Università degli Studi di Napoli Federico II

Final project

Scuola Politecnica e delle Scienze di Base

"Visione Computazionale"

Done by:

Víctor Borrero López - Matricola N.0001/14078

David Romero Pastor – Matricola N.0001/14079

Index

1. Objective

In this project the main objective is the Spanish vertical traffic signs detection, organised this way:

- **Red**: prohibition, danger or yield. It can be octogonal, a triangle or a circle.
- **Yellow**: working area. It can be a circle,a triangle or a rectangle.
- **Blue**: recommendation or obligation. It can be a circle or a rectangle.
- **White**: end of prohibition. It can be a circle or a triangle.

First of all, focus will be on detection colors. Once it's done, the next step consists of differentiating between triangles, rectangles or squares and circles.

At this point it's made the assumption that all detected traffic signs are being seen already spun. This means it's considered the signs are seen in its perfect shape, correctly placed.

2. Code explanation

The structure of the code will consist of:

- Creating a function where you insert the color you want to study and that makes the traffic sign detection.
- Main code (where all the color will be studied)
- Initialized parameters.

First of all, libraries must be inserted.

```
2 import time
3 import random as rng
4 import cv2 as cv
5 import numpy as np
6 import math
```

- **<u>Initialized parameters</u>**

Here are introduced all the color values, with their own inferior and superior limits, so that the study can be more successful. Each color has two limits, each one with three values, that are the HSV values (Hue, Saturation, Value), which determines the color it is. There is one more color, called "null" (is basicly black), whose use will be explained later.

Also a kernel is created to apply the masks and to reduce the noise.

```
11 #Establecemos el rango de colores que vamos a detectar
12 #En este caso de verde oscuro a verde-azulado claro
13 sensivity=15
14 yellow=30
15 rojo_bajos_0 = np.array([0, 100, 100], dtype=np.uint8)
```

```
16 rojo_altos_0 = np.array([sensivity, 255, 255], dtype=np.uint8)
17 rojo_bajos_1 = np.array([180-sensivity, 100, 100], dtype=np.uint8)
18 rojo_altos_1 = np.array([180, 255, 255], dtype=np.uint8)
19
20 azul_bajos = np.array([100, 65, 75], dtype=np.uint8)
21 azul_altos = np.array([130, 255, 255], dtype=np.uint8)
22
23 blanco_bajos = np.array([0, 0, 255-sensivity], dtype=np.uint8)
24 blanco_altos = np.array([180, sensivity, 255], dtype=np.uint8)
25
26 amarillo_bajos = np.array([yellow-sensivity, 100, 255], dtype=np.uint8)
27 amarillo_altos = np.array([yellow+sensivity, 100, 255], dtype=np.uint8)
28
29 null=np.array([0, 0, 0], dtype=np.uint8)
30 #Crear un kernel de '1' de 3x3
31 kernel = np.ones((3,3),np.uint8)
32
```

- **Function 'Detection'**

First of all, the function is defined, with 6 inputs.

```
33 def detection(color_low,color_high,color_low_plus,
color_high_plus,color,imagen):
```

The parameters "**color_low**" and "**color_high**" are the inferior and superior limits of the interval of each color.

The parameter "**color**" will be the same color of the one which is going to be studied.

The parameter "**imagen**" will be the image we introduce in the function to see if there are traffic signs in it.

Finally, the parameters "**color_low_plus**" and "**color_high_plus**" will only be used in the case of color red. That's because we have two intervals where red can be detected, so we take those two intervals and unite them, as we can see in lines from **37** to **39**. Two masks are created for each interval and them both are connected. In the case of the rest of colors, these both parameters will be "null" (basicly black).

```
34 #Crear una mascara con solo los pixeles dentro del rango de verdes
35 mask = cv.inRange(hsv, color_low, color_high)
36
37 if color_low_plus[0]!=0 and color_low_plus[1]!=0 and color_low_plus[2]!=0:
38 mask2=cv.inRange(hsv, color_low_plus, color_high_plus)
39 mask=cv.bitwise_or(mask,mask2)
40
```

Here a transformation is applied so that noise can be reduced (or even deleted).

```
41 #Se aplica la transformacion: Opening
42 trans1 = cv.morphologyEx(mask,cv.MORPH_OPEN,kernel)
43 trans2 = cv.morphologyEx(trans1,cv.MORPH_CLOSE,kernel)
44 #trans3 = cv.morphologyEx(mask,cv.MORPH_HITMISS,kernel)
```

```
45
46 src=trans2
```

Here the code tries to find contours in the mask created.

```
47 # Find contours
48 contours, hierarchy = cv.findContours(src, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
```

Here moments are got, which is useful to calculate the detected contours areas. In this case, only contours with areas between the values 1000 and 30000 are studied. Finally, depending on the area obtained fits in the limits, "v" will be 1. "V" is a variable created to study if the results are correct or not (meaning if the color to study has been detected or not).

```
49 # Get the moments
50 mu = [None]*len(contours)
51 v = [0]*len(contours)
52 for i in range(len(contours)):
53 mu[i] = cv.moments(contours[i])
54 if mu[i]['m00']>3000 and mu[i]['m00']<75000:
55 v[i]=1
56
```

Here mass centers are calculated, only if before areas fit in the limits ("**v**" value equal to 1).

```
57 # Get the mass centers
58 mc = [None]*len(contours)
59 for i in range(len(contours)):
60 # add 1e-5 to avoid division by zero
61 if v[i]==1:
62 mc[i] = (mu[i]['m10'] / (mu[i]['m00'] + 1e-5), mu[i]['m01'] /
(mu[i]['m00'] +
1e-5))
63
```

Here contours are drawn (again, only if "**v**" is equal to 1).

```
64 # Draw contours
65 drawing = np.zeros((len(src), len(src[0]), 3), dtype=np.uint8)
66 drawing=imagen
67 for i in range(len(contours)):
68 if v[i]==1:
69 cv.drawContours(drawing, contours, i, (255,0,255), 1)
70
```

