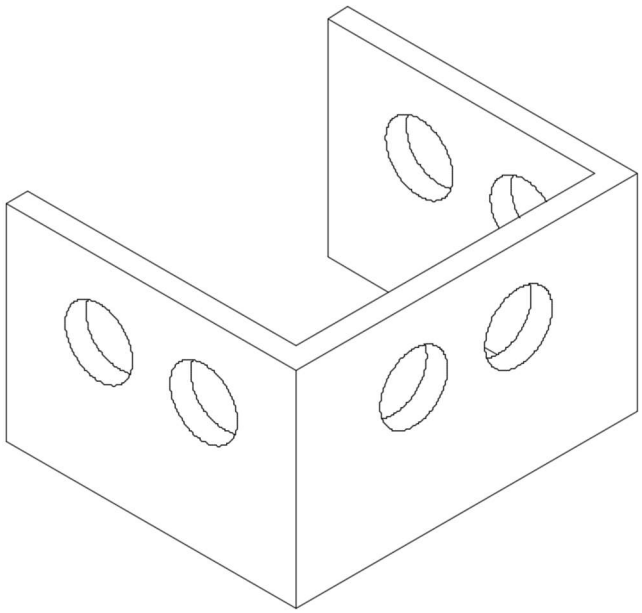
Front View



Top View



Side View



Isometric View

METHODOLOGY

The Robot consists of 2 major processes to achieve the given objectives. They are

Line Follower:

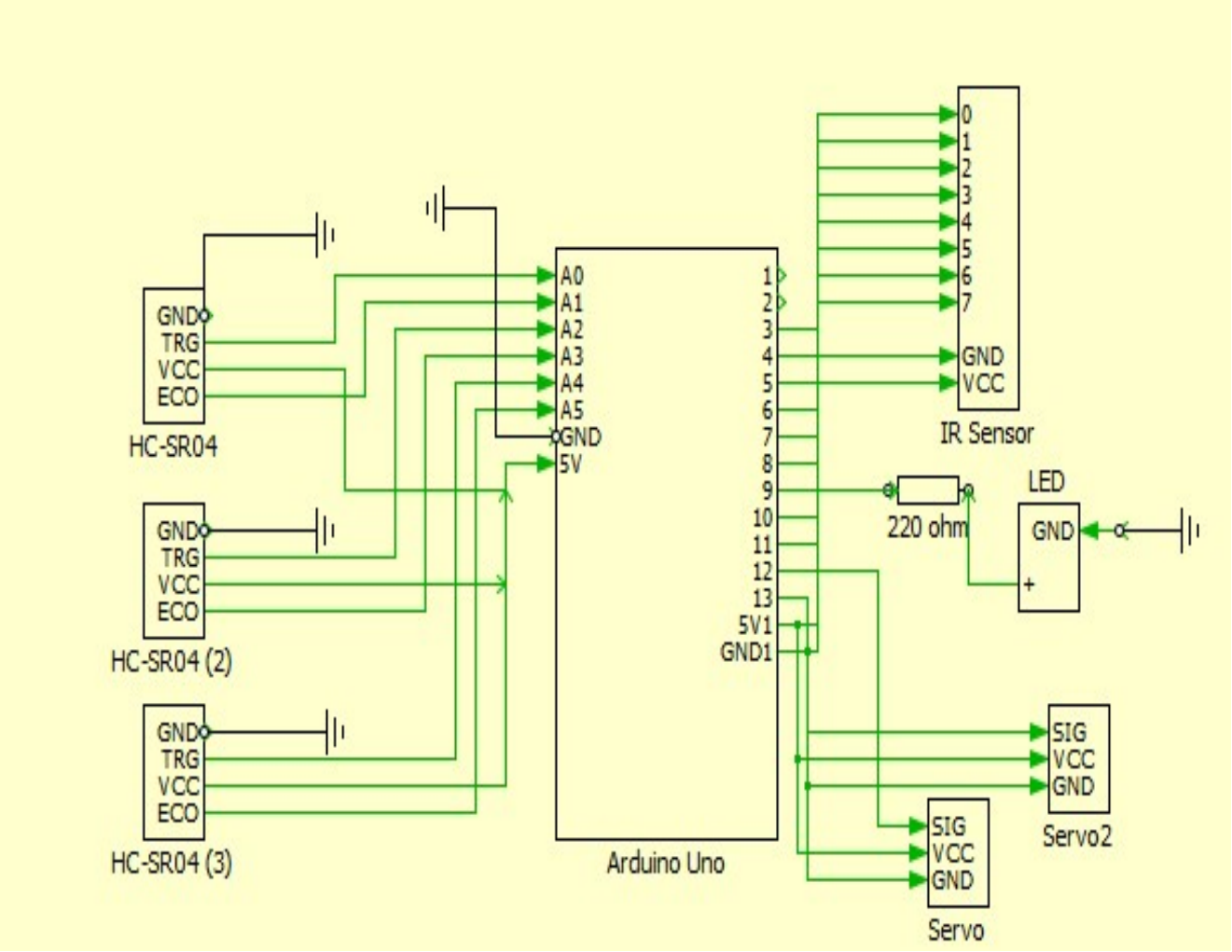
The QTC-8RC IR sensor array is used, the middle three IR sensors were used to detect whether the Robot is on the center of the path, the outer IR sensors sensor 1 and sensor 5 are used to detect whether car is or not in the middle of the path. The Robot used all of 8 sensors to determine whether it reached the intersection point or not. Firstly, the sensors are to be calibrated before every run just to make sure IR sensors detect the difference between dark and light colors, so that the robot moves flawlessly under every ambient conditions. When using the calibration method, the sensor values range from 0 – 1000. We gave a thresh-hold of 950 as we found out from testing the array on the track. So as to detect the intersection when all eight sensors have values above 950 then and only then do, we say that an intersection is detected. When the Boe-Bot will have a forward motion only when the Sensors 0 and 1 have values greater than 900 and Sensor 4 has a value of less than 800 that we have identified that the robot sees the black line on the left side and thus is making a motion towards the right. We achieve these small adjustments by giving pulses to both the right and the left side servo. For giving adjustment turns for the left side we do the same with the sensors, Sensor 4 and Sensor 5, when their values are greater than 900 and the Sensor value of Sensor 1 is less than 800. When it does so the bot sees the line on the right side and thus is making small adjustment turns towards the right. This is the gist of the method we are using to code the line follower part of this project.

