

# Algoritmo para el reconocimiento de objetos y movimiento de brazo robótico

Nicolás Barraza Estrada, 2022-2  
Ingeniería Civil en Informática



# Índice

<b>Introducción</b>	<b>2</b>
<b>Instalación de Python</b>	<b>3</b>
Actualización de PIP	4
<b>Instalación de Librerías</b>	<b>6</b>
Instalación de OpenCV	6
Instalación de Numpy	6
Instalación de PySimpleGui	6
Instalación de PySerial	7
<b>Instalación del IDE</b>	<b>8</b>
<b>Instrucciones de uso de la interfaz</b>	<b>9</b>
<b>Precauciones</b>	<b>11</b>
<b>Imagenes de Muestra</b>	<b>12</b>
<b>Explicación del código</b>	<b>14</b>
<b>Conclusión</b>	<b>27</b>

# Introducción

Durante la realización de la práctica 1 en el Laboratorio CIM, se solicitó hacer una programa de escritorio que pueda reconocer el tipo de pieza de madera mediante una cámara y enviar órdenes al brazo robótico, para ello se hizo uso del lenguaje de programación Python y las librerías OpenCV, Numpy, PySerial y PySimpleGui

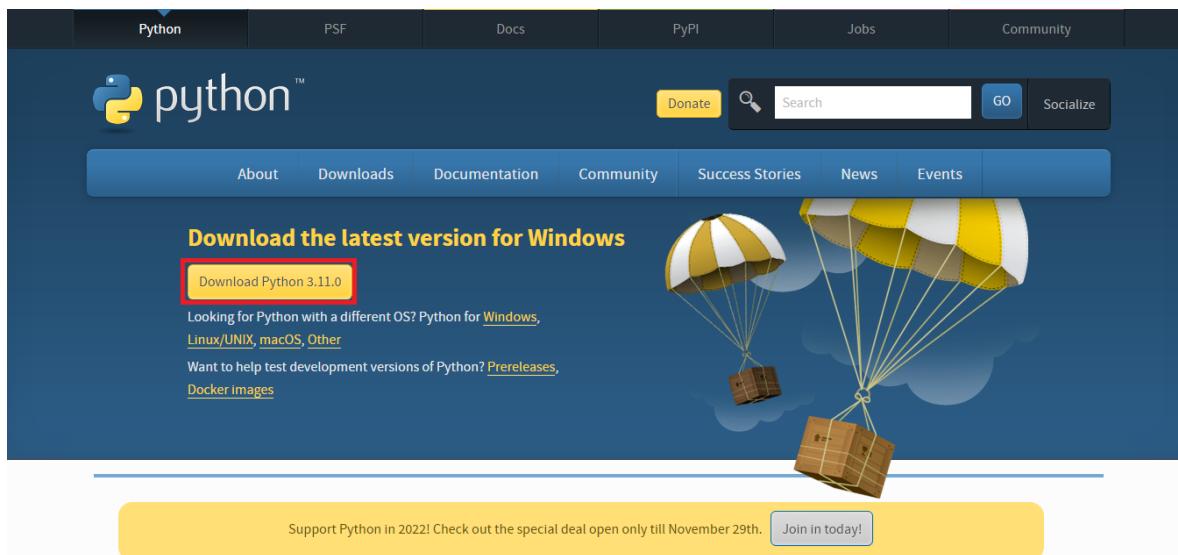
Se utilizaron como guía los tutoriales entregados por el profesor Luis Vera, el canal de YouTube OMES y algunos post de StackOverflow.

Esta documentación proporciona las preparaciones para su ejecución, junto con un instructivo de su uso y una explicación del código.

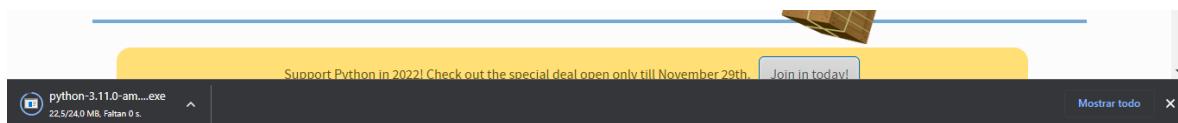


# Instalación de Python

Para la ejecución del programa se necesita instalar la versión de Python más reciente para, se puede descargar desde el siguiente link: <https://www.python.org/downloads/>

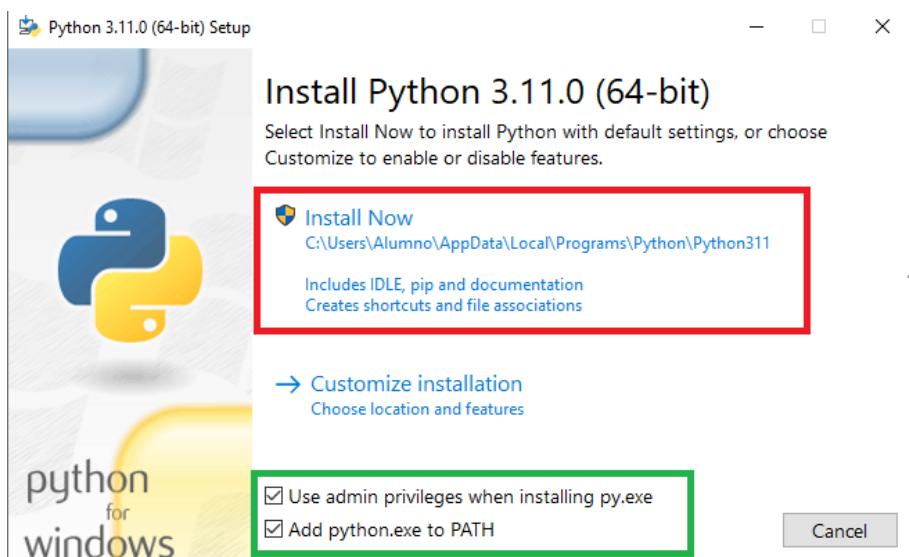


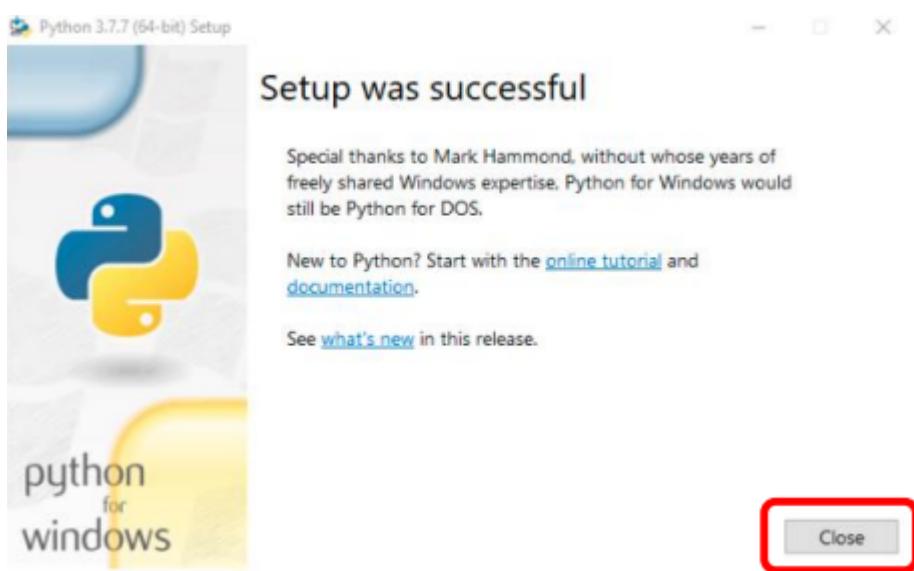
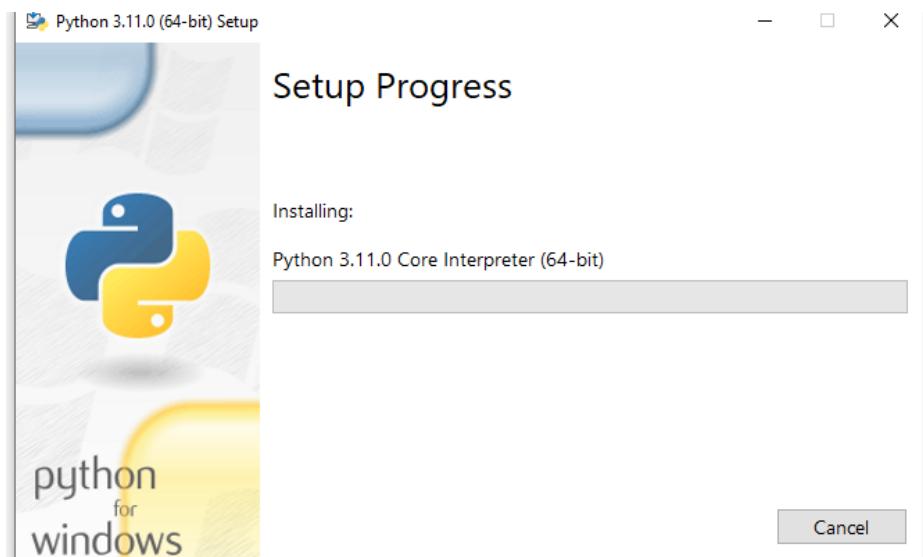
Si su sistema operativo no es Windows seleccione el que más se acomode y comenzará la descarga.



Se ejecuta el instalador, en caso de que pida permisos de administrador seleccione 'ejecutar'.

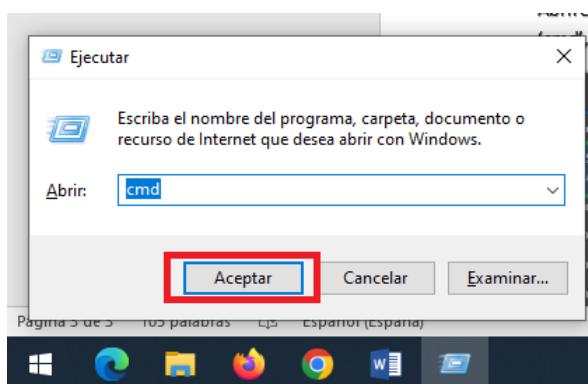
Una vez abierto debe marcar la casilla 'Add Python.exe to PATH' y dar click en 'Install Now' y espere a que finalice la instalación.

