Cosmic Team

EVA movements algorithm and programming

Nada Bzoor
PROGRAMMER

Contents

Figures

1. Introduction

The vehicle EVA is designed using two microcontrollers: Raspberry Pi, and Arduino. The Arduino is programmed on a Linux environment, which is installed on Raspberry pi, And it is connecting to the Raspberry Pi using a serial port, and the platform that is used for programming and uploading the code on Arduino is Arduino IDE ARM version.

Depending on the previous, the C++ language is used for Arduino, and the Python language is used for Raspberry pi.

the C++ language is used to program the Arduino microcontroller, to control the movement of the motor, steering wheel, and sensors (Ultrasonic) while the python is used for image processing depending on the suitable library so that the Open CV v4.5.1 library has been downloaded suitable to python. it is used for processing the image coming from the camera, clustering the red and green pillars, and sending commands for each case according to the color.

The code consist of three main section which is :

The first section consists of the libraries that used to control the servo motor movement, and the time library.

The second section consists of global variables for the main and sub-functions, that carried names with programmatic significance, such as the normalSpeed variable to determine the vehicle's speed, and the turnSpeed .... etc.

The third section consists of functions, and the most important of these functions are : turnLeft (), turnRight (), the direction to the left and the right are determined by Ultrasonic Sensors, and this is performed according to the following: When the vehicle turns counter-clockwise, the left Ultrasonic

sensor value is large so the vehicle turns to the left , and when the vehicle turns in clockwise, the right one is large so it turns to the right. also, the getForward (), getRight (), and getLeft () functions are used to read each ultrasonic values separately.

The required configurations are adjusted within the setup() function and the main code was also written inside the loop () function. and it is also added A function for tuning the vehicle's path so that it is as much as possible in the middle of the path, in addition to other functions such as controlling the rotation of the servo motor that is used in the steering and so on.

The procedures which are mentioned above are used for the game without pillars, but in the case of the presence of the pillars, some fundamental changes have been made to the code, which are:

The rotation method has been processed using the straightBack (), turnLeft(), turnRight() functions, in addition to adding a function to read the serial that takes its orders from the Raspberry Pi controller, in addition to adding the cases that the vehicle must do in the case of seeing the green and red pillars according to the requirements of the game. Other functions have also been added to achieve the rotation of the vehicle to any degree, as well as the angle of the steering.

2. Stage one “without pillars “:

In the absence of pillars, we have made an algorithm for the rotation of the vehicle to include all the positions that the vehicle can be in, which can rotate the vehicle in a smooth way.

This was done by calculating the angle of slope at which the vehicle is traveling and implementing the rotation based on this angle.

The angle of slope of the vehicle was calculated by measuring the distance from the outer tire when the vehicle started (tmp0), and after it traveled for a certain distance, we rounded it to

100 cm (D). We measured the distance from the outer tire (tmp1) and then found the angle of slope using the following equation:

$$Sin^{-1}\frac{\Delta tmp}{d} * \frac{180}{\pi} \qquad (1)$$

We multiply it by $\frac{180}{\pi}$ because in C++ it gives the value of Sin^{-1} in radians, so we multiply to convert it to degrees.

So, if the vehicle is tilted to the right, when it turns 90 degrees, it is not straight, so depending on its angle of inclination, if the vehicle rotates 90 degrees in addition to the angle of its inclination, in this case it becomes straight after the rotation. (Figure 1)

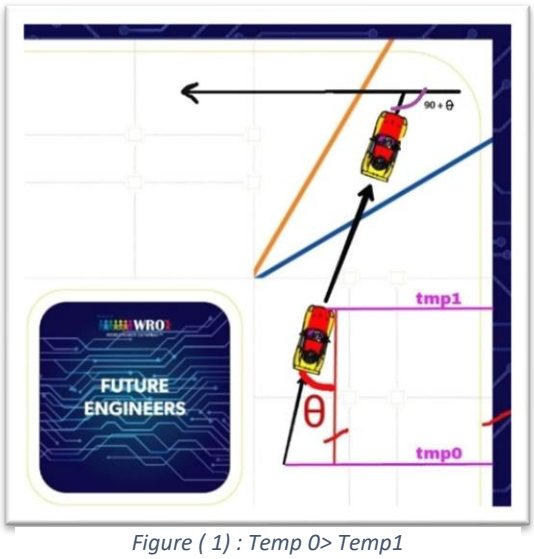

Figure (1) : Temp 0> Temp1

But if the vehicle is tilted to the left, the vehicle needs to rotate 90 degrees and subtract its angle of slope to become straight after the rotation.(Figure 2)

And if it is straight without any angle of inclination, then the vehicle rotates a full 90 degrees to become straight.

