Propeller 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 grid map similar to the streets of manhattan and accomplish certain tasks without breaking traffic rules. This project also brings some challenges such as avoiding obstacles and rerouting when necessary and also following one-way roads. This report describes in detail the research and development work done to accomplish this task.**

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 to reduce the spread and contain the virus requires, the human element during the transportation and delivery should be absent, since humans are the host and transmitters of the virus.

This paper provides a detailed overview of the robot delivery system, 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. Several sections are one way only indicated by the green arrow marks.

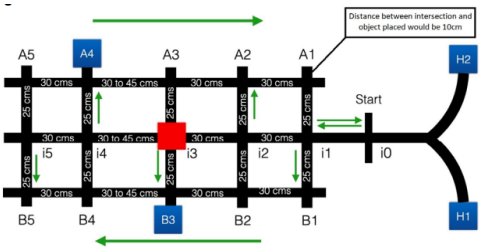

Fig 1: *Diagram of the Map*

b. Intersection Detection: The robot should be capable of detecting the intersections on the map. The intersections are named from i0 to i5, B1 to B5, and A1 to A5 (i.e., A total of 16 intersections are located). The robot has to perform some actions at each intersection based on its environment.

c. Obstacle Detection: An obstacle (shown in red color on the above map) will be located at the intersections i2,i3, or i5. The robot should be capable of

detecting, indicating the obstacle, and rerouting to get to the target locations to deliver the covid test kits.

d. 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.

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

All the actions should be seamless and the robot shouldn't stop at any intersections to ensure a smooth and fast delivery system.

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. Propeller Activity Board WX: This board is the brain of the robot. The Propeller Activity Board WX features the 8-core Propeller 1 microcontroller pre-wired to a host of popular peripherals for fast and fun experiments. The Propeller 1 microcontroller on this board makes complex projects easier to manage. Timing-sensitive processes like controlling motors, checking sensors, or playing audio can be handled by separate cores without interfering with each other. All the documentation and libraries are well written and available on the parallax website.

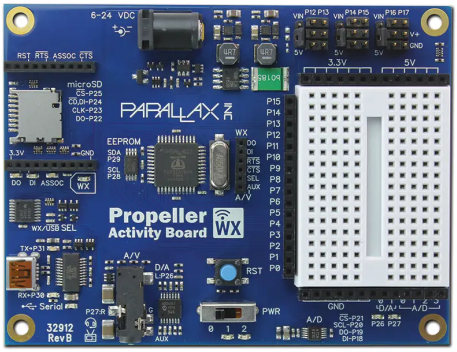

Fig 2: *Image of Propeller Activity Board WX*

b. 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.

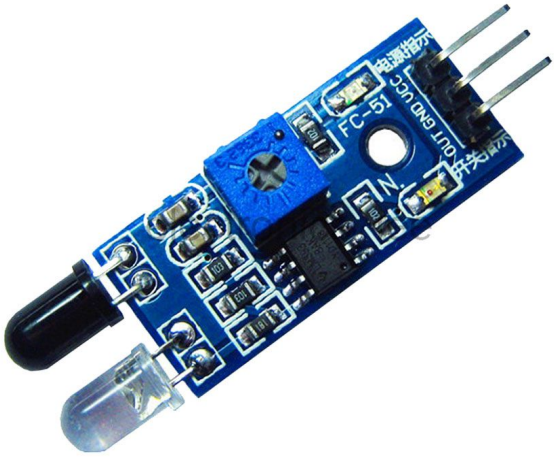

Fig 3: *Image of the IR lane following sensor*

c. 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. The ultrasonic sensor used in this project has 4 pins VCC, Trig, Echo, and GND. A pulse to the Trig pin sends an ultrasonic sound pulse and the Echo pin is the output that goes high when the sound is returned back. The time between the transmission and reception of the ultrasonic sound is used to calculate the distance between the sensor and the object.