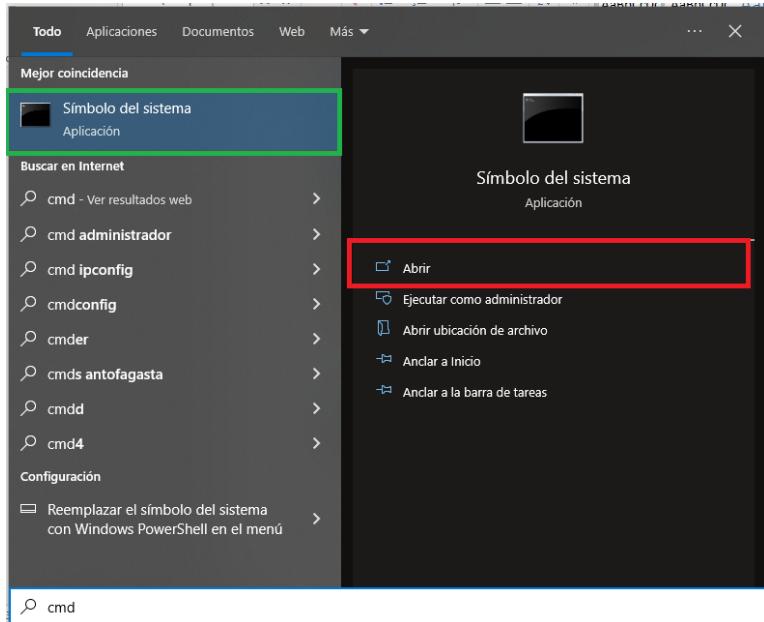




## Actualización de PIP

Abra la consola de Windows (símbolo del sistema) con la tecla ‘Windows’ + ‘r’ y escriba ‘cmd’, o vaya a ‘inicio’ y escriba ‘cmd’.





Y se abrirá la siguiente ventana:



En ella escriba el siguiente comando “ python –m pip install --upgrade pip ”, presione “Enter” y espere. Puede usar “ pip --version ” antes y después para comprobar si se ha actualizado a una versión más reciente.

```
C:\Users\ >pip --version
pip 22.3.1 from C:\Users\ :\AppData\Local\Programs\Python\Python310\lib\site-packages\pip (python 3.10)
```

# Instalación de Librerías

En la cmd ingrese los comandos que se muestran más abajo (sin las comillas) para la instalación de las librerías.

## Instalación de OpenCV

Open Computer Vision es una librería dedicada al procesamiento de imágenes, se usó para capturar imágenes, generar una imagen binaria basada en los colores y medir el área de los objetos para revisar si la forma es rectangular

Comando : pip install opencv-python

```
C:\Users\    >pip install opencv-python
Collecting opencv-python
  Using cached opencv_python-4.6.0.66-cp36-abi3-win_amd64.whl (35.6 MB)
Collecting numpy>=1.14.5
  Downloading numpy-1.23.5-cp310-cp310-win_amd64.whl (14.6 MB)
    ..... 14.6/14.6 MB 3.4 MB/s eta 0:00:00
Installing collected packages: numpy, opencv-python
Successfully installed numpy-1.23.5 opencv-python-4.6.0.66
```

Comando : pip install opencv-contrib-python

```
C:\Users\    >pip install opencv-contrib-python
Collecting opencv-contrib-python
  Using cached opencv_contrib_python-4.6.0.66-cp36-abi3-win_amd64.whl (42.5 MB)
Requirement already satisfied: numpy>=1.17.3 in c:\users\    \appdata\local\programs\python\python310\lib\site-packages
  (from opencv-contrib-python) (1.23.5)
Installing collected packages: opencv-contrib-python
Successfully installed opencv-contrib-python-4.6.0.66
```

## Instalación de Numpy

Numpy es una librería usada principalmente para álgebra y el cálculo de vectores y matrices, la usamos para guardar en vectores el rango de los colores en HSV. También se planeaba usar para guardar matrices con los píxeles, aunque opencv proporciona métodos para ello. Si se siguió el orden, numpy debió haberse instalado con openCV.

Comando : pip install numpy

```
C:\Users\    >pip install numpy
Requirement already satisfied: numpy in c:\users\    \appdata\local\programs\python\python310\lib\site-packages (1.23.5)
```

## Instalación de PySimpleGui

PySimpleGui es una librería para realizar interfaces, si bien es menos estético que librerías como ktinker, es suficiente para programas en el ámbito industrial. Se basa en la composición de columnas, se colocan elementos en una lista (que representa una columna) y luego se junta con más columnas para formar la ventana.

Comando : pip install PySimpleGUI

```
C:\Users\          pip install PySimpleGUI
Collecting PySimpleGUI
  Using cached PySimpleGUI-4.60.4-py3-none-any.whl (509 kB)
Installing collected packages: PySimpleGUI
Successfully installed PySimpleGUI-4.60.4
```

## Instalación de PySerial

PySerial es una librería que facilita la comunicación con puertos seriales, la usamos para conectar el programa con el brazo robótico.

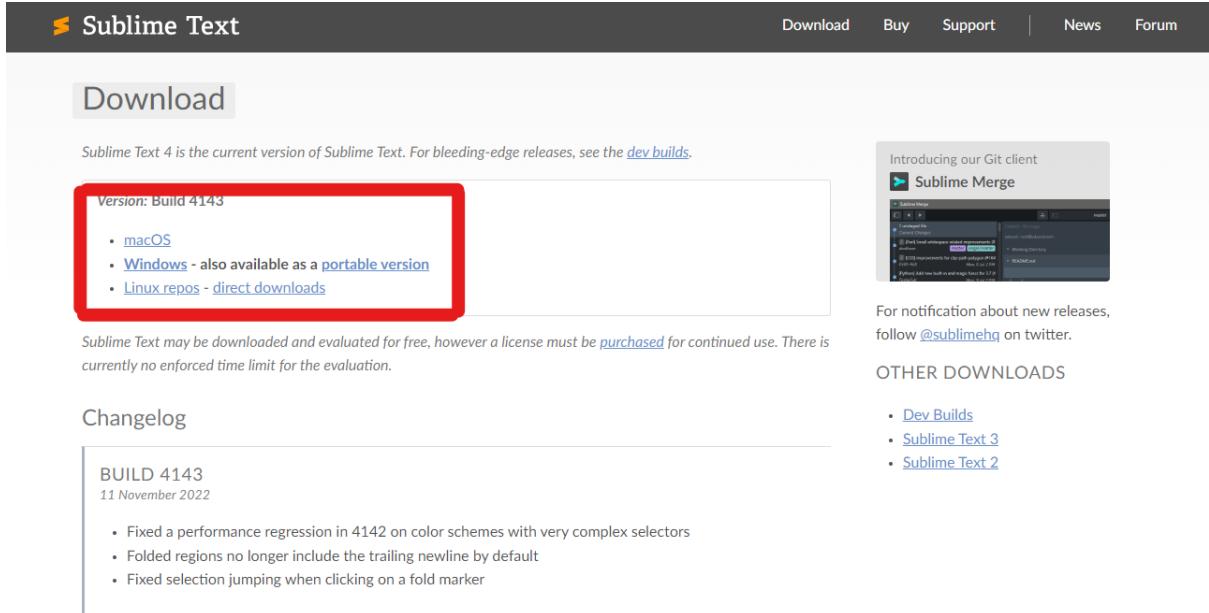
Comando : pip install pyserial

```
C:\Users\          >pip install pyserial
Collecting pyserial
  Downloading pyserial-3.5-py2.py3-none-any.whl (90 kB)
                                         90.6/90.6 kB 1.3 MB/s eta 0:00:00
Installing collected packages: pyserial
Successfully installed pyserial-3.5
```

Para finalizar use el comando “ pip list ” y le mostrará todas las librerías instaladas, entre ellas deberán estar al menos estas cuatro (cinco, ya que openCV se repite).

# Instalación del IDE

Pueden usar cualquier editor de texto o IDE, en este caso se usó Sublime Text. Para descargarlo se ingresa en <https://www.sublimetext.com/download> y se elige el más compatible con su sistema operativo.



Version: Build 4143

- macOS
- Windows - also available as a [portable version](#)
- Linux repos - [direct downloads](#)

Sublime Text may be downloaded and evaluated for free, however a license must be [purchased](#) for continued use. There is currently no enforced time limit for the evaluation.