Object Detection:

We have defined the right-side ultrasonic sensor to be Ultrasonic 1, the left side ultrasonic sensor to be Ultrasonic 2 and the forward ultrasonic to be Ultrasonic 3. When an intersection is detected apart from the first one i.e., the start line position, the ultra-sonic sensors are activated, and detects for the objects at intersections. A function in the code is used to detect the objects proximity ranging from 30 cm to 50 cm. 30 cm and 50 cm is the distance where the object will be placed according to the objective given. When the object is placed on the left side the robot turns left and starts following line that leads to the object placed at the end of the intersection where the object is detected. Then it is commanded to turn right until it detects the line again, then follow the same line until it detects the intersection when it detects the intersection then it takes a left turn again until it finds the line and then again starts following the line again until the next intersection. Then it takes the same steps similarly for all the conditions where the objects at the intersection might be. The list of states depending on objects at intersections are as follows:

1. Left object detected
2. Right Object Detected
3. Both Left and Right Object detected
4. Forward Object detected
5. Forward and left Object detected
6. Forward and Right Object detected
7. Forward, Left and Right Object Detected

CIRCUIT DIAGRAM



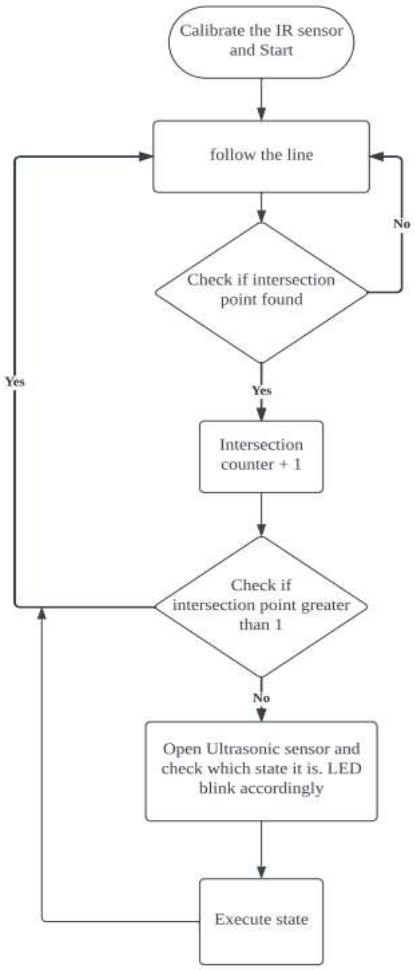
The electronic components are connected to one another in the most optimum way possible. The 8 pins of the 8RC Reflectance sensor are directly interfaced to the Arduino uno board. The Continuous Servo Motors power pins and ground pins are connected to Gnd and 5V pins on the Arduino board and signal pins of the servos are connected to Digital Pins 12 and 13. As the analog pins on Arduino uno board can also be used as digital pins the trigger and echo pins of the three ultrasonic sensors were connected to Analog Pins A0 to A5 and all the Vcc and Gnd pins of ultrasonic sensors are connected to respective pins on the Arduino board.