In some cases, due to an error reading the sensors, theta becomes greater than 10, but this is an unacceptable value, so we convert any value greater than 10 to zero, so the vehicle moves in a straight line

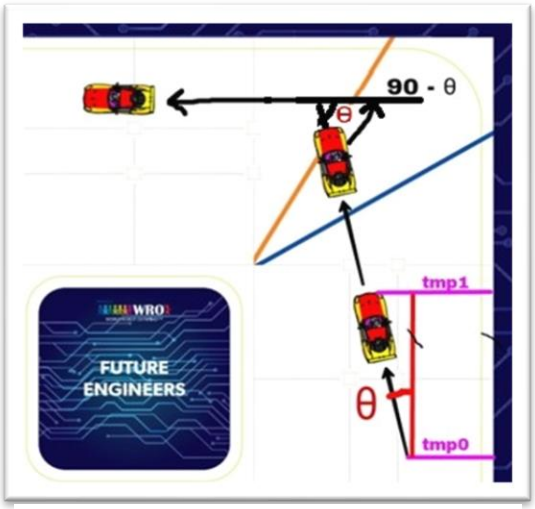

Figure (2) : Temp 0< Temp1

<u>And that was done by the following code:</u>

```
void turnLeft(int delta)
{


    double drvEncoderVar = 84+ delta ;
    Stearing.write(30);
    drvEncoderSpeed(drvEncoderVar,turnSpeed);
    resetStearingi();
    MyDelay(2);
    counter ++ ;
    encoderWalkStraight(50,999,normalSpeed);

}
```

3. Stage two “with pillars “:

In the case of the presence of pillars, we have created an algorithm for rotation, considering all the positions that the vehicle can be in.

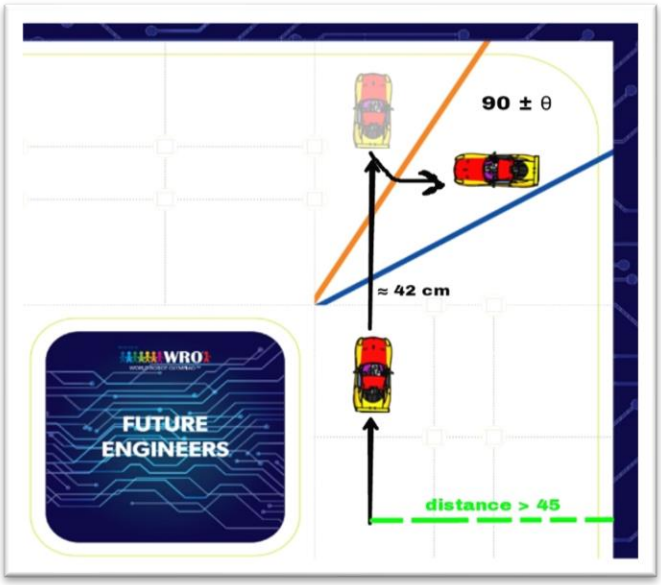

Figure (3): Straight Back Turn

The first case is that while driving the vehicle, the distance between it and the outer frame is greater than 45 cm, so we use the straightBack() function (Figure 3), which is implemented as follows :

The vehicle walks half the distance to the front wall, and it is rounded to 42 cm, then turns back by 90 degrees plus or minus the angle of slope(theta) and it becomes straight, so he completes reading the case in front of it and implementing it.We found that each degree in radians when the steering wheel is set at 45 degrees is equal to the distance of 0.76 cm , so that the 90 degrees is 65 cm .

