# Autonomous Robot to Eliminate Enemy Intruder


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* - The purpose of this project is to demonstrate a clever system that detects enemy intruders among civilians and eliminate them in a city. The robot must be capable of detecting Aruco tags located on each object and eliminate if they are enemies. Friendly numbered 1-10 must never be disturbed. To simulate real-world situations the map has several one-way directions and the robot should follow the rules. The Robot should navigate the city, avoid obstacles search for enemies and eliminate them by scouring the streets while following all the traffic rules.**


## I. INTRODUCTION

The NYPD revealed the city's crime statistics for the month of March, announcing an overall crime index increase of 36.5% compared to the same time last year. Despite the NYPD's stepped-up enforcement, many New Yorkers are still deeply worried about crime, both in the subways and on the streets. Even though the budget for NYPD is increased to fight more crimes, there is a necessity for more advanced and efficient systems to fight crimes. The system should be autonomous and should be capable of monitoring the streets and safely taking out criminals. It should be cost-effective and shouldn't eliminate or disturb civilians.

This paper provides a detailed overview of the project 'Autonomous Robot to Eliminate Enemy Intruders', along with hardware used, software architecture, electronics circuit, and testing of the complete 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.

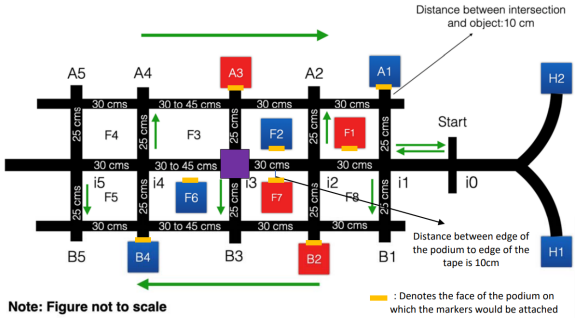


***Fig 1:*** *Diagram of the Map*

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 purple 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 accordingly.

d. Platform detection: The robot should continuously detect the platform on either side of the road. The Aruco tags will be located on the platform.
e. Aruco tag detection: The robot should detect the Aruco codes located in each platform and perform certain actions to hit the arco code referring to enemies.
f. 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 that the robot will not be causing a traffic jam.

## 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 robot's brain. 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 activity board has an onboard voltage regulator and servo headers. 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.

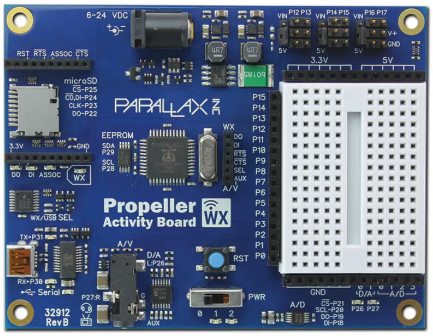


***Fig 2:*** *Image of Propeller Activity Board WX*

b. Raspberry Pi 4B(2GB): This board is the vision system for the robot. Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. The Raspberry Pi project originally leaned towards the promotion of teaching basic computer science in schools and in developing countries. Raspberry Pi is a powerful SBC to connect the camera and work on a vision system to detect Aruco tags and hit the enemies.

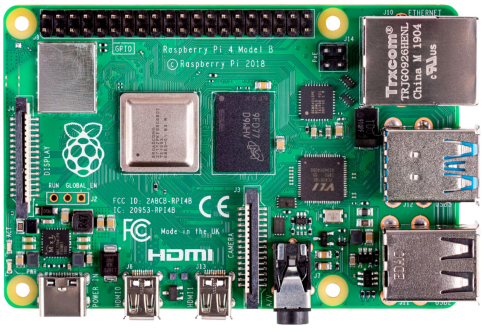

***Fig 2:*** *Image of the Raspberry pi 4B.*

c. Raspberry Pi camera: This camera directly connects to the raspberry pi using a ribbon cable. Arducam 5MP Camera for Raspberry Pi is used which

has an OV5647 sensor. This is equivalent to Raspberry pi Camera Module V1.

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

d. Buck Converter: A buck converter (step-down converter) is a DC-to-DC power converter that steps down voltage (while drawing less average current) from its input (supply) to its output (load). It is a class of switched-mode power supply (SMPS). This is used to step down the battery input (voltage) from 13.3 - 16.7V(4s Lipo) to 5V for Raspberry Pi.

