Visión Artificial

Curso 2022-2023

Alumno: Gonzalo Rodriguez Rovira **Grupo: 1.3**

DNI: 26557227R

Índice

CALIBRACIÓN

a) Realiza una calibración precisa de tu cámara mediante múltiples imágenes de un ***chessboard***.

Para realizar una calibración precisa de mi cámara ejecutare el programa calibrate.py pasándole como parámetros unas fotos realizadas con la cámara a calibrar. Pero antes de ello al tomar mi cámara fotos en alta resolución y con el formato .jpg debo aplicarles a todas las fotos un "convert -resize 20% *.jpg r.png", de esta manera si poder utilizarlas en mi programa calibrate.py, destacar de tuve que modificar el programa ya que este estaba diseñado para detectar *chessboard* de un tamaño mayor al mío el cual es un 8x8.

El resultado obtenido es el siguiente:

RMS: 0.623315167274942

camera matrix:

[[613.81777024 0. 306.95216681]

[0. 614.27775787 414.30159544]

[0. 0. 1.]]

distortion coefficients: [1.08767824e-01 -2.23879625e-01 -5.24661512e-05 -3.37519432e-04

2.46443829e-01]

Una vez sabemos que la f = 613.81777024, puedo despejar el FOV de la cámara con la siguiente formula.

$$\tan\left(\frac{FOV}{2}\right) = \frac{\frac{w}{2}}{f}$$

Sabiendo que la resolución de las fotos tomadas es de 605 x 806 podré despejar el FOV horizontal y vertical.

FOV horizontal:

tan(FOVh/2)=605/2/613.8

FOVh=52.47

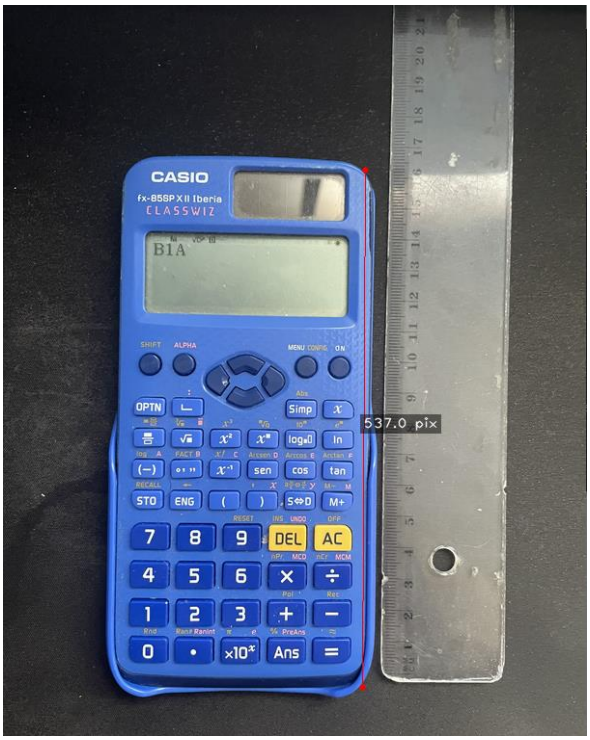
FOV vertical:

tan(FOVv/2)=806/2/613.8

FOVv=66.575

b) Haz una calibración aproximada con un objeto de tamaño conocido y compara con el resultado anterior.

Para este apartado tomare una foto de una calculadora cuyo tamaño es de X=16cm, la distancia es de Z=20cm y finalmente he calculado mediante el uso de medidor.py el número de pixeles en la foto u=537pix (He tenido que realizar de nuevo un convert a la imagen).



Aplicando la siguiente formula podré despejar la f:

$$u = f\frac{X}{Z}$$

f=429pix

Al igual que el ejercicio anterior despejare el FOV con la misma fórmula, dando de resultado:

FOVh=70.377

FOVv=86.42

Podemos observar que al calcular el FOV de manera aproximada da valores algo superiores a la forma mas precisa, siendo en el caso horizontal de un 34.1% de diferencia a favor de la forma imprecisa y en el caso vertical de un 29.8% de diferencia a favor de la forma imprecisa.

c) Determina a qué altura hay que poner la cámara para obtener una vista cenital completa de un campo de baloncesto.

Para realizar este apartado en primer lugar debo despejar la f, para ello supondré que mi FOV horizontal es correcto a 52.47, además sabiendo la resolución que es de 605 x 806 podré despejar la f de aquí:

$$\tan\left(\frac{FOV}{2}\right) = \frac{\frac{w}{2}}{f}$$

Siendo f =613.81, el cual es el mismo valor que originalmente nos daba calibrate.py.

Una vez teniendo la f y suponiendo el tamaño de campo a 28 metros X=2800cm y que ocupara los mismos pixeles que en el otro ejemplo u=537pix por lo tanto, solo tendré que despejar la distancia que es Z.

$$u = f\frac{X}{Z}$$

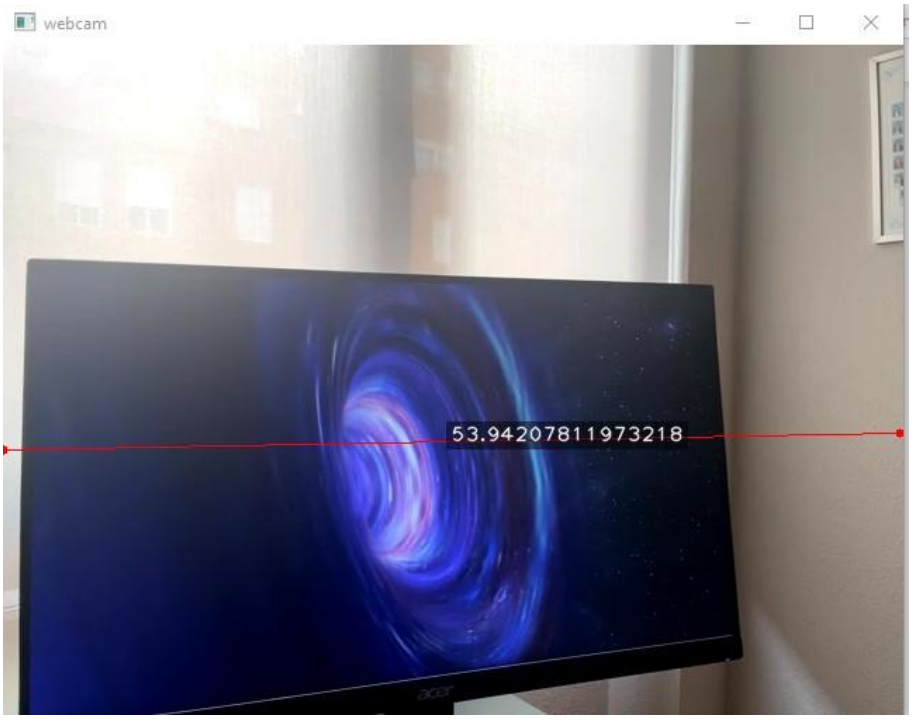
Z=3200.5cm = 32,2m de altura.

d) Haz una aplicación para medir el ángulo que definen dos puntos marcados con el ratón en la imagen.