BLOCK DIAGRAM

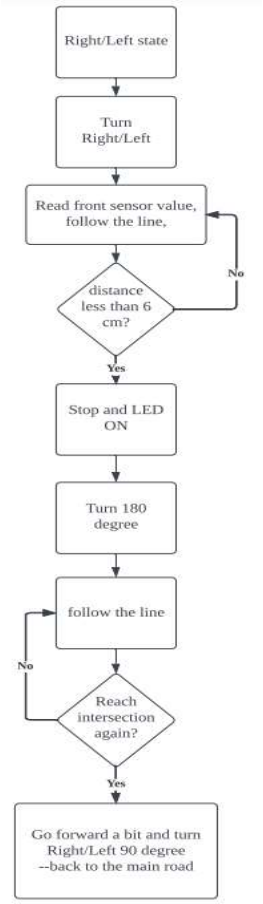
Various decision-making sub-programs were written in the code to the Robot to achieve the given tasks. The decisions making programs used are as follows:

1. Intersection detection
2. Ultrasonic object detection at intersection
3. Line Following program

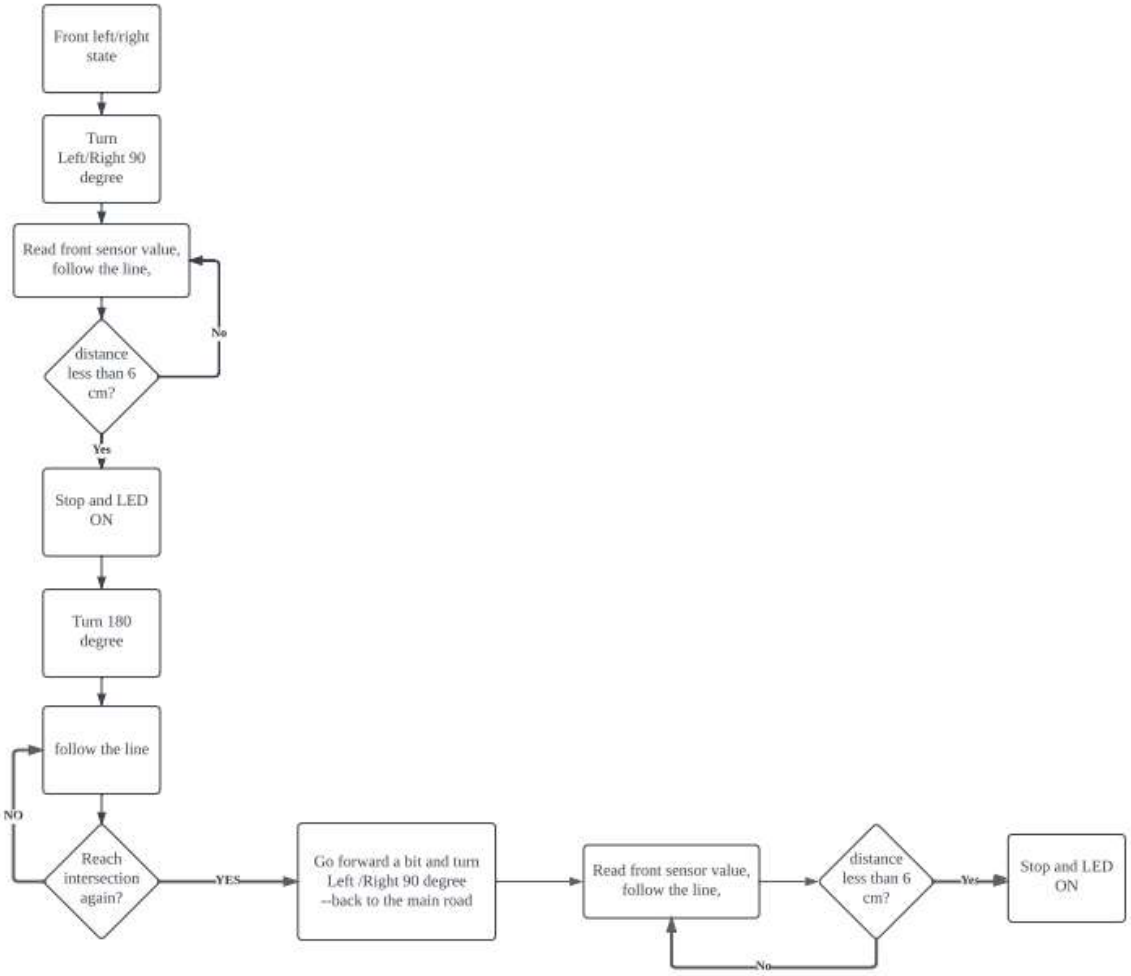
The flowchart depicting the decision-making process in our program are as follows:



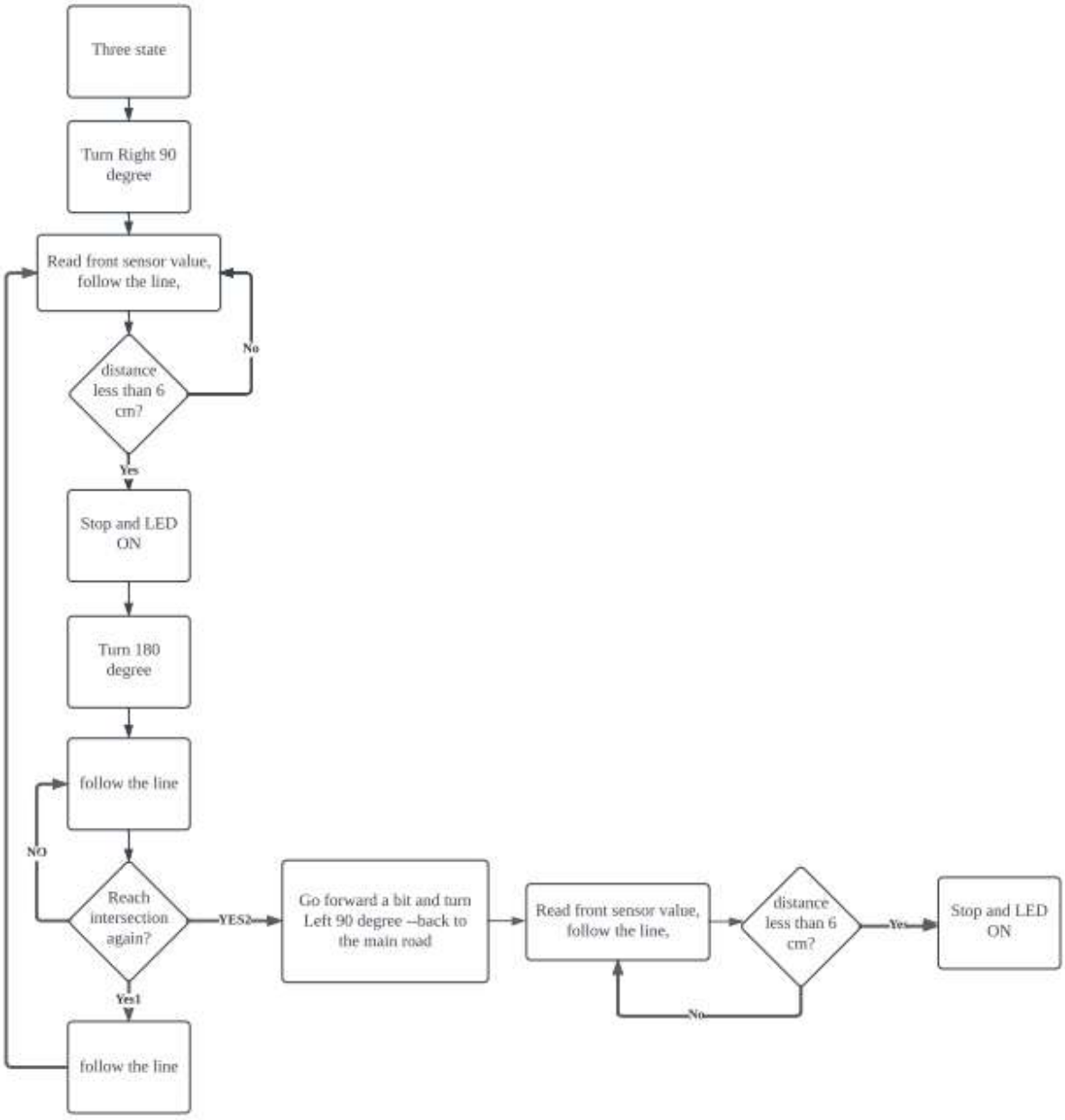
Intersection Detection Block Diagram



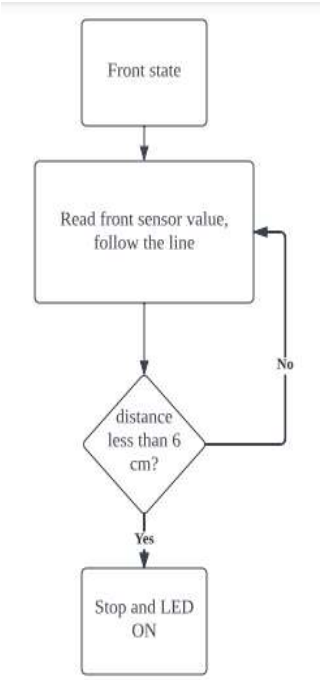
Ultrasonic Detection Block Diagram



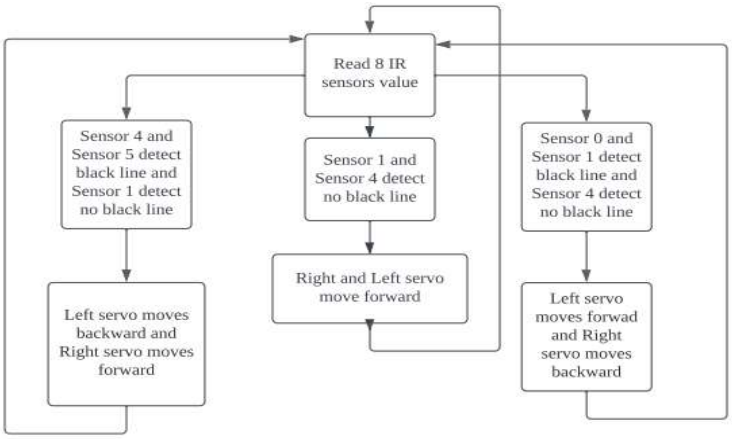
Object Detection Block Diagram (For Front and Left or Right)



Object Detection Block Diagram (For Front, Left, and Right)



Object Detection Block Diagram (Object in Front)



Line Following Block Diagram

DISCUSSION AND CONCLUSION:

The Robot achieves the given objective i.e., starting from home position and reaching the start position at which point it shouldn't look for any object at start point intersection. After the start position intersection, the robot should stop at every intersection and look for objects at the intersection. If the Robot find an object it should approach the object and stop at a distance and indicate to the user that it reached the object. This process is to be repeated at all the intersections. The Robot achieves all the given tasks and objectives. One of the prominent problems of the robot is it should be supplied with enough power source to run the high peripherals like servos. Powering the servo motors using Arduino uno board may affect the working of the Robot as motors use 6V DC at max speed no load conditions but Arduino can only supply a 5V output so the working of servos may be limited and if the peripherals are not properly connected without enough power, they may end up causing Arduino board to go into brown- out mode which may affect the functionality of the Robot. The Robot can be