The straightBack() function:

```
void straightBack (int delta)
{  int deltaStrightDifferenceVar = deltaStrightDifference(15);
  if (LastState == 0 || LastState == 1 ){
      encoderWalkStraight(50,999);
      encoderTurnBack(65 +delta+3);
      resetStearing();
      MyDelay(2);
      revEncoder(10);
      }
      else if (LastState == 5 )
      {
      encoderWalkStraight(55,999);
      encoderTurnBack(65 +delta+3);
      resetStearing();
      MyDelay(2);
      revEncoder(15);
        }
counter++;
}
```

As for the second case, if the distance between the vehicle and the outer frame is less than or equal to 40 cm, then we use the turnLeft() or turnRight() functions (Figure 4), which are implemented as follows:

We drew a circle in the ring with a radius = 42 cm, which is half the length of the distance in the corner area, then we calculated the length of the arc in the corner area, and it was according to the following equation:

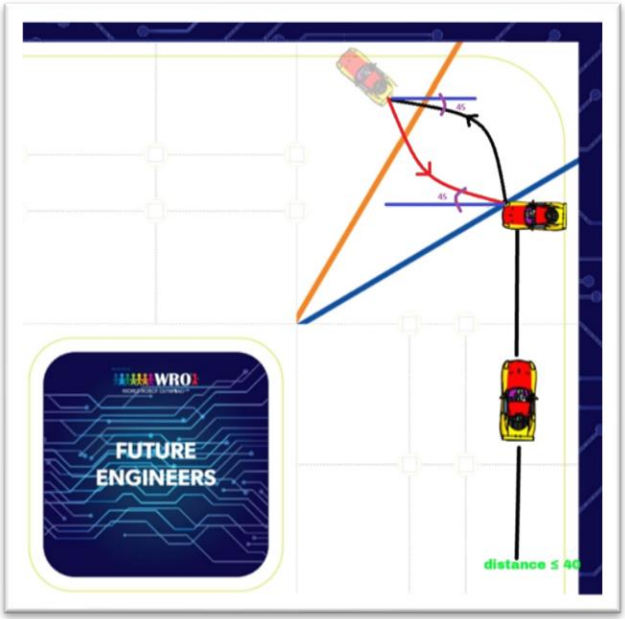

Figure (4): Turn Left

$$Arc = \ r * \frac{\pi}{4} \qquad (2)$$

Arc : arc length r : radius

In the calculations, it was found that the length of one arc is equal to 32 cm, and the length of the two arcs together is equal to 65 cm, but through experience, we have established that the length of the first arc = 37.25 and the second arc = 32.25, and this difference was due to the slope of the vehicle from the beginning, so it became almost inclined by 5 cm.

The rotation process is by walking the vehicle, the first arc, while moving the steering wheel 45 degrees 37 cm, and then returning to the back to walk the second arc 32 cm.

The turnLeft() and turnRight() functions:

```
void turnRight(int delta)
{
    Stearing.write(135);
    drvEncoder(38.25);
    Stearing.write(45);
    double revEncoderVar = 36.25 + delta * 0.76 ;
    revEncoder(revEncoderVar);
    resetStearingii();
    MyDelay(2);
    drvEncoder(10);
    Run(0);
    counter ++ ;}


void turnLeft(int delta)
{
    Stearing.write(45);
    drvEncoder(37.25);
    Stearing.write(135);
    double revEncoderVar = 32.25 + delta * 0.76 ;
    revEncoder(revEncoderVar);
    resetStearingi();
    MyDelay(2);
    drvEncoder(10);
    Run(0);
    counter ++ ;
}
```

That method was used in the counterclockwise, but in clockwise we found a problem in the steering so we made solution for that by making the vehicle walks in a straight line, then turn to the right and return and becomes straight (Figure 5), and the action will be as the following code ：

```
void straightBack (int delta)
{  int deltaStrightDifferenceVar = deltaStrightDifference(15);
  if (LastState == 0 || LastState == 1 ){
       encoderWalkStraight(50,999);
       Stearing.write(45);
       encoderWalkStraight(27,999);
       encoderTurnBack(35 +delta);
       resetStearingi();
       MyDelay(2);
       revEncoder(15);
       }
       else if (LastState == 5 )
       {
       encoderWalkStraight(63,999);
       Stearing.write(45);
       encoderWalkStraight(27,999);
       encoderTurnBack(50 +delta);
       resetStearingi();
       MyDelay(2);
      revEncoder(15);
         }
counter++;
}
```