Primero mostraré mi código con toda su explicación:

```
#!/usr/bin/env python

import cv2 as cv
import math
from umucv.stream import autoStream
from collections import deque
import numpy as np
from umucv.util import putText


points = deque(maxlen=2)#Declaro una lista de maximo dos elementos para poder guardas las cordenadas x y de mis dos puntos
f=613.81#Parametro f sacado de calibrate.py
w=605#Parametro ancho en pixeles de las fotos de mi camara
h=806#Parametro alto en pixeles de las fotos de mi camara

def fun(event, x, y, flags, param):#Defino una función para que en caso de dar un click en la pantalla me guarde las cordendas x,y del punto clikeado
    if event == cv.EVENT_LBUTTONDOWN:
        points.append((x,y))

cv.namedWindow("webcam")#Inicializo mi ventana donde monstrare el video
cv.setMouseCallback("webcam", fun)#Asigno a mi ventana  la funcion previamente declarada

for key, frame in autoStream():

    for p in points:#Para cada punto que forma mi lista de puntos les dibujo un circulo en sus cordenadas

        cv.circle(frame, p,3,(0,0,255),-1)

    if len(points) == 2:#Una vez que tengo dos puntos puedo empezar a calcular sobre estos los angulos que forman

        cv.line(frame, points[0],points[1],(0,0,255))#Dibujo una linea entre mis dos puntos
        c = np.mean(points, axis=0).astype(int)#Calculo el punto medio de mis dos puntos para poder colocar en su posición el valor del angulo que forman

        u=[(points[0])[0]-w/2,(points[0])[1]-h/2,f]#Calculo mi matriz K completa
        v=[(points[1])[0]-w/2,(points[1])[1]-h/2,f]


        a=(np.dot(u,v))/(np.linalg.norm(u)*np.linalg.norm(v))#Despejo mi angulo al calcular el producto escalar de mis matrices y lo divido entre la multiplicacion de los
modulos de mis vectores
        a= np.rad2deg((math.acos(a)))#Finalmente paso a grados mi valor

        putText(frame,f'{a}', c)#Coloco mi valor del angulo

    cv.imshow('webcam',frame)

cv.destroyAllWindows()
```

A continuación, muestro una imagen de ejemplo obtenida de mi programa, siendo el FOVh=53



ACTIVIDAD

a) Utilizando un substractor de fondo de opencv como en los ejemplos backsub0.py y backsub.py.

Para este apartado al usar esos dos ejemplos llego a la conclusión de que resuelven el problema mediante el uso de sustracción de fondos de createBackgroundSubtractorMOG2, de esta manera podre crear un frame en negro que volverá sus pixeles blancos al detectar movimiento y mediante la suma de los pixeles blancos podré detectar el movimiento.

```
if region.roi:#Si tengo ya una zona marcadaSS

    if seleccionado==False:#Fase de seleccionado de zona a observar

        [x1,y1,x2,y2] = region.roi

    if key == ord('c'): #Empiezo a capturar movimiento en la region

        seleccionado=True#Indico que ya he seleccionado un parte a observar y puede pasar a la fase de observacion
        trozo1 = frame[y1:y2+1, x1:x2+1]#Obtengo la imagen inicial la cual comparare a la hora de buscar movimiento, al usar
                                        #bgsub sirve con comparar una imagen negra con la fmask que se generara al detectar movimiento ya que no sera negra
        height, width, channels = trozo1.shape
        minimoMovimiento=height*width*0.3 #Constantes usadas para delimitar los rangos en los ue grabar
        maximoMovimiento=height*width
```

En esta parte del código si se cumple que ya he seleccionado mi región del frame a observar en primer lugar guardare la región, después indicare que si se pulsa la tecla c se comenzara con la detección del movimiento, marcando la variable de seleccionado a True que indica que ya esta seleccionada la zona a observar y guardare un frame nuevo que corresponde a la sección a observar, una vez hecho esto sacare la altura y anchura de este trozo y obtendré las variables minimoMovimiento y maximoMovimiento para controlar la activación de la grabación.

```
if key == ord('x'): #Dejo de capturar movimiento en la region

    region.roi = []#Vacio las cordenadas de mi rectangulo

    if seleccionado==True:#En el caso de que tenga algo seleccionado y lo tenga en observacion al pusar x se destruira la ventana que detecta cambios

        cv.destroyWindow('mask')#En caso de que quiera dejar de observar una zona debo elimnar las ventamas que me transmitian la informacion de esa zona
        cv.destroyWindow('object')
        seleccionado=False#Indico que debo volver a la fase de seleccionado
        video.ON=False#Al eliminar la region observada se para el video si estaba siendo capturado
        video = Video(fps=15, codec="MJPG",ext="avi")#Cada vez que selecciono una nueva zona debo crear nuevas secuencias de videos, no todo en el mismo archivo
```

En esta otra parte trato el tema de eliminación de la zona de observación en caso de querer observar otra parte de mi frame, para ello siempre que se pulse la tecla x se eliminara mi región de observación (la parte física que se muestra, es decir el rectángulo verde) y si esa zona estaba en observación (había sido pulsada la tecla c) se destruirán las ventanas que nos mostraban información sobre esa zona, se pasara la variable de seleccionado a False para indicar que ya no esta seleccionada ninguna zona y se parará el video en caso de que estuviera grabando y se creará un nuevo video para la zona nueva, para tener videos separados de cada zona.

```
if seleccionado==True:#Fase de observacion
    trozo = frame[y1:y2+1, x1:x2+1]#Recorto del frame la zona a observar y la guardo en otro frame
    fgmask = bgsub.apply(trozo)#Le aplico la deteccion de movimiento a esa zona
    cv.imshow('mask', fgmask)
    fgmask2 = bgsub.apply(trozo, learningRate = -1 if seleccionado else 0)#Realizo la sustraccion de fondo que monstrara recortada solo al zona del movimiento
    fgmask2 = cv.erode(fgmask2,kernel,iterations = 1)
    fgmask2 = cv.medianBlur(fgmask2,3)
    masked = trozo.copy()
    masked[fgmask2==0] = 0
    cv.imshow('object', masked)
```