Changelog

BUILD 4143  
11 November 2022

- Fixed a performance regression in 4142 on color schemes with very complex selectors
- Folded regions no longer include the trailing newline by default
- Fixed selection jumping when clicking on a fold marker

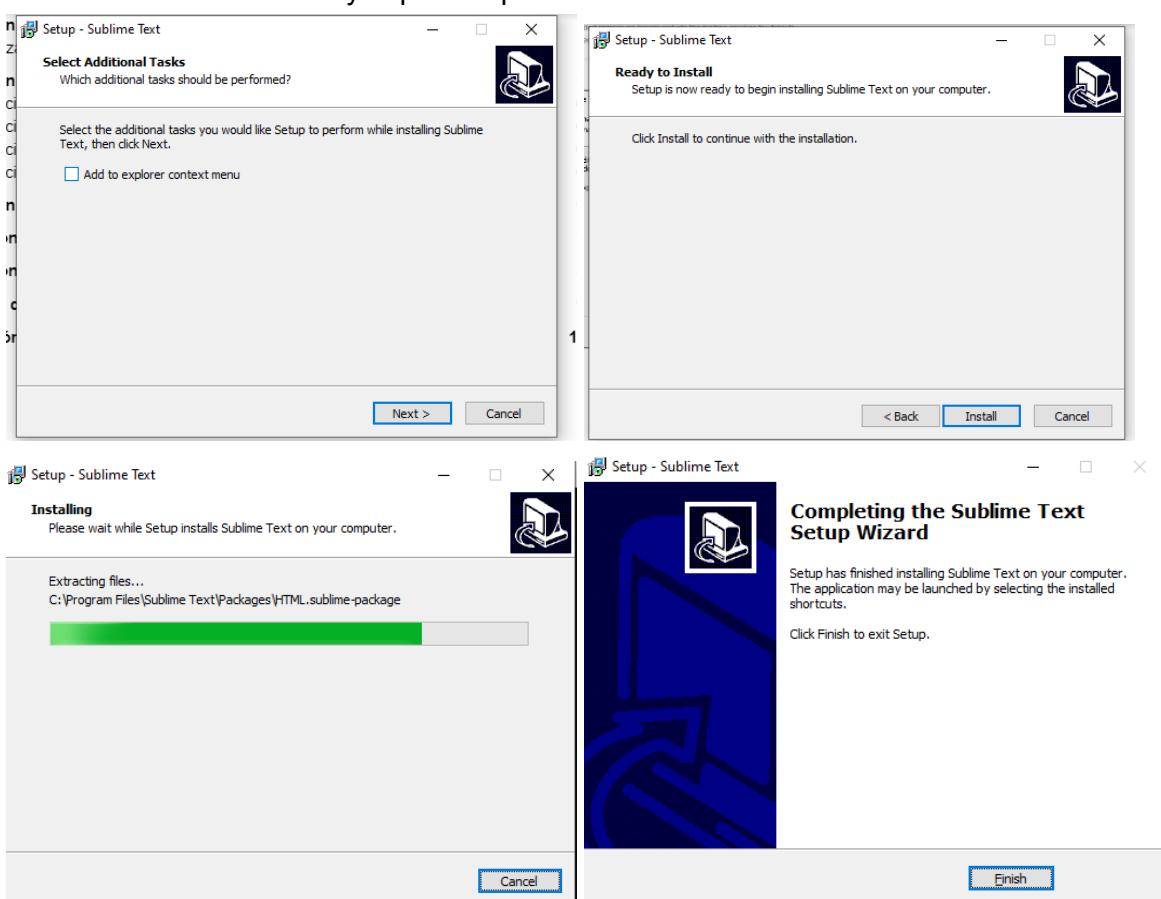
Introducing our Git client  
Sublime Merge

For notification about new releases, follow [@sublimehq](#) on twitter.

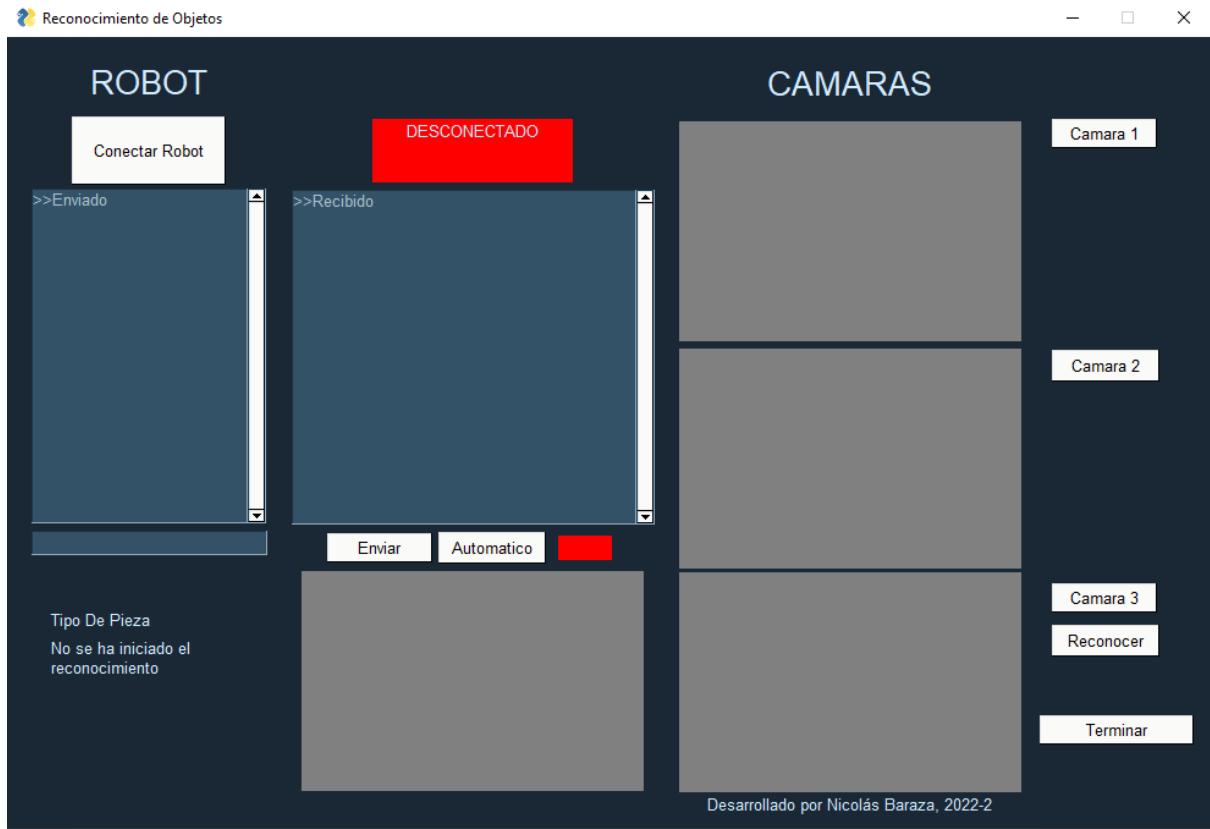
OTHER DOWNLOADS

- [Dev Builds](#)
- [Sublime Text 3](#)
- [Sublime Text 2](#)

Click en >> next >> install y espere a que finalice



# Instrucciones de uso de la interfaz



La interfaz tiene dos secciones, a la izquierda el robot y a la derecha las cámaras. Para conectar el robot debe asegurarse de que el programa hyperterminal esté cerrado o que no esté conectado en el puerto COM1, y se presiona el botón “Conectar Robot”.

En la caja de texto de la izquierda será donde se registraran las peticiones para el brazo robótico que se han enviado (desde el input que está justo abajo y se presiona “Enviar”), esta petición se verá reflejada también en la caja de texto de la derecha junto con el mensaje “Done” si es que el comando existe, de lo contrario avisará del error.

El botón “Automático” requiere que el robot y la cámara 3 estén conectados, o avisará el error en la caja de textos recibidos. Al presionarlo el robot se moverá hacia la posición 600 (“move 600”) y comenzará a reconocer la pieza hasta que se vuelva a presionar “Automático” para desactivarlo.

En el lado derecho están las cámaras, se encienden con los botones “Camara 1”, “Camara 2” y “Cámara 3”, sin embargo las cámaras 1 y 2 no son necesarias para la ejecución del programa, ya que están pensadas para la supervisión del entorno.

Para poder usar “Reconocer” se debe tener encendida la Cámara 3, y debajo de “Tipo de piezas” aparecerá el tipo de pieza que es, las opciones son; Pieza Virgen, Pieza Macho, Pieza Hembra o No se ha detectado nada.

Se dejan registrada las posiciones 600, 601 y 602 para realizar el reconocimiento de objetos.

El comando run 1 es igual a move 600, run 2 mueve la pieza abajo a la izquierda, run 3 mueve la pieza abajo a la derecha, y run 4 es la posición de descanso.

La pieza verde está en la posición 600, la azul es 601 y la roja está en la posición 602. Se accede mediante el comando move [número]. Ejemplo: move 600, move 601, move 602.



Según las pruebas realizadas, Windows asigna las cámaras 0,1,2,etc, dependiendo del orden en el que son conectadas al pc. Se recomienda cerrar y volver a abrir el programa cada vez que quiera modificar el orden las cámaras para evitar algún posible problema.

# Precauciones

Para poder usar el reconocimiento es necesario que las piezas sigan un patrón de colores, estos deben pintarse solamente en el área trabajada del color Rojo, Verde o Azul. Las piezas hembras deben pintarse por el borde rectangular, incluso si este fue trabajado, véase el ejemplo de la pieza hembra roja.



Sobre el color negro, es posible hacerlo funcionar, sin embargo se deben hacer dos modificaciones, primero debe cambiar el color de fondo, ya que no es posible detectar una pieza hembra negra en un fondo negro, tampoco es posible reconocer directamente el interior color madera (cafe claro), ya que puede hacer interferencia con las puntas metálicas y las cintas blanco/gris que tienen algunas mesas.

De intentarlo se recomienda usar un color que no se utilice en el programa, durante las pruebas se usó un fondo blanco, como una hoja o la tapa trasera de un cuaderno.

