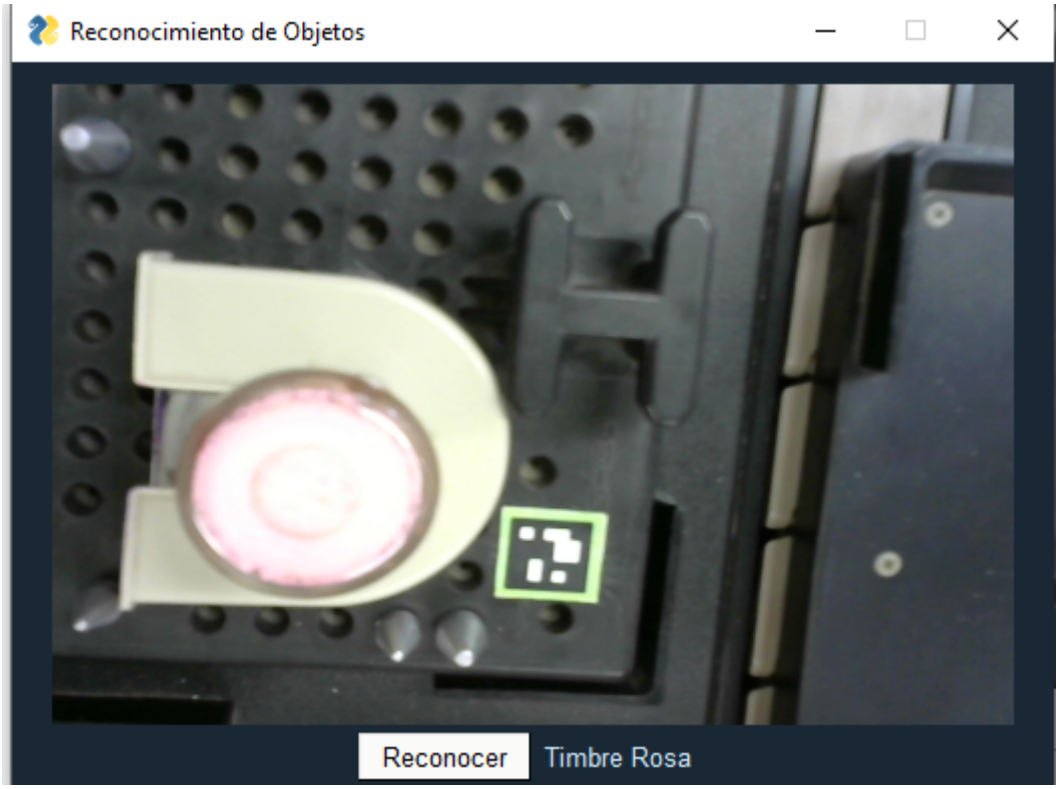
Instructivo de Apoyo para el reconocimiento de objetos

Nicolás Barraza Estrada, 2022-2
Ingeniería Civil en Informática

Paso 0 - Modo de uso

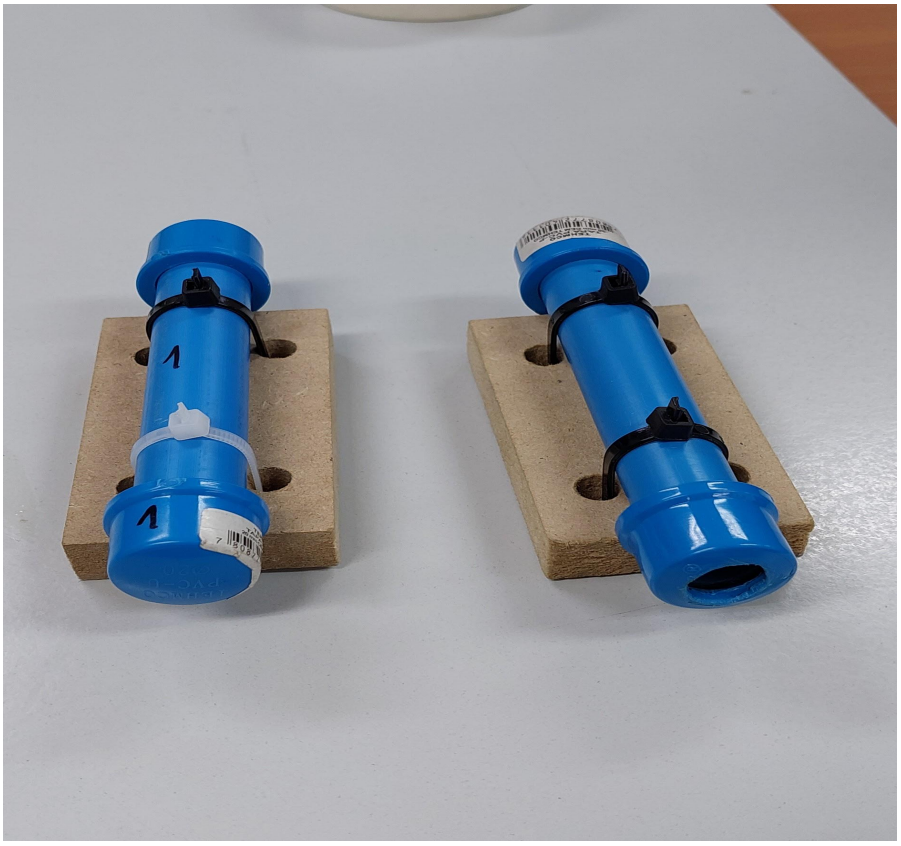
Ejecutar “Reconocimiento de Objetos 2.py”