seen as a scaled down prototype of contactless delivery mobile Robot which has potential applications in the pandemic situation. Furthermore, it can also be used as a In-house delivery bot in large office spaces to replace the man force deployed to transport various documents in the workspace. Finally, by using a proper power source and calibrating the sensors used in the Robot properly, the Robot can efficiently and perform the given tasks in the known environments.

APPENDIX

Code:

The programming of the Arduino is done using Arduino IDE. Libraries like Servo and QTRSensors are used to interface the continuous servo motors and QTR-8RC Reflectance sensor array

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

// initialize servo motors
Servo LeftServo;
Servo RightServo;

// wheel speed constants
// can be adjusted for desired speed - about 90 is zero speed
int count = 0;
int count2 = 0;
int leftGo = 101;
int rightGo = 81;
int leftTurn = 93;
int rightTurn = 86;
int leftReverse = 86;
int rightReverse = 93;
int STOP1 = 93;
int STOP2 = 94;
int RturnL90 = 180;
int RturnR90 = 94;
int intcheckflag = 0;
int ultracheck1 = 0;
int ultracheck2 = 0;
int ultracheck3 = 0;
int time1;

//Ultrasonic sensor pin define
const int trigPin1 = A0;
const int echoPin1 = A1;
const int trigPin2 = A2;
const int echoPin2 = A3;
```

```
const int trigPin3 = A4;
const int echoPin3 = A5;

long duration1;
int distance1;
long duration2;
int distance2;
long duration3;
int distance3;
int ultrathred = 25;
int ultrathred1 = 35;
int ultrathred2 = 50;