Here contours are approximated to polygons. In addition, bounding rects and circles are obtained. In this piece of code it's studied if there's a circle. The way it's done is creating an ellipse that best fits the contour obtained, and then, if both areas (ellipse and contour obtained) are almost equals, then a circle has been detected. If a circle is finally detected, the variable" "**v**" is set to 2 and "CIRCULO_SIGNAL" is printed on screen.

```
71 # Approximate contours to polygons + get bounding rects and circles
72 contours_poly = [None]*len(contours)
```

```
73 boundRect = [None]*len(contours)
74 minEllipse = [None]*len(contours)
75 minRect = [None]*len(contours)
76 N=15
77 for i,cin enumerate(contours):
78 contours_poly[i] = cv.approxPolyDP(c, 0.03*cv.arcLength(c,True), True)
79 boundRect[i] = cv.boundingRect(contours_poly[i])
80 if v[i]==1:
81 if c.shape[0]>5:
82 minEllipse[i] = cv.fitEllipse(c)
83 minRect[i] = cv.minAreaRect(c)
84 area_ellipse=minEllipse[i][1][0]*minEllipse[i][1][1]
85 area_rect=minRect[i][1][0]*minRect[i][1][1]
86 if math.fabs(area_ellipse-area_rect)<150 and
math.fabs(minEllipse[i][0][0]-minRect[i][0][0])<3 and
math.fabs(minEllipse[i][0][1]-minRect[i][0][1])<3:
87 v[i]=2
88 print("CIRCULO SIGNAL")
89 cv.ellipse(drawing, minEllipse[i], (0,255,0), 2)
90
```

If a circle is not obtained then all other cases are studied. As the command "**cv.approxPolyDP**" returns the number of vertexes, with that value the shape of the signs is determined.

About triangles, once three vertexes are detected and the color is yellow, red or white (Spanish signs that are triangles), then it's studied if this detected triangle is equilateral.

The parameters "**lato1**","**lato2**","**lado3**" are the values of the sides of the triangle, compared between themselves to determine if it is equilateral or not. If a triangle is finally detected, the variable "**v**" is set to 3 and "TRIANGULO_SIGNAL" is printed on screen.

```
91 if v[i]==1:
92 if len(contours_poly[i])==3 and (color==(255,255,0) or color==(0,0,255) or
color==(255,255,255)):
93 print("TRIANGULO")
94
lato1=(contours_poly[i][0][0][0]-contours_poly[i][1][0][0])*(contours_poly
[i][0][0][0]-contours_poly[i][1][0][0])+(contours_poly[i][0][0][1]-contour
s_poly[i][1][0][1])*(contours_poly[i][0][0][1]-contours_poly[i][1][0][1])
95
lato2=(contours_poly[i][1][0][0]-contours_poly[i][2][0][0])*(contours_poly
[i][1][0][0]-contours_poly[i][2][0][0])+(contours_poly[i][1][0][1]-contour
s_poly[i][2][0][1])*(contours_poly[i][1][0][1]-contours_poly[i][2][0][1])
96
lato3=(contours_poly[i][2][0][0]-contours_poly[i][0][0][0])*(contours_poly
[i][2][0][0]-contours_poly[i][0][0][0])+(contours_poly[i][2][0][1]-contour
s_poly[i][0][0][1])*(contours_poly[i][2][0][1]-contours_poly[i][0][0][1])
97 dif=10000
98 #print("lati:",lato1,lato2,lato3)
99 if math.fabs(lato1-lato2)<dif or math.fabs(lato2-lato3)<dif or
math.fabs(lato3-lato1)<dif:
100 v[i]=3
101 print("TRIANGULO SIGNAL")
102 cv.drawContours(drawing, contours_poly, i, (0,255,0), 2)
```

In the case of rectangles the same technique is applied. Once four vertexes are detected and the color is yellow, blue or white (Spanish signs that are rectangles), then the sides of the rectangle are analyzed and compared two by two. As it's done with triangles, if finally a correct rectangle is obtained, its contour is drawn and "CUADRADO SIGNAL" is printed on screen.Also the variable "**v**" is set to 4.

```
103 elif len(contours_poly[i])==4 and (color==(255,255,0) or color==(255,0,0)
or
color==(255,255,255)):
104 print("CUADRADO")
105
lato1=(contours_poly[i][0][0][0]-contours_poly[i][1][0][0])*(contours_poly
[i][0][0][0]-contours_poly[i][1][0][0])+(contours_poly[i][0][0][1]-contour
s_poly[i][1][0][1])*(contours_poly[i][0][0][1]-contours_poly[i][1][0][1])
106
lato2=(contours_poly[i][1][0][0]-contours_poly[i][2][0][0])*(contours_poly
[i][1][0][0]-contours_poly[i][2][0][0])+(contours_poly[i][1][0][1]-contour
s_poly[i][2][0][1])*(contours_poly[i][1][0][1]-contours_poly[i][2][0][1])
107
lato3=(contours_poly[i][2][0][0]-contours_poly[i][3][0][0])*(contours_poly
[i][2][0][0]-contours_poly[i][3][0][0])+(contours_poly[i][2][0][1]-contour
s_poly[i][3][0][1])*(contours_poly[i][2][0][1]-contours_poly[i][3][0][1])
108
lato4=(contours_poly[i][3][0][0]-contours_poly[i][0][0][0])*(contours_poly
[i][3][0][0]-contours_poly[i][0][0][0])+(contours_poly[i][3][0][1]-contour
s_poly[i][0][0][1])*(contours_poly[i][3][0][1]-contours_poly[i][0][0][1])
109 dif=10000
110 #print("lati:",lato1,lato2,lato3,lato4)
111 if math.fabs(lato1-lato3)<dif and math.fabs(lato2-lato4)<dif:
112 v[i]=4
113 print("CUADRADO SIGNAL")
114 cv.drawContours(drawing, contours_poly, i, (0,255,0), 2)
```