Posicionar un objeto aproximadamente en el centro de la cámara y presionar el botón “Reconocer”:



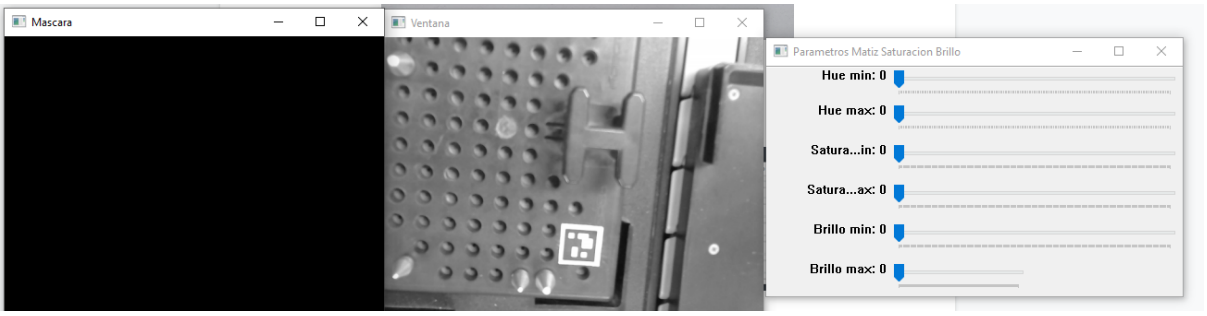
Paso 1 - Elegir un objeto

Definir un objeto para detectar. Para este instructivo se usará uno de los PVC del laboratorio CIM.

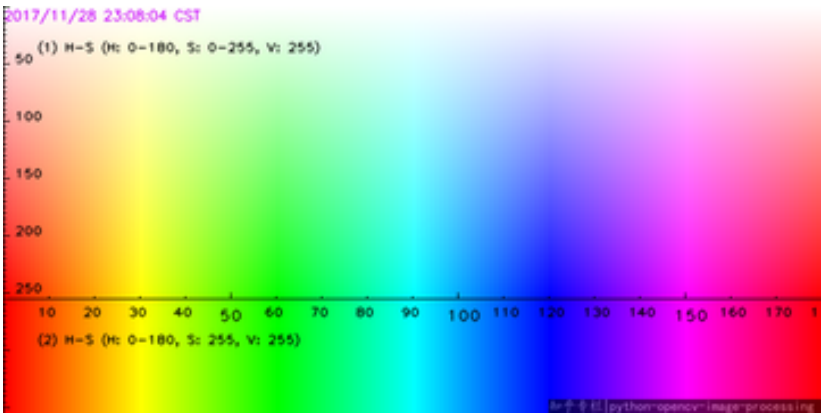


Paso 2 - Definir las características

Encontrar sus características principales, ,para ello debe abrir el programa “tutorialResaltado.py”