Fig 4: *An image of Generic Ultrasonic Sensor - HC-SR04*

d. 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.

e. Boe-Bot Chassis: Sturdy aluminum chassis is the central hardware for mounting all the electronic components and hardware. This chassis is a Durable yet lightweight 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.

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

IV. ELECTRICAL DESIGN

The Activity Board WX has a breadboard for quick prototyping and wiring. It also has servo leads and various peripheral connections. These features make it easier to connect the sensors, continuous servo motors, and LEDs to the Activity Board. The circuit is connected using jumper cables and suitable resistors are connected between LEDs and the pinout of Activity Board WX.

Component	**Input/ output name**	**Activity Board WX pin**
Yellow LED (Intersection)	3.3v	9
Green LED (Object detection)	3.3v	10
Led (obstacle detection)	3.3v	11
Left Servo	Signal	12

	(PWM)	
Right Servo	Signal (PWM)	13
Intersection IR	Left	2
	Right	3
Line following IR	Left	1
	Right	0
Front Ultrasonic sensor	Trigger	14
	Echo	15
Side Ultrasonic sensor	Trigger	6
	Echo	7

TABLE 1: Component connection to corresponding Arduino UNO pin

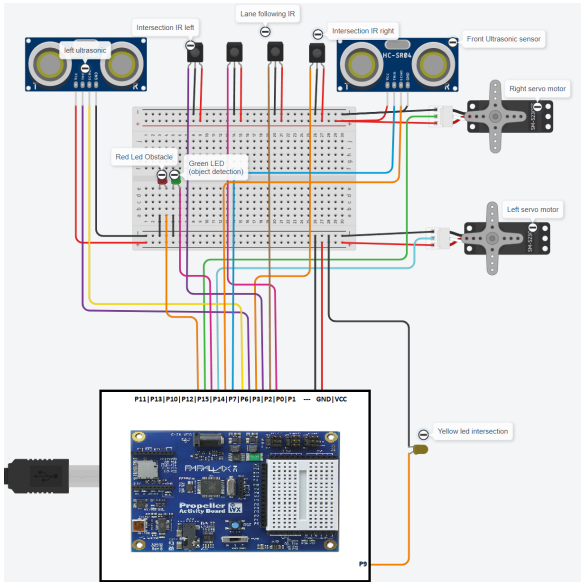

Fig 5: *Circuit Diagram*

V. SOFTWARE / PROGRAMMING

The entire code is written in C programming language. It is a single file consisting of all the functions and instructions to drive the robot. Out of the 8 cores, only 2 cores(cogs) are being used since it is sufficient for the given application. One cog(core) is used for path following, detecting the obstacle, and for traffic rules. The second cog(Core) is used for LEDs and shutting down the first cog at certain intersections where it should turn.