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

e. 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.

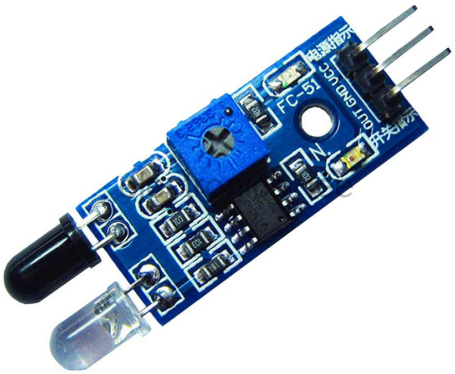


***Fig 3:*** *Image of the IR lane following sensor*

f. 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*

g. 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.

h. Standard Servo Motor: This servo motor is attached to the side of the Robot to rotate and arm and hit enemies, whenever detected. Servo motors have three wires: power, ground, and signal. The power wire is typically red and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the board. The signal pin is typically yellow or orange and should be connected to the PWM pin on the board. The recommended PWM frequency for servos is typically in the range of 40-200 Hz, with most servos using 50 Hz.

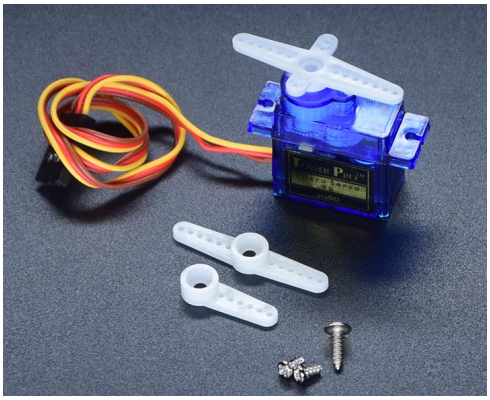

***Fig 4:*** *An image of SG-90 Servo Motor*

i. 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.

j. 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 raspberry pi serial communication makes it very easy to communicate with other devices such as the Activity Board WX, Arduino, etc via a USB port. The Raspberry pi also has a camera connector which makes it very easy to implement Computer Vision(OpenCV). 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.

| Component | Input/ output name | Activity Board WX pin |
|---|---|---|
| Yellow LED (Intersection) | 3.3v | 9 |
| Green LED (Friend detection) | 3.3v | 11 |
| Red Led (Enemy detection) | 3.3v | 4 |
| Blue Led (Detect Obstacle) | 3.3v | 10 |
| Left Servo (Continous) | Signal (PWM) | 12 |
| Right Servo (Continous) | Signal (PWM) | 13 |
| Knocker Servo (Continous) | Signal (PWM) | 17 |
| 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

The circuit is connected using jumper cables and suitable resistors are connected between LEDs and the pinout of Activity Board WX. The above table shows all the Pins with respect to components.

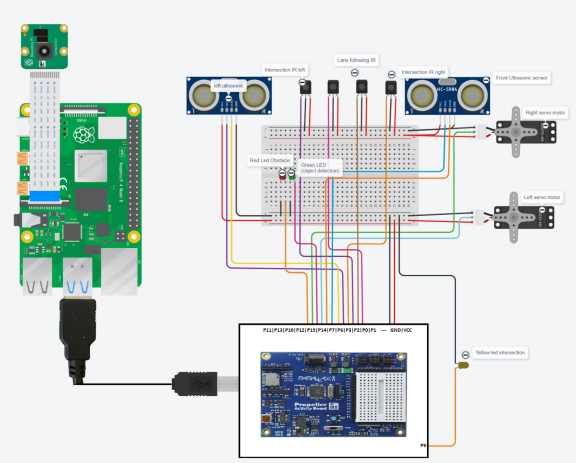

***Fig 5:*** *Circuit Diagram*

## V. SOFTWARE / PROGRAMMING

The Raspberry pi uses python programming language script to detect the Aruco tags using the OpenCV library. It has a class that returns “0” for friendly tags, and “1” for enemy tags, and it will return “2” if no tags are detected.This is also sent to Activity Board WX through serial communication. The Activity Board WX is programmed with C programming language and contains all the information regarding the navigation. This is the hardware that is connected to all the sensors and actuators. 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. The Cog 3 receives information through serial communication from Rasberry pi to actuate servo motors. Several global variables and flags are used to communicate or perform actions in different cogs.