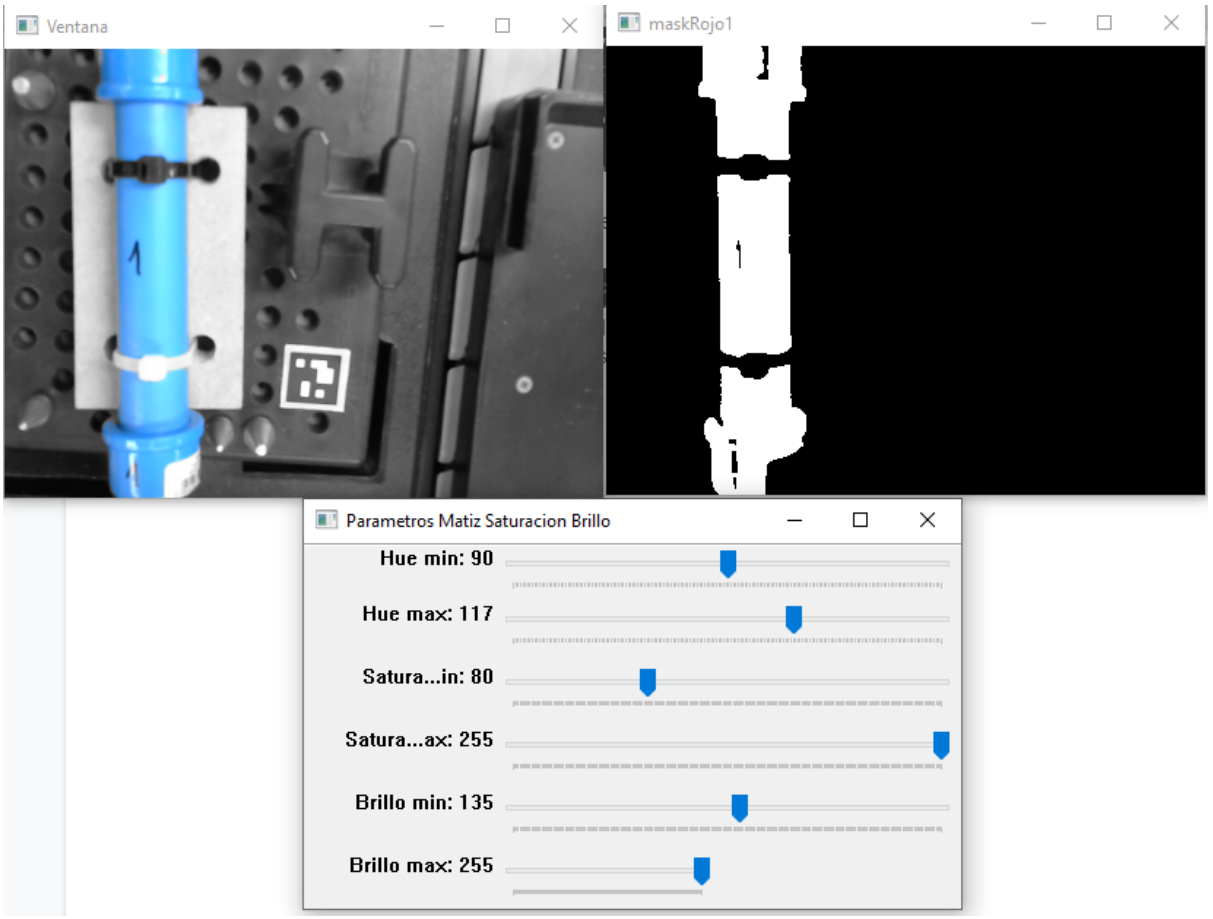


Para resaltar un color posicione el parámetro Hue Min donde comienza el rango de colores que quiere detectar, y Hue Max donde termina. Guiese de la siguiente imagen:



Para obtener el color celeste el rango Hue es 90 a 117.

Posicione la saturación Máx y Brillo Máx en el máximo (255) y varie Saturación Min y Brillo Min hasta que solo vea el color celeste.



Para cerrar el programa presione la tecla “ESC”. Deberá realizar este procedimiento para cada color que quiera resaltar, en este caso falta el café-claro

En la cmd obtendrá listas similares a estas:

Celeste

```
C:\Users\cim\Desktop\NicolasBarraza NO BORAR\0_0_2_1>tutorialResaltado.py
3
[90, 117, 80, 255, 135, 255, 0, 4135.5]
[90, 117, 80, 255, 135, 255, 0, 5684.0]
[90, 117, 80, 255, 135, 255, 0, 4337.0]
```

Cafe-claro

```
C:\Users\cim\Desktop\NicolasBarraza NO BORAR\0_0_2_1>tutorialResaltado.py
2
[12, 24, 10, 255, 184, 255, 0, 6569.0]
[12, 24, 10, 255, 184, 255, 0, 4078.0]
```

El contenido de estas listas es:

[HueMin,HueMax,SatuMin,SatuMax,BrilloMin,BrilloMax,Cámara,AreaDelContorno]

El motivo por el que aparece más de uno es porque al haber interferencia en medio de un color se trata como imágenes separadas, lo cual es bueno porque proporciona más información para diferenciarlo de otros objetos.

