Arduino Covid kit Delivery System

Jagennath Hari
New York University
Tandon School of Engineering
New York City, USA
jh7454@nyu.edu

Sudharsan Ananth
New York University
Tandon School of Engineering
New York City, USA
sa6788@nyu.edu

Naveen Kumar
New York University
Tandon School of Engineering
New York City, US
nk3090@nyu.edu

***Abstract* - COVID-19 has disrupted the delivery of medical equipment (especially test kits) to people in need. Since hand-delivering test kits could cause the potential spread of the COVID-19 virus, there seems to be a need for an automated bot to deliver medical supplies to homes. We have been tasked to simulate a robot to perform automated delivery tasks in a given map and accomplish certain tasks. This report describes in detail the research and development that has been made during this project.**

I. INTRODUCTION

COVID-19 pandemic has disrupted all the services across the world. Several Doctors, Health care workers, and the entire medical industry have been struggling to reduce the spread of this virus. Since the start of the COVID-19 pandemic, there have been many developments in the way jobs are operated, medical equipment is transported, etc. But in order to reduce the spread and contain the virus required human element during the transportation and delivery to be absent, since humans are the host and transmitters of the virus.

In this paper, a detailed overview of the robot delivery system is provided along with hardware used, software architecture, electronics circuit, and testing of the robot systems. All the components used are carefully chosen to reduce the price of the robot and to make it accessible for anyone to build the project.

II. OBJECTIVES / TASKS

a. Line Following: The Robot should navigate through the given map using the black line. It will start from the home location (H1 or H2) and the robot should follow the line to get to the intersections and task locations. The width of the black line is 25mm.

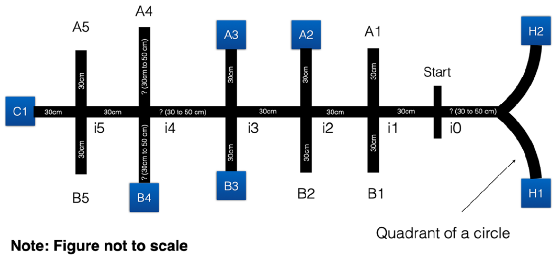


Fig 1: Diagram of the Map

b. Intersection Detection: The robot should be capable of detecting the intersections on the map. The intersections are named from i0 to i5 (i.e., A total of 6 intersections are located). The robot has to perform some actions at each intersection based on its environment.

c. Object detection: At each location, the robot should check for objects in either direction. If an object is detected at either end of the intersection. The robot should turn and navigate to the object and also blink an LED to indicate that it has accomplished its task.

d. Human Interface Device (HID): The robot should indicate its status several times. When a Robot reaches an intersection, detects an object, or reaches to the object, the robot should indicate its status with an LED.

III. COMPONENTS USED

In order to make it completely autonomous, it requires several electrical, electronic and mechanical components to work together. Various sensors were used to feed the data to the microcontroller that handles the logic and sends PWM signals to the actuators to control and maneuver the robot.

a. IR Sensors: Infrared sensors consist of an IR transmitter (LED) and an IR receiver. The IR light bounces if there is a white surface and outputs digital HIGH. When the IR sensor is placed on top of a black line the light coming from the transmitter is absorbed and the sensor outputs a LOW signal as the receiver is not able to detect the sufficient amount of the IR light reflected. The sensitivity of this sensor can be tuned by adjusting the potentiometer.
b. Ultrasonic Sensor: 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.
c. Continuous Servo Motor: A continuous rotation servo (sometimes called a full rotation servo) looks like a regular hobby servo. While a regular servo motor only turns over a narrow range, with precise control over position, a continuous rotation servo has a shaft that spins continuously, with control over its speed and direction. A regular servo can be converted to a continuous Servo motor by disconnecting the potentiometer from the shaft and cutting any material that limits the rotation. The control is performed using a pulse train signal, typically with pulses that vary from 1 to 2 milliseconds, sent every 20 milliseconds (50 Hz). A one-millisecond pulse corresponds to full speed in one direction, while a two-millisecond pulse is a full speed in the other direction. These pulses are easy to generate using the pulse-width-modulation hardware on a modest microcontroller.
d. Boe-Bot Chassis: Sturdy aluminum chassis is the central hardware for mounting all the electronic components and hardware. This chassis is Durable yet light-weight chassis and consists of numerous holes and slots for mounting robot drive and control systems. Rectangular cut-outs fit most of our Continuous Rotation Servos motors. It also consists of mounting options for four AA battery pack holders.
e. Arduino UNO: Arduino UNO is a microcontroller board based on the ATmega328P chip. 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. Arduino UNO is a low-cost, flexible, and easy-to-use programmable open-source microcontroller board that can be integrated into a variety of electronic projects. This board can be interfaced with other Arduino boards, Arduino shields, Raspberry Pi boards and can

