BOT_PROGRAMMING-

1. PID CONTROLLER – Cuboid (Refer Coppeliasim Model)

```
function sysCall_init()
    -- do some initialization here
    RobotBase =
sim.getObjectAssociatedWithScript(sim.handle_self)
    M1 = sim.getObjectHandle('M1')
    M2 = sim.getObjectHandle('M2')
    M3 = sim.getObjectHandle('M3')
    M4 = sim.getObjectHandle('M4')

    P11 = sim.getObjectHandle('P11')
    P12 = sim.getObjectHandle('P12')
    P13 = sim.getObjectHandle('P13')
    P14 = sim.getObjectHandle('P14')
    P15 = sim.getObjectHandle('P15')
    P16 = sim.getObjectHandle('P16')

    sim.setJointTargetVelocity(M1,0)
    sim.setJointTargetVelocity(M2,0)
    sim.setJointTargetVelocity(M3,0)
    sim.setJointTargetVelocity(M4,0)

    totError = 0.0
    lastError = 0.0
    number = 0

    file = io.open("CentreCoordinates.txt", "r")

end

function sysCall_actuation()

        if(D11 == nil) then
    D11=1
    end
        if(D12 == nil) then
    D12=1
    end
        if(D13 == nil) then
    D13=1
    end
        if(D14 == nil) then
    D14=1
    end
        if(D15 == nil) then
    D15=1
    end
        if(D16 == nil) then
    D16=1
    end

    if(D15>0.3) then
      c=1
    elseif(D15<0.3) then
      c=2
    end


  print(D11)
  print(D12)
 V01 = PID(D11,D12)

         if(c==1) then
          sim.setJointTargetVelocity(M1,-(1+V01))
          sim.setJointTargetVelocity(M2,-(1+V01))
          sim.setJointTargetVelocity(M4,-(1-V01))
          sim.setJointTargetVelocity(M3,-(1-V01))
         elseif(c==2) then
          moveRight()
         end
 end


function sysCall_sensing()
    -- put your sensing code here
    D11 = getDistance(P11)
    D12 = getDistance(P12)
    D13 = getDistance(P13)
    D14 = getDistance(P14)
    D15 = getDistance(P15)
    D16 = getDistance(P16)
    simTime=sim.getSimulationTime()


end

function sysCall_cleanup()
    -- do some clean-up here
end

-- See the user manual or the available code snippets for
additional callback functions and details
function getDistance(sensor)
  local detected, distance
   detected, distance = sim.readProximitySensor(sensor)
  if (detected<1) then
   distance = 1.0
  end
  return distance
end
  -- ABC
function PID(Distance1, Distance2)
  T  = 0.5
  Kp = 25
  Kd = 22

  p=0.7
  q=0.7
 error = (Distance1-Distance2)*p + (0.2-
(Distance1+Distance2)/2)*q
 totError = error + totError
 deltaError = error - lastError
 simTime1=sim.getSimulationTime()
 Control = Kp*error + (Kd)*deltaError

 lastError = error
 return(Control)
end

function moveRight()
sim.setJointTargetVelocity(M1,-1)
sim.setJointTargetVelocity(M2,-1)
sim.setJointTargetVelocity(M3,0)
sim.setJointTargetVelocity(M4,0)
end

function moveLeft()
sim.setJointTargetVelocity(M4,-1)
sim.setJointTargetVelocity(M3,-1)
sim.setJointTargetVelocity(M2,0)
sim.setJointTargetVelocity(M1,0)
end
```

2. Image Processing (Refer OpenCV)

```
import cv2
import numpy as np

def find_centre2(white_loc):
    n = len(white_loc)
    #print(n)
    if n<120:
        print("No Flame Detected")
        return
    arr = np.array(white_loc)
    x,y = np.sum(arr,axis=0)/n
    return (round(y), round(x))


def operate(img):

    img_in = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img_out = np.zeros(img_in.shape)
    img_out[img_in>190]=255
    img_out[img_in>220]=0
    return img_out

def find_white_loc(img_in):
    white_loc=[]
    m,n = img_in.shape
    for i in range(m):
        for j in range(n):
            if img_in[i,j]==255:
                white_loc.append([i,j])
    return white_loc
count=0
while count<100:
    count=count+1
    video = cv2.VideoCapture('Fire.jpg')
    while True:
        ret,frame = video.read()
        if ret== False:
            break

        frame = cv2.resize(frame, (960, 540))

        blur = cv2.GaussianBlur(frame, (21, 21), 0)
        hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

        lower = [18, 50, 50]
        upper = [60, 255, 255]
        lower = np.array(lower, dtype="uint8")
        upper = np.array(upper, dtype="uint8")

        mask = cv2.inRange(hsv, lower, upper)
        output = cv2.bitwise_and(frame, hsv, mask=mask)
        no_red = cv2.countNonZero(mask)

        mask = operate(frame)
        white_loc = find_white_loc(mask)
        centre = find_centre2(white_loc)

        if int(no_red) > 15000:
            print("FIRE DETECTED")
            print(centre)

            text_file = open("CentreCoordinates.txt", "w")
            text_file.write('1')
            text_file.write(str(centre[0]))

            text_file.write(' ')
            text_file.write(str(centre[1]))

            text_file.close()

        else:

            text_file = open("CentreCoordinates.txt", "w")
            text_file.write('2')
            text_file.close()

        cv2.circle(frame,centre,60,(255,0,255),2)
        #cv2.imshow('gray',mask)
        cv2.imshow('flame',frame)
        #cv2.imshow('output',output)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

cv2.destroyAllWindows()
video.release()
```