The same will be done for octagons. Once eight vertexes are detected and the color is red, the sides of the octagon are studied and compared them two by two. As it's done with triangles, if finally a correct rectangle is obtained, its contour is drawn and "OCTOGONO SIGNAL" is printed on screen. Also the variable "**v**" is set to 5.

```
115 elif len(contours_poly[i])==8 and color==(0,0,255):
116 print("OCTOGONO")
117
lato1=(contours_poly[i][0][0][0]-contours_poly[i][1][0][0])*(contours_poly
[i][0][0][0]-contours_poly[i][1][0][0])+(contours_poly[i][0][0][1]-contour
s_poly[i][1][0][1])*(contours_poly[i][0][0][1]-contours_poly[i][1][0][1])
118
lato2=(contours_poly[i][1][0][0]-contours_poly[i][2][0][0])*(contours_poly
[i][1][0][0]-contours_poly[i][2][0][0])+(contours_poly[i][1][0][1]-contour
s_poly[i][2][0][1])*(contours_poly[i][1][0][1]-contours_poly[i][2][0][1])
119
lato3=(contours_poly[i][2][0][0]-contours_poly[i][3][0][0])*(contours_poly
[i][2][0][0]-contours_poly[i][3][0][0])+(contours_poly[i][2][0][1]-contour
s_poly[i][3][0][1])*(contours_poly[i][2][0][1]-contours_poly[i][3][0][1])
120
lato4=(contours_poly[i][3][0][0]-contours_poly[i][4][0][0])*(contours_poly
```

```
[i][3][0][0]-contours_poly[i][4][0][0])+(contours_poly[i][3][0][1]-contour
s_poly[i][4][0][1])*(contours_poly[i][3][0][1]-contours_poly[i][4][0][1])
121
lato5=(contours_poly[i][4][0][0]-contours_poly[i][5][0][0])*(contours_poly
[i][4][0][0]-contours_poly[i][5][0][0])+(contours_poly[i][4][0][1]-contour
s_poly[i][5][0][1])*(contours_poly[i][4][0][1]-contours_poly[i][5][0][1])
122
lato6=(contours_poly[i][5][0][0]-contours_poly[i][6][0][0])*(contours_poly
[i][5][0][0]-contours_poly[i][6][0][0])+(contours_poly[i][5][0][1]-contour
s_poly[i][6][0][1])*(contours_poly[i][5][0][1]-contours_poly[i][6][0][1])
123
lato7=(contours_poly[i][6][0][0]-contours_poly[i][7][0][0])*(contours_poly
[i][6][0][0]-contours_poly[i][7][0][0])+(contours_poly[i][6][0][1]-contour
s_poly[i][7][0][1])*(contours_poly[i][6][0][1]-contours_poly[i][7][0][1])
124
lato8=(contours_poly[i][7][0][0]-contours_poly[i][0][0][0])*(contours_poly
[i][7][0][0]-contours_poly[i][0][0][0])+(contours_poly[i][7][0][1]-contour
s_poly[i][0][0][1])*(contours_poly[i][7][0][1]-contours_poly[i][0][0][1])
125 dif=10000
126 if math.fabs(lato1-lato5)<dif and math.fabs(lato2-lato6)<dif and
math.fabs(lato3-lato7)<dif and math.fabs(lato4-lato8)<dif:
127 v[i]=5
128 print("OCTOGONO SIGNAL")
129 cv.drawContours(drawing, contours_poly, i, (0,255,0), 2)
130
```

The next process is used to detect if there are smaller contours inside other contours, so that smaller ones are not detected and can't create a misunderstanding for the image processing. The way it is done is comparing the bounding rectangle which fits to the contour with the mass center of the object to study.

```
117 for i in range(len(contours)):
118 if v[i]>1:
119 j=0
120 aux=0
121 while aux==0 and j<len(contours):
122 if v[j]>1:
123 if i!=j and mc[i][0]>boundRect[j][0] and
mc[i][0]<boundRect[j][0]+boundRect[j][2] and mc[i][1]>boundRect[j][1]
andmc[i][1]<boundRect[j][1]+boundRect[j][3]:
124 if boundRect[i][2]*boundRect[i][3] >boundRect[j][2]*boundRect[j][3]:
125 v[j]=1
126 else:
127 v[i]=1
128 aux=1
129 j+=1
130
```

Finally, depending on the value of "v", it it is bigger than 1, that means that the detection has been successful and a red point in the mass center of the detected contour. The function returns the value of **src** (transformed mask without noise) and **drawing**, where all the contours drawn are saved.

```
145
146 for i in range(len(contours)):
147 if v[i]>1:
```

```
148 cv.circle(drawing, (int(mc[i][0]), int(mc[i][1])), 3, (0,255,0), -1)
149 cv.rectangle(drawing, (int(boundRect[i][0]), int(boundRect[i][1])), \
150 (int(boundRect[i][0]+boundRect[i][2]),
int(boundRect[i][1]+boundRect[i][3])),
(125,125,0), 1)
151
152 return drawing,src;
```

- **Main code**

The main code will be a loop where a picture is read with **cv.imread** and changed to hsv values with **cvt.Color.**

```
154 while(1):
155 #Capturamos una imagen y la convertimos de RGB -> HSV
156 #_, imagen = captura.read()
157 imagen=cv.imread('signal1.jpg',cv.IMREAD_COLOR)
158 hsv = cv.cvtColor(imagen, cv.COLOR_BGR2HSV)
159
```

Firstly, four calls to the function “detection” are done. Each one is a call with a different color, returning a different value of drawing and src. Finally, it’s printed on screen the picture with the green contours in case they’ve been detected (via imshow).

```
160 draw1,src1=detection(azul_bajos,azul_altos,null,null,(255,0,0),imagen)
161
draw2,src2=detection(rojo_bajos_0,rojo_altos_0,rojo_bajos_1,rojo_altos_1,(0,0
,255),draw1)
162
draw3,src3=detection(blanco_bajos,blanco_altos,null,null,(255,255,255),draw2)
163
draw4,src4=detection(amarillo_bajos,amarillo_altos,null,null,(255,255,0),draw
3)
164
165 draw=draw4
166 src=src1+src2+src3+src4
167
168 # Show in a window
169 cv.imshow('mask',src)
170 cv.imshow('Contours', draw)
171
172 #tecla = cv.waitKey(5) & 0xFF
173 #time.sleep(2)
174 #if tecla == 27:
175 # break
```

Finally, a loop created to stop the process anytime we want, just pressing Ctrl+C.