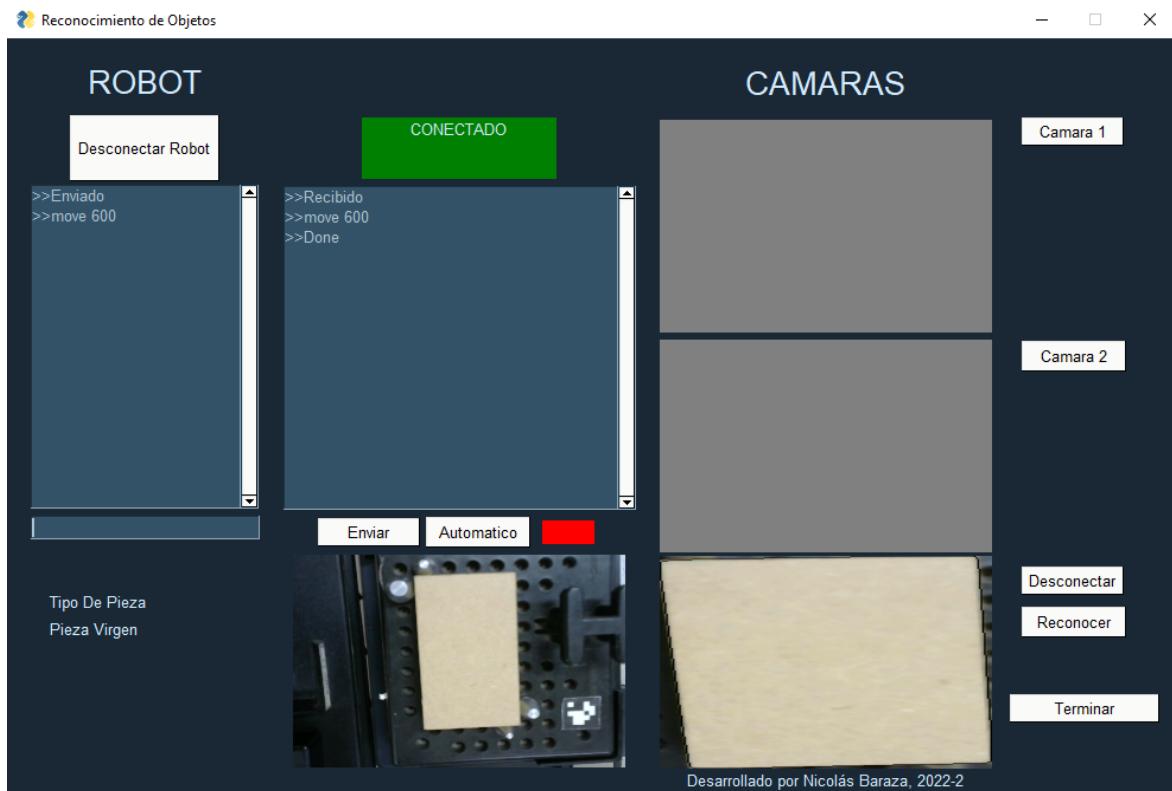
Lo segundo es modificar ligeramente una línea de código, abrimos el código “Reconocimiento de Objetos.py” con el editor de texto o IDE, en la línea 41 veremos una lista que contiene las imágenes a procesar discriminando por colores:

```
39  
40     maskVerde = cv2.inRange(frameHSV,verdeAbajo,verdeArriba)  
41     maskNegro = cv2.inRange(frameHSV,black_lower,black_upper)  
42     arrayMascaras = [maskRojo,maskAzul,maskVerde]#agregar maskNegro|  
contornosMadera, hierarchy0 = cv2.findContours(maskMadera, cv2.RETR_EXTERNAL)
```

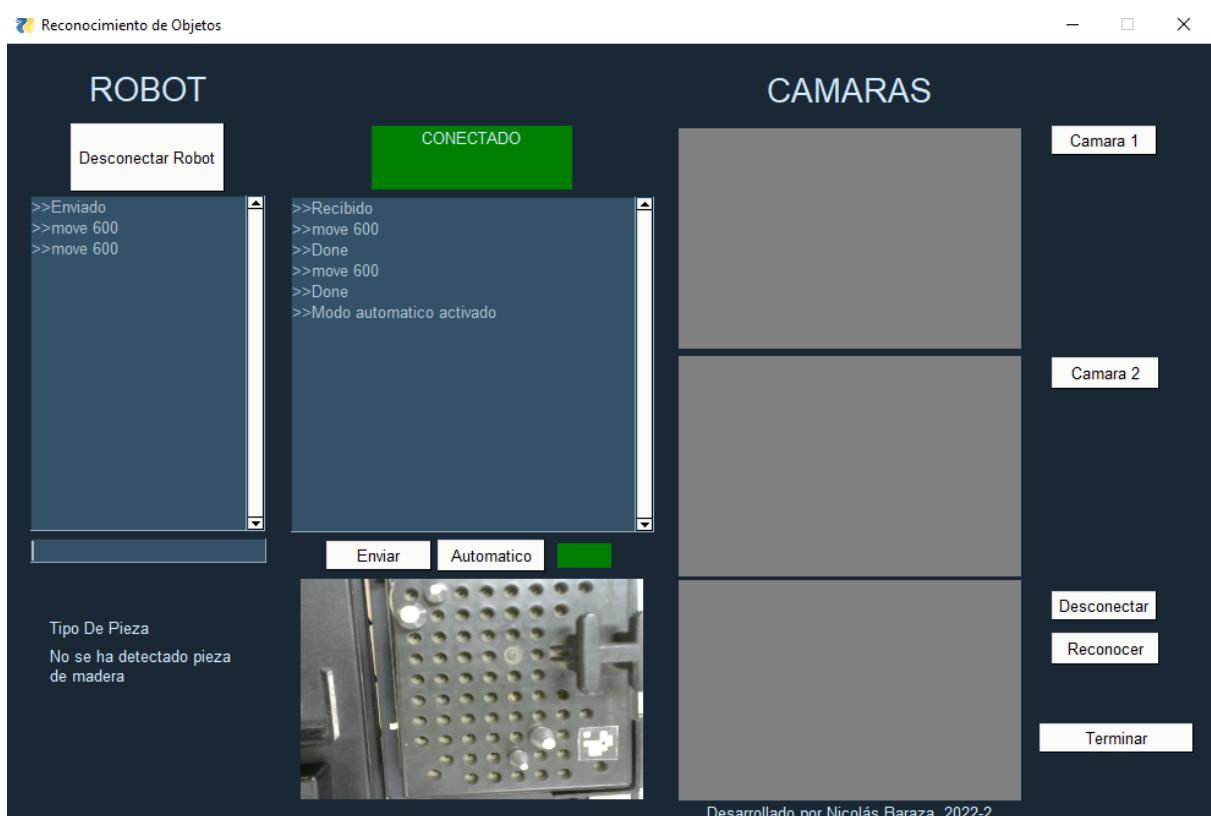
Seguido de maskRojo, maskAzul y maskVerde agregaremos maskNegro y listo, ya podemos usar el color negro:

```
40     maskNegro = cv2.inRange(frameHSV,black_lower,black_upper)  
41     arrayMascaras = [maskRojo,maskAzul,maskVerde,maskNegro]#agregar maskNegro  
42     contornosMadera, hierarchy0 = cv2.findContours(maskMadera, cv2.RETR_EXTERNAL)
```