#define NUM_SENSORS   8     // number of sensors used
#define TIMEOUT       2500  // waits for 2500 microseconds for sensor outputs to go low
#define EMITTER_PIN   2     // emitter is controlled by digital pin 2

// sensors 0 through 7 are connected to digital pins 3 through 10, respectively
QTRSensorsRC qtrrc((unsigned char[]) {3,4,5,6,7,8,10,11},
NUM_SENSORS, TIMEOUT, EMITTER_PIN);
unsigned int sensorValues[NUM_SENSORS];



#define TURN_RIGHT 0
#define TURN_LEFT 1
#define FORWARD 2
#define STOP 3
#define BOTHSIDE 4
#define THREESIDE 5
#define FRONTRIGHT 6
#define FRONTLEFT 7
#define FORWARD2 8

bool joint_flag;
unsigned int direction_flag = FORWARD;
unsigned int joint_num = 0;

void LEDON() {
   digitalWrite(9, HIGH);
   delay(500);
   digitalWrite(9, LOW);
}

void LEDblink() {
  if(count > 1){
```

```
   digitalWrite(9, HIGH);
   delay(200);
   digitalWrite(9, LOW);
   delay(200);
   digitalWrite(9, HIGH);
   delay(200);
   digitalWrite(9, LOW);
  }
}

void Stop(){
  LeftServo.write(90); // Tuned
  RightServo.write(89); // Tuned
  delay(1000);
  }

int joint_check(){//check if the interseciton point found
  if(count  < 1){
    if(sensorValues[0] > 950 && sensorValues[2] > 950 && sensorValues[3] > 950 &&
sensorValues[5] > 950 && sensorValues[6] > 950 && sensorValues[1] > 950 &&
sensorValues[4] > 950 && sensorValues[7] > 950)


  {
    count = count + 1;

    return true;
  }
  else{
    return false;
    }
  }

  if(sensorValues[0] > 950 && sensorValues[2] > 950 && sensorValues[3] > 950 &&
sensorValues[5] > 950 && sensorValues[6] > 950)
  {
    count = count + 1;
    if (count > 1){
    LEDON();
    }
    return true;
  }
  else{
    return false;
    }
```

```
  }
int direction_check(){// check which state is detected by the three sensors
  readdata();
  if((distance1 <= ultrathred1 && distance2 > ultrathred2 && distance3 > ultrathred1) ||
(distance1 <= ultrathred2 && distance2 > ultrathred1 && distance3 > ultrathred1))  //need to
tune
  {
    return TURN_RIGHT;
  }
  else if((distance2 <= ultrathred1 && distance1 > ultrathred2 && distance3 > ultrathred1) ||
(distance2 <= ultrathred2 && distance1 > ultrathred2 && distance3 > ultrathred1))  //need to
tune
  {
    return TURN_LEFT;
  }
  else if(distance3 <= ultrathred1 && distance1 > ultrathred1 && distance2 > ultrathred1) //need
to tune
  {
    return FORWARD2;
  }
    else if(distance3 <= ultrathred1 && distance1 <= ultrathred1 && distance2 <= ultrathred1)
//need to tune
  {
    return THREESIDE;
  }
      else if(distance3 > ultrathred1 && distance1 <= ultrathred2 && distance2 <= ultrathred2)
//need to tune
  {
    return BOTHSIDE;
  }
      else if(distance3 <= ultrathred1 && distance1 <= ultrathred1 && distance2 > ultrathred2)
//need to tune
  {
    return FRONTRIGHT;
  }
      else if(distance3 <= ultrathred1 && distance1 > ultrathred2 && distance2 <=  ultrathred1)
//need to tune
  {
    return FRONTLEFT;
  }

  return STOP;
  }
```

```
void turnright(){
  LeftServo.write(130);  // need to tune  // Tuned
  RightServo.write(130);   // need to tune  // Tuned
  delay(650);  //need to tune   // Tuned
}
void turnleft(){
  LeftServo.write(57);  // need to tune    // Tuned
  RightServo.write(57);   // need to tune   // Tuned
  delay(700); //need to tune   // Tuned
  }
void forward(){
  LeftServo.write(101);
  RightServo.write(81);
  delay(500);
}
void forward2(){
  LeftServo.write(101);
  RightServo.write(81);
  delay(1000);
}
void right_state(){ //
  bool joint_flag2 = false;
  turnright();
  delay(100);
  followline();
  delay(100);
  readdata();



  while(distance3 > 5){ //need to tune the value to see what is the number when it face the object
at 5-8cm
    readdata();
    followline();
  }