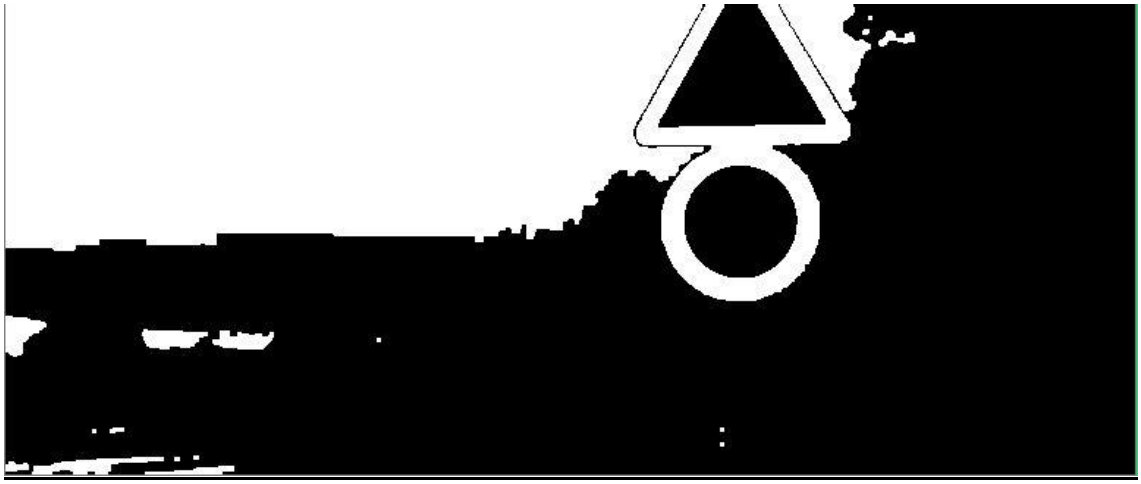
```
176 while(1):
177 tecla = cv.waitKey(5) &0xFF
178 if tecla == 27:
179 break
180
181 cv.destroyAllWindows()
```

3. Results

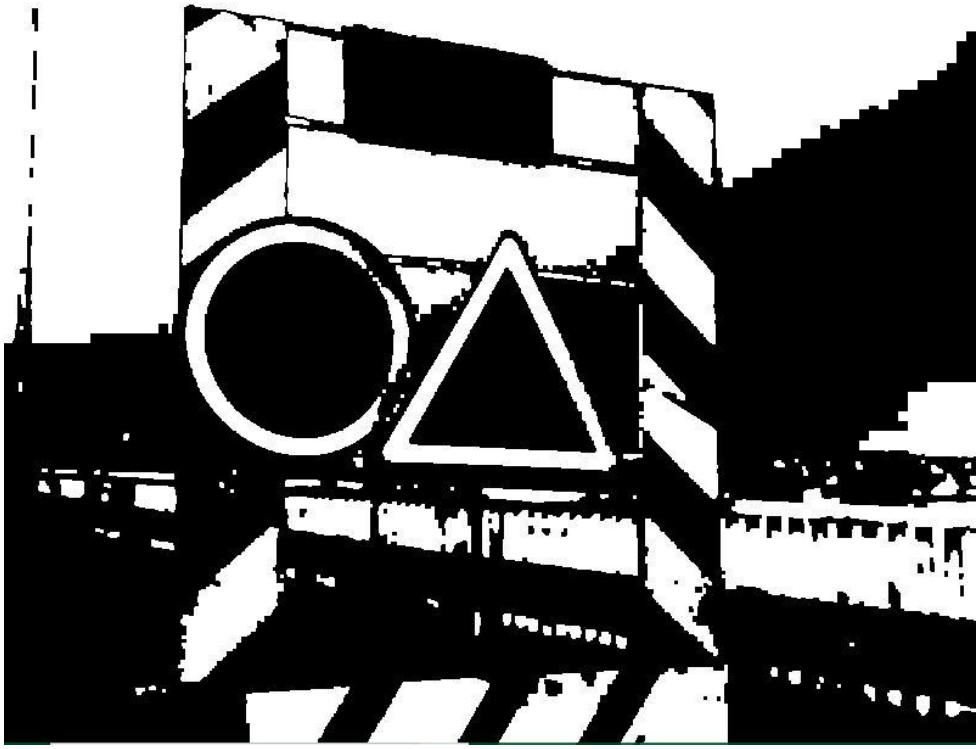
In this section, the results of the simulation will be presented. For each picture there will be two images obtained. One in white and black, where the white color indicates where one of the four colors has been detected. And another where the contours have been painted in purple and the signal detected have been marked in green and bounded by a blue box.

As expected, not every signal has been detected due to some situations that have not been taken into account. For the same reason, other elements are marked as a signal, not being the case, however this last case would be solved in the signal recognition part, not implemented in this case.