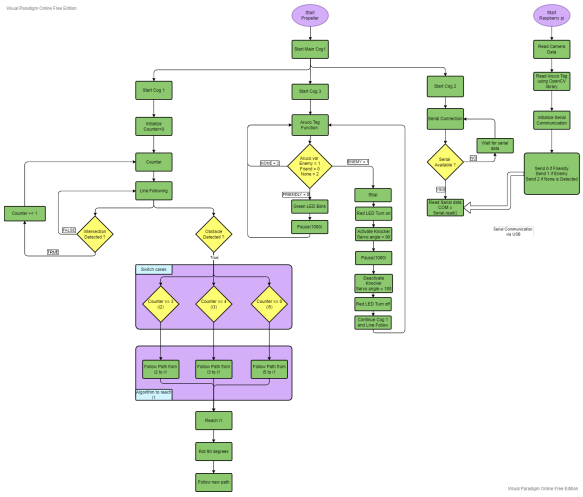

***Fig 6:*** *Flow chart of Algorithm*

## VI. PROTOTYPING

Propeller Activity Board WX is mounted on top of the Boe-bot chassis. The Raspberry pi is sandwiched between the Bee-bot chassis and the Propeller activity board. 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.

During the prototyping phase, various mounts were developed. Initially, the ultrasonic sensor and camera were mounted on the front of the Bee-bot using along with the IR sensor array in a 3d printed mounting system. The robot was hitting objects and obstacles during the turning maneuver. Hence a new mounting method was designed to accommodate the sensors.

Initially, the knocking mechanism was a Rack and pinion system, due to reliability issues and space constraints, this model was scratched and a simple rotary servo with a long arm is used for our final prototype.

Various battery and power supplies have also been tested. Lipo with a buck converter is selected since it can power the Raspberry pi and Bee-Bot along with the propeller Activity board without any struggles. And the Lipo battery is 4s(14.7v) 1500mah, which is rated for 300 cycles. This battery can power the robot for 1 hour and 30 mins continuously.

The Final prototype is compact and 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(enemy and friendly) on the sides and front. The robot first moves towards the obstacle and decides which route to take. Then the robot will return to the i0 position and run a different route. While running the route the robot will check for enemies and friendlies using the raspberry pi and camera on the right side and whenever an enemy is detected it will stop and blink an LED and then knock the enemy. If a friendly is detected it will stop blinks the Green LED and then continues.

## 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 Activity Board. The final Prototype successfully executed all the tasks and completed the objectives.

## VIII. RESULTS

The prototype can execute complex tasks and navigate the map, avoid obstacles, detect enemy Aruco Tags and take action(strike enemy targets). 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 with great accuracy. The system never eliminated any civilian(friendly) target during the testing. This prototype showcased the multicore capabilities of Propeller Activity Board WX and the ease of use of this board for quick prototyping. This project also highlighted the vision capabilities and communication system of an SBC like Raspberry pi.

## IX. CONCLUSION

The project demonstrated a fully working prototype of a robot that can be used for the crime reduction and elimation 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 sensors such as LIDAR and powerful Single Board computers like Jetson Nano 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 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. Raspberry pi Website and help
https://www.raspberrypi.org/