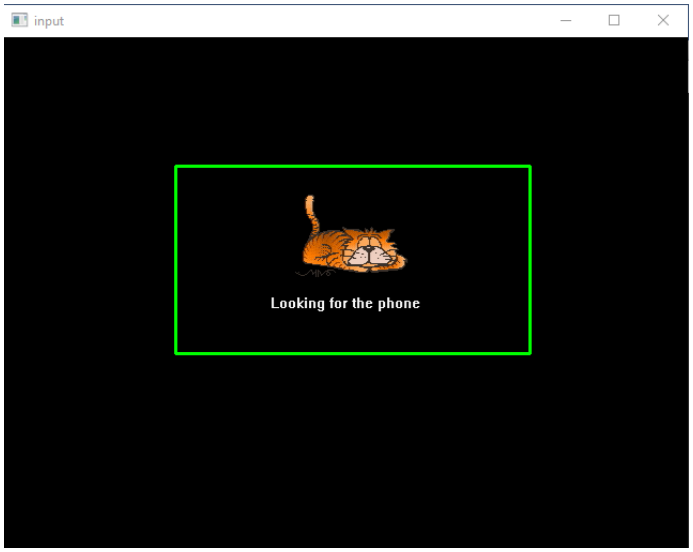
En esta otra parte se puede apreciar la parte de observación (tenemos un trozo seleccionado y se pulso la tecla c) en esta parte le aplico a mi trozo la sustracción de fondo, obteniendo por un lado la detección de movimiento a esa zona en fgmask (ventana mask) y la zona de movimiento recortada en masked (ventana object).

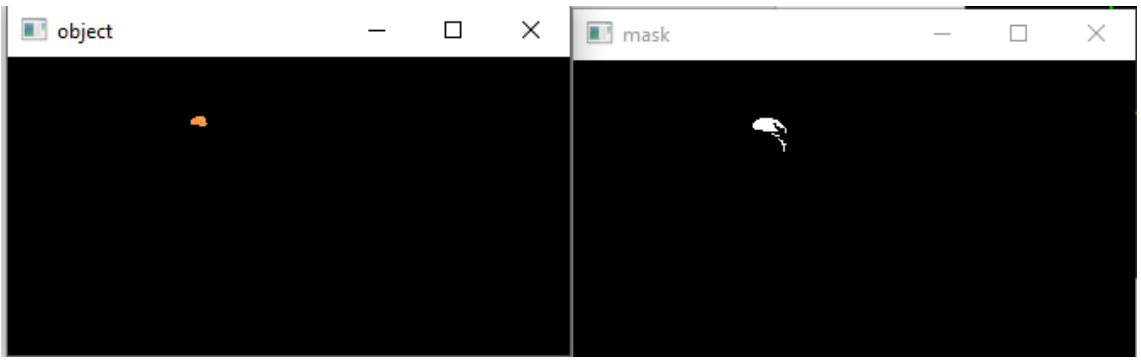
```
        if fgmask.sum()>minimoMovimiento and fgmask.sum()<maximoMovimiento: #Al detectar movimiento comparando la imagen en negro(inicial) con la que se esta capturando
            video.ON=True#Al encontrar que son distintas empiezo a grabar en caso contrario detengo

        else:
            video.ON=False

    cv.rectangle(frame, (x1,y1), (x2,y2), color=(0,255,0), thickness=2)#Dibujo el rectangulo de la zona seleccionada
```

Finalmente, en esta parte trato el tema de la grabación con las variables ya comentadas y muestro el rectángulo que delimita la zona a observarse.

Quiero comentar que este problema tiene algunos falsos positivos cuando está expuesto a la iluminación ya que la ventana mask detecta movimiento al haber cambios de luz por lo que no es muy exacto en zonas en las que no se puede controlar la iluminación totalmente.

A continuación, mostraré un ejemplo en el que se puede ver un video de un gato moviendo su cola y se detecta movimiento y se comienza con la grabación:





b) Mediante un procedimiento sencillo que construya un modelo de fondo con frames anteriores y compare con el actual.

Para este ejercicio me he basado en deque.py, al no poder realizar la sustracción de fondos he creado una lista de 20 frame la cual en caso de añadir uno estando llena elimina el frame más viejo, de esta manera puedo detectar el movimiento sacando la diferencia de el mas reciente con el mas antiguo de esa lista y sumando todos los pixeles y comparándolos con la constante minimoMovimientoya que en caso de ser mayor la diferencia a este valor iniciare la grabación, uso este valor para no caer en falsos positivos por la iluminación.

En este ejercicio sigo el mismo procedimiento que en apartado anterior solo cambiando en este parte en la que realizo la diferencia como ya he comentado.

```
    if (cv.absdiff(d[0],d[-1])).sum()>minimoMovimiento: #Al detectar movimiento comparando la imagen en negro(inicial) con la que se esta capturando
                                        #al encontrar que son distintas empiezo a grabar en caso contrario detengo
        video.ON=True#Al encontrar que son distintas empiezo a grabar en caso contrario detengo
    else:
        video.ON=False

cv.rectangle(frame, (x1,y1), (x2,y2), color=(0,255,0), thickness=2)#Dibujo el rectangulo de la zona seleccionada
```

Quiero comentar que este problema tiene algunos falsos positivos cuando está expuesto a la iluminación ya que la ventana mask detecta movimiento al haber cambios de luz por lo que no es muy exacto en zonas en las que no se puede controlar la iluminación totalmente.

COLOR

a) Construye un contador de objetos que tengan un color característico en la escena, simplemente pinchando con el ratón en dos o tres de ellos.

Para este ejercicio en primer lugar me cree una función para guardas los colores una vez dados click en formato HSV.

```
def seleccionaColor(event, x, y, flags, param):#Función de callback del mouse para obtener los colores

    if event == cv.EVENT_LBUTTONUP:
            hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)# Obtengo el valor HSV de cada pixel seleccionado
            colors.append(hsv[y,x])#Añado ese valor a mis lista de colores
```

Plantee el programa separándolo en dos secciones, la sección de seleccionado de colores y la sección de búsqueda de colores y tratado de estos.