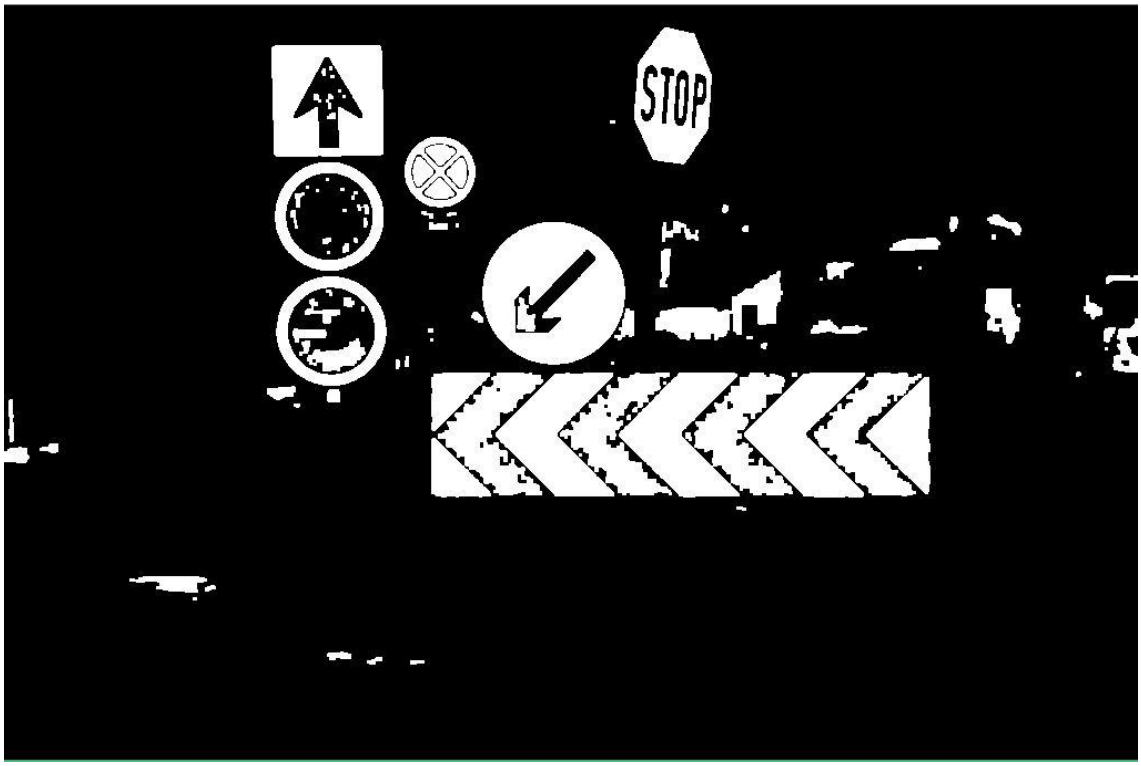
- **Image 1**



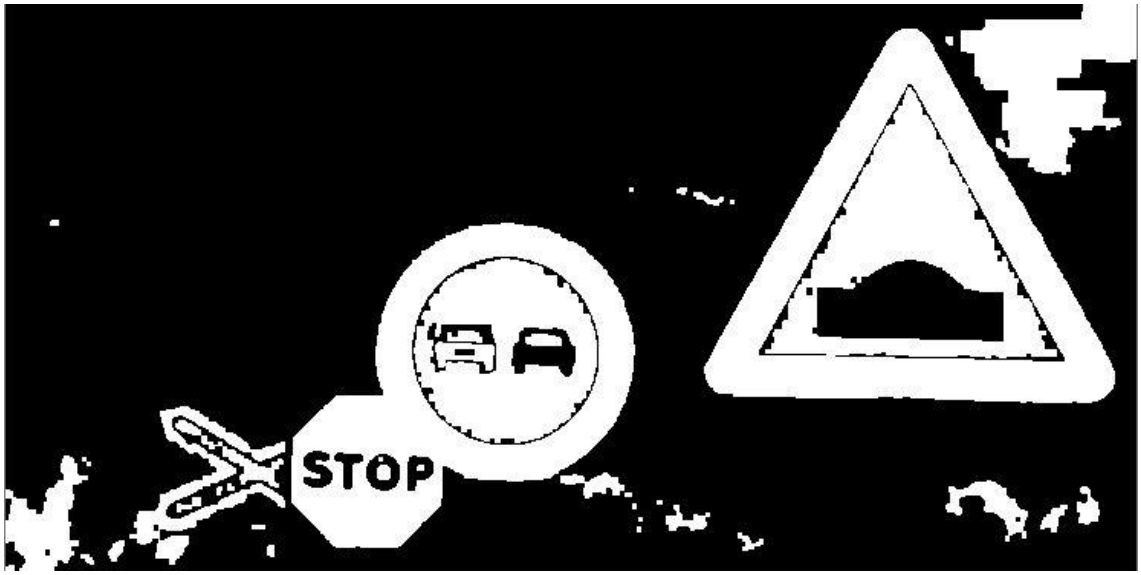
- **Image 2**



- **Image 3**



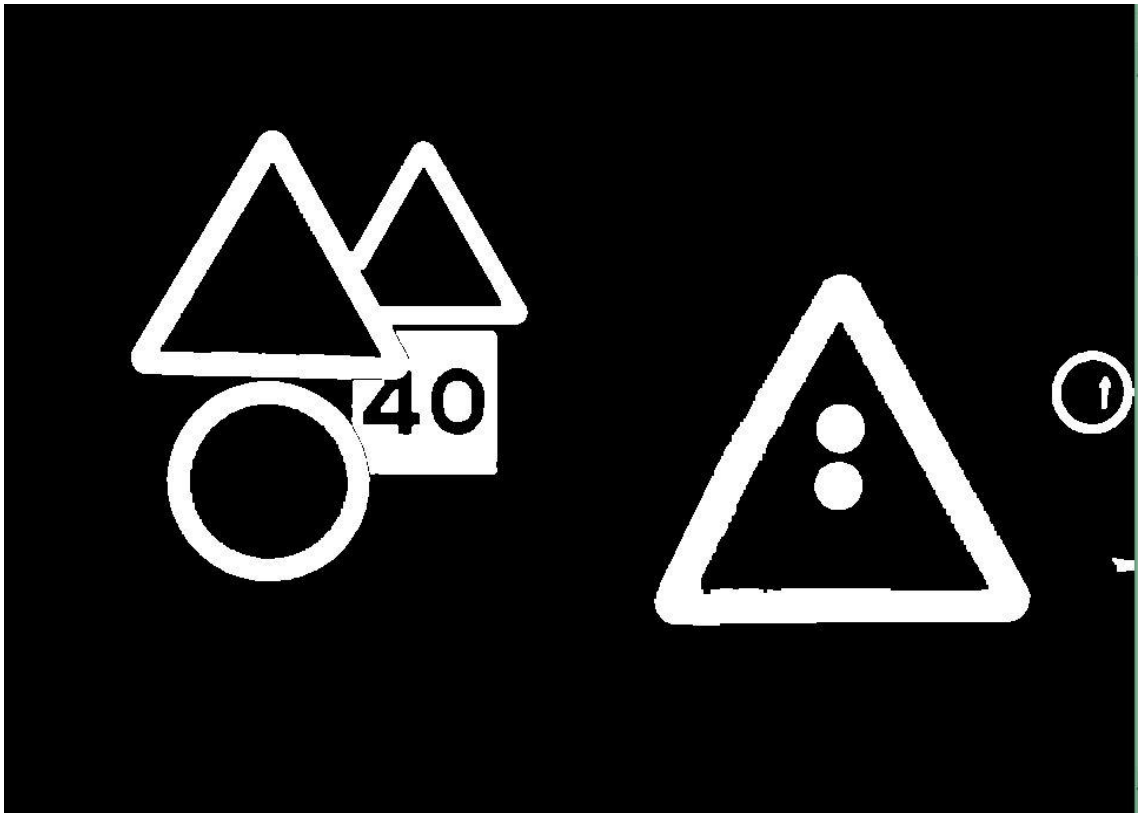
- **Image 4**



- **Image 5**


MAZATLAN Libre
EL SALTO
MEXICO 40
EL SALTO
MEXICO 40
SANTA LUCIA
MEXICO 40
MAZATLAN CUOTA
40
km/h
SALIDA

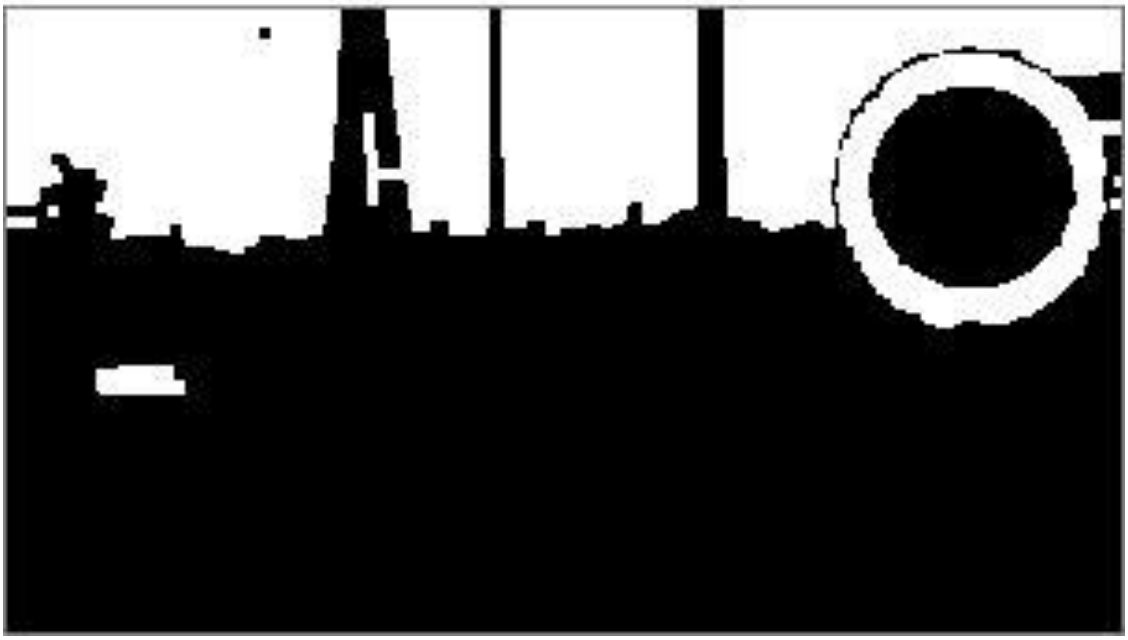
- **Image 6**



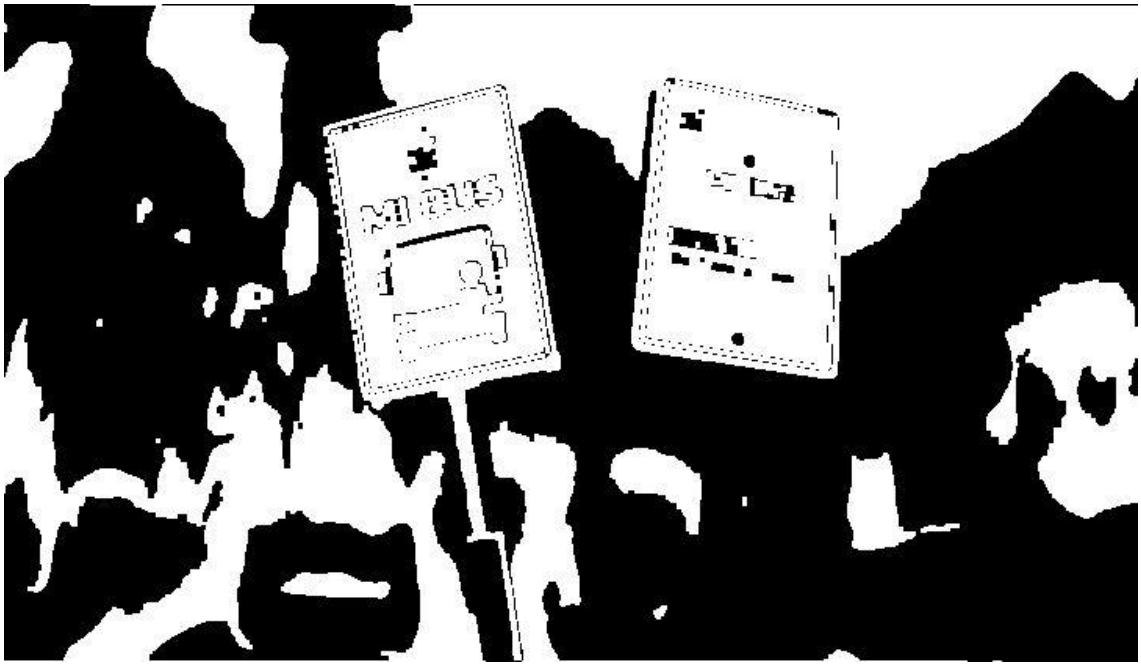
- **Image 7**

- **Image 8**

- **Image 9**



- **Image 10**



4. Full code

```
1
2 import time
3 import random as rng
4 import cv2 as cv
5 import numpy as np
6 import math
7
8 #Iniciamos la camara
9 captura = cv.VideoCapture(1)
10
11 #Establecemos el rango de colores que vamos a detectar
12 #En este caso de verde oscuro a verde-azulado claro
13 sensivity=15
14 yellow=30
15 rojo_bajos_0 = np.array([0, 100, 100], dtype=np.uint8)
16 rojo_altos_0 = np.array([sensivity, 255, 255], dtype=np.uint8)
17 rojo_bajos_1 = np.array([180-sensivity, 100, 100], dtype=np.uint8)
18 rojo_altos_1 = np.array([180, 255, 255], dtype=np.uint8)
19
20 azul_bajos = np.array([100, 65, 75], dtype=np.uint8)
21 azul_altos = np.array([130, 255, 255], dtype=np.uint8)
22
23 blanco_bajos = np.array([0, 0, 255-sensivity], dtype=np.uint8)
24 blanco_altos = np.array([180, sensivity, 255], dtype=np.uint8)
25
26 amarillo_bajos = np.array([yellow-sensivity, 100, 255], dtype=np.uint8)
27 amarillo_altos = np.array([yellow+sensivity, 100, 255], dtype=np.uint8)
28
29 null=np.array([0, 0, 0], dtype=np.uint8)
30 #Crear un kernel de '1' de 3x3
31 kernel = np.ones((3,3),np.uint8)
32
33 def detection(color_low,color_high,color_low_plus,
color_high_plus,color,imagen):
34 #Crear una mascara con solo los pixeles dentro del rango de verdes
35 mask = cv.inRange(hsv, color_low, color_high)
36
37 if color_low_plus[0]!=0 or color_low_plus[1]!=0 or color_low_plus[2]!=0:
38 mask2=cv.inRange(hsv, color_low_plus, color_high_plus)
39 mask=cv.bitwise_or(mask,mask2)
40
41 #Se aplica la transformacion: Opening
42 trans1 = cv.morphologyEx(mask,cv.MORPH_OPEN,kernel)
43 trans2 = cv.morphologyEx(trans1,cv.MORPH_CLOSE,kernel)
44 #trans3 = cv.morphologyEx(mask,cv.MORPH_HITMISS,kernel)
45
46 src=trans2
47 # Find contours
48 contours, hierarchy = cv.findContours(src, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
49 # Get the moments
50 mu = [None]*len(contours)
51 v = [0]*len(contours)
52 for i in range(len(contours)):
53 mu[i] = cv.moments(contours[i])
54 if mu[i]['m00']>3000 and mu[i]['m00']<75000:
55 v[i]=1
56
```

```
57 # Get the mass centers
58 mc = [None]*len(contours)
59 for i in range(len(contours)):
60 # add 1e-5 to avoid division by zero
61 if v[i]==1:
62 mc[i] = (mu[i]['m10'] / (mu[i]['m00'] + 1e-5), mu[i]['m01'] /
(mu[i]['m00'] +
1e-5))
63
64 # Draw contours
65 drawing = np.zeros((len(src), len(src[0]), 3), dtype=np.uint8)
66 drawing=imagen
67 for i in range(len(contours)):
68 if v[i]==1:
69 cv.drawContours(drawing, contours, i, (255,0,255), 1)
70
71 # Approximate contours to polygons + get bounding rects and circles
72 contours_poly = [None]*len(contours)
73 boundRect = [None]*len(contours)
74 minEllipse = [None]*len(contours)
75 minRect = [None]*len(contours)
76 N=15
77 for i,cin enumerate(contours):
78 contours_poly[i] = cv.approxPolyDP(c, 0.03*cv.arcLength(c,True), True)
79 boundRect[i] = cv.boundingRect(contours_poly[i])
80 if v[i]==1:
81 if c.shape[0]>5:
82 minEllipse[i] = cv.fitEllipse(c)
83 minRect[i] = cv.minAreaRect(c)
84 area_ellipse=minEllipse[i][1][0]*minEllipse[i][1][1]
85 area_rect=minRect[i][1][0]*minRect[i][1][1]
86 if math.fabs(area_ellipse-area_rect)<150 and
math.fabs(minEllipse[i][0][0]-minRect[i][0][0])<3 and
math.fabs(minEllipse[i][0][1]-minRect[i][0][1])<3:
87 v[i]=2
88 print("CIRCULO SIGNAL")
89 cv.ellipse(drawing, minEllipse[i], (0,255,0), 2)
90
91 if v[i]==1:
92 if len(contours_poly[i])==3 and (color==(255,255,0) or color==(0,0,255) or
color==(255,255,255)):
93 print("TRIANGULO")
94
lato1=(contours_poly[i][0][0][0]-contours_poly[i][1][0][0])*(contours_poly
[i][0][0][0]-contours_poly[i][1][0][0])+(contours_poly[i][0][0][1]-contour
s_poly[i][1][0][1])*(contours_poly[i][0][0][1]-contours_poly[i][1][0][1])
95
lato2=(contours_poly[i][1][0][0]-contours_poly[i][2][0][0])*(contours_poly
[i][1][0][0]-contours_poly[i][2][0][0])+(contours_poly[i][1][0][1]-contour
s_poly[i][2][0][1])*(contours_poly[i][1][0][1]-contours_poly[i][2][0][1])
96
lato3=(contours_poly[i][2][0][0]-contours_poly[i][0][0][0])*(contours_poly
[i][2][0][0]-contours_poly[i][0][0][0])+(contours_poly[i][2][0][1]-contour
s_poly[i][0][0][1])*(contours_poly[i][2][0][1]-contours_poly[i][0][0][1])
97 dif=10000
98 #print("lati:",lato1,lato2,lato3)
99 if math.fabs(lato1-lato2)<dif or math.fabs(lato2-lato3)<dif or
math.fabs(lato3-lato1)<dif:
100 v[i]=3
```

```
101 print("TRIANGULO SIGNAL")
102 cv.drawContours(drawing, contours_poly, i, (0,255,0), 2)
103 elif len(contours_poly[i])==4 and (color==(255,255,0) or color==(255,0,0)
or