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

d. SG-90 servo motor
https://datasheetspdf.com/pdf/791970/TowerPro/SG90/1

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

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

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

h. "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

```
// ALL INCLUDE STATEMENTS

#include "servo.h"
#include "fdserial.h"
#include "servodiffdrive.h"
#include "simpletools.h"

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

//ALL COGS

int *cog1; //cog 1 reference
int *cog2; //cog 2 reference
int *cog3; //cog 3 reference

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

//ALL STACKS



unsigned int stack1[40+100]; //stack1 value
unsigned int stack2[40+100]; //stack2 value
unsigned int stack3[40+100]; //stack3 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 obstacle detectiions indication
#define objectFriendly 11 //object detection led for a friendly
#define objectEnemy 4 //object detection led for an enemy
#define knocker 17 // standard servo for knocking


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

//GLOBAL VARIABLES
fdserial *tags; //serial object for communication with rasberry pie
volatile char com; //variable to store values of serial communication
volatile unsigned int counter = 0; // global counter variable
volatile unsigned int middleCounter = 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
unsigned int nowStop = 0; //global int for signifing stopping
unsigned int nowKnock = 0; //global unsigned int signifing knock

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

// 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
```

```
68 int distance(const unsigned int trigPin, const unsigned int echoPin); //forward declaration of function to find distance
   from ultrasonic sensor
69 void servoInit(const unsigned int leftpin, const unsigned int rightpin); //forward declaration of function to attach
   pins
70 void forward(); //forward declaration of move forward function
71 void backward(); //forward declaration of move backward function
72 void left(); //forward declaration of  move left function
73 void right(); //forward declaration of  move right function
74 void stop(const unsigned int time); //forward declaration of  stationary function
75 void lineFollow(); //forward declaration of line following function
76 int intersection(); //forward declaration of detecting an intersection
77 int intersectionCount(); //forward declaration of intersection counter
78 int frontObj(); //forward declaration of function to see if there is a front object
79 void rotateLeft(); // forward declaration of function to rotate the bot left
80 void rotateRight(); //forward declaration of function to rotate the bot right
81 void detectJamAndReturn(); //forward declaration of function to find where Jam is and come back to start
82 void autoRotateLeft(); //forward declaration of function to rotate left till it finds black line
83 void autoRotateRight(); //forward declaration of function to rotate right till it finds black line
84 void followPath(); //forward declaration of function to follow path set which is got to B1-B2-B3-B4-i4-A4-A3-A2-A1-i1-
   B1-B2-B3-B4-B5
85 void exitIntersection(const unsigned int translation); //forward declaration of function to exit intersection
86 int middleIntersectionCount(); //forward declaration of middle lane intersection counter
87 void initSer(); //forward declaration of function to intialize serial communication
88 void detectArUcoTags(); //forward declaration of function to detect arUco tags
89 void knock(); //foward declaration of function to knock enemy
90
91
92 //-------------------------------------------------------------------------------------------------------------------
   -----
93
94 // MAIN LOOP
95
96
97 int main()
98 {
99   servoInit(leftServoPin, rightServoPin); //initalizing servo
100  initSer(); //instalizing the serial communication
101  cog1 = cogstart((void*)detectJamAndReturn, NULL, stack1, sizeof(stack1)); //starting cog1
102  cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2));  //starting cog2
103  cog3 = cogstart((void*)knock, NULL, stack3, sizeof(stack3)); //starting cog 3
104 }
105
106
107 //-------------------------------------------------------------------------------------------------------------------
   -----
108
109 // ALL FUNCTIONS
110
111 //-------------------------------------------------------------------------------------------------------------------
   -----
112 //FUNCTION TO TURN ON LED
113
114 void ledOn(const unsigned int pin) //function to turn on led
115 {
116   high(pin); //high pin
117 }
118 //-------------------------------------------------------------------------------------------------------------------
   -----
119 //FUNCTION TO TURN OFF LED
120
121 void ledOff(const unsigned int pin) //function to turn off led