Paso 3 - Editar el código

Agregar esta información al código:

Se crea la misma lista, un np.array para el HSV menor y uno para el HSV mayor, se guarda el área en caso de querer usarla, más abajo en el código yo lo escribí manualmente.

```
 99  arrayPVC = [90, 117, 80, 255, 135, 255, 0, 2830.0]
100  PVCBajo = np.array([arrayPVC[0] , arrayPVC[2], arrayPVC[4]], np.uint8)
101  PVCAlto = np.array([arrayPVC[1] , arrayPVC[3], arrayPVC[5]], np.uint8)
102  areaPVC = arrayPVC[7]
103
104  arrayMadera = [12, 24, 10, 255, 185, 255, 0, 3686.5]
105  maderaBajo = np.array([arrayMadera[0] , arrayMadera[2], arrayMadera[4]], np.uint8)
106  maderaAlto = np.array([arrayMadera[1] , arrayMadera[3], arrayMadera[5]], np.uint8)
107  areaMadera = arrayMadera[7]
108
```

Se agrega la máscara:

```
127              maskHsv     = cv2.inRange(frameHSV,HsvBajo,HsvAlto)
128              maskPVC     = cv2.inRange(frameHSV,PVCBajo,PVCAlto)
129              maskMadera  = cv2.inRange(frameHSV,maderaBajo,maderaAlto)
```

Se buscan los contornos:

```
137      contornosPVC, hierarchy5 = cv2.findContours(maskPVC, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
138      contornosMadera, hierarchy5 = cv2.findContours(maskMadera, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
```

Estos contornos son listas, por lo que se debe iterar sobre cada uno de ellos para encontrar los objetos buscados.

Paso 4 - Agregar características

Ahora se deben agregar condiciones para discriminar el objetos de otros similares, lo más básico es empezar por el área, se sabe que el área del PVC está en entre 10000 px y 20000 px-desde la posición de la cámara instalada-así que lo agregamos de la siguiente manera:

```
145         for c in contornosPVC:
146             areaContorno = cv2.contourArea(c)
147             epsilon = 0.02*cv2.arcLength(c,True)
148             approx = cv2.approxPolyDP(c,epsilon,True)
149             if areaContorno > 10000  and areaContorno < 26000:
150                 nuevoContorno = cv2.convexHull(c)
```

Si queremos más precisión se puede ingresar la cantidad de lados que debe tener el contorno con el siguiente código:

```
148             approx = cv2.approxPolyDP(c,(0.02*cv2.arcLength(c,True)),True)
```

approx será un número entero, por ejemplo 4, si es un cuadrado, sin embargo se debe tener cuidado ya que las esquinas redondeadas son tratadas como lados, si quisieran un circulo seria por lo menos 12 lados.

Buscará por lo menos 2 veces el contorno celeste (dos, ya que no todas traen doble amarre)

```
80                  if areaContorno > 10000  and areaContorno < 26000:
81                      nuevoContorno = cv2.convexHull(c)
82                      contornoList.append(nuevoContorno)
83                      count = count + 1
84                      if count == 2:
```

y cuando lo encuentre hará el mismo procedimiento para el color café-claro

```
87                  for c in contornosMadera:
88                      areaContorno1 = cv2.contourArea(c)
89                      epsilon1 = 0.02*cv2.arcLength(c,True)
90                      approx1 = cv2.approxPolyDP(c,epsilon,True)
91                      if areaContorno1 > 11000 and areaContorno1 < 15000:
92                          nuevoContorno1 = cv2.convexHull(c)
93                          contornoList1.append(nuevoContorno1)
94                          count1 = count1 + 1
95                          if count1 == 2:
96                              cv2.drawContours(frame, [contornoList1[0]], 0, (255,0,0
97                              cv2.drawContours(frame, [contornoList1[1]], 0, (255,0,0
98                              cv2.drawContours(frame, [contornoList[0]], 0, (255,0,0)
99                              cv2.drawContours(frame, [contornoList[1]], 0, (255,0,0)
100                             texto = 'PCV'
101                             flag = True
```

Y ahora se tiene un código que reconoce PVC con piezas de madera… o no, en general, reconoce cualquier objeto cuyos colores son celeste y café claro de los tamaños definidos.