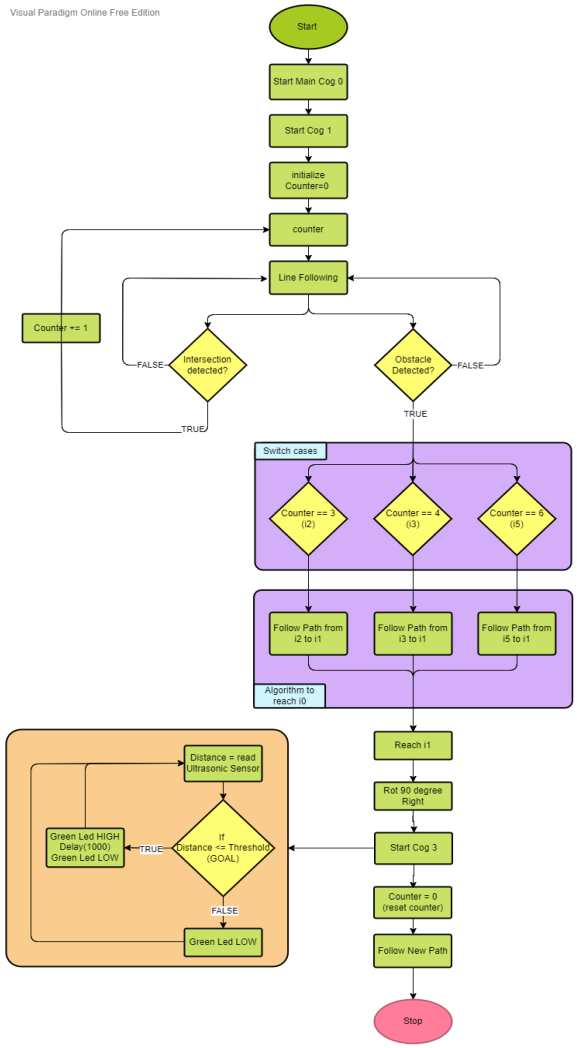

Fig 6: *Flow chart of Algorithm*

VI. PROTOTYPING

Propeller Activity Board WX is mounted on top of the Boe-bot chassis. Two IR sensors are placed at the front-center of the chassis for lane detection, in such a way that sensors will output HIGH when it's going out of the line. This way using only 2 IR sensors the lane following can be achieved. Two more IR is attached at the rear

of the Boe-bot chassis inner side of the frame for intersection detection. Most of the wiring is done using jumper caples since it offers flexibility for prototyping. The ultrasonic sensor is also mounted at the sides and front of the Boe-bot using soft double-sided tape to absorb vibrations.

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.

During the prototyping phase we first used 3 ultrasonic sensors, but later realized that only 2 sensors will be sufficient and opted for 2 ultrasonic sensor approaches to reduce complexity.

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 obstacles and objects on the sides and front. The robot first makes a trip to see where the obstacle is and decides which route to take. Then the robot will return to the i0 position and run a different route. Now while running the new route the robot will check for objects in the intersections only on the right side. The robot will stop once 2 targets or object has been found. Otherwise, it will proceed to intersection B5 and stop.

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 the 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 can execute complex tasks and navigate the map. 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 multicore capabilities of Propeller Activity Board WX and the ease of use of this board 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 consider many factors. For a real working prototype of this project, We need to use an 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. Propeller activity board WX Documentation
https://cdn-shop.adafruit.com/datasheets/32910-Propeller-Activity-Board-Guide-v1.0.pdf

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

d. IR Sensor module datasheet
https://www.rhydolabz.com/documents/26/IR_line_obstacle_detection.pdf

e. Propeller activity board resources
https://www.parallax.com/product/propeller-activity-board-wx/

f. “Implementation Of Autonomous Line Follower Robot”, Conference: ICIEV 2012, by *Mahmud Kazi, Abdullah Nahid, Abdullah Al Mamun*
https://www.researchgate.net/publication/260249901_Implementation_Of_Autonomous_Line_Follower_Robot