control relays, LEDs, servos, and motors as an output.

f. Arduino Prototyping Shield: The Prototyping Shield is designed to facilitate prototyping - it allows for easy connections between a breadboard and an Arduino. Shields are designed to sit on top of an Arduino, extending the functionality of the popular microprocessor platform.

g. LEDs as HID: LEDs are here used as HID to show the presence of black lines intersection, detection of object etc.

IV. ELECTRICAL DESIGN

The Circuit has been put together using an Arduino prototype shield. This shield provides a breadboard to be mounted on top of the Arduino UNO. All the components VCC/+5v and GND wires are connected with VCC and GND of Arduino using the prototype board. The signal and PWM wires are connected to Arduino directly using jumper cables. The sensors, LEDs, and servo connections with Arduino UNO are given in the table below. This circuit can be easily modified to test and modify for prototyping purposes.

Component	**Input/output name**	**Arduino pin**
LED (Intersection)	3.3v	13
LED (Object detection)	3.3v	12
Led (target Approach)	3.3v	A1
Left Servo	Signal (PWM)	11
Right Servo	Signal(PWM)	10
Intersection IR	Left	4
	Right	5
Line following IR	Left	2
	Right	3
Front Ultrasonic sensor	Trigger	8
	Echo	9
Side Ultrasonic sensor	Trigger	6
	Echo	7

TABLE 1: Component connection to corresponding Arduino UNO pin

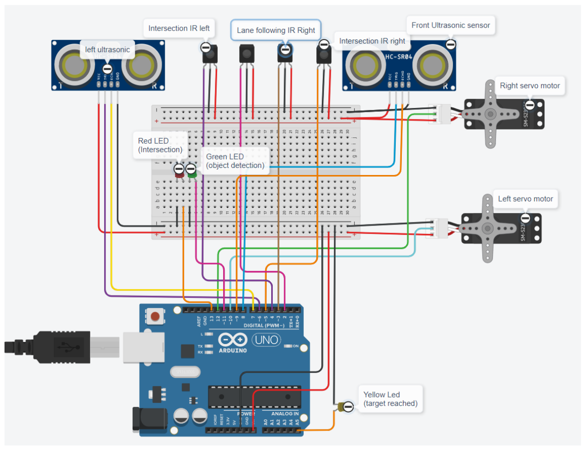
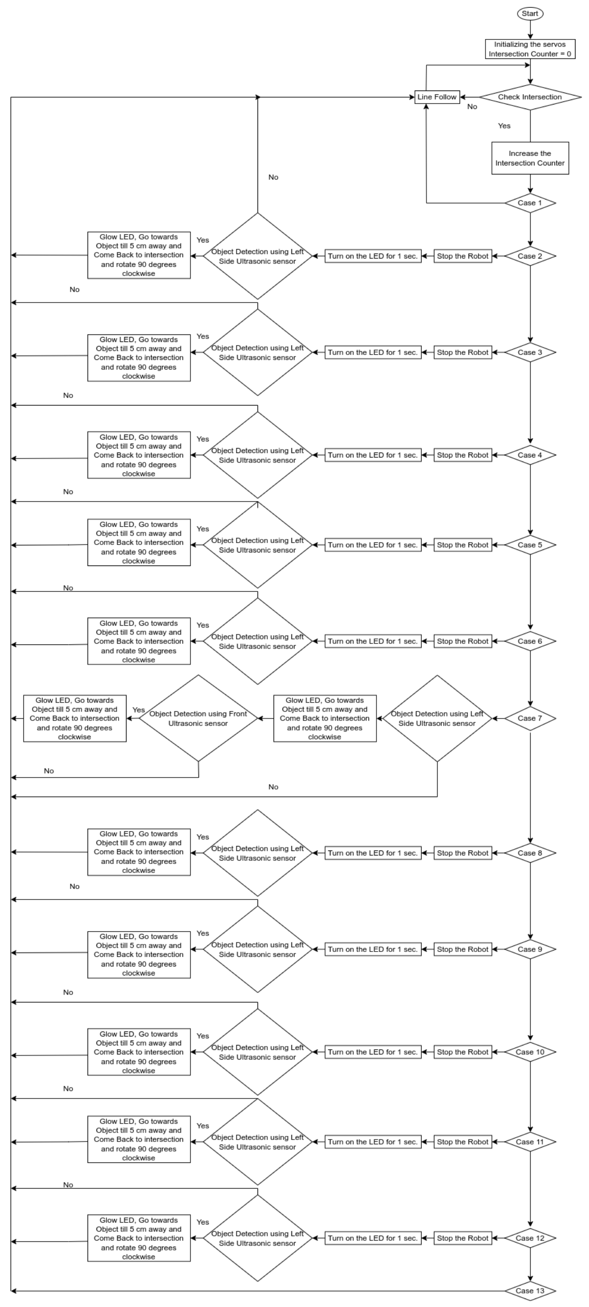