Así que dejaré otros códigos que puedan ser de utilidad.

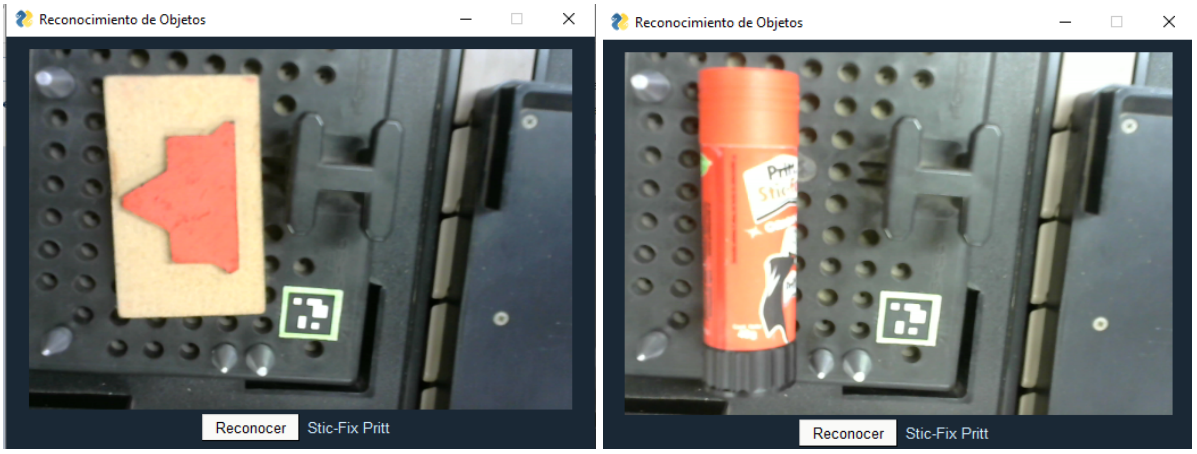
```
if len(approx)==4 and ( areaContorno > (area * 0.9) ):
    cv2.drawContours(frame, [nuevoContorno], 0, (0,0,0), 1)
    flag = True
    texto = 'Pieza Virgen'
    x,y,w,h = cv2.boundingRect(c)
    madera = frame[y:y+h,x:x+w]
    #cv2.imshow('madera',madera)
```

```
if area != 0:
    M = cv2.moments(c)
    if (M["m00"]==0): M["m00"]=1
        x = int(M["m10"]/M["m00"])
        y = int(M['m01']/M['m00'])
```

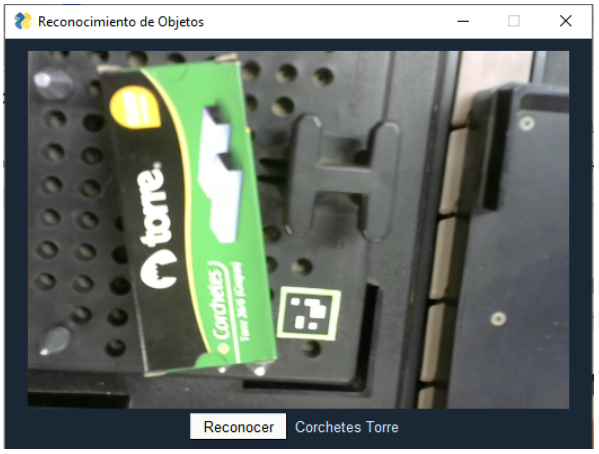
El primero recibe la posición (x,y) donde empieza el contorno, w width (ancho) y h height (alto), así podemos buscar específicamente otras características en una área en particular.

El segundo calcula el centro (x,y), puede usarse para comprobar si un objeto está incluido dentro de otro

El problema de no representar bien las características de un objeto.



En este caso el Stick-Fix fue programado para reconocer un objeto rojo con un área mayor a 10500 px y con 4 o más lados, por lo que puede reconocer -erróneamente- más objetos de los que debería.



En el caso de los corchetes Torre vusca principalmente un gran area verde para luego buscar un circulo(approx > 12) amarillo de un tamaño menor a 500 px, por lo que no lo confunde con otros objetos parecidos.

Fin

(Para una mejor aplicacion consultar por “Reconcimiento de Objetos.py”, codigo orientado al reconocimiento de piezas y movimiento de brazo robótico)