XI. APPENDIX ‘A’

```
// ALL INCLUDE STATEMENTS

#include "servodiffdrive.h"
#include "simpletools.h"

//-----------------------------------------------------------------------------------------------------------------

//ALL COGS

int *cog1; //cog 1 reference
int *cog2; //cog 2 reference
int *cog3; //cog 3 reference
int *cog4; //cog 4 reference
int *cog5; //cog 5 reference
int *cog6; //cog 6 reference
int *cog7; //cog 7 reference

//-----------------------------------------------------------------------------------------------------------------

//ALL STACKS

unsigned int stack1[40+25]; //stack1 value
unsigned int stack2[40+25]; //stack2 value
unsigned int stack3[40+25]; //stack3 value
unsigned int stack4[40+25]; //stack4 value
unsigned int stack5[40+25]; //stack5 value
unsigned int stack6[40+25]; //stack6 value
unsigned int stack7[40+25]; //stack7 value

//-----------------------------------------------------------------------------------------------------------------

// PINS
#define pinIrMiddleLeft 1 //ir middle left pin
#define pinIrMiddleRight 0 //ir middle right pin
#define pinIrLeft 2 //ir right pin
#define pinIrRight 3 //ir right pin
#define frontTrigPin 14 // trigger Pin of front ultrasonic sensor
#define frontEchoPin 15 // echo Pin of front ultrasonic sensor
#define leftServoPin 12 //left Servo Pin
#define rightServoPin 13 //right Servo Pin
#define pinIntersectionIndicator 9 //pin for led for intersection indication
#define sideTrigPin 6 //trigger Pin of side(left) ultrasonic sensor
#define sideEchoPin 7 //trigger Pin of side(left) ultrasonic sensor
#define ledPin 10 //led pin for object detectiions(goals)
#define object 11 //object detection led

//-----------------------------------------------------------------------------------------------------------------

//GLOBAL VARIABLES

volatile unsigned int counter = 0; // global counter variable
unsigned int locationOfJam; //global int storing location of traffic jam
unsigned int goalsDetected = 0; //global int for storing how many goal locations it has seen

//-----------------------------------------------------------------------------------------------------------------

// FORWARD DELARATIONS OF FUNCTIONS

void ledOn(const unsigned int pin); //forward declaration of function to turn on led
void ledOff(const unsigned int pin); //foward declaration of function to turn off led
int IR(const unsigned int pin); //forward declaration of function to get IR input
int distance(const unsigned int trigPin, const unsigned int echoPin); //forward declaration of function to find
distance from ultrasonic sensor
void servoInit(const unsigned int leftpin, const unsigned int rightpin); //forward declaration of function to attach
pins
void forward(); //forward declaration of move forward function
void backward(); //forward declaration of move backward function
void left(); //forward declaration of  move left function
void right(); //forward declaration of  move right function
void stop(); //forward declaration of  stationary function
```

```
void lineFollow(); //forward declaration of line following function
int intersection(); //forward declaration of detecting an intersection
int intersectionCount(); //forward declaration of intersection counter
int frontObj(); //forward declaration of function to see if there is a front object
void rotateLeft(); // forward declaration of function to rotate the bot left
void rotateRight(); //forward declaration of function to rotate the bot right
void detectJamAndReturn(); //forward declaration of function to find where Jam is and come back to start
void autoRotateLeft(); //forward declaration of function to rotate left till it finds black line
void autoRotateRight(); //forward declaration of function to rotate right till it finds black line
void followPath(); //forward declaration of function to follow path set which is got to i1-B1-B2-B3-B4-i4-A4-A3-A2-A1-i1-B1-B2-B3-B4-B5
void goalLocations(); //forward declaration of function to get how many goal locations it has seen so far
void goalLocationsTest(); //forward declaration of function to get how many goal locations it has seen so far

//-----------------------------------------------------------------------------------------------------------------

// MAIN LOOP


int main()
{