Fig 2: Circuit Diagram

V. SOFTWARE / PROGRAMMING

The code consists of 3 files and the program is mentioned in the Appendix. The first file is the header file ("AdvancedMech.h"), which initializes the class objects, functions, variables. The CPP file named "AdvancedMech.cpp" contains the class constructor with all the information and subsequent functions to complete the task. The ino file

(“AdvancedMech1.ino”) calls the class with a specified constructor parameter. And inside the setup is object initialization(attaching servos). Finally, inside the loop, it calls the object’s/class’s function which is programmed to complete the assigned task. Arduino IDE is used for programming the ino file. Whereas the cpp file that contains most of the code is done using a simple text editor.



Start
Initializing the servos
Intersection Counter = 0
Check Intersection
Line Follow
No
Yes
Increase the Intersection Counter
Case 1
Case 2
Stop the Robot
Turn on the LED for 1 sec.
Object Detection using Left Side Ultrasonic sensor
Glow LED, Go towards Object till 5 cm away and Come Back to intersection and rotate 90 degrees clockwise
Case 3
Case 4
Case 5
Case 6
Case 7
Object Detection using Front Ultrasonic sensor
Case 8
Case 9
Case 10
Case 11
Case 12
Case 13

Fig 3: Flow chart of Algorithm

VI. PROTOTYPING

The Circuits have been mounted on the Boe-bot chassis. The Arduino UNO with the prototyping shield is mounted on the upper surface of the Boe-Bot using an insulated foam that has glue on both sides. The Prototyping shield mounted on the Arduino provides the platform for rapid prototyping and testing different sensors. The Boe-Bot chassis is a sturdy aluminum chassis for various mounting holes for various sensors.

During the prototyping face, various sensors and methods have been tested. The initial prototype contained only 3 IR sensors, due to false sensing at the joint between 2 home positions this method was incapable of navigating. Further development showed that it is possible to use only IR sensors to navigate but due to time constraints and difficulty this prototype was dropped and a new prototype was designed.

The Final prototype contains 4 IR sensor, 2 of which is used for line following and 2 are used for intersection detection. Two ultrasonic sensors are used for the detection of objects in the sides and front. The robot will check for objects in the intersections only on the right side. After the last intersection, the robot will turn back and check for objects on the other side of the map.