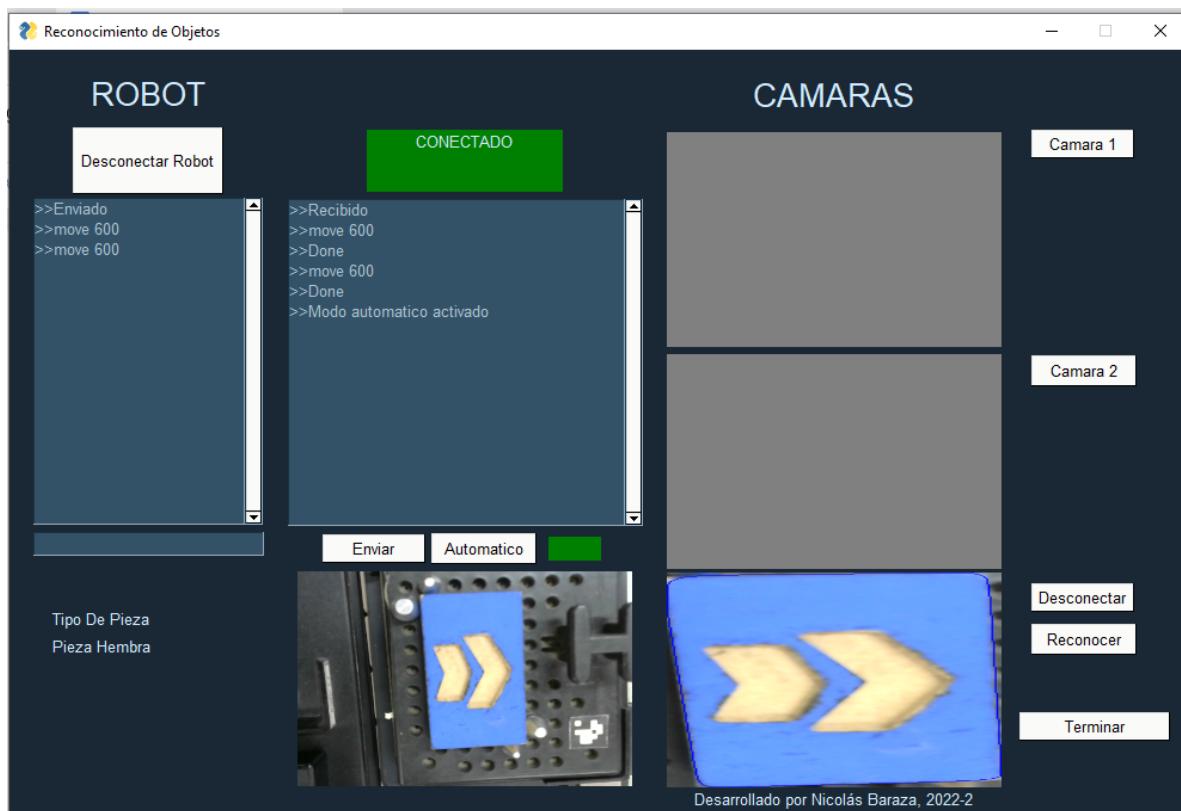
# Imagenes de Muestra



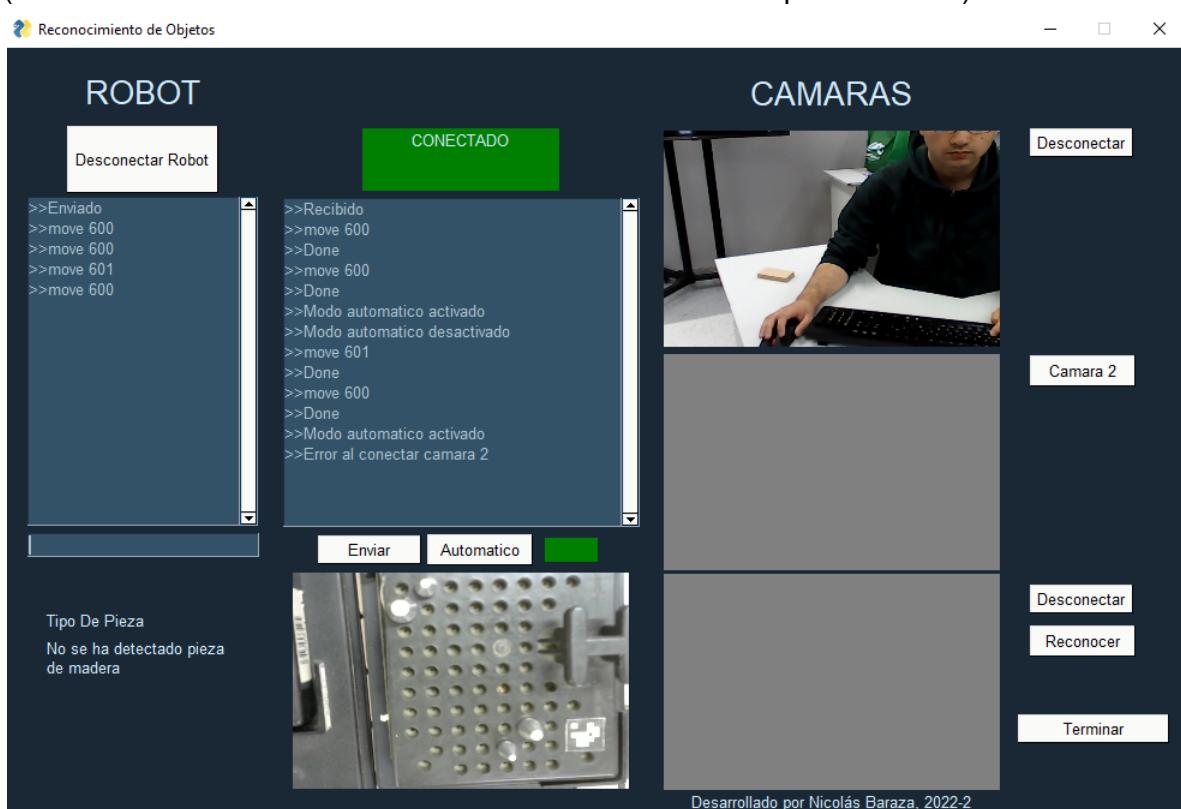
(Se usó “Move 600” y “Reconocer” sobre una pieza virgen)



(Se usó el botón “Automático” cuando no había nada sobre la mesa)



(Se mantuvo el modo automático mientras se colocaba una pieza hembra)



(Se desactivo el modo automático, se movió el brazo, y se volvió activar cuando no había nada, también se activo la cámara 2, pero al no estar conectada da un error, se activa la cámara 1)

# Explicación del código

“Reconomiento de Objetos.py” fue entregado al profesor.

Se importan las librerías:

```
1  import cv2
2  import PySimpleGUI as sg
3  import numpy as np
4  import serial
5  import time
6  import sys
7
```

Se define el algoritmo de reconocimiento de piezas:

Se inició una variable texto como ‘No se ha detectado pieza de madera’, que más adelante cambiará a ‘pieza virgen’ , ‘pieza macho’ o ‘pieza hembra’, dependiendo de las condiciones.

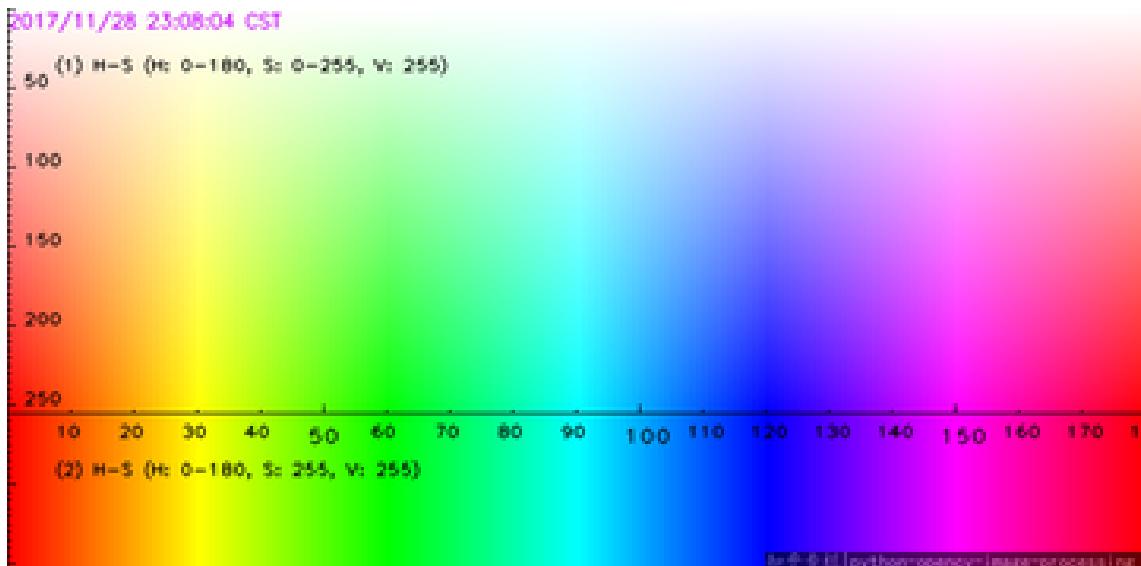
HSV es la nomenclatura para Hue, Saturation, Value (Matiz, Saturación, Valor), para openCV se deben indicar dos valores creando un rango de colores donde se procesan las imágenes, creando un vector de la siguiente forma:

VectorHsv = [HueMín, HueMáx, SaturationMin, SaturationMax, ValueMin, ValueMax]

```
8
9  def reconocimiento_pieza(video_capture3):
10     texto = 'No se ha detectado pieza de madera'
11     arrayHsv = [15, 33, 15, 255, 140, 255, 0, 27058.0]
12     arrayHsv2 = [99, 132, 105, 255, 170, 255, 0, 26158.0]
13     maderaBajo = np.array([15 , arrayHsv[2], arrayHsv[4]], np.uint8)
14     maderaAlto = np.array([35, arrayHsv[3], arrayHsv[5]], np.uint8)
15
16     rojoBajo = np.array([0 , arrayHsv2[2], arrayHsv2[4]], np.uint8)
17     rojoAlto = np.array([10 , arrayHsv2[3], arrayHsv2[5]], np.uint8)
18
19     azulBajo = np.array([105 , arrayHsv2[2], arrayHsv2[4]], np.uint8)
20     azulAlto = np.array([130 , arrayHsv2[3], arrayHsv2[5]], np.uint8)
21
22     verdeBajo = np.array([50 , arrayHsv2[2], arrayHsv2[4]], np.uint8)
23     verdeAlto = np.array([95 , arrayHsv2[3], arrayHsv2[5]], np.uint8)
24
25     black_lower = np.array([0,0,0],np.uint8)
26     black_upper = np.array([180,255,100],np.uint8)
27
28     area = arrayHsv[7]
29     count = 0
30     while count <= 4:
31         flag = False
```

En la línea 11 se creó un arrayHsv específico para el color de la madera (café claro), y en la 12 uno general para el resto de colores. Al final de los vectores hay un 0 y un valor sobre 20.000, el 0 es la cámara asignada para el reconocimiento (que se usó durante las pruebas) y el segundo número es un aproximado del área de las piezas de madera, en píxeles y calculado manualmente con códigos desarrollados anteriormente. No se recomienda eliminar el 0 ya que cambiará el índice del área que se asigna más abajo.

Desde la línea 16 a 23 se crearon dos np.array por cada color, el mínimo y el máximo, sin embargo la Saturación y el Valor ya la tenemos, por lo que solo se debe especificar el el Tono. Pueden guiarse de la siguiente imagen:



El rojo parte desde el 0 hasta aproximadamente el 10, el verde desde 40 hasta aprox. 70, si quisieramos agregar el rosa-fucsia sería desde el 145 hasta 160. También se puede observar que el rojo está al principio y al final, sin embargo con la saturación y valor que se usan no hay diferencia, en otras situaciones podría necesitar crear otro array desde el 170.

En la línea 28 se asignó el área, cualquiera de las dos ya que luego se le configuró un rango de error, se inició un contador en 0.

```

30     while count <= 4:
31         flag = False
32         ret,frame = video_capture3.read()
33         #frame = frame0[]
34         if ret==True:
35             frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
36             maskMadera = cv2.inRange(frameHSV,maderaBajo,maderaAlto)
37             maskRojo = cv2.inRange(frameHSV,rojoBajo,rojoAlto)
38             maskAzul = cv2.inRange(frameHSV,azulBajo,azulAlto)
39             maskVerde = cv2.inRange(frameHSV,verdeBajo,verdeAlto)
40             maskNegro = cv2.inRange(frameHSV,black_lower,black_upper)
41             arrayMascaras = [maskRojo,maskAzul,maskVerde]#agregar maskNegro
42             contornosMadera, hierarchy0 = cv2.findContours(maskMadera, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
43             for mascaras in arrayMascaras:
44                 if flag == False:
45                     contornosMascara, hierarchy = cv2.findContours(mascaras, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
46                     for c in contornosMascara:

```

Desde la línea 30 comienza un While donde se realizó el procesamiento de la imagen, si bien el contador llega hasta 4 es solo en caso de imperfecciones en la pieza, ya que se retorna el texto apenas se detecte la pieza.

Se definió una Flag, si es False significa que el código no ha encontrado la pieza y continuará los bucles. La variable `videocapture3` se recibe desde el main y al aplicarle `.read()` devuelve dos valores, el primero es un booleano que indica si existe la conexión con la cámara y el segundo es la imagen, por lo tanto en la línea 34 se revisó si `ret` es True.

En la línea 35 se usa `cvtColor` para transformar la imagen de BGR (openCV los recibe en ese orden) a HSV. Las máscaras son imágenes donde solo se ve el rango de colores que se le indica, para ello se usa `inRange()`. Para evitar repetir el código se puso cada máscara en la lista `arrayMascaras` para luego iterarla, excepto `maskMadera`.