122 {
123   low(pin); // low pin
124 }
125
126 //-------------------------------------------------------------------------------------------------------------------
   -----
127 //FUNCTION TO TAKE IR READINGS
128
129 int IR(const unsigned int pin) //function to get IR input
130 {
131   volatile int i = input(pin); //volatile int storing input of pin
132   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);
  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(-30,-30); //full forward
}
//-------------------------------------------------------------------------------------------------------------------------------
//FUNCTION TO DRIVE BACKWARD
void backward() //drive backward
{
  drive_speeds(30,30); //full backward
}
//-------------------------------------------------------------------------------------------------------------------------------
//FUNCTION TO TILT LEFT
void left() //turn left
{
  drive_speeds(-30, 0); //keeping left wheel stationary
}
//-------------------------------------------------------------------------------------------------------------------------------
//FUNCTION TO TILT RIGHT
void right() //turn right
{
  drive_speeds(0, -30); //keeping right wheel stationary
}
//-------------------------------------------------------------------------------------------------------------------------------
//FUNCTION FOR STOPPING
void stop(const unsigned int time) //bot stationary
{
  for (int i = 0; i <= time; i++)
  {
    drive_stop(); //stopping the servos
  }
}
//-------------------------------------------------------------------------------------------------------------------------------
//FUNCTION TO ROTATE RIGHT
void rotateRight() //bot rotate right
{
  drive_speeds(30, -30); //rotating both wheels same speed in opposite direction to make it rotate right
}
//-------------------------------------------------------------------------------------------------------------------------------
//FUNCTION TO ROATE LEFT
void rotateLeft() //bot rotate left
{
  drive_speeds(-30, 30); //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(300); //stop motors
  while(IR(pinIrMiddleRight) == 0) // while loop to get back into black line
  {
    rotateLeft(); //rotate left
  }
  stop(300); //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(300); //stop motors
  while(IR(pinIrMiddleLeft) == 0) // while loop to get back into black line
  {
    rotateRight(); //rotate right
  }
  stop(300); //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(1); //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);
    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
  return counter; //returning the count
}
//-------------------------------------------------------------------------------------------------------------------------
-----
//FUNCTION FOR COUNTING INTERSECTION ONLY FOR THE MIDDLE LANE
int middleIntersectionCount() //function to count the intersection
{

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

  if (intersection() == 1 && middleTruth == 0) //checking if there is an intersection and avoid recounts
  {
    middleCounter++; //incrementing the counter
  }
  middleTruth = intersection(); //setting the fake object to see if it still at an intersection
  return middleCounter; //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) <= 2) //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 EXIT INTERSECTION
void exitIntersection(const unsigned int translation) //function to exit intersection
{
  while(intersection() == 1) //while loop to exit intersection
  {
    lineFollow(); //line follow
  }
  for(int i = 0; i <= translation; i++) //for loop to give a very minor translation
  {
    lineFollow(); // line loop
  }
  stop(60); //stopping the bot for saftey
}

//-------------------------------------------------------------------------------------------------------------------------
-----
//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
```

```
{
  //cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //starting cog2
  while((frontObj() != 3) && (frontObj() != 4) && (frontObj() != 6) ) //while loop to ensure line follow keep executing till condition not satisfied
  {
    lineFollow(); //line follow
    //detectArUcoTags(); //detect tags
  }
  //cogstop(cog2); //stopping cog2
  counter = 0; //resetting the counter
  stop(50); //stopping the bot for smooth motion
  for (int i = 0; i <= 600; i++) //for loop for going back
  {
    backward(); //bakward
  }
  autoRotateLeft(); //180 degree rotation
  ledOff(ledPin); //swtiching off led for front object detection

  //cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //stopping cog2
  switch (locationOfJam) //switch case for the where the jam is at
  {
    case 3 : //case 3
      while(intersectionCount() != 1) //till count = 2 it will keep following the line
      {