color==(255,255,255)):
104 print("CUADRADO")
105
lato1=(contours_poly[i][0][0][0]-contours_poly[i][1][0][0])*(contours_poly
[i][0][0][0]-contours_poly[i][1][0][0])+(contours_poly[i][0][0][1]-contour
s_poly[i][1][0][1])*(contours_poly[i][0][0][1]-contours_poly[i][1][0][1])
106
lato2=(contours_poly[i][1][0][0]-contours_poly[i][2][0][0])*(contours_poly
[i][1][0][0]-contours_poly[i][2][0][0])+(contours_poly[i][1][0][1]-contour
s_poly[i][2][0][1])*(contours_poly[i][1][0][1]-contours_poly[i][2][0][1])
107
lato3=(contours_poly[i][2][0][0]-contours_poly[i][3][0][0])*(contours_poly
[i][2][0][0]-contours_poly[i][3][0][0])+(contours_poly[i][2][0][1]-contour
s_poly[i][3][0][1])*(contours_poly[i][2][0][1]-contours_poly[i][3][0][1])
108
lato4=(contours_poly[i][3][0][0]-contours_poly[i][0][0][0])*(contours_poly
[i][3][0][0]-contours_poly[i][0][0][0])+(contours_poly[i][3][0][1]-contour
s_poly[i][0][0][1])*(contours_poly[i][3][0][1]-contours_poly[i][0][0][1])
109 dif=10000
110 #print("lati:",lato1,lato2,lato3,lato4)
111 if math.fabs(lato1-lato3)<dif and math.fabs(lato2-lato4)<dif:
112 v[i]=4
113 print("CUADRADO SIGNAL")
114 cv.drawContours(drawing, contours_poly, i, (0,255,0), 2)
115 elif len(contours_poly[i])==8 and color==(0,0,255):
116 print("OCTOGONO")
117
lato1=(contours_poly[i][0][0][0]-contours_poly[i][1][0][0])*(contours_poly
[i][0][0][0]-contours_poly[i][1][0][0])+(contours_poly[i][0][0][1]-contour
s_poly[i][1][0][1])*(contours_poly[i][0][0][1]-contours_poly[i][1][0][1])
118
lato2=(contours_poly[i][1][0][0]-contours_poly[i][2][0][0])*(contours_poly
[i][1][0][0]-contours_poly[i][2][0][0])+(contours_poly[i][1][0][1]-contour
s_poly[i][2][0][1])*(contours_poly[i][1][0][1]-contours_poly[i][2][0][1])
119
lato3=(contours_poly[i][2][0][0]-contours_poly[i][3][0][0])*(contours_poly
[i][2][0][0]-contours_poly[i][3][0][0])+(contours_poly[i][2][0][1]-contour
s_poly[i][3][0][1])*(contours_poly[i][2][0][1]-contours_poly[i][3][0][1])
120
lato4=(contours_poly[i][3][0][0]-contours_poly[i][4][0][0])*(contours_poly
[i][3][0][0]-contours_poly[i][4][0][0])+(contours_poly[i][3][0][1]-contour
s_poly[i][4][0][1])*(contours_poly[i][3][0][1]-contours_poly[i][4][0][1])
121
lato5=(contours_poly[i][4][0][0]-contours_poly[i][5][0][0])*(contours_poly
[i][4][0][0]-contours_poly[i][5][0][0])+(contours_poly[i][4][0][1]-contour
s_poly[i][5][0][1])*(contours_poly[i][4][0][1]-contours_poly[i][5][0][1])
122
lato6=(contours_poly[i][5][0][0]-contours_poly[i][6][0][0])*(contours_poly