`findContours()` devuelve un array con todos los contornos, en este caso, de los objetos cafe-claro, ya que siempre habrá una parte de la madera que no esté pintada.

```

41 arrayMascaras = [maskRojo,maskAzul,maskVerde]#agregar maskNegro
42 contornosMadera, hierarchy0 = cv2.findContours(maskMadera, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
43 for mascaras in arrayMascaras:
44     if flag == False:
45         contornosMascara, hierarchy = cv2.findContours(mascaras, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
46     for c in contornosMascara:
47         areaContorno = cv2.contourArea(c)
48         #print(area)
49         epsilon = 0.02*cv2.arcLength(c, True)
50         approx = cv2.approxPolyDP(c, epsilon, True)
51     if areaContorno > 1000:
52         nuevoContorno = cv2.convexHull(c)
53     if len(approx)==4 and ( areaContorno > (area * 0.9) ):
54         #cv2.drawContours(frame, [nuevoContorno], 0, (255,0,0), 1)
55         flag = True
56         texto = 'Pieza Hembra'
57         x,y,w,h = cv2.boundingRect(c)
58         madera = frame[y:y+h,x:x+w]
59         #cv2.imshow('madera',madera)
60     elif areaContorno < (area * 0.9):
61         for c in contornosMadera:
62             areaContorno = cv2.contourArea(c)
63             #print(area)
64             epsilon = 0.02*cv2.arcLength(c, True)

```

En la línea 43 se comienza a iterar sobre el arrayMascaras, si la flag es falsa se usa `findContours()` para revisar si existe el color y se itera sobre los contornos obtenidos. En la línea 47 se usó `contourArea()` para extraer el área del contorno. En la línea 49 y 50 se utiliza un algoritmo para extraer la cantidad de lados del contorno, el caso que se busca es que `aprox` sea 4. En la línea 51 se comprueba que el área sea mayor a 1000 pixeles, para evitar interferencia con otros objetos pequeños que puedan haber en los bordes de la mesa. El método `convexHull()` se usa junto con `drawContours()` para dibujar el contorno encontrado, pero no se usa en el código final (se deja para realizar prueba, de ser necesarias).

En este punto se sabe que el contorno existe, en la línea 53 se pregunta si la cantidad de lados es 4 y el área es por lo menos el 90% del área medidas durante las pruebas realizadas manualmente, entonces es una pieza hembra y se cambia la Flag. El método `boundingRect()` devuelve cuatro valores; punto inicial, punto final, ancho y largo. En la línea 58 se recorta la imagen original para extraer solo la pieza de madera.

```

58         madera = frame[y:y+h,x:x+w]
59         #cv2.imshow('madera',madera)
60     elif areaContorno < (area * 0.9):
61         for c in contornosMadera:
62             areaContorno = cv2.contourArea(c)
63             #print(area)
64             epsilon = 0.02*cv2.arcLength(c, True)
65             approx = cv2.approxPolyDP(c, epsilon, True)
66     if areaContorno > 1000:
67         nuevoContorno = cv2.convexHull(c)
68     if len(approx)==4 and ( areaContorno > (area * 0.9) ):
69         #cv2.drawContours(frame, [nuevoContorno], 0, (255,0,0), 1)
70         texto = 'Pieza Macho'
71         flag = True
72         #cv2.drawContours(frame, [nuevoContorno], 0, (0,0,0), 1)
73         x,y,w,h = cv2.boundingRect(c)
74         madera = frame[y:y+h,x:x+w]
75         #cv2.imshow('madera',madera)
76     if flag == False:
77         for c in contornosMadera:

```

Si lo anterior es falso en la línea 60 se confirma que el área existe pero es menor que el 90% del área registrada. Para saber si es una pieza de madera o un objeto haciendo interferencia se usó la variable contornosMadera definida en la línea 42, y se vuelve a realizar el código anterior, si el área es menor a 1000 pixeles es un objeto interfiriendo y se sale mas rapido del código, si la cantidad de lados es 4 y borde es similar a al área registrada entonces es una pieza macho.

Si aún no encuentra la pieza, revisará si es una pieza virgen.

```

71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
    flag = True
    #cv2.drawContours(frame, [nuevoContorno], 0, (0,0,0), 1)
    x,y,w,h = cv2.boundingRect(c)
    madera = frame[y:y+h,x:x+w]
    #cv2.imshow('madera',madera)

if flag == False:
    for c in contornosMadera:
        areaContorno = cv2.contourArea(c)
        #print(area)
        epsilon = 0.02*cv2.arcLength(c,True)
        approx = cv2.approxPolyDP(c,epsilon,True)
        if areaContorno > 1000:
            nuevoContorno = cv2.convexHull(c)
            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)

cv2.putText(frame,texto,(10,20), cv2.FONT_HERSHEY_SIMPLEX, 0.7,(255, 255, 255), 1)
#cv2.imshow('frame',frame)

```

En la línea 77 itera sobre contornosMadera y realiza lo mismo que en la línea 61, la diferencia es que si flag llegó a este punto siendo False es porque se no se encontró un diseño dentro de la pieza, pero se sabe que la pieza está, por ende, solo puede ser virgen.

En la línea 94 se aumenta el contador.

Finalmente, si el texto es diferente de “No se ha detectado la pieza”, retorna el nuevo texto, la imagen y la flag, si el While acaba significa que no se ha encontrado la pieza.

```

87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
    texto = 'Pieza Virgen'
    x,y,w,h = cv2.boundingRect(c)
    madera = frame[y:y+h,x:x+w]
    #cv2.imshow('madera',madera)

    #cv2.putText(frame,texto,(10,20), cv2.FONT_HERSHEY_SIMPLEX, 0.7,(255, 255, 255), 1)
    #cv2.imshow('frame',frame)
    count = count + 1
    if cv2.waitKey(1) & 0xFF == 27 or texto != 'No se ha detectado pieza de madera':
        #print(texto)
        count = 0
        madera = cv2.resize(madera, frameSize)
        break
    if count == 4:
        #print(texto)
        madera = 0
        count = 0
        break
return texto,madera,flag
def movimiento_brazo(textEnviodo):

```

En la línea 107 comienza el algoritmo para el movimiento del brazo robot, recibimiento el comando que se trata de enviar.

Primero se verifica que el texto exista, entonces se crea la variable “res”, que almacenará la respuesta del robot y la guardará en la lista “list\_res” y una flag para el caso de error.

El método str.encode() transforma el string a un conjunto de bytes y se envía al robot con el método write().

```
106
107     def movimiento_brazo(textEnviado):
108         if len(textEnviado) >0:
109             res = ''
110             list_res = []
111             flag_no_resq = False
112             order = str.encode(textEnviado)
113             robot.write(order+b'\r')
114             while 'Done' not in res:
115                 res = str(robot.readline())
116                 if 'r*** E' in res :
117                     flag_no_resq = True
118                     break
119                 if (res not in list_res) and (len(res)>4) and ('\\r\\n' not in res):
120                     list_res.append(res)
121             for indice,res in enumerate(list_res):
122                 if('Done.' in res):
123                     list_res[indice] = (f'>>{res[6:-2]}')
124                 else:
125                     list_res[indice] = (f'>>{res[2:-1]}')
126             if flag_no_resq == True:
127                 list_res.append('Comando no encontrado')
128                 flag_error = True
129             return list_res,flag_error
130         else:
131             flag_error = False
132         return list_res,flag_error
133
134
```

El método readline() recibe la respuesta del robot.

Se necesita más de una respuesta del robot, estas son; la confirmación del comando que se le envió, la confirmación de que sigue conectado, y el mensaje en caso de que el comando no exista, sin embargo el robot solo responde lo que se le pregunta en el momento, por lo tanto lo más probable es que se reciba un string vacío. Por ello en la línea 114 se puso un While mientras no se haya recibido el “Done”(confirmación de conexión).

En la línea 116 se preguntó por “r\*\*\* E”, este es el mensaje que devuelve el robot en caso de que el comando no exista. Ya que se pregunta tantas veces por segundo la mayoría de string's que se recibirán estarán vacíos, en la linea 119 se comprobará que la respuesta venga en un formato válido y la agregara a la lista. En el for de la línea 121 se acomoda el texto a una más legible.

Ej: \\r\\l\\Done\\. pasa a ser 'Done'. Si la flag cambio a True se agrega un 'Comando no encontrado' a la lista y se retorna la lista y la flag, si la flag es False, se envía de inmediato.

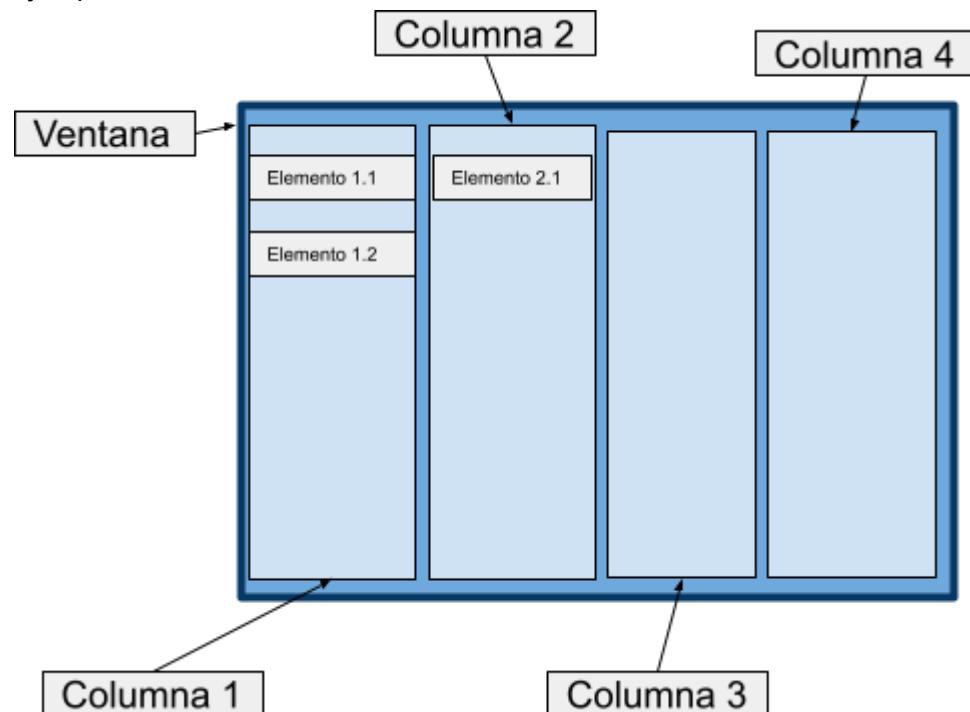
A partir de la línea 35 inicia el código. Se define el ancho y largo de la casilla de las cámaras en la interfaz y se juntan en frameSize. El método videoCapture() realiza la asignación de las cámaras.

```
134
135     camera_Width = 280 # 480 # 640 # 1024 # 1280
136     camera_Height = 180 # 320 # 480 # 780 # 960
137     frameSize = (camera_Width, camera_Height)
138
139     video_capture1 = cv2.VideoCapture(1)
140     video_capture2 = cv2.VideoCapture(2)
141     video_capture3 = cv2.VideoCapture(0)
142
143     flag_robot = 'DESCONECTADO'
144     flag_serial = True
145     flag_no_resq = False
146     flag_cam1 = False
147     flag_cam2 = False
148     flag_cam3 = False
149     flag_auto = False
150     count = 0
```