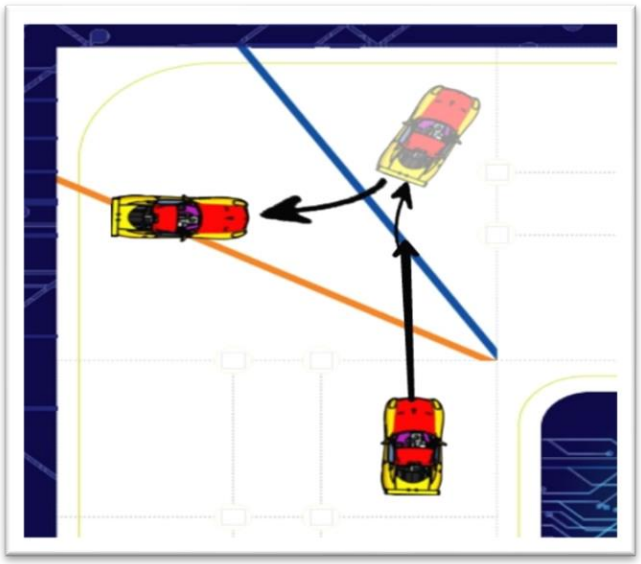

Figure (5) : Enhanced Straight Back

actions for red and green colors:

In case of seeing the red color, the vehicle must turn on its right, so we made an algorithm for that as the following (Figure 6):

We calculated d in Pythagorean theorem, and we move the vehicle depending on it.

$$\sqrt{a^2 + b^2} = c$$

We assume a= b so the equation become

$$\sqrt{a^2 + a^2} = c$$

$$a\sqrt{2} = c \qquad (3)$$

```
if (state == "1") {
if (counterClockWise == 1 ){

int getRightVar = getRight() ;

double d= 1.414 * getRightVar - 7;
Stearing.write(135);
drvEncoder((d/4));
drvEncoder((d/5));
resetStearingi();
drvEncoder((d/3));
drvEncoder((d/10));
Stearing.write(45);
drvEncoder((d/3));
drvEncoder((d/8));
resetStearingiL();
drvEncoder(10);
LastState = 1;
}
```

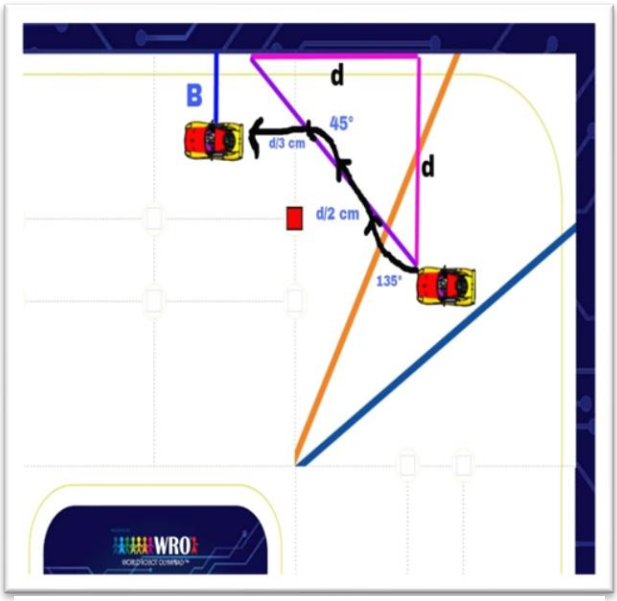

Figure (6) : CCW Red Action

```
else if (counterClockWise == 2 ){
   int getLeftVar = getLeft() ;
   double d= 127-(1.414 * getLeftVar );
   Stearing.write(135);
   drvEncoder((d/2));
   resetStearingi();
   drvEncoder((d*2/5)-5);
   Stearing.write(45);
   drvEncoder((d*2/4)+2);
   resetStearingi();
   drvEncoder(10);
   LastState = 1;
  }
}
 }
```

And in the case of seeing the green color, the vehicle must on its left, so we made an algorithm for that as the following (Figure 7):