  servoInit(leftServoPin, rightServoPin); //initalizing servo
  cog1 = cogstart((void*)detectJamAndReturn, NULL, stack1, sizeof(stack1)); //starting cog1
}

//-----------------------------------------------------------------------------------------------------------------

// ALL FUNCTIONS

//-----------------------------------------------------------------------------------------------------------------
//FUNCTION TO TURN ON LED

void ledOn(const unsigned int pin) //function to turn on led
{
  high(pin); //high pin
}
//-----------------------------------------------------------------------------------------------------------------
//FUNCTION TO TURN OFF LED

void ledOff(const unsigned int pin) //function to turn off led
{
  low(pin); // low pin
}

//-----------------------------------------------------------------------------------------------------------------
//FUNCTION TO TAKE IR READINGS

int IR(const unsigned int pin) //function to get IR input
{
  volatile int i = input(pin); //volatile int storing input of pin
  //print("%d\n", i); //printing value for testing and calibration
  return i; //returns the IR sensor reading
}
//-----------------------------------------------------------------------------------------------------------------
//FUNCTION TO RECORD DISTANCE FRO ULTRASONIC SENSOR
int distance(const unsigned int trigPin, const unsigned int echoPin) //function to get distance from ultrasonic sensor
{
  low(trigPin); // setting trig Pin to low
  pulse_out(trigPin, 10); //pulse out of trigger Pin
  volatile long tEcho = pulse_in(echoPin, 1); //pulse in of echo Pin
  volatile int cmDist = tEcho / 58; //getting distance
  //print("%d cm\n", cmDist); //printing distance for testing
  return cmDist; //returning distance

}
//-----------------------------------------------------------------------------------------------------------------
//FUNCTION TO INTIALISE SERVO
```

```
void servoInit(const unsigned int leftpin, const unsigned int rightpin) //function to attach servos
{
  drive_pins(leftpin, rightpin); //attaching servos
  drive_stop(); //stopping the servos
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION TO DRIVE FORWARD
void forward() //drive forward
{
  drive_speeds(-200,-200); //full forward
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION TO DRIVE BACKWARD
void backward() //drive backward
{
  drive_speeds(200,200); //full backward
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION TO TILT LEFT
void left() //turn left
{
  drive_speeds(-200, 0); //keeping left wheel stationary
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION TO TILT RIGHT
void right() //turn right
{
  drive_speeds(0, -200); //keeping right wheel stationary
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION FOR INV LEFT
void invLeft() //inverse move left function
{
 drive_speeds(200, 0); //keeping left wheel stationary
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION FOR INV RIGHT
void invRight() //inverse move left function
{
 drive_speeds(0, 200); //keeping left wheel stationary
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION FOR STOPPING
void stop() //bot stationary
{
  drive_stop(); //stops the servos
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION TO ROTATE RIGHT
void rotateRight() //bot rotate right
{
  drive_speeds(200, -200); //rotating both wheels same speed in opposite direction to make it rotate right
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION TO ROATE LEFT
void rotateLeft() //bot rotate left
{
  drive_speeds(-200, 200); //rotating both wheels same speed in opposite direction to make it rotate left
}
//-------------------------------------------------------------------------------------------------------------------
//FUNCTION FOR AUTO ROTATION TO LEFT
void autoRotateLeft() //function to rotate left till it finds black line
```

```
{
  while(IR(pinIrMiddleRight) == 1) //while loop to get out of black line
  {
    rotateLeft(); //rotate left
  }
  stop(); //stop motors
  while(IR(pinIrMiddleRight) == 0) // while loop to get back into black line
  {
    rotateLeft(); //rotate left
  }
  stop(); //stop motors
}
//-------------------------------------------------------------------------------------------------------------------
------
//FUNCTION FOR AUTO ROTATION RIGHT