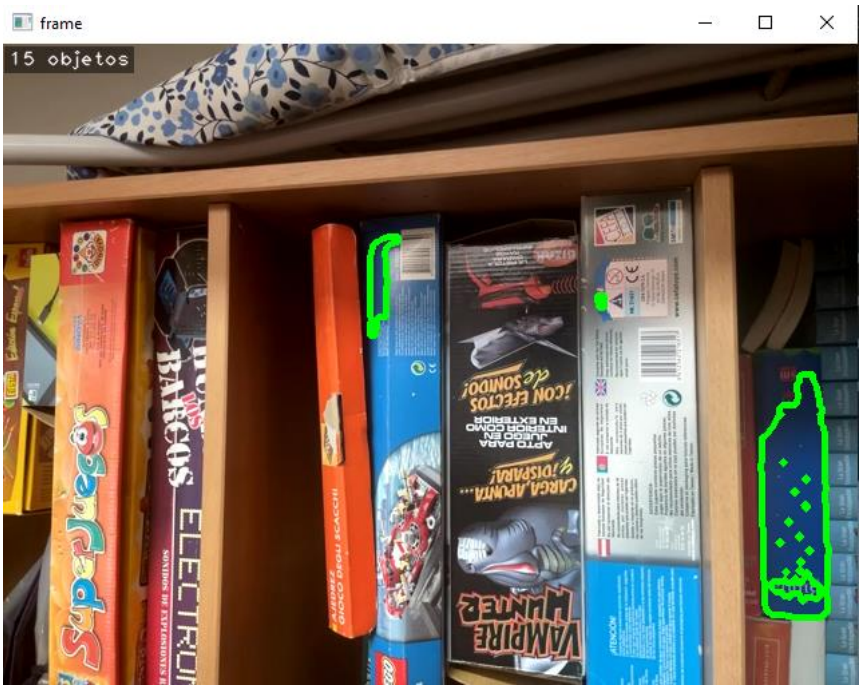
En la primera parte, asigno mi función creada al click del ratón, posteriormente paso mi frame a HSV y finalmente compruebo cuando tengo mis colores seleccionados que en este caso es al dar tres clicks, en el momento que se cumple esa condición puedo pasar a la siguiente fase.

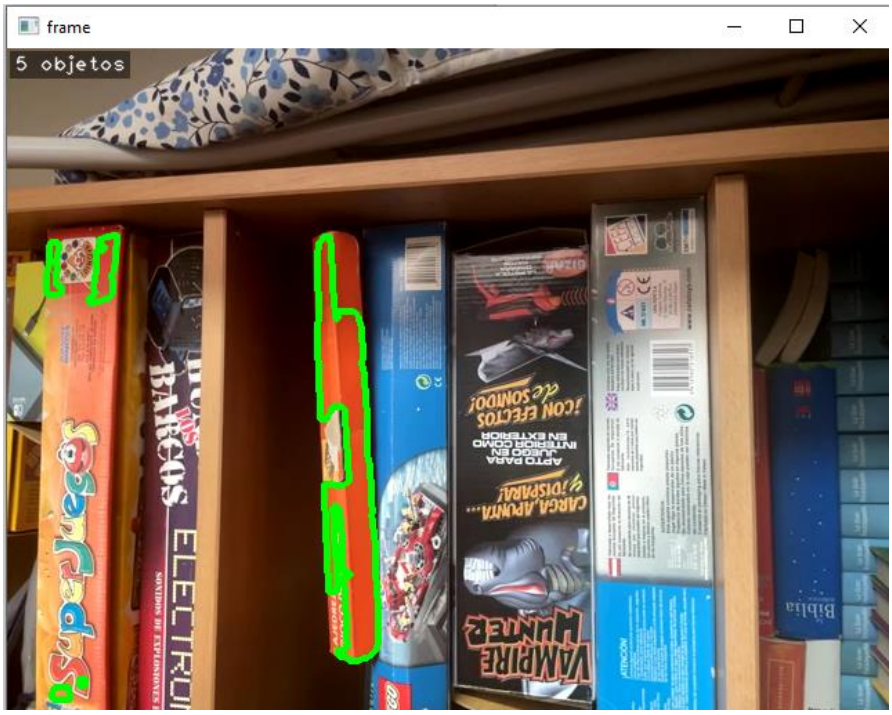
```
for key, frame in autoStream():

    if not seleccionado:#Parte de seleccionado

        cv.imshow('frame', frame)
        frame2=frame#Guardo el frame original para el caso de que quiera seleccionar un nuevo colo vuelva a la original antes de realizar los
                    #tres click en un nuevo tono

        hsv_frame = cv.cvtColor(frame, cv.COLOR_BGR2HSV)#Convierto la imagen a HSV

        if len(colors) == 3:#Mientras muestro la  imagen  esperaro a que los colores sean seleccionados, cuando sean seleccionados paso
                            #a la parte de  buscar y dibujar los contronos de estos
            seleccionado= True
```

En la segunda parte, realizo la media de mis colores seleccionados y obtengo el rango de error de colores que van a ser tomados tanto con diferencia por arriba como por abajo, una vez tengo esto suavizo mi imagen para detectar colores en esta, realizo una eliminación de objetos no deseados y finalmente obtengo los contornos de los objetos que tienen los mismos colores los cuento y los imprimo tanto su valor numérico como su contorno en el frame. En caso de dar pulsar la tecla x se entenderá que voy a seleccionar un nuevo color y deberé dar tres nuevamente para saber que color nuevo debo buscar en la imagen.

```
    if seleccionado:#Parte buscado y dibujado de contornos

        mean_color = np.mean(colors, axis=0) #Calculo la media de los valores HSV de los colores seleccionados
        lower_color = np.array([mean_color[0]-10, mean_color[1]-50, mean_color[2]-50]) #Defino el rango, límites superiores e inferiores de rango de color en HSV
        upper_color = np.array([mean_color[0]+10, mean_color[1]+50, mean_color[2]+50])

        mask = cv.inRange(hsv_frame, lower_color, upper_color)#Aplico la máscara a la imagen
        opening = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)#Operación para eliminar pequeños objetos no deseados
        contours, hierarchy = cv.findContours(opening, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)#busco los contornos de los objetos en la máscara

        cv.drawContours(frame, contours, -1, (0,255,0), 3)#Dibujo los contornos encontrados sobre el frame original con el mismo tono de color
        putText(frame, f'{len(contours)} objetos')#Muestro el numero de objetos encontrados del mismo tono

        if key == ord('x'):#En caso de que seleccione un nuevo tono

            seleccionado=False#Indico que debo volver a la patr de seleccion
            colors.clear()#Vacio todos los colores
            frame=frame2#Vuelvo a mi frame sin los contronos dibujados

        cv.imshow('frame', frame)
```

Finalmente mostraré algunos ejemplos del funcionamiento del programa:





FILTROS

a) Amplía el código de la práctica 4 para mostrar en vivo el efecto de diferentes filtros, seleccionando con el teclado el filtro deseado y modificando sus parámetros (p.ej. el nivel de suavizado) con trackbars. Aplica el filtro en un ROI para comparar el resultado con el resto de la imagen.