        lineFollow(); //line follow
        //detectArUcoTags(); //detect tags
      }
      stop(50); //stopping the bot for smooth motion
      break; //break statement
    case 4 : //case 4
      while(intersectionCount() != 2) //till count = 3 it will keep following the line
      {
        lineFollow(); //line follow
        //detectArUcoTags(); //detect tags
      }
      stop(50);  //stopping the bot for smooth motion
      break; //break statement
    case 6 : //case 6
      while(intersectionCount() != 4) //till count = 5 it will keep following the line
      {
        lineFollow(); //line follow
        //detectArUcoTags(); //detect tags
      }
      stop(50); //stopping the bot for smooth motion
      break;  //break statement
  }
  stop(1);  //stopping the bot for smooth motion
  //cogstop(cog2); //stoping cog2
  exitIntersection(400); //exit intersection
  autoRotateRight(); //90 degree rotation
  stop(1); //stop
  counter = 0; //resetting the counter
  //cog2 = cogstart((void*)followPath, NULL, stack2, sizeof(stack2)); //starting cog2
  //cogstop(*cog1);
  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 B1-B2-B3-B4-i4-A4-A3-A2-A1-i1-B1-B2-B3-B4-B5
{

  counter = 1; //resetting the counter to 1 for safety
  while(intersectionCount() != 2) //letting the bot reach B1
  {
    lineFollow(); //line follow
  }
  stop(1); //stopping the bot
```

```
  exitIntersection(200); //exit intersection //NEW
  autoRotateRight(); //rotate right
  //cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //starting cog2
  while(intersectionCount() != 5) //letting the bot reach B4
  {
    lineFollow(); //line follow
    //detectArUcoTags(); //detect tags
  }
  //cogstop(cog2);
  stop(1); //stopping the bot
  exitIntersection(600); //exit intersection //NEW
  autoRotateRight(); //rotate right
  while(intersectionCount() != 6) //letting the bot reach i4
  {
    lineFollow(); //line follow
  }
  stop(1); //stopping the bot
  //cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //starting cog2
  switch (locationOfJam) //switch case for the where the jam is at
  {
    case 3:
      exitIntersection(400); //exit intersection //NEW
      autoRotateRight(); //rotate 90 right
      //cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //starting cog2
      while ((middleIntersectionCount() <= 1) && (distance(frontTrigPin, frontEchoPin) >= 2)) //while loop to keep
following line till front abject is detected
      {
        lineFollow(); //line follow
        //detectArUcoTags(); //detect tags
      }
      stop(100); //stop
      ledOn(ledPin); //turning on led
      stop(100);
      for (int i = 0; i <= 800; i++) //for loop for going back
      {
        backward(); //bakward
      }
      //cogstop(cog2); //stopping cog 2
      autoRotateRight(); //rotate 180 right
      stop(100);
      ledOff(ledPin); // turning off led
      //cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //starting cog2
      while (middleIntersectionCount() != 4) //while loop for the bot to reach i5
      {
        lineFollow(); //line follow
        //detectArUcoTags(); //detect tags
      }
      stop(1); //stop
      //cogstop(cog2); //cog stop
      exitIntersection(400); //exit intersection //NEW
      autoRotateRight(); //rotate 90 right
      stop(2000); //a big stop statement to for a smooth trasistion
      autoRotateRight(); //rotate 90 right
      stop(1); //stop
      //cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //starting cog2
      while (middleIntersectionCount() != 5) //let the bot reach i4
      {
        lineFollow(); //line following
        //detectArUcoTags(); //detect tags
      }
      //cogstop(cog2); //cog stop 2
      exitIntersection(400); //exit intersection //NEW
      autoRotateLeft(); //rotate 90 left
      break; //break statement
    case 4:
      exitIntersection(400); //exit intersection //NEW
      autoRotateRight(); //rotate 90 right
      //stop(1000); //stopping for a very less time to avoid front detection early
      //cog3 = cogstart((void*)detectArUcoTags2, NULL, stack3, sizeof(stack3)); //starting cog2
      while ((middleIntersectionCount() == 0) && (distance(frontTrigPin, frontEchoPin) >= 2)) //while loop to keep
following line till front abject is detected
      {
```

```
      lineFollow(); //line follow
      //detectArUcoTags(); //detect tags
    }
    //cogstop(cog3); //cog stop 2
    stop(100);
    ledOn(ledPin); //turning on led
    stop(100);
    for (int i = 0; i <= 800; i++) //for loop for going back
    {
      backward(); //bakward
    }
    autoRotateRight(); //rotate 180 right
    stop(100);
    ledOff(ledPin); // turning off led
    //cog3 = cogstart((void*)detectArUcoTags2, NULL, stack3, sizeof(stack3)); //starting cog2
    while (middleIntersectionCount() != 2) //while loop for the bot to reach i5
    {
      lineFollow(); //line follow
      //detectArUcoTags(); //detect tags