Desde la línea 143 hasta 149 se crearon una serie de flag's que se ocuparan en la activación de eventos en la interfaz.

Para la interfaz usamos PySimpleGui, su uso es simple, se define una ventana de fondo y crean diferentes columnas, y en ella cada fila representa elementos:

Ejemplo:



En la línea 152, se definió un tema de fondo para la interfaz, en la 153 y 154 listas para registrar todos los comandos enviados a través de la interfaz y los recibidos desde el robot.

En la línea 155 se define la primera columna; que contendrá un texto diciendo que son los botones para el robot, el botón para conectarlo, una caja de texto con todos los comandos enviados, un input para escribir los comandos y dos textos para indicar el tipo de pieza reconocida.

```

150 count = 0
151
152 sg.theme("DarkBlue")
153 Enviado_array = ['>>Enviado']
154 req_array = ['>>Recibido']
155 columnna1 = [
    [sg.Text(text='ROBOT', font=("Arial", 20), pad=(0,10))],
    [sg.Button("Conectar Robot", key='conectarR', size=(15, 3), pad=(10, 0))],
    [sg.Listbox(values=Enviado_array, key='Enviado'), size=(25, 16)],
    [sg.InputText('', key='EnviadoText'), size=(27, 5)],
    [sg.Text(text='', size=(20,1), pad=(0,0))],
    [sg.Text(text='', size=(20,1), pad=(0,0))],
    [sg.Text(text='Tipo De Pieza', size=(20,1), pad=(0,0))],
    [sg.Text(text='No se ha iniciado el reconocimiento', key='Pieza', size=(20,2))],
    [sg.Text(text=' ', size=(20,1), pad=(0,35))],
    [sg.Text(text=' ', pad=(0,0), justification='center')]]
165 colwebcam1 = sg.Column(columnna1, element_justification='center')
166
167 columnna2 = [
    [sg.Text(text=' ', font=("Arial", 20), pad=(0,10))],
    [sg.Text(text='DESCONECTADO', key='colorR', size=(20,3), background_color='red',
        justification='center')],
```

Para editar el texto de los elementos se le agrega “ “text=’...’” , para cambiar la letra es “ font=('tipoLetra, tamaño') ”, para cambiar el tamaño es “ size=(ancho,largo) ”, para agregar margen es “ pad=(x,y) ”, sin embargo el pad agrega el margen por ambos lados, es decir, si se le da un pad de 10px en el eje y se los dará tanto por arriba como por debajo, para no alterar el orden simplemente se llenó el espacio vacío con cajas de texto vacías. element\_justification = ‘center’ centra todos los elementos de la columna

El atributo key debe ser un valor único con el que se reconocerá el evento.

```

165 colwebcam1 = sg.Column(columnna1, element_justification='center')
166
167 columnna2 = [
    [sg.Text(text=' ', font=("Arial", 20), pad=(0,10))],
    [sg.Text(text='DESCONECTADO', key='colorR', size=(20,3), background_color='red',
        justification='center')],  

    [sg.Listbox(values=req_array, key='Recibido'), size=(40, 16)],
    [sg.Button("Enviar", key='enviarR', size=(10, 1), pad=(0,0)), sg.Button("Automatico",
        key='auto', size=(10,1)), sg.Text(text=' ', key='colorAuto',
        size=(5,1), background_color='red')],
    [sg.Image(filename="", key="cam3", size=(280,180), background_color='gray')],
    [sg.Text(text=' ', pad=(0,0), justification='center')]]
176
177 colwebcam2 = sg.Column(columnna2, element_justification='center')
178
179 columnna3 = [
    [sg.Text(text='CAMARAS', font=("Arial", 20), pad=(0,10))],
    [sg.Image(filename="", key="cam1", size=(280,180), background_color='gray')],
    [sg.Image(filename="", key="cam2", size=(280,180), background_color='gray')],
    [sg.Image(filename="", key="cam4", size=(280,180), background_color='gray', pad=(0,0))],
    [sg.Text(text='Desarrollado por Nicolás Baraza, 2022-2', pad=(0,0), justification='center')]
184 colwebcam3 = sg.Column(columnna3, element_justification='center')
185
186 columnna4 = [
    [sg.Text(text=' ', font=("Arial", 20), pad=(0,10))],
    [sg.Button("Camara 1", key='conectar1', size=(10, 1), pad=(10, 0))],
    [sg.Button("Camara 2", key='conectar2', size=(10, 1), pad=(10, 165))],
    [sg.Button("Camara 3", key='conectar3', size=(10, 1), pad=(10, 0))],
    [sg.Button("Reconocer", key='Reconocer3', size=(10, 1), pad=(10, 10))],
    [sg.Button("Terminar", key='terminar', size=(15,1), pad=(0,38))],
    [sg.Text(text=' ', pad=(0,0), justification='center')]]
193 colwebcam4 = sg.Column(columnna4)
194 colslayout = [colwebcam1, colwebcam2, colwebcam3, colwebcam4]
195 layout = [colslayout]
```

En la línea 194 se guardaron todas las lista que representan las columnas en una nueva lista, generando una matriz, esta se guardó en la variable layout, que es el nombre genérico

que se usa en la interfaces. En la línea 197 se inicia la venta, se le asigna el nombre, de se entrega el layout, el resto de atributos vienen por efecto, excepto location, que representa la esquina superior izquierda donde se inicia la ventana.

```

192     [sg.Text(text=' ', pad=(0,0), justification='center')] ]
193 colwebcam4 = sg.Column(columna4)
194 colslayout = [colwebcam1,colwebcam2,colwebcam3, colwebcam4]
195 layout = [colslayout]
196
197 window = sg.Window("Reconocimiento de Objetos", layout,
198 no_titlebar=False, alpha_channel=1, grab_anywhere=False,
199 return_keyboard_events=True, location=(200, 10))
200
201 while True:

```

En la línea 201 comienza un While donde estará el resto del código, el método windows.read() devuelve dos valores, el evento y su valor, por ejemplo el click de un botón reconocido por una única key es un evento, y el texto ingresado dentro de un input es un valor. Si el evento es la equis en la esquina superior izquierda o el botón terminar se termina el While y en la última línea del código se cierra el programa.

```

200
201 while True:
202     start_time = time.time()
203     event, values = window.read(timeout=20)
204     if event == sg.WIN_CLOSED or event == 'terminar':
205         break
206     if event == 'conectarR':
207         #conectar robot...
208         if flag_robot == 'DESCONECTADO':
209             try:
210                 robot = serial.Serial(port='COM1',baudrate=9600,timeout=0)
211             except:
212                 textEnviado = 'No se pudo conectar la serial'
213                 req_array.append(f'>{textEnviado}')
214                 window["Recibido"].update(req_array)
215                 window["EnviadoText"].update('')
216                 flag_serial = False
217             if flag_serial == True:
218                 #order0 = str.encode('move 600')
219                 #robot.write(order0+b'\r')
220                 window["conectarR"].update('Desconectar Robot')
221                 window["colorR"].update('CONECTADO',background_color="green")
222                 flag_robot = 'CONECTADO'
223             elif flag_robot == 'CONECTADO':
224                 window["conectarR"].update('Conectar Robot')
225                 window["colorR"].update('DESCONECTADO',background_color="red")
226                 flag_robot = 'DESCONECTADO'
227                 robot.close()
228
229     if event == 'enviarR':

```

Línea 206, si el evento es la conexión del robot, y su flag es “DESCONECTADO” se intenta conectar con pySerial al puerto COM1, de lo contrario se agregara el mensaje de error a la lista con los mensajes recibidos por el programa. El método windows[“{key}”].update(...) actualiza un elemento del layout. En la 114 se actualiza la lista con mensajes, y en la 115 se resetea el input para que quede vacío.

Si la flag es True significa que no se produjo el error y actualiza el texto a “Desconectar Robot” y el color del botón cambia a verde, mostrando que está conectado. Si la flag del robot es “CONECTADO” realiza el proceso contrario y cierra la conexión con el robot con robot.close().

```

227         robot.close()
228
229     if event == 'enviarR':
230         if flag_robot=='CONECTADO':
231             textEnviado = values['EnviadoText']
232             list_res, flag_error = movimiento_brazo((textEnviado))
233             Enviado_array.append(f'>>{textEnviado}')
234             window["Enviado"].update(Enviado_array)
235             window["EnviadoText"].update('')
236             if flag_error == False:
237                 req_array.extend(list_res)
238                 window["Recibido"].update(req_array)
239             else:
240                 text_req = 'Comando no encontrado'
241                 req_array.append(f'>>{text_req}')
242                 window["Recibido"].update(req_array)
243             elif flag_robot=='DESCONECTADO':
244                 textEnviado = 'El robot no esta conectado'
245                 req_array.append(f'>>{textEnviado}')
246                 window["Recibido"].update(req_array)
247                 window["EnviadoText"].update('')
248         if event == 'auto':
249             if flag_auto == False:

```

Si el evento es “enviarR” y el robot está conectado extrae el valor del input ejecuta el código del movimiento del brazo, si la flag\_error que devuelve es False agrega la lista de respuestas al lista de respuestas de la interfaz y la actualiza.