Para este ejercicio he tomado las siguientes decisiones de diseño: En primer lugar he decido que solo se puede aplicar un filtro cada vez y que para retirar el filtro solo pueda hacerse mediante el uso del 0 ya que este elimina el filtro actual y permite poner uno nuevo pulsando su respectiva tecla, otra decisión tomada ha sido la de seleccionar manualmente la región donde se aplicarán los filtros en caso de que esta ya este seleccionada y que queramos aplicar los filtros solo en esa sección, una vez colocado (para ello debes de estar en el modo de aplicar el filtro solo en la sección aunque aún no exista) una vez elegida la sección se debe pulsar la tecla m para fijar la sección, ya que una vez seleccionada esta no se puede recolocar y todos los filtros se aplicaran en esa sección si te encuentras en el modo de aplicar filtros en la sección.

En esta primera parte podemos ver la creación del menú y las variables de control que serán utilizadas

```
help = Help(#Menu de ayuda
"""
BLUR FILTERS

0: do nothing
1: Box
2: Gaussian
3: Diferencia
4: Premask
5: Mask
6: Masked

c: color/monochrome
r: only roi
m: fijar region

h: show/hide help
""")

color=True#Varaiable para controlar si se usa blanco y negro o color
roi=False#Variable para controlar si ya ha sido marcada la zona del filtro
todo=False#Variable de control para saber si aplicacar el filtro en toda la ventana o solo en una seccion
seleccionado=False#Varaibles para el control de la actividad de los filtros
seleccionado1=False
seleccionado2=False
seleccionado3=False
seleccionado4=False
seleccionado5=False
seleccionado6=False
```

En esta parte podemos ver por un lado la acción de eliminar cualquier filtro, posteriormente la selección de aplicar el filtro a toda la imagen o solo a la zona ya seleccionada, la acción de mostrar el menú, la confirmación de la zona seleccionada y la selección de si mostrar en blanco y negro o a color la imagen.

```
for key,frame in autoStream():

    if key == ord('0'):#En caso de pulsar 0 se quitara el filtro activo

        seleccionado=False
        seleccionado1=False
        seleccionado2=False
        seleccionado3=False
        seleccionado4=False
        seleccionado5=False
        seleccionado6=False

    if key == ord('r'):#En caso de pulsar r el flitro se aplicara a toda la ventana o a la region ya seleccionada

        todo=not todo

    help.show_if(key, ord('h'))#Monstrar menu de ayuda

    if region.roi or todo:#Solo se monstraran los filtros si tengo o bien una region seleccionada donde aplicarlo o bien estoy aplicando el fultro en toda mi ventana

        frameTodo=frame#Inicializo el frame que monstrará tanto el frame original como la zona con un filtro

        if not roi and not todo:#En el caso de que aun no tenga seleccionado mi region y no estar en el modo de toda la pantalla

            [x1,y1,x2,y2] = region.roi #Obtengo los valores de mi sección

        if key == ord('m'):#Confirmo que quiero esa sección para aplicar los filtros

            roi=True

        if key == ord('c'):#Intercambio entre color y blanc y negro

            color= not color

        if not color:

            result = cv.cvtColor(frame,cv.COLOR_RGB2GRAY)#Transformo mi imagen a color en blanco y negro
            result = cv.cvtColor(result,cv.COLOR_GRAY2RGB)#Para seguir teniendo tres canales y poder sustituir en mi imagen a color una imagen en blanco y negro debo
volverla a RGB para tener 3 canales
            if roi:#En caso de tener una seccion ya seleccionada
                frame[y1:y2+1, x1:x2+1]=result[y1:y2+1, x1:x2+1]#Reenplazo la seccion por ella misma en blanco y negro

        else:

            result = frame#En caso de ser a color mantengo mi frame en result

        if todo: #En caso de estar aplicando los filtros en la ventana entera le asigno result ya que indicara si mantengo color o blanco y negro

                frameTodo=result
```

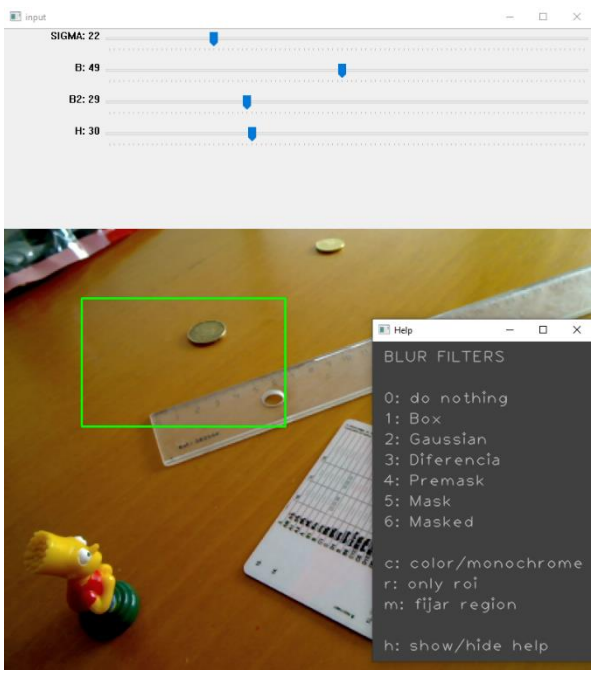
En esta parte se puede ver la creación de un filtro y su activación además de tratar la aplicación solo a la zona de este filtro o su aplicación a toda la imagen.