    }
    //cogstop(cog3); //cog stop 2
    stop(1); //stop
    exitIntersection(400); //exit intersection //NEW
    autoRotateRight(); //rotate 90 right
    stop(2000); //a big stop statement to for a smooth trasistion
    autoRotateRight(); //rotate 90 right
    stop(1); //stop
    //cog3 = cogstart((void*)detectArUcoTags2, NULL, stack2, sizeof(stack2)); //starting cog2
    while (middleIntersectionCount() != 3) //let the bot reach i4
    {
      lineFollow(); //line following
      //detectArUcoTags(); //detect tags
    }
    //cogstop(cog3); //cog stp 2
    exitIntersection(400); //exit intersection //NEW
    autoRotateLeft(); //rotate 90 left
    break; //break statement
  case 6:
    exitIntersection(400); //exit the current intersection
    break; //break statement
}
//cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //starting cog2
while(intersectionCount() != 7) //letting the bot reach A4
{
  lineFollow(); //line follow
}
stop(1); //stopping the bot
exitIntersection(400); //exit intersection //NEW
autoRotateRight(); //rotate right
//cog4 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //starting cog2
while(intersectionCount() != 10) //letting the bot reach A1
{
  lineFollow(); //line follow
  //detectArUcoTags(); //detect tags
}
stop(1); //stopping the bot
//cogstop(cog4); //cog stop 2
exitIntersection(600); //exit intersection //NEW
autoRotateRight(); //rotate right
while(intersectionCount() != 12) //letting the bot reach B1
{
  lineFollow(); //line follow
}
stop(1); //stopping the bot
exitIntersection(400); //exit intersection //NEW
autoRotateRight(); //rotate right
while(intersectionCount() != 15) //letting bot reach B4
{
  lineFollow(); //line follow
}
//cog2 = cogstart((void*)detectArUcoTags, NULL, stack2, sizeof(stack2)); //starting cog2
while(intersectionCount() != 16) //letting the bot reach B5
```

```
  {
    lineFollow(); //line follow
    //detectArUcoTags(); //detect tags
  }
  exitIntersection(800);
  stop(1);


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

  //SPECIAL STATEMENTS FOR COG CONTROL
  //cogstop(*cog2); //ending the cog2

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

}
//-------------------------------------------------------------------------------------------------------------------------
-----
//FUNCTION TO INTIALIZE SERIAL COMMUNICATION
void initSer() //function to intialize serial communication
{
  tags = fdserial_open(31, 30, 0, 115200);
}
//-------------------------------------------------------------------------------------------------------------------------
-----
//FUNCTION TO DETECT TAGS

void detectArUcoTags() //function to detect arUco Tags
{
  initSer(); //instalizing the serial communication
  while(1){
  com = fdserial_rxChar(tags); //reading the communication from rasberry pie which encodes as UTF-8
  if((distance(sideTrigPin, sideEchoPin) >= 3 && distance(sideTrigPin, sideEchoPin) <= 12)) //if ultrasonic detects an
object
  {
    if (com != -1) //only if com is transmitting data it will exceute
    {
      if (com == '2') //if char is 2 it means no tags Ids were detected (false positive from Ultrasonic sensor)
      {
        ledOff(objectFriendly); //led off
        ledOff(objectEnemy); //led off
        nowKnock = 0; //knocking flag set to false
      }
      else if (com == '0') //if rasberry pie transmites 0 which means friendly
      {
        ledOn(objectFriendly); //led on for friendly object
        ledOff(objectEnemy); //led off for enemy
        nowKnock = 0; //knocking flag set to false
      }
      else if (com == '1') //if rasberry pie transmites 1 which means enemy
      {
        ledOff(objectFriendly); //led off for friendly object
        ledOn(objectEnemy); //led on object enemy
        nowKnock = 1; //Knocking flag set to true
      }
      else
      {
        ledOff(objectFriendly); //led off for friendly object
        ledOn(objectEnemy); //led on object enemy
        nowKnock = 0; //Knocking flag set to true
      }
    }
  }
  else
  {
    ledOff(objectFriendly); //led off
    ledOff(objectEnemy); //led off
    nowKnock = 0; //Knocking flag set to false
  }
}
}
```

```

//-------------------------------------------------------------------------------------------------------------------
-----
//FUNCTION TO KNOCK ENEMIES
void knock() //function to knock enemy
{
  while(1) //infinite loop
  {
    if(nowKnock == 1) //signifing knocking as 1
    {
      servo_angle(knocker, 700); //knocking to 90 deg
    }
    else if (nowKnock == 0)
    {
      servo_angle(knocker, 1800); //knocking to 0 deg
    }
  }
}
//-------------------------------------------------------------------------------------------------------------------
-----
```

```python
import cv2 as cv  # importing openCV library as cv
import cv2.aruco as aruco  # importing openCV aruco module as aruco