VII. TESTING

During the prototyping phase the robot has been tested several times to check task completion. All the prototyping is done using the given map in maker space at NYU Tandon. During the testing of the robot, several hurdles have been encountered. All these obstacles were successfully solved with clever thinking and fast prototyping. One challenge we faced during the testing is the calibration of IR sensors changes every time a new battery is used this is due to a change in potential. This is solved by connecting the IR sensors to the 5v regulated pin on the Arduino. The final Prototype successfully executed all the tasks and completed the objectives.

VIII. RESULTS

The prototype is able to execute complex tasks and navigate the map. It also returns to the home location. Even though it is not a redundant system(fail-safe) and has few glitches at times. It performs all the actions with great speed and completes all the objectives. This prototype showcased the capabilities of Arduino and the ease of use of Arduino for quick prototyping.

IX. CONCLUSION

The project demonstrated a fully working prototype of a robot that can be used for the covid kit delivery system. This prototype is a conceptual model and does not take many factors into consideration. For a real working prototype of this project, We need to use advanced microcontroller and Single Board computers to understand and analyze the environment and navigate in the real world. In the future to make this project deployable in the real world, further improvements can be made. Such improvements can use computer vision and machine learning to observe the world with greater understanding and perform tasks in a changing environment.

X. REFERENCES

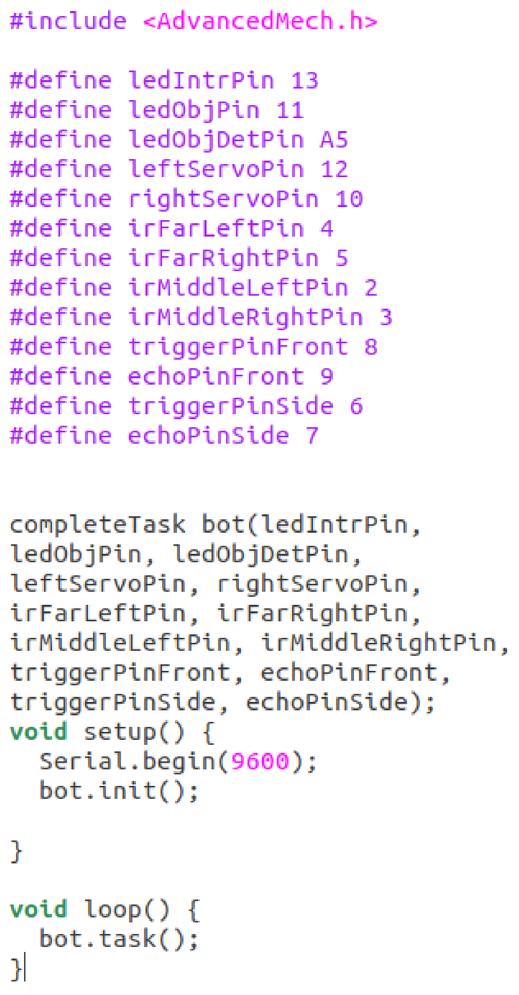
a. Arduino Documentation https://docs.arduino.cc/hardware/uno-rev3

b. Parallax Continous servo motor datasheet https://docs.rs-online.com/870b/0900766b8123f8a1.pdf

c. Ultrasonic Sensor datasheet https://web.eece.maine.edu/~zhu/book/lab/HC-SR04%20User%20Manual.pdf