Si la flag\_error es True agrega “Comando no encontrado” y actualiza. Si el robot no está conectado agrega “El robot no está conectado” y repite lo mismo.

```

247         window["EnviadoText"].update('')
248     if event == 'auto':
249         if flag_auto == False:
250             if flag_robot=='CONECTADO':
251                 textEnviado = 'move 600'
252                 list_res, flag_error = movimiento_brazo((textEnviado))
253                 Enviado_array.append(f'>>{textEnviado}')
254                 window["Enviado"].update(Enviado_array)
255                 window["EnviadoText"].update('')
256                 if flag_error == False:
257                     req_array.extend(list_res)
258                     window["Recibido"].update(req_array)
259                 else:
260                     text_req = 'Comando no encontrado'
261                     req_array.append(f'>>{text_req}')
262                     window["Recibido"].update(req_array)
263                     if flag_cam3==True:
264                         if rot3 == True:

```

Si el evento es el botón “Automático” , su flag está en False y el robot está conectado realiza el mismo evento anterior, pero envía el texto “move 600”, esta es una posición ya registrada en el robot y es para la cual se hizo el reconocimiento de objetos.

```

262 |         window["Recibido"].update(req_array)
263 |     if flag_cam3==True:
264 |         if ret3 == True:
265 |             flag_auto = True
266 |             textEnviado = 'Modo automatico activado'
267 |             window['colorAuto'].update('',background_color='green')
268 |             req_array.append(f'>{textEnviado}')
269 |             window["Recibido"].update(req_array)
270 |         else:
271 |             window["conectar3"].update('Camara 3')
272 |             textEnviado = 'Error al conectar cam3'
273 |             req_array.append(f'>{textEnviado}')
274 |             window["Recibido"].update(req_array)
275 |             window["EnviadoText"].update('')
276 |     elif flag_robot=='DESCONECTADO':
277 |         textEnviado = 'El robot no esta conectado'
278 |         req_array.append(f'>{textEnviado}')
279 |         window["Recibido"].update(req_array)
280 |         window["EnviadoText"].update('')
281 |     else:
282 |         flag_auto = False
283 |         textEnviado = 'Modo automatico desactivado'
284 |         window['colorAuto'].update('',background_color='red')
285 |         req_array.append(f'>{textEnviado}')
286 |         window["Recibido"].update(req_array)
287 |
288 |     if flag_auto:

```

En la línea 263 comprueba si la cámara está conectada y encendida y activa la flag\_auto que ocuparemos luego, el resto de código hasta la línea 286 ya se vieron más arriba y son condiciones de rechazo y actualización de la caja de respuestas.

```

286 |         window["Recibido"].update(req_array)
287 |
288 |     if flag_auto:
289 |         texto , madera ,flag = reconocimiento_pieza(video_capture3)
290 |         if flag != False:
291 |             window["Pieza"].update(texto)
292 |             imgbytes = cv2.imencode(".png", madera)[1].tobytes()
293 |             window["cam4"].update(data=imgbytes)
294 |         elif madera == False:
295 |             window["Pieza"].update(texto)
296 |             window["cam4"].update('',size=(280,180))
297 |
298 |

```

Si la flag\_auto quedó en True llama al reconocimiento\_pieza(...) y si la flag que devuelve no es False(pieza no encontrada) actualiza la casilla de la cam4 en la interfaz, esta debe estar en formato de bytes(línea 293).

Si el evento es el botón de la cámara 1 su flag es False (desconectada), la cambia a True y actualiza el texto del botón, si ya estaba activada cambia la flag a False elimina la imagen de la casilla cam1 en la interfaz y cambia el botón “Desconectar” a “Cámara 1”

```
295         window[ "rieza" ].update( texto )
296         window[ "cam4" ].update( '' , size=(280,180) )
297
298
299 if event == 'conectar1':
300     if flag_cam1 == False:
301         flag_cam1 = True
302         window[ "conectar1" ].update('Desconectar')
303     elif flag_cam1 == True:
304         flag_cam1 = False
305         window[ "cam1" ].update( '' , size=(280,180) )
306         window[ "conectar1" ].update('Camara 1')
307
308 if flag_cam1==True:
309     ret1, frameOrig1 = video_capture1.read()
310     if ret1 == True:
```

Ahora que la flag\_cam1 se mantiene en True para cada ciclo del While, comprueba que esté llegando la imagen desde la cámara y en la línea 311 cambia su tamaño con el método resize de openCV, pasa la imagen a bytes y actualiza la cam 1.

```
305         window[ "cam1" ].update( '' , size=(280,180) )
306         window[ "conectar1" ].update('Camara 1')
307
308 if flag_cam1==True:
309     ret1, frameOrig1 = video_capture1.read()
310     if ret1 == True:
311         frame1 = cv2.resize(frameOrig1, frameSize)
312         imgbytes = cv2.imencode(".png", frame1)[1].tobytes()
313         window[ "cam1" ].update(data=imgbytes)
314     else:
315         window[ "conectar1" ].update('Camara 1')
316         textEnviado = 'Error al conectar camara 1'
317         req_array.append(f'>>{textEnviado}')
318         window[ "Recibido" ].update(req_array)
319         window[ "EnviadoText" ].update('')
320         flag_cam1=False
321
322 if event == 'conectar2':
323     if flag_cam2 == False:
324         flag_cam2 = True
```

Se repite exactamente el mismo código para la cámara 2:

```

320         flag_cam1=False
321
322     if event == 'conectar2':
323         if flag_cam2 == False:
324             flag_cam2 = True
325             window["conectar2"].update('Desconectar')
326         elif flag_cam2 == True:
327             flag_cam2 = False
328             window["cam2"].update('',size=(280,180))
329             window["conectar2"].update('Camara 2')
330     # get camera frame
331     if flag_cam2==True:
332         ret2, frameOrig2 = video_capture2.read()
333         if ret2 == True:
334             frame2 = cv2.resize(frameOrig2, frameSize)
335             imgbytes = cv2.imencode(".png", frame2)[1].tobytes()
336             window["cam2"].update(data=imgbytes)
337     else:
338         window["conectar2"].update('Camara 2')
339         textEnviado = 'Error al conectar camara 2'
340         req_array.append(f'>>{textEnviado}')
341         window["Recibido"].update(req_array)
342         window["EnviadoText"].update('')
343         flag_cam2=False
344
345     if event == 'conectar3':
346         if flag_cam3 == False:

```

El código para la cámara 3 es similar, sin embargo se modificó un poco para agregar el evento del reconocimiento.

```

343         flag_cam2=False
344
345     if event == 'conectar3':
346         if flag_cam3 == False:
347             flag_cam3 = True
348             window["conectar3"].update('Desconectar')
349         elif flag_cam3 == True:
350             flag_cam3 = False
351             window["cam3"].update('',size=(280,180))
352             window["cam4"].update('',size=(280,180))
353             window["Pieza"].update('Camara Desconectada')
354             window["conectar3"].update('Camara 3')
355     # get camera frame
356     if flag_cam3==True:
357         if flag_recognition==False:

```

Si la flag:cam3 esta en True, la imagen se está actualizando correctamente y se activa el evento de reconocimiento, llama a reconocimiento\_piezas() y actualiza la imagen igual que en el evento de “Automático”.

```

354         window["conectar3"].update('Camara 3')
355     # get camera frame
356     if flag_cam3==True:
357         ret3, frameOrig3 = video_capture3.read()
358         if ret3 == True:
359             frame3 = cv2.resize(frameOrig3, frameSize)
360             imgbytes = cv2.imencode(".png", frame3)[1].tobytes()
361             window["cam3"].update(data=imgbytes)
362             if event == 'Reconocer3':
363                 texto , madera ,flag = reconocimiento_pieza(video_capture3)
364                 if flag != False:
365                     window["Pieza"].update(texto)
366                     imgbytes = cv2.imencode(".png", madera)[1].tobytes()
367                     window["cam4"].update(data=imgbytes)
368                 elif madera == False:
369                     window["Pieza"].update(texto)
370                     window["cam4"].update('',size=(280,180))
371             else:
372                 window["conectar3"].update('Camara 3')
373                 textEnviado = 'Error al conectar cam3'
374                 req_array.append(f'>>{textEnviado}')
375                 window["Recibido"].update(req_array)
376                 window["EnviadoText"].update('')
377

```

Si se presionaba la equis o el evento “Terminar” se rompe el ciclo While, se destruyen las imágenes generadas y termina el programa.

```

375         window["Recibido"].update(req_array)
376         window["EnviadoText"].update('')
377
378
379     cv2.destroyAllWindows()

```

# Conclusión

Este proyecto tuvo como principal objetivo la creación de un programa capaz de reconocer piezas de madera. Este objetivo se logró conseguir por medio del procesamiento de imágenes con OpenCV., ya que se sabía de antemano cómo serían las piezas fue posible utilizar visión artificial para identificar específicamente la forma y el color para poder discriminar el tipo de pieza.

Sin embargo, aun así se presentaron algunas dificultades durante la realización del código, puesto que originalmente se tenía previsto utilizar solo el color negro, esto debió cambiar debido a la imposibilidad de diferenciar entre la pieza y la mesa. Finalmente se concluyó que lo mejor era simplemente utilizar diferentes colores para asegurar una clara distinción entre las piezas y el entorno.

El código se ha enviado junto con esta documentación y se ha comentado casi todo el código para facilitar su comprensión. Esperando que este programa les sea de utilidad y que el próximo avance sea igual de próspero, me despido.