void autoRotateRight() //function to rotate right till it finds black line
{
  while(IR(pinIrMiddleLeft) == 1) //while loop to get out of black line
  {
    rotateRight(); //rotate right
  }
  stop(); //stop motors
  while(IR(pinIrMiddleLeft) == 0) // while loop to get back into black line
  {
    rotateRight(); //rotate right
  }
  stop(); //stop motors
}
//-------------------------------------------------------------------------------------------------------------------
------
//FUNCTION FOR LINE FOLLOW

void lineFollow() //line follow function
{
  if (IR(pinIrMiddleLeft) == 1 && IR(pinIrMiddleRight) == 1) //if both middle IR read 1 it will move forward
  {
    forward(); //calling forward function
  }
  else if (IR(pinIrMiddleLeft) == 1 && IR(pinIrMiddleRight) == 0) //if left IR reads 1 and right IR reads 0
  {
    left(); //turn left
  }
  else if (IR(pinIrMiddleLeft) == 0 && IR(pinIrMiddleRight) == 1) //if left IR reads 0 and right IR reads 1
  {
    right(); //turn right
  }
  else //any other condition
  {
    stop(); //stop
  }
}
//-------------------------------------------------------------------------------------------------------------------
------
//FUNCTION FOR INTERSECTION DETECTION

int intersection() //function to detect if there is an intersection
{
  if (IR(pinIrLeft) == 1 && IR(pinIrRight) == 1) //if both far IR's read 1
  {
    //print("1 \n"); //printing value for testing
    ledOn(pinIntersectionIndicator);
    return 1; //return 1
  }
  else //else
  {
    ledOff(pinIntersectionIndicator);
    //print("0 \n"); //printing value for testing
    return 0; //return 0
  }
}
//-------------------------------------------------------------------------------------------------------------------
```

```
------
//FUNCTION FOR COUNTING INTERSECTION

int intersectionCount() //function to count the intersection
{

  static unsigned int truth = 0; //fake object to avoid multiple counts

  if (intersection() == 1 && truth == 0) //checking if there is an intersection and avoid recounts
  {
    counter++; //incrementing the counter
  }
  truth = intersection(); //setting the fake object to see if it still at an intersection
  //print("number of intersections so far = %d\n" ,counter); //printing value for testing
  //print("Total intersections so far = %d\n", totalIntersections); //printing total intersections so far for testing
  return counter; //returning the count
}
//-------------------------------------------------------------------------------------------------------------------
------
//FUNCTION TO DETECT WHERE THE JAM IS AT RETURN ITS LOCATION
int frontObj() //function to see if there is an object in front
{
  unsigned int trafficAt = intersectionCount();
  if (distance(frontTrigPin, frontEchoPin) <= 7) //checking if distance is less than or equal to 5
  {
    //print("Traffic jam at %d\n", trafficAt +1); //printing for testing
    ledOn(ledPin); //truning on led for front object detection
    locationOfJam = trafficAt + 1; //setting global variable to store where Jam is located
    return trafficAt + 1; //return the location of the jam
  }
  else //else statement
  {
    return 0; //returns 0 if false
  }
}
//-------------------------------------------------------------------------------------------------------------------
------
//FUNCTION TO START FRO HOME AND COME BACK TO I0 AFTER DETECTING WHERE THE JAM IS AT
void detectJamAndReturn() //function to find where Jam is and come back to start
{
  while((frontObj() != 3) && (frontObj() != 4) && (frontObj() != 6) ) //while loop to ensure line follow keep executing
till condition not satisfied
  {
    lineFollow(); //line follow
  }
  counter = 0; //resetting the counter
  for (int i =0; i<=50;i++) //for loop
  {
    stop(); //stopping the bot for smooth motion
  }
  autoRotateLeft(); //180 degree rotation
  ledOff(ledPin); //swtiching off led for front object detection
  switch (locationOfJam) //switch case for the where the jam is at
  {
    case 3 : //case 3
      while(intersectionCount() != 2) //till count = 2 it will keep following the line
      {
        lineFollow(); //line follow
      }
      for (int i =0; i<=50;i++) //for loop
      {
        stop(); //stopping the bot for smooth motion
      }
      break; //break statement
    case 4 : //case 4
      while(intersectionCount() != 3) //till count = 3 it will keep following the line