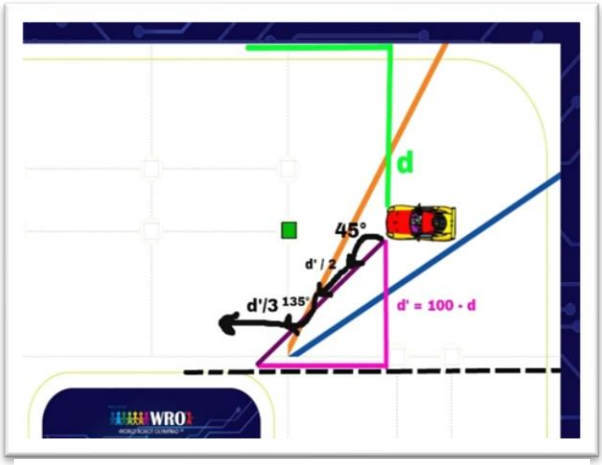

Figure (7): CCW Green Action

```
if (state == "5")
    {
      if (counterClockWise == 1 )  {
     int getRightVar=getRight();
     double d= 135- (1.414 * getRightVar );
     Stearing.write(45);
     drvEncoder(d/2);
     resetStearingi();
     drvEncoder((d/4));
     drvEncoder((d/14));
     Stearing.write(125);
     drvEncoder((d/4));
     drvEncoder((d/8));
     LastState = 5 ;
     resetStearingiR();
     drvEncoder(10);
    }


    else if (counterClockWise == 2 ) {
     int getLeftVar = getLeft() ;
     double d=(1.414 * getLeftVar )-7;
     Stearing.write(45);
     drvEncoder((d/2));
      drvEncoder((d/8));
     resetStearingi();
     drvEncoder((d/2));
     Stearing.write(135);
     drvEncoder((d/5));
     drvEncoder((d/5));
     LastState = 5 ;
     resetStearingii();
     drvEncoder(10);
    }
    }
```

4. Tuning:

We created a function for tuning the vehicle's path so that it is as much as possible in the middle of the path, and that was by measuring the distance of the vehicle from the outer or inner edge :

When the distance is less than 15 from the outer edge, the vehicle moves to the left, and if it is from the inner edge, it moves to the right.

(Figure 8), (Figure 9)

also, if the distance is greater than 22 From the outer edge, the vehicle will move to the right, and if it is from the inner edge, it will move to the left. And that is the code for the tuning :

```
void encoderWalkStraight(int distance,int side){
   if (counterClockWise == 1 ) {
   int getRightVar =0;
   drvEncoder(distance /4);
   if (LastState == 1 )
   {
       drvEncoder(distance /4);
       getRightVar = getRight();
       if (side !=999 )
       {
          if
(getRightVar<17){  Stearing.write(45);MyDelay(3);Stearing.write(120);MyDelay(2);resetStearingi();}
          else if
(getRightVar>22){  Stearing.write(120);MyDelay(2);Stearing.write(60);MyDelay(2);resetStearingi();}
       }
       drvEncoder(distance /2);
       }
}
```

Figure (8) : Near Edges Tunning

```
      else if (LastState==5)
       {
        int  getLeftVar = getLeft();
        drvEncoder(distance /4);
        getLeftVar = getLeft();
        if (side !=999  )
            {
             if (getLeftVar<17
){  Stearing.write(120);MyDelay(1);Stearing.write(60);MyDelay(1);resetStearingiR();}
             }
        drvEncoder(distance /2);
        }
        else
        {
         drvEncoder(distance );
        }
```

Figure (9): Far Edges Tunning

5. DC Encoder :

   At first, we had an encoder 60 RPM, but we replace it with an encoder 130 RPM which is faster , and smoother .

   Using the first encoder we found that each cm equals to 29 pulses in the encoder ,

   But when we replace with the 130 RPM , the number of pulses varies according to the distance traveled, and therefore we have a relationship that relates the distance traveled to the number of pulses, which is the following equation :

$$0.08 \times x + 20$$

$$x = \frac{cm - 20}{0.08} \qquad (4)$$

X = the pulses value

And it is true on the values greater than 20 cm. and this is , The code:

```
void drvEncoderSpeed(int cm,int speed0)
{
  encoderPos=0;
if (cm>25){
float enc = (cm-20)/0.077;

while (encoderPos < int(enc))
{
  Run(speed0);
  }
}
else
{
float enc = (cm)/0.23;
while (encoderPos < int(enc))
{
  Run(speed0);
  }
}
}
```

6. Camera coding:

The pre-existing Python language on Raspberry Pi's operating system was used to operate the camera and process the images taken from it.

First , we converted the color scheme of the image from RGB to HSV.

```
hsvFrame = cv2.cvtColor(imageFrame, cv2.COLOR_BGR2HSV)
```

Second , we created a color mask which we determined the color range of green and red in the HSV system.

```
# Set range for red color and
# define mask
red_lower = np.array([170, 100, 50], np.uint8)
red_upper = np.array([180, 255, 255], np.uint8)
red_mask = cv2.inRange(hsvFrame, red_lower, red_upper)

# Set range for green color and
# define mask
green_lower = np.array([50,80, 50], np.uint8)
green_upper = np.array([80, 255, 255], np.uint8)
green_mask = cv2.inRange(hsvFrame, green_lower, green_upper)
```