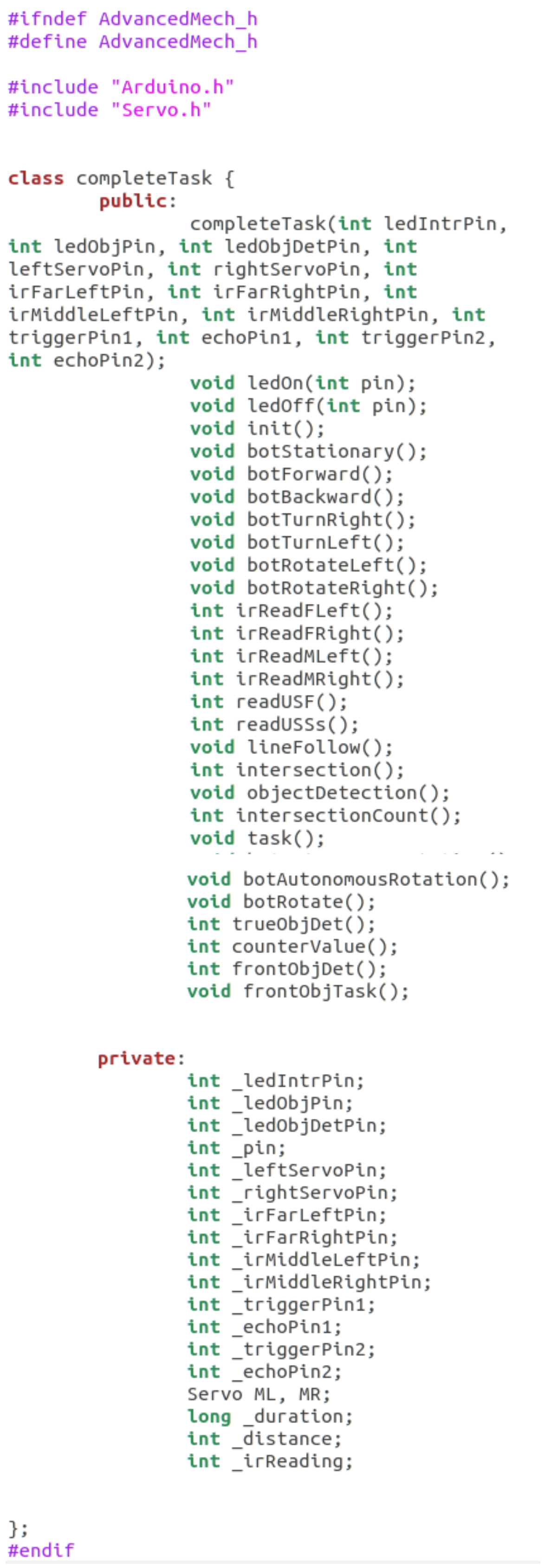
XI. APPENDIX ‘A’

“AdvancedMech1.ino” file



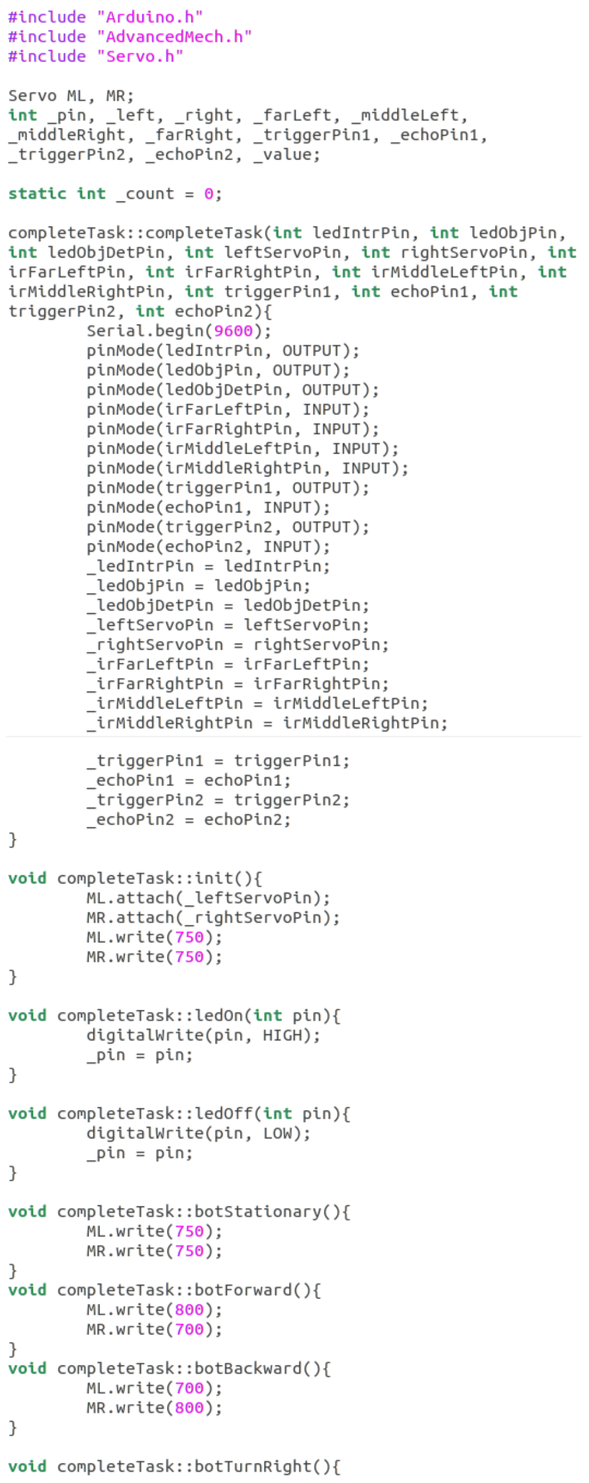
```
#include <AdvancedMech.h>

#define ledIntrPin 13
#define ledObjPin 11
#define ledObjDetPin A5
#define leftServoPin 12
#define rightServoPin 10
#define irFarLeftPin 4
#define irFarRightPin 5
#define irMiddleLeftPin 2
#define irMiddleRightPin 3
#define triggerPinFront 8
#define echoPinFront 9
#define triggerPinSide 6
#define echoPinSide 7


completeTask bot(ledIntrPin,
ledObjPin, ledObjDetPin,
leftServoPin, rightServoPin,
irFarLeftPin, irFarRightPin,
irMiddleLeftPin, irMiddleRightPin,
triggerPinFront, echoPinFront,
triggerPinSide, echoPinSide);
void setup() {
  Serial.begin(9600);
  bot.init();

}

void loop() {
  bot.task();
}
```