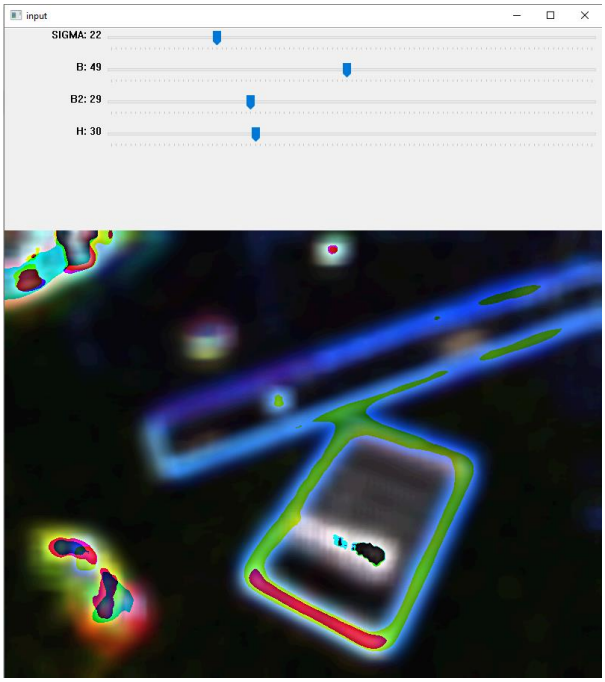
```
        #Creacion de filtro 2
        t1 = time.time()
        smooth = cv.GaussianBlur(result,(0,0),SIGMA[0])
        t2=time.time()
        putText(smooth,f'sigma={SIGMA[0]:.1f}')
        putText(smooth,f'{1000*(t2-t1):5.1f}ms',orig=(5,35))

        if key == ord('2') and not seleccionado: #En caso de querer activar el filtro 2 y no tener otro activo

            seleccionado2=not seleccionado2#Indico que selecciono el filtro 2
            seleccionado= not seleccionado#Indico que hay un filtro activo

        if seleccionado2:#En caso de tener el filtro seleccionado

            if todo:#En caso de aplicarse a toda la ventana

                frameTodo=cv.addWeighted(result,0,smooth,1,0)#Le doy una mayor prioridad a smooth para que se muestre

            else:#En caso de aplicarse solo a la zona seleccionada

                frame[y1:y2+1, x1:x2+1]=smooth[y1:y2+1, x1:x2+1]#Coloco la parte seleccionada de smooth en mi frame original
```

La creación del resto de filtros es bastante similar al previamente mostrado solo que aplicación las transformaciones pertinentes a la imagen para que se muestre como se desea.

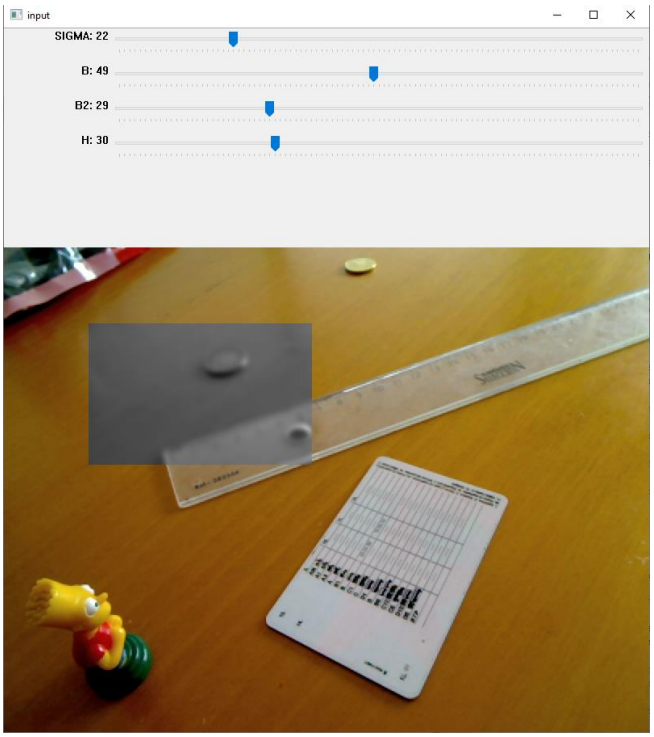
Finalmente mostraré algunos ejemplos de ejecución del programa:

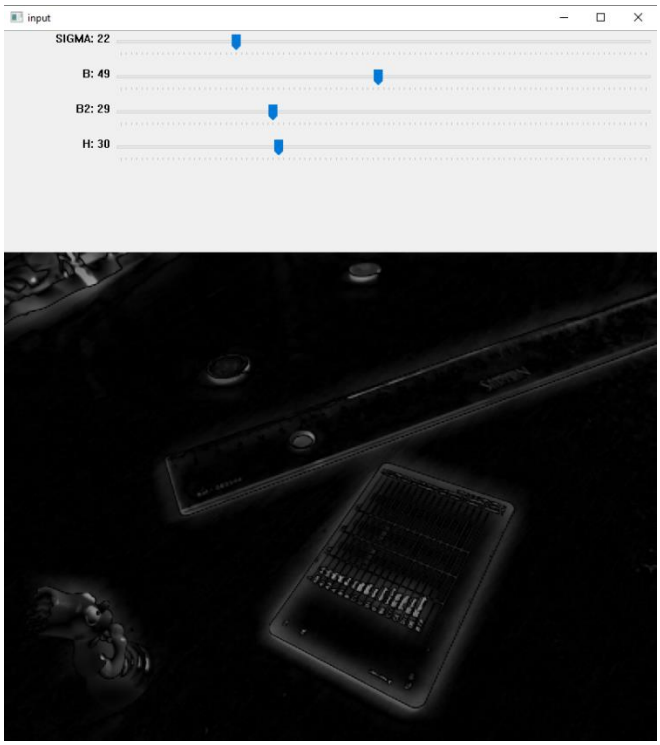
En este ejemplo se muestra el menú y la parte previa a la marcación (antes de pulsar la tecla m), en la segunda imagen se puede ver la aplicación del filtro 4 a toda la imagen.





En estos siguientes ejemplos se puede ver la selección de color blanco y negro y aplicándose en la sección marcada, así como la aplicación del filtro 1, en la segunda imagen se puede ver la aplicación del filtro 2 a toda la imagen con la selección del color en blanco y negro.





SIFT

a) Escribe una aplicación de reconocimiento de objetos (p. ej. carátulas de CD, portadas de libros, cuadros de pintores, etc.) con la webcam basada en el número de coincidencias de keypoints.