[i][5][0][0]-contours_poly[i][6][0][0])+(contours_poly[i][5][0][1]-contour
s_poly[i][6][0][1])*(contours_poly[i][5][0][1]-contours_poly[i][6][0][1])
123
lato7=(contours_poly[i][6][0][0]-contours_poly[i][7][0][0])*(contours_poly
[i][6][0][0]-contours_poly[i][7][0][0])+(contours_poly[i][6][0][1]-contour
s_poly[i][7][0][1])*(contours_poly[i][6][0][1]-contours_poly[i][7][0][1])
124
```

```
lato8=(contours_poly[i][7][0][0]-contours_poly[i][0][0][0])*(contours_poly
[i][7][0][0]-contours_poly[i][0][0][0])+(contours_poly[i][7][0][1]-contour
s_poly[i][0][0][1])*(contours_poly[i][7][0][1]-contours_poly[i][0][0][1])
125 dif=10000
126 if math.fabs(lato1-lato5)<dif and math.fabs(lato2-lato6)<dif and
math.fabs(lato3-lato7)<dif and math.fabs(lato4-lato8)<dif:
127 v[i]=5
128 print("OCTOGONO SIGNAL")
129 cv.drawContours(drawing, contours_poly, i, (0,255,0), 2)
130
131
132 for i in range(len(contours)):
133 if v[i]>1:
134 j=0
135 aux=0
136 while aux==0 and j<len(contours):
137 if v[j]>1:
138 if i!=j and mc[i][0]>boundRect[j][0] and
mc[i][0]<boundRect[j][0]+boundRect[j][2] and mc[i][1]>boundRect[j][1]
andmc[i][1]<boundRect[j][1]+boundRect[j][3]:
139 if boundRect[i][2]*boundRect[i][3] >boundRect[j][2]*boundRect[j][3]:
140 v[j]=1
141 else:
142 v[i]=1
143 aux=1
144 j+=1
145
146 for i in range(len(contours)):
147 if v[i]>1:
148 cv.circle(drawing, (int(mc[i][0]), int(mc[i][1])), 3, (0,255,0), -1)
149 cv.rectangle(drawing, (int(boundRect[i][0]), int(boundRect[i][1])), \
150 (int(boundRect[i][0]+boundRect[i][2]),
int(boundRect[i][1]+boundRect[i][3])),
(125,125,0), 1)
151
152 return drawing,src;
153
154 while(1):
155 #Capturamos una imagen y la convertimos de RGB -> HSV
156 #_, imagen = captura.read()
157 imagen=cv.imread('signal1.jpg',cv.IMREAD_COLOR)
158 hsv = cv.cvtColor(imagen, cv.COLOR_BGR2HSV)
159
160 draw1,src1=detection(azul_bajos,azul_altos,null,null,(255,0,0),imagen)
161
draw2,src2=detection(rojo_bajos_0,rojo_altos_0,rojo_bajos_1,rojo_altos_1,(0,0
,255),draw1)
162
draw3,src3=detection(blanco_bajos,blanco_altos,null,null,(255,255,255),draw2)
163
draw4,src4=detection(amarillo_bajos,amarillo_altos,null,null,(255,255,0),draw
3)
164
165 draw=draw4
166 src=src1+src2+src3+src4
167
168 # Show in a window
169 cv.imshow('mask',src)
170 cv.imshow('Contours', draw)
```

```
171
172 #tecla = cv.waitKey(5) & 0xFF
173 #time.sleep(2)
174 #if tecla == 27:
175 # break
176 while(1):
177 tecla = cv.waitKey(5) &0xFF
178 if tecla == 27:
179 break
180
181 cv.destroyAllWindows()
```