      {
        lineFollow(); //line follow
      }
      for (int i =0; i<=50;i++) //for loop
      {
```

```
      stop();  //stopping the bot for smooth motion
    }
    break; //break statement
  case 6 : //case 6
    while(intersectionCount() != 5) //till count = 5 it will keep following the line
    {
      lineFollow(); //line follow
    }
    for (int i =0; i<=50;i++) //for loop
    {
      stop(); //stopping the bot for smooth motion
    }
    break;  //break statement
 }

 for (int i =0; i<=800;i++) //for loop
 {
   backward(); //going back
 }
 stop();  //stopping the bot for smooth motion
 autoRotateLeft(); //90 degree rotation
 stop(); //stop
 counter = 0; //resetting the counter
 followPath(); //starting next function follow Path on the same cog



}
//-------------------------------------------------------------------------------------------------------------------
------
//FUNCTION TO FOLLOW ASSIGNED PATH

void followPath() //function to follow path set which is go to i1-B1-B2-B3-B4-i4-A4-A3-A2-A1-i1-B1-B2-B3-B4-B5
{

 counter = 0; //resetting the counter to 0 for safety
 while(intersectionCount() == 0) //letting the bot reach I1
 {
   lineFollow(); //line follow
 }
 stop(); //stopping the bot
 for (int i =0; i<=250;i++) //for loop
 {
   backward(); //going back
 }
 for(int i =0;i<=100;i++) // for loop
 {
   stop(); //stopping the bot for safety
 }
 autoRotateLeft(); //rotate left
 while(intersection() == 0) //while loop to enter intersection
 {
  lineFollow(); //line follow
 }
 while(intersection() == 1) //while loop to exit intersection
 {
  lineFollow(); //line follow
 }
 while(intersectionCount() == 1) //letting the bot reach B1
 {
   lineFollow(); //line follow
 }
 stop(); //stopping the bot
 for (int i =0; i<=350;i++) //for loop
 {
   backward(); //going back
 }
 autoRotateRight(); //rotate right
 for(int i =0; i<100;i++) //for loop
 {
```

```
  stop(); //stop the bot for safety
}
//cog3 = cogstart((void*)goalLocationsTest, NULL, stack3, sizeof(stack3)); //starting the cog for object detection
cog3 = cogstart((void*)goalLocations, NULL, stack3, sizeof(stack3)); //starting the cog for object detection
while(intersection() == 0) //while loop to enter intersection
{
 lineFollow(); //line follow
}
while(intersection() == 1) //while loop to exit intersection
{
  lineFollow(); //line follow
}
while(intersectionCount() != 5) //letting the bot reach B4
{
  lineFollow(); //line follow
}
stop(); //stopping the bot
cogstop(cog3); //stopping the cog to avoid sensor noise
for (int i =0; i<=250;i++) //for loop
{
  backward(); //going back
}
for(int i =0;i<=100;i++) //for loop
{
  stop(); //stopping the bot for safety
}
autoRotateRight(); //rotate right
for(int i =0; i<100;i++) //for loop
{
  stop(); //stop the bot for safety
}
//cog3 = cogstart((void*)goalLocationsTest, NULL, stack3, sizeof(stack3)); //starting the cog
cog3 = cogstart((void*)goalLocations, NULL, stack3, sizeof(stack3)); //starting the cog for object detection
while(intersection() == 0) //while loop to enter intersection
{
 lineFollow(); //line follow
}
while(intersection() == 1) //while loop to exit intersection
{
  lineFollow(); //line follow
}
while(intersectionCount() != 7) //letting the bot reach A4
{
  lineFollow(); //line follow
}
stop(); //stopping the bot
cogstop(cog3);
for (int i =0; i<=250;i++) //for loop
{
  backward(); //going back
}
for(int i =0;i<=100;i++) //for loop
{
  stop();  //stopping the bot for safety
}
autoRotateRight(); //rotate right
//cog3 = cogstart((void*)goalLocationsTest, NULL, stack3, sizeof(stack3)); //starting the cog
cog3 = cogstart((void*)goalLocations, NULL, stack3, sizeof(stack3)); //starting the cog for object detection
for(int i =0; i<100;i++) //for loop
{
  stop(); //stopping the bot for safety
}
while(intersection() == 0) //while loop to enter intersection
{
 lineFollow(); //line follow
}
while(intersection() == 1) //while loop to exit intersection
{
  lineFollow(); //line follow
}
```

```
  while(intersectionCount() != 10) //letting the bot reach A1
  {
    lineFollow(); //line follow
  }
  cogstop(cog3); //stopping the cog
  stop(); //stopping the bot
  for (int i =0; i<=250;i++) //for loop
  {
    backward(); //going back
  }
  for(int i =0;i<=100;i++) //for loop
  {
    stop(); //stopping the bot