  Stop();
  LEDON();

  turnright();
  delay(100);
  turnright();

  while(joint_flag2 == false){
    joint_flag2 = joint_check();
    followline();
```

```
  }
  forward();
  turnright();
  time1 = millis();
  while(millis()-time1 < 500){ //go forward for 0.5s or sometime in case its not at the right path
    followline();
  }
  joint_flag = false;
  direction_flag = FORWARD;
}
void left_state2(){ //
  bool joint_flag2 = false;
  followline();
  delay(100);
  readdata();



  while(distance3 > 4){ //need to tune the value to see what is the number when it face the object
at 5-8cm
    readdata();
    followline();
  }

  Stop();
  LEDON();

  turnright();
  delay(100);
  turnright();

  while(joint_flag2 == false){
    joint_flag2 = joint_check();
    followline();
  }
  forward();
  turnleft();
  time1 = millis();
  while(millis()-time1 < 500){ //go forward for 0.5s or sometime in case its not at the right path
    followline();
  }
  joint_flag = false;
  direction_flag = FORWARD;
}

void left_state(){ //
```

```
  bool joint_flag2 = false;
  turnleft();
  delay(100);
  followline();
  delay(100);
  readdata();



  while(distance3 > 4){ //need to tune the value to see what is the number when it face the object
at 5-8cm
    readdata();
    followline();
  }

  Stop();
  LEDON();

  turnright();
  delay(100);
  turnright();

  while(joint_flag2 == false){
    joint_flag2 = joint_check();
    followline();
  }
  forward();
  turnleft();
  time1 = millis();
  while(millis()-time1 < 500){ //go forward for 0.5s or sometime in case its not at the right path
    followline();
  }
  joint_flag = false;
  direction_flag = FORWARD;
}

void bothside_state(){
  bool joint_flag2 = false;
  turnright();
  delay(100);
  followline();
  delay(100);
  readdata();

  while(distance3 > 5){ //need to tune the value to see what is the number when it face the object
at 5-8cm
```

```
   readdata();
   followline();
  }

  LEDON();
  Stop();

  turnright();
  delay(100);
  turnright();

  while(joint_flag2 == false){
   joint_flag2 = joint_check();
   followline();
  }

  left_state2();

  }

void forward_state(){
  unsigned int time1 = millis(); // cheat method, go forward for 0.5s
   followline();
  readdata();
  while(distance3 > 5){ //need to tune the value to see what is the number when it face the object
at 5-8cm
   readdata();
   followline();
  }
  LEDON();
  while(1){
   Stop(); //
  }
}
void followline(){
unsigned int position = qtrrc.readLine(sensorValues);
  // print the sensor values as numbers from 0 to 1000, where 0 means maximum reflectance and
  // 1000 means minimum reflectance, followed by the line position
  for (unsigned char i = 0; i < NUM_SENSORS; i++)
  {
   Serial.print(sensorValues[i]);
   Serial.print('\t');
  }
  //Serial.println(); // uncomment this line if you are using raw values
  Serial.println(position); // comment this line out if you are using raw values
```

```
  //delay(250);
 // read in sensor values from both photoresistors
 // need to test whether those sensor number is correct or not

  if (sensorValues[1] < 100 && sensorValues[4] < 100 ) {  // sees no line on either side, goes
straight until a line is seen
    LeftServo.write(leftGo);    // left wheel turns
    RightServo.write(rightGo);  // right wheel turns
  }
  else if (sensorValues[1] > 900 && sensorValues[0] > 900 && sensorValues[4] < 800) {  // sees
line on right side, is making a right turn
    LeftServo.write(leftTurn);      // left wheel turns
    RightServo.write(rightReverse); // right wheel reverses
  }
  else if (sensorValues[4] > 900 && sensorValues[5] > 900 && sensorValues[1] < 800) {  // sees
line on left side, is making a left turn
    LeftServo.write(leftReverse);  // left wheel reverses
    RightServo.write(rightTurn);   // right wheel turns
  }

  /*else if (sensorValues[1] > 950 && sensorValues[2] > 950 && sensorValues[3] > 950 &&
sensorValues[4] > 950) {
    //intcheckflag = intcheckflag++;
    LeftServo.write(STOP1);  // stop
    RightServo.write(STOP2);   // stop
    digitalWrite(9, HIGH);
    delay(500);
    digitalWrite(9, LOW);
    }*/

  else {  // both are seeing line, goes straight until light is seen
    LeftServo.write(leftTurn);    // left wheel turns
    RightServo.write(rightTurn);  // right wheel turns