3. Nozzle Motion for Audrino IDE

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

int servoPin_y = 6;  //for rotation in vertical plane
int servoPin_x = 5;  //for rotation in horizontal plane
int pumpPin = 4;
Servo Servo_y;
Servo Servo_x;
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(100);
  // put your setup code here, to run once:
   Servo_y.attach(servoPin_y);
   Servo_x.attach(servoPin_x);

   pinMode(pumpPin, OUTPUT);
}

void loop() {
  int counter = 0;
  digitalWrite(pumpPin, HIGH);

  float plane_theta=360,z_rotation=360;
  String serialData=Serial.readString();
  plane_theta=extractY(serialData);
  z_rotation=extractX(serialData);
  counter = (int)extractZ(serialData);
  delay(100);


  if(plane_theta < 180 && z_rotation < 180)
  {
    Servo_y.write(round(95+plane_theta)); //mean
positions angles adjusted by real world experimentation
    delay(500);
    Servo_x.write(round(100+z_rotation));
    delay(2000);

    if(counter!= 0)
    {
      digitalWrite(pumpPin, HIGH);
      delay(4000);
      digitalWrite(pumpPin, LOW);
      }
    }
}

//buffer string =
"X<z_rotation>Y<plane_theta>Z<flame_status>"

int extractX(String data){
  data.remove(data.indexOf("Y"));
  data.remove(data.indexOf("X"),1);
  return data.toFloat();
  }
int extractY(String data){
  data.remove(data.indexOf("Z"));
  data.remove(0,data.indexOf("Y")+1);
  return data.toFloat();
  }
int extractZ(String data){
data.remove(0,data.indexOf("Z")+1);
return data.toFloat();
}
```

4. Fire Detection Using Camera

```
import numpy as np
import serial
import time
import cv2
from math import *

# All params are in m.

# Distance of screen from camera = D
D=1.46

# Height of the camera = H
H= 0.25

# Height of the nozzle = h
h=0.30

# Distance of the nozzle from screen
d=0.20

# m/pixel ratio= D/640 for a distance D of the screen
from camera
fact= D/640

#velocity = 4m/s
u=4

#gravity
g = 9.81

arduino = serial.Serial(port='COM7', baudrate=9600,
timeout=0.1) #port number may vary

def find_centre2(white_loc):
    n = len(white_loc)
    #print(n)
    if n<120:
        print("No Flame Detected")
        return
    arr = np.array(white_loc)
    x,y = np.sum(arr,axis=0)/n
    return (round(y),round(x))


def operate(img):

    img_in = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img_out = np.zeros(img_in.shape)
    img_out[img_in>230]=255
    return img_out

def find_white_loc(img_in):
    white_loc=[]
    m,n = img_in.shape
    for i in range(m):
        for j in range(n):
            if img_in[i,j]==255:
                white_loc.append([i,j])
    return white_loc

def plane_theta(centre):
        ## coordinates with respect to camera
        x = centre[0]*fact
        y = centre[1]*fact

        ## coordinates with respect to nozzle. Obtained by
shifting of origin
        X = sqrt(d**2+(x)**2)
        Y = y + H - h
        print("X:",X, "Y: ", Y)

        Q1 = ( atan( ( (u**2) + sqrt( u**4 - 2*Y*g*u**2 -
(g**2)*(X**2) ) ) / (g*X) ) ) * 180/3.1415
        Q2 = ( atan( ( (u**2) - sqrt( u**4 - 2*Y*g*u**2 -
(g**2)*(X**2) ) ) / (g*X) ) ) * 180/3.1415

        if (abs(Q1)<abs(Q2)):
            angle = Q1
        else:
            angle = Q2
        print("Vertical Angle is : ",angle)
        return angle

def z_rotation(centre):

            ## coordinates with respect to camera
        x = centre[0]*fact
        y = centre[1]*fact


        # coordinates with respect to nozzle. Obtained by
shifting of origin
        X = x
        Y = y + H - h

        angle=(atan(X/d))*180/3.1415

        print("Horizontal Angle is :", angle)
        return angle

while True:
        video = cv2.VideoCapture('household-heating-fires-
1.jpg')
        ret,frame = video.read()
        if ret== False:
            break

        frame = cv2.resize(frame, (960, 540))

        blur = cv2.GaussianBlur(frame, (21, 21), 0)
        hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

        lower = [18, 50, 50]
        upper = [35, 255, 255]
        lower = np.array(lower, dtype="uint8")
        upper = np.array(upper, dtype="uint8")

        mask = cv2.inRange(hsv, lower, upper)
        output = cv2.bitwise_and(frame, hsv, mask=mask)
        no_red = cv2.countNonZero(mask)

        img = np.copy(frame)
        mask = operate(frame)
        white_loc = find_white_loc(mask)
        centre = find_centre2(white_loc)

        if int(no_red) > 15000:
            print("FIRE DETECTED")
            print(centre)
            if int(centre[0]) > 0 :
                print("1")
            if int(centre[0]) == 0 :
                print("1")
            if int(centre[0]) < 0 :
```

```
                print("-1")

        cv2.circle(frame,centre,60,(255,0,255),2)
        cv2.imshow('frmae',frame)
        cv2.imshow('output',output)
        if centre!=0:
                        adj_centre=(centre[0]-320,-
centre[1]+240)
                        if centre!= None:
                            print(adj_centre)
                            img =
cv2.circle(img,centre,40,(255,0,0),3)
                            img =
cv2.circle(img,centre,0,(255,0,0),5)
                            img =
cv2.circle(img,(320,240),0,(255,0,0),5)


        else:

arduino_out="X"+str(360)+"Y"+str(360)+"Z"+str(0)

        arduino.write(bytes(arduino_out, 'utf-8'))
        arduino.flush()
        time.sleep(0.5)
        print(arduino_out)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

video.release()
cv2.destroyAllWindows()
```