En primer lugar, cree una constante para controlar el número de keypoints además de la detección de características, un objeto de coincidencia de características, obtengo las imágenes a comparar de mi carpeta de imágenes y creo una lista donde colocaré las imágenes a buscar dentro de mi carpeta, para solo obtener imágenes en mi lista que sean imágenes del tipo .jpg o .png creo una función para filtrar el contenido de la carpeta.

```
KEYPOINTS=100#Cantidad de caracteristicas a observar de la imagen

sift = cv.SIFT_create(nfeatures=KEYPOINTS)#Creo la deteccion de caracteristicas

matcher = cv.BFMatcher()#Creo un objeto de coincidencia de características
imagenesAux = os.listdir('imagenes')#Obtengo las imagenes de mi carpeta de imagenes
imagenes=[]#Creo una lista de mis imagenes a buscar


for imagen in imagenesAux:#Me aseguro de coger solo imagenes de mi carpeta con formato .png o .jpg

    if '.jpg' in imagen or '.png' in imagen:
        imagenes.append(imagen)
```

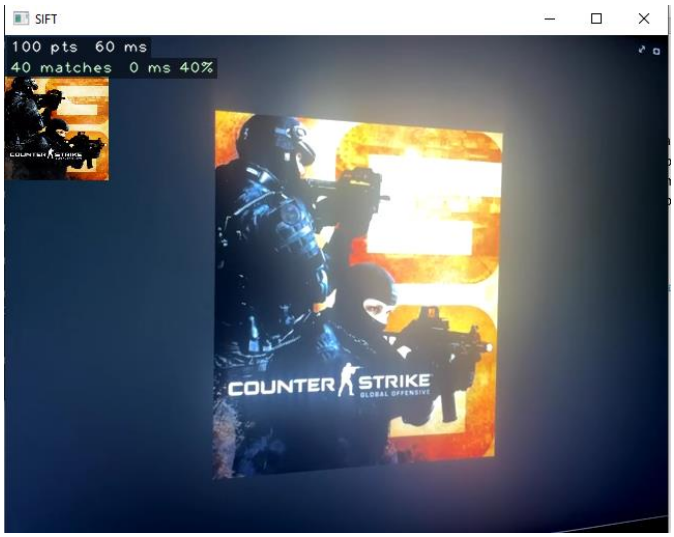
Creo un bucle para recorrer el nombre de todas mis imágenes y con la función imread le añado a la ruta ese nombre para obtener las imágenes, llamo al detector para cada una de mis imágenes, solicito las dos mejores coincidencias de cada punto y creo una lista para guardarme las mejores coincidencias.

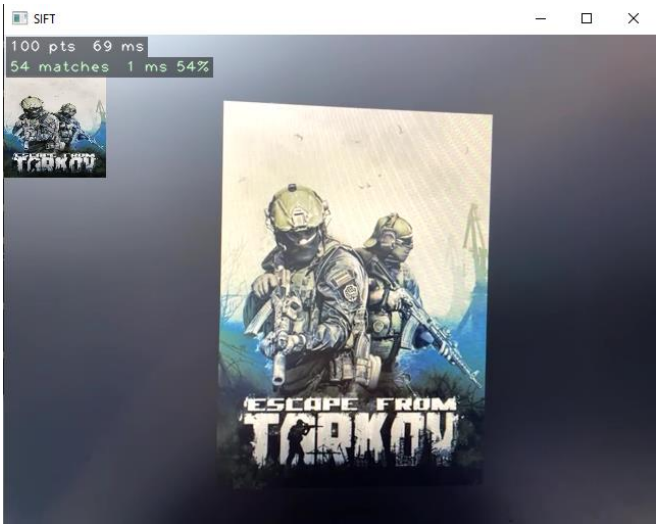
```
for imagen in imagenes:#Recorro mis imagenes buscando cual de todas es la que corresponde con la imagen capturada

        frame2=cv.imread('imagenes/'+imagen)#Obtengo un frame de cada imagen a buscar
        keypoints2 , descriptors2 = sift.detectAndCompute(frame2, mask=None)#LLamo al detector para cada una de mis imagenes a buscar
        k0, d0, x0 = keypoints2, descriptors2, frame2
        t2 = time.time()
        matches = matcher.knnMatch(descriptors, d0, k=2)#Solicito las dos mejores coincidencias de cada punto, no solo la mejor
        t3 = time.time()
        good = []#Creo una lista de mis coincidencias correctas

        for m in matches:#Recorro mis coincidencias y las validas las añado a mi lista de correctas

            if len(m) >= 2:
                best,second = m
                if best.distance < 0.75*second.distance:
                    good.append(best)
```

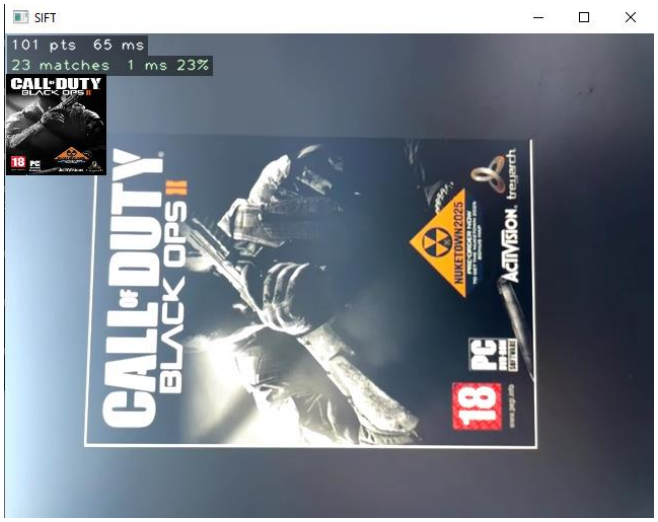
Finalmente compruebo la longitud de mis buenas coincidencias en caso de ser mayor al 20% de mis keypoints, es decir que tengo un acierto del 20% tomare como que he reconocido una imagen y por lo tanto la mostraré en mi imagen y indicare con texto datos del tiempo, así como el numero de coincidencias en porcentaje, en caso de no superar el 20% mostrare el frame normal sin mostrar la coincidencia con alguna imagen.

```
if (len(good)>KEYPOINTS*0.2):#En caso de tener una coincidencia mayor al 20% se tomara como si se ha encontrado el objeto

    frame[40:140,0:100]=cv.resize(frame2,(100,100))#Coloco la imagen con la que tiene coincidencia mi frame capturado
    putText(frame ,f'{len(good)} matches  {1000*(t3-t2):.0f} ms {len(good)}%', #Imprimo informacion
              orig=(5,36), color=(200,255,200))
    cv.imshow("SIFT",frame)
    break #Una vez encontrado el objeto ya no es necesario seguir comprobando el conjunto

else:

    cv.imshow('SIFT', frame)
```

Finalmente mostraré algunos ejemplos del uso de este programa, en estos podemos apreciar que funciona (de manera menos efectiva) aunque se modifique la orientación y el ángulo:







RECTIF

a) Rectifica la imagen de un plano para medir distancias (tomando manualmente referencias conocidas). Por ejemplo, mide la distancia entre las monedas en coins.png o la distancia a la que se realiza el disparo en gol-eder.png. Las coordenadas reales de los puntos de referencia y sus posiciones en la imagen deben pasarse como parámetro en un archivo de texto. Aunque puedes mostrar la imagen rectificada para comprobar las operaciones, debes marcar los puntos y mostrar el resultado sobre la imagen original. Verifica los resultados con imágenes originales tomadas por ti.