“AdvancedMech.h” file



```
#ifndef AdvancedMech_h
#define AdvancedMech_h

#include "Arduino.h"
#include "Servo.h"


class completeTask {
        public:
                completeTask(int ledIntrPin,
int ledObjPin, int ledObjDetPin, int
leftServoPin, int rightServoPin, int
irFarLeftPin, int irFarRightPin, int
irMiddleLeftPin, int irMiddleRightPin, int
triggerPin1, int echoPin1, int triggerPin2,
int echoPin2);
                void ledOn(int pin);
                void ledOff(int pin);
                void init();
                void botStationary();
                void botForward();
                void botBackward();
                void botTurnRight();
                void botTurnLeft();
                void botRotateLeft();
                void botRotateRight();
                int irReadFLeft();
                int irReadFRight();
                int irReadMLeft();
                int irReadMRight();
                int readUSF();
                int readUSSs();
                void lineFollow();
                int intersection();
                void objectDetection();
                int intersectionCount();
                void task();

                void botAutonomousRotation();
                void botRotate();
                int trueObjDet();
                int counterValue();
                int frontObjDet();
                void frontObjTask();


        private:
                int _ledIntrPin;
                int _ledObjPin;
                int _ledObjDetPin;
                int _pin;
                int _leftServoPin;
                int _rightServoPin;
                int _irFarLeftPin;
                int _irFarRightPin;
                int _irMiddleLeftPin;
                int _irMiddleRightPin;
                int _triggerPin1;
                int _echoPin1;
                int _triggerPin2;
                int _echoPin2;
                Servo ML, MR;
                long _duration;
                int _distance;
                int _irReading;


};
#endif
```