Third , we used functions supported by the OpenCV library to extract the regions that contain these masks in the image.

```
# For red color
red_mask = cv2.dilate(red_mask, kernal)
res_red = cv2.bitwise_and(imageFrame, imageFrame,
                          mask=red_mask)

# For green color
green_mask = cv2.dilate(green_mask, kernal)
res_green = cv2.bitwise_and(imageFrame, imageFrame,
                            mask=green_mask)
```

Fourth , we used functions to extract the area depending on these masks, taking into account their rectangular shape.

```
# Creating contour to track red color
contours, hierarchy = cv2.findContours(red_mask,
                                       cv2.RETR_TREE,
                                       cv2.CHAIN_APPROX_SIMPLE)
# Creating contour to track green color
contours, hierarchy = cv2.findContours(green_mask,
                                       cv2.RETR_TREE,
                                       cv2.CHAIN_APPROX_SIMPLE)
```

Fifth , we calculated the area for these masks and ignored areas less than 300 pixels

```
# FOR RED COLOR
for pic, contour in enumerate(contours):

    area = cv2.contourArea(contour)
    redarea = area
    pt1 = (320, 0)
    pt2 = (320, 480)

    color = (255, 255, 0)
    cv2.line(imageFrame, pt1, pt2,color)
    if (area > 300):
        redcct=redcct+1
        x, y, w, h = cv2.boundingRect(contour)
     redarea = w * h
       # print (w)
        dd= int(Distance_finder(60.16,5,w/10))
        pt1 = (x, y)
        pt2 = (320, 480)
        if x>320 :
            pt1 = (x+w, y)
        pt3 = (320,y)
       # print (pt1)
      #  slope = (480-y) / (320 -x)
        color = (0, 0, 255)
        cv2.line(imageFrame, pt1, pt2,color)
        cv2.line(imageFrame, pt3, pt1,color)
        imageFrame = cv2.rectangle(imageFrame, (x, y),
                                   (x + w, y + h),
                                   (0, 0, 255), 2)


        z=0
        if x>320 :
            z=x-320+w
        else:
            z=320-x
        theta = math.degrees(math.atan((z/10)/dd))
        cv2.putText(imageFrame, "L="+str(z/10) , (x+int((z/2)), y+30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.0,
                    (0, 255, 255))
        cv2.putText(imageFrame, "R d="+str(dd) +" @="+str(int(theta)), (x, y),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.0,
                    (0, 255, 255))
```

.

```
 FOR GREEN COLOR
or pic, contour in enumerate(contours):

   area = cv2.contourArea(contour)
   greanarea = area
   if (area > 300):
       greencct=greencct+1
       x, y, w, h = cv2.boundingRect(contour)
       greanarea = h * w
       pt1 = (x, y)
       pt2 = (320, 480)
       if x>320 :
           pt1 = (x+w, y)
       pt3 = (320,y)
      # slope = (480-y) / (320 -x)
       color = (0, 255, 0)
       cv2.line(imageFrame, pt1, pt2,color)
       cv2.line(imageFrame, pt3, pt1,color)
       dd=int( Distance_finder(60.16,5,w/10))
       imageFrame = cv2.rectangle(imageFrame, (x, y),
                                  (x + w, y + h),
                                  (0, 255, 0), 2)


       z=0
       if x>320 :
           z=x-320 +w
       else:
           z=320-x
       theta = math.degrees(math.atan((z/10)/dd))

       cv2.putText(imageFrame, "L="+str(z/10) , (x+int(z/2), y+30),
                   cv2.FONT_HERSHEY_SIMPLEX, 1.0,
                   (0, 255, 255))
       cv2.putText(imageFrame, "G d="+str(dd) +" @="+str(int(1)), (x, y),
                   cv2.FONT_HERSHEY_SIMPLEX,
                   1.0, (0, 255, 0))
```

Sixth , we create global variables to count the number of masks of both colours.

```
counter = counter +1
mingreen=10
minred =10
```

Seventh , the least possible number of these masks of both colors was used to repeat 10 times to reduce the noise that might occur during the live video.

```
if greencct < mingreen :
    mingreen =greencct
if redcct < minred :
    minred =redcct
```

Eighth , we connected the Raspberry Pi controller to the Arduino through the serial port including the serial library.

```
ser = serial.Serial('/dev/ttyACM0', 115200, timeout=1)
ser.reset_input_buffer()
```

Ninth , we sent the commands through the serial port of the Arduino so that the Arduino takes the appropriate action.