  }
  autoRotateRight(); //rotate right
  for(int i =0; i<100;i++) //for loop
  {
    stop(); //stopping the bot
  }
  //cog3 = cogstart((void*)goalLocationsTest, NULL, stack3, sizeof(stack3)); //starting the cog
  cog3 = cogstart((void*)goalLocations, NULL, stack3, sizeof(stack3)); //starting the cog for object detection
  while(intersection() == 0) //while loop to enter intersection
  {
   lineFollow(); //line follow
  }
  while(intersection() == 1) //while loop to exit intersection
  {
    lineFollow(); //line follow
  }
  while(intersectionCount() != 12) //letting the bot reach B1
  {
    lineFollow(); //line follow
  }
  stop(); //stopping the bot for safety
  cogstop(cog3); //stopping the cog
  for (int i =0; i<=350;i++) //for loop
  {
    backward(); //going back
  }
  for(int i =0;i<=100;i++) //for loop
  {
    stop(); //stopping the bot
  }
  autoRotateRight(); //rotate right
  for(int i =0; i<100; i++) //for loop
  {
    stop(); //stopping the bot
  }
  //cog3 = cogstart((void*)goalLocationsTest, NULL, stack3, sizeof(stack3)); //starting the cog
  cog3 = cogstart((void*)goalLocations, NULL, stack3, sizeof(stack3)); //starting the cog for object detection
  while(intersection() == 0) //while loop to enter intersection
  {
   lineFollow(); //line follow
  }
  while(intersection() == 1) //while loop to exit intersection
  {
    lineFollow(); //line follow
  }
  while(intersectionCount() != 16) //letting the bot reach B5
  {
    lineFollow(); //line follow
  }
  stop(); //stopping the bot

//------------------------------------------------------------------------------------------------------------------
------

  //SPECIAL STATEMENTS FOR COG CONTROL
  cogstop(cog1); //ending the cog1
  cogstop(cog3); //ending the cog3
```

```
540
541 //-------------------------------------------------------------------------------------------------------------------
    ------
542
543 }
544
545 //-------------------------------------------------------------------------------------------------------------------
    ------
546 //FUNCTION FOR GOAL LOCATIONS
547
548 void goalLocations() //function to get how many goal locations it has seen so far
549 {
550   static unsigned int truthObj = 0; //fake object to avoid multiple counts
551   while(goalsDetected != 2) //while loop for excetuting till it has seen all goal locations
552   {
553     if((distance(sideTrigPin, sideEchoPin) <= 10) && (truthObj == 0)) //checking if distance is less and 10cm and to
    ensure no recounts happens
554     {
555
556        goalsDetected++; //incrementing the goal locations
557        ledOn(object); //turning on led
558        truthObj = 1; //setting fake object to 1
559     }
560     else if((intersection() == 1) && truthObj == 1) //else statement
561     {
562       ledOn(object); //keeping the led On
563       truthObj = 1; //setting the fake object to 0
564     }
565     else if((distance(sideTrigPin, sideEchoPin) >= 11)) //distance check
566     {
567       ledOff(object); //turning off the led
568       truthObj = 0; //setting fake object to 0
569     }
570     //print("%d\n", goalsDetected);
571   }
572 //-------------------------------------------------------------------------------------------------------------------
    ------
573   //SPECIAL STATEMENTS FOR COG CONTROL
574
575   cogstop(cog1); //ending the cog2
576   stop(); //stopping the bot (for safety)
577
578 //-------------------------------------------------------------------------------------------------------------------
    ------
579
580 }
581
582 //-------------------------------------------------------------------------------------------------------------------
    ------
583 //FUNCTION FOR GOAL LOCATIONS
584
585  void goalLocationsTest() //forward declaration of function to get how many goal locations it has seen so far
586  {
587    while(1) //infinte loop
588    {
589    if (distance(sideTrigPin, sideEchoPin) <= 7) //checking the distance
590    {
591      ledOn(object); //turning on the led
592    }
593    else if((distance(sideTrigPin, sideEchoPin) > 10)) //checking the distance
594    {
595      ledOff(object); //led off
596    }
597   }
598  }
599
600
601 //-------------------------------------------------------------------------------------------------------------------
    -
```