  }


}

void readfront()
{ digitalWrite(trigPin3, LOW);
    delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
    digitalWrite(trigPin3, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin3, LOW);
```

```
  // Reads the echoPin, returns the sound wave travel time in microseconds
    duration3 = pulseIn(echoPin3, HIGH);
  // Calculating the distance
    distance3 = duration3 * 0.034 / 2;
  // Prints the distance on the Seria2 Monitor
    Serial.print("Distance3: ");
    Serial.println(distance3);
    }

void readdata()//
{     // read ultra 1
  // Clears the trigPin
    digitalWrite(trigPin1, LOW);
    delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
    digitalWrite(trigPin1, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin1, LOW);

  // Reads the echoPin, returns the sound wave travel time in microseconds
    duration1 = pulseIn(echoPin1, HIGH);
  // Calculating the distance
    distance1 = duration1 * 0.034 / 2;
  // Prints the distance on the Serial Monitor
    Serial.print("Distance1: ");
    Serial.println(distance1);

  //ultra 2
  // Clears the trigPin
    digitalWrite(trigPin2, LOW);
    delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
    digitalWrite(trigPin2, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin2, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
    duration2 = pulseIn(echoPin2, HIGH);
  // Calculating the distance
    distance2 = duration2 * 0.034 / 2;
  // Prints the distance on the Serial Monitor
    Serial.print("Distance2: ");
    Serial.println(distance2);

      //ultra 3
      // Clears the trigPin
    digitalWrite(trigPin3, LOW);
```

```
  delayMicroseconds(2);
 // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin3, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin3, LOW);
 // Reads the echoPin, returns the sound wave travel time in microseconds
  duration3 = pulseIn(echoPin3, HIGH);
 // Calculating the distance
  distance3 = duration3 * 0.034 / 2;
 // Prints the distance on the Seria2 Monitor
  Serial.print("Distance3: ");
  Serial.println(distance3);
  }

void setup() {
 delay(500);
 pinMode(trigPin1, OUTPUT);
 pinMode(echoPin1, INPUT);
 pinMode(trigPin2, OUTPUT);
 pinMode(echoPin2, INPUT);
 pinMode(trigPin3, OUTPUT);
 pinMode(echoPin3, INPUT);
 pinMode(9, OUTPUT);
 pinMode(13, OUTPUT);
 digitalWrite(13, HIGH);    // turn on Arduino's LED to indicate we are in calibration mode
 for (int i = 0; i < 400; i++)  // make the calibration take about 10 seconds
 {
  qtrrc.calibrate();       // reads all sensors 10 times at 2500 us per read (i.e. ~25 ms per call)
 }
 digitalWrite(13, LOW);     // turn off Arduino's LED to indicate we are through with calibration

 // print the calibration minimum values measured when emitters were on
 Serial.begin(9600);
 for (int i = 0; i < NUM_SENSORS; i++)
 {
  Serial.print(qtrrc.calibratedMinimumOn[i]);
  Serial.print(' ');
 }
 Serial.println();

 // print the calibration maximum values measured when emitters were on
 for (int i = 0; i < NUM_SENSORS; i++)
 {
////    Serial.print(qtrrc.calibratedMaximumOn[i]);
  Serial.print(' ');
 }
```

```
  Serial.println();
  Serial.println();
  delay(1000);

  LeftServo.attach(12);  // attach left servo to pin 6 on arduino
  RightServo.attach(13); // attach right servo to pin 9 on arduino
  digitalWrite(9, LOW);

}

void loop() {


  followline();

  joint_flag = joint_check(); //check if is intersection point

  if(joint_flag == true){
    count2 = count2 + 1;
    if (count2 > 1){
    Stop();
    direction_flag = direction_check(); //chech which state it is
    joint_num++;//add joint number, if it is 1 then no need to operate anything
    }

    if(joint_num <= 1){
      joint_flag = false;
    }

    if(joint_num > 1){//go to predesign path
      switch(direction_flag){
        case TURN_RIGHT:LEDblink(); forward(); right_state(); break;
        case TURN_LEFT: LEDblink(); forward(); left_state(); break;
        case FORWARD2: LEDblink();forward_state(); break;
        case BOTHSIDE: LEDblink();forward(); bothside_state(); break;
        case THREESIDE:LEDblink();forward(); bothside_state(); forward_state(); break;
        case FRONTRIGHT: LEDblink();forward(); right_state();forward_state();break;
        case FRONTLEFT: LEDblink();forward(); left_state();forward_state(); break;
      }
      }
    }
}
```