import serial as sr  # importing pyserial library as sr
from picamera.array import PiRGBArray  # importing picamera array
from picamera import PiCamera  # importing picamera
# https://stackoverflow.com/questions/26169633/capturing-video-with-raspberry-using-opencvpicamera-stream-io


class arUcoDetector:  # Defining class
    def __init__(self, resolution = (1920, 1080), fps = 32, minMarkerPerimeterRate = 0.1, maxMarkerPerimeterRate = 4,
,                 minDistanceToBorder = 3, port = '/dev/ttyUSB0',
,                 baudRate = 115200):  # Defining the initialization parameters with default values
        self.camera = PiCamera()  #  creating object picamera
        self.camera.resolution = resolution  #  setting the resolution
        self.camera.framerate = fps  # setting the frame rate
        self.rawCapture = PiRGBArray(self.camera, size = resolution)  #  turing it raw data
        self.arUcoDict = aruco.Dictionary_get(aruco.DICT_6X6_50)  # Creating variable for aruco dictionary
        self.arUcoParams = aruco.DetectorParameters_create()  # Creating variable for parameters
        self.arUcoParams.minMarkerPerimeterRate = minMarkerPerimeterRate  # Setting the parameters
        self.arUcoParams.maxMarkerPerimeterRate = maxMarkerPerimeterRate  # Setting the parameters
        self.arUcoParams.minDistanceToBorder = minDistanceToBorder  # Setting the parameters
        while True:  # infinite loop
        try:  # try block
        self.ser = sr.Serial(str(port), baudRate)  # Setting the ser object by specifying port and baud rate
        except (FileNotFoundError, sr.SerialException):  # Except block to allow only these error
        pass  # Pass the error to continue in the loop
        else:  # Else statement
        break  # break statement

    def detect(self):  # defining function detect
        for image in self.camera.capture_continuous(self.rawCapture, format = "bgr", use_video_port = True):  # for loop
            frame = image.array  # reading the video and saving it to a matrix frame for each time step
            frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)  # Removing the BGR Channels to grayscale so only 500 x 500 x 1
pixels
            #frame = cv.rotate(frame, cv.ROTATE_180) #rotating frame for easy visuals
            [corners, ids, _] = aruco.detectMarkers(frame, self.arUcoDict, parameters = self.arUcoParams)  # getting
corners and ids
            frame = aruco.drawDetectedMarkers(frame, corners, ids = ids)  # Drawing markers on each frame
            if ids is None:  # If no tags are detected
                self.ser.write(str.encode("2"))  # Send 2 to propeller
                print("2")  # Printing 2 for testing
            elif 0 <= ids <= 9:  # If tag ID are in range of 0 to 9
                self.ser.write(str.encode("0"))  # It will send 0 to propeller
                print("0")  # print 0 for testing
            elif 10 <= ids <= 19:  # If tag ID are in the range 10 to 19
                self.ser.write(str.encode("1"))  # It will send 1 to propeller
                print("1")  # print 1 for testing
            cv.imshow("arUco Tags", frame)  # show each frame
            self.rawCapture.truncate(0)  # clearing data from the array for next frame
            if cv.waitKey(1) & 0xFF == ord('q'):  # If wait is done for 1 sec bit wise and is the ordinal value q which
is 127 on keyboard
                break  # break statement to exit infinite loop
        cv.destroyAllWindows()  # destroying all windows


while True: #  Infinite loop
    try:  # Try block
        detector = arUcoDetector((240, 160), 90 ,0.4, 100, 3, '/dev/ttyUSB0', 115200)  # Creating an object with
parameters
        detector.detect()  # Calling function to detect
    except:  # Except block to only allow these errors
    del detector  # Deleting the object
    else:  # Else statement
        break #  Break statement
```

Start Propeller

Start Main Cog1

Start Cog 1

Initialize Counter=0

Counter

Line Following

Intersection Detected ?

FALSE

TRUE

Counter += 1

Obstacle Detected ?

True

Switch cases

Counter == 3 (i2)

Counter == 4 (i3)

Counter == 6 (i5)

Follow Path from i2 to i1

Follow Path from i3 to i1

Follow Path from i5 to i1

Algorithm to reach i1

Reach i1

Rot 90 degrees

Follow new path

Start Cog 3

Aruco Tag Function

Aruco var
Enemy = 1
Friend = 0
None = 2

NONE = 2

FRIENDLY = 0

ENEMY = 1

Green LED Blink

Pause(1000)

Stop

Red LED Turn on

Activate Knocker
Servo angle = 90

Pause(1000)

Deactivate Knocker
Servo angle = 180

Red LED Turn off

Continue Cog 1 and Line Follow

Start Cog 2

Serial Connection

Serial Available ?

NO

Wait for serial data

YES

Read Serial data
COM = Serial.read()

Start Raspberry pi

Read Camera Data

Read Aruco Tag using OpenCV library

Initialize Serial Communication

Send 0 if Friendly
Send 1 if Enemy
Send 2 if None is Detected

Serial Communication via USB