"ArduinoMech.cpp"



```
#include "Arduino.h"
#include "AdvancedMech.h"
#include "Servo.h"

Servo ML, MR;
int _pin, _left, _right, _farLeft, _middleLeft,
_middleRight, _farRight, _triggerPin1, _echoPin1,
_triggerPin2, _echoPin2, _value;

static int _count = 0;

completeTask::completeTask(int ledIntrPin, int ledObjPin,
int ledObjDetPin, int leftServoPin, int rightServoPin, int
irFarLeftPin, int irFarRightPin, int irMiddleLeftPin, int
irMiddleRightPin, int triggerPin1, int echoPin1, int
triggerPin2, int echoPin2){
        Serial.begin(9600);
        pinMode(ledIntrPin, OUTPUT);
        pinMode(ledObjPin, OUTPUT);
        pinMode(ledObjDetPin, OUTPUT);
        pinMode(irFarLeftPin, INPUT);
        pinMode(irFarRightPin, INPUT);
        pinMode(irMiddleLeftPin, INPUT);
        pinMode(irMiddleRightPin, INPUT);
        pinMode(triggerPin1, OUTPUT);
        pinMode(echoPin1, INPUT);
        pinMode(triggerPin2, OUTPUT);
        pinMode(echoPin2, INPUT);
        _ledIntrPin = ledIntrPin;
        _ledObjPin = ledObjPin;
        _ledObjDetPin = ledObjDetPin;
        _leftServoPin = leftServoPin;
        _rightServoPin = rightServoPin;
        _irFarLeftPin = irFarLeftPin;
        _irFarRightPin = irFarRightPin;
        _irMiddleLeftPin = irMiddleLeftPin;
        _irMiddleRightPin = irMiddleRightPin;

        _triggerPin1 = triggerPin1;
        _echoPin1 = echoPin1;
        _triggerPin2 = triggerPin2;
        _echoPin2 = echoPin2;
}

void completeTask::init(){
        ML.attach(_leftServoPin);
        MR.attach(_rightServoPin);
        ML.write(750);
        MR.write(750);
}

void completeTask::ledOn(int pin){
        digitalWrite(pin, HIGH);
        _pin = pin;
}

void completeTask::ledOff(int pin){
        digitalWrite(pin, LOW);
        _pin = pin;
}

void completeTask::botStationary(){
        ML.write(750);
        MR.write(750);
}
void completeTask::botForward(){
        ML.write(800);
        MR.write(700);
}
void completeTask::botBackward(){
        ML.write(700);
        MR.write(800);
}

void completeTask::botTurnRight(){
        ML.write(800);
        MR.write(750);
}

void completeTask::botTurnLeft(){
        ML.write(750);
        MR.write(700);
}

void completeTask::botRotateLeft(){
        ML.write(700);
        MR.write(700);
}

void completeTask::botRotateRight(){
        ML.write(800);
        MR.write(800);
}


int completeTask::irReadFLeft(){
        _irReading = digitalRead(_irFarLeftPin);
        return _irReading;
}

int completeTask::irReadFRight(){
        _irReading = digitalRead(_irFarRightPin);
        return _irReading;
}

int completeTask::irReadMLeft(){
        _irReading = digitalRead(_irMiddleLeftPin);
        return _irReading;
}

int completeTask::irReadMRight(){
        _irReading = digitalRead(_irMiddleRightPin);

        return _irReading;
}

int completeTask::readUSF(){
        digitalWrite(_triggerPin1, LOW);
        delayMicroseconds(2);
        digitalWrite(_triggerPin1, HIGH);
        delayMicroseconds(10);
        digitalWrite(_triggerPin1, LOW);
        _duration = pulseIn(_echoPin1, HIGH);
        _distance = _duration * 0.034 / 2;
        return _distance;
}

int completeTask::readUSSs(){
        digitalWrite(_triggerPin2, LOW);
        delayMicroseconds(2);
        digitalWrite(_triggerPin2, HIGH);
        delayMicroseconds(10);
        digitalWrite(_triggerPin2, LOW);
        _duration = pulseIn(_echoPin2, HIGH);
        _distance = _duration * 0.034 / 2;
        return _distance;
}

void completeTask::lineFollow(){
        if((irReadMLeft() == 1) && (irReadMRight() == 1)){
                botForward();
        }
        else if((irReadMLeft() == 1) && (irReadMRight()
== 0)){
                botTurnLeft();
        }
        else if((irReadMLeft() == 0) && (irReadMRight()
== 1)){
                botTurnRight();
        }
```

```
        else{
                botStationary();
        }
}

int completeTask::intersection(){
        if((irReadFLeft() == 1) && (irReadFRight() == 1)){
                ledOn(_ledIntrPin);

                return 1;
        }

        else {
                ledOff(_ledIntrPin);
                return 0;
        }
}




void completeTask::botAutonomousRotation(){
        while(irReadMRight() == 0){
                botRotateLeft();
        }

}


void completeTask::botRotate(){
        botBackward();
        delay(340);
        botStationary();
        delay(500);
        Serial.println("rotate");

        botRotateLeft();
        delay(600);
        botAutonomousRotation();
        botStationary();
        delay(100);
}


void completeTask::objectDetection(){
        if ( intersection() == 1 && readUSSs() >10 &&
readUSSs() <40 ){
                ledOn(_ledObjPin);
                delay(500);
                ledOff(_ledObjPin);
                botRotate();
                delay(500);
                while(readUSF()>3){
                        lineFollow();
                }
                botStationary();
                ledOn(_ledObjDetPin);
                delay(600);
                ledOff(_ledObjDetPin);
                botRotate();
                delay(1000);
                Serial.println("Test4");
                while(intersection() == 0){
                Serial.println(intersection());
                lineFollow();
                }
                botStationary();
                botRotate();
                delay(1500);
                delay(70);
                while(intersection()==0){
                        lineFollow();
        botRotateLeft();
        delay(600);
        botAutonomousRotation();
        botStationary();
        delay(100);
}


void completeTask::objectDetection(){
        if ( intersection() == 1 && readUSSs() >10 &&
readUSSs() <40 ){
                ledOn(_ledObjPin);
                delay(500);
                ledOff(_ledObjPin);
                botRotate();
                delay(500);
                while(readUSF()>3){
                        lineFollow();
                }
                botStationary();
                ledOn(_ledObjDetPin);
                delay(600);
                ledOff(_ledObjDetPin);
                botRotate();
                delay(1000);
                Serial.println("Test4");
                while(intersection() == 0){
                Serial.println(intersection());
                lineFollow();
                }
                botStationary();
                botRotate();
                delay(1500);
                delay(70);
                while(intersection()==0){
                        lineFollow();

                        delay(60);
                }

                botStationary();
                delay(2000);
                }
                else if(intersection() == 1 && readUSSs()
>40 ){
                        while(intersection() == 1){
                                lineFollow();
                                delay(30);
                        }
                        botStationary();
                        delay(5000);

                }


}

int completeTask::intersectionCount(){
        static boolean truth = false;
        if(intersection() == 1 && truth == false){
                _count++;
        }
        truth = intersection();
        return _count;
}

int completeTask::trueObjDet(){
        if (readUSSs() >= 10 && readUSSs() <= 24 ){
                return 1;
        }
        else return 0;

}
```

```
                        delay(60);
                }

                botStationary();
                delay(2000);
                }
                else if(intersection() == 1 && readUSSs()
>40 ){
                        while(intersection() == 1){
                                lineFollow();
                                delay(30);
                        }
                        botStationary();
                        delay(5000);

                }


}

int completeTask::intersectionCount(){
        static boolean truth = false;
        if(intersection() == 1 && truth == false){
                _count++;
        }
        truth = intersection();
        return _count;
}

int completeTask::trueObjDet(){
        if (readUSSs() >= 10 && readUSSs() <= 24 ){
                return 1;
        }
        else return 0;

}
```

```
void completeTask::task(){
        volatile int x;
        if(intersection() == 1){
                x = intersectionCount();
                switch(x){
                                case 1:
                                case 12:
                                        botStationary();
                                        delay(250);

while(intersection()==1){

lineFollow();
                                        }
                                        break;
                                case 2:
                                case 3:
                                case 4:
                                case 5:
                                case 7:
                                case 8:
                                case 9:
                                case 10:
                                case 11:
                                        botStationary();
                                        delay(250);
                                        objectDetection();
                                        break;
                                case 6:
                                        botStationary();
                                        delay(1000);
                                        objectDetection();
                                        frontObjTask();
                                        break;
                                default:
                                        botStationary();
                                        delay(1000);
while(intersection()==1){

lineFollow();
                                        }
                                        break;
                }
                Serial.println(intersectionCount());

        }
        else{
                lineFollow();
        }
}
```