Para este ejercicio he tomado las siguientes decisiones de diseño: En primer lugar a la hora de seleccionar mi objeto de referencia este debe ser marcado con cuatro puntos en sus esquinas ya que solo funciona con objeto de 4 lados, el programa no realizara la normalización de la imagen hasta que esos cuatro puntos que delimitan el objeto sean seleccionados, una vez seleccionados esos cuatro puntos se mostrara una ventana con la imagen normalizada, por otro lado se podrá realizare la marcación para medir distancias tanto en la imagen normal como en la normalizada y finalmente se puede borrar todos los puntos colocados, tanto los de medición como los de seleccionado de objeto al pulsar la tecla x.

Para comenzar con este problema he creado dos funciones para colocar puntos, la primera solo se usará en la imagen original, esta tiene dos usos en primer lugar marcar los cuatro primeros puntos para obtener la referencia y cuando cumpla esa condición se podrá usar para marcar puntos de medición en la imagen original. La segunda función solo se podrá usar una vez estén colocados los puntos de medición y se usará únicamente en la imagen rectificada para marcar puntos de medición.

```
def funFrame(event, x, y, flags, param):#Funcion para obtener valores al hacer cicl en mi frame
    if event == cv.EVENT_LBUTTONDOWN:
        if len(points) < 4:
            points.append((x, y))
        else:
            medidor.append((x, y))

def funRec(event, x, y, flags, param):#Funcion para obtener valores al hacer cicl en mi imagen rectificada
    if event == cv.EVENT_LBUTTONDOWN:
        if len(points) == 4:
            medidor3.append((x,y))
```

altura de un lado y en otra línea los del otro.

```
def readFile(fichero):#Funcion para leer de un archivo los datos de la medidas totales
    with open(fichero, 'r') as f:
        lineas = f.readlines()
        lista = []
        for l in lineas:
            elementos = map(float, l.strip().split())
            lista.append(tuple(elementos))
    return lista
```

que mantiene las proporciones de los valores obtenidos.

```
medidas=readFile("datos.txt")#Obtengo las medidas

real = np.array([#Posicion con medidas reales escalas para el tamaño de mi frame donde esta la imagen escalada
    [  400.,   (200+(medidas[0])[1]*12)],
    [ (400+(medidas[0])[0]*12),   (200+(medidas[0])[1]*12)],
    [ (400+(medidas[0])[0]*12),  200.],
    [  400.,  200.]])
```

A continuación, en el bucle principal de captura de imagen recorro mis puntos ya seleccionado colocándolos en la imagen hasta tener los cuatro que delimitan mi objeto de referencia, después los transformo a un np.array para trabajar con el y poder calcular mi homografía y poder crear mi imagen rectificada.

```
for p in points:

    cv.circle(frame, p,3,(0,255,0),-1)#Dibujo todos los puntos que seleccionan mi objeto

    if len(points) == 4:#En caso de ser 4 ya puedo tener el objeto medido ya que supongo que todos los objetos serán de 4 lados

        points2=np.array([[(points[0])[0],(points[0])[1]],[(points[1])[0],(points[1])[1]],[(points[2])[0],(points[2])[1]],[(points[3])[0],(points[3])[1]]])#Transformo
        mis puntos seleccionados en un np.array
        H,_ = cv.findHomography(points2, real)
        rec = cv.warpPerspective(frame,H,(600,600))#Creo mi frame rectificado con mi matriz
        rec=cv.rotate(cv.flip(rec, 0), cv.ROTATE_90_CLOCKWISE)#Coloco la orientacion correcta a la referencia D
```

Una vez tengo mis puntos delimitadores obtenidos obtengo la distancia entre ellos y la imprimo en las rectas que formo entre ellos para delimitar el objeto de referencia, como se podrá apreciar en los ejemplo del programa las medidas de pixeles no son las misma de cada lado aun sabiendo en la realidad que los lados son iguales dos a dos, pero en pixeles de la imagen no se refleja por lo que para tener una medición precisa se debe normalizar la imagen para que esos valores sean correctos y realizar la medición sobre esa referencia.

```
    d1 = np.linalg.norm(np.array(points[1])-points[0])#Calculo las medidas de mis puntos seleccionados
    d2= np.linalg.norm(np.array(points[2])-points[1])
    d3= np.linalg.norm(np.array(points[3])-points[2])
    d4= np.linalg.norm(np.array(points[0])-points[3])

    cv.line(frame, points[0],points[1],(0,255,0))#Dibujo para mi figura su medida tanto en pixeles como en cm en su linea
    c = np.mean((points[0],points[1]), axis=0).astype(int)
    putText(frame,f'{d1:.1f} pix//{(medidas[0])[0]}cm',c)
    cv.line(frame, points[1],points[2],(0,255,0))
    c = np.mean((points[1],points[2]), axis=0).astype(int)
    putText(frame,f'{d2:.1f} pix//{(medidas[0])[1]}cm',c)
    cv.line(frame, points[2],points[3],(0,255,0))
    c = np.mean((points[2],points[3]), axis=0).astype(int)
    putText(frame,f'{d3:.1f} pix//{(medidas[1])[0]}cm',c)
    cv.line(frame, points[3],points[0],(0,255,0))
    c = np.mean((points[3],points[0]), axis=0).astype(int)
    putText(frame,f'{d4:.1f} pix//{(medidas[1])[1]}cm',c)
```

En esta parte realizo la transformación de mis puntos a la escala correcta y obtengo los valores de medición con los que hago una media del valor la cual esta si es precisa y podre usar para medir distancia.

```
    pointsr = htrans(H,points)
    d1r = np.linalg.norm(np.array(pointsr[1])-pointsr[0])#Calculo la escala de pix por cm para mi normalizado
    d2r= np.linalg.norm(np.array(pointsr[2])-pointsr[1])
    d3r= np.linalg.norm(np.array(pointsr[3])-pointsr[2])
    d4r= np.linalg.norm(np.array(pointsr[0])-pointsr[3])
    prop1r=d1r/(medidas[0])[0]
    prop2r=d2r/(medidas[0])[1]
    prop3r=d3r/(medidas[1])[0]
    prop4r=d4r/(medidas[1])[1]
    propr=np.mean((prop1r,prop2r,prop3r,prop4r))
```

Esta parte trata la colocación y medición de puntos sobre la imagen normaliza en la que usaré mis puntos para dibujar una línea entre ellos y calcular con la proporción sacada anteriormente y esa distancia de pixeles podre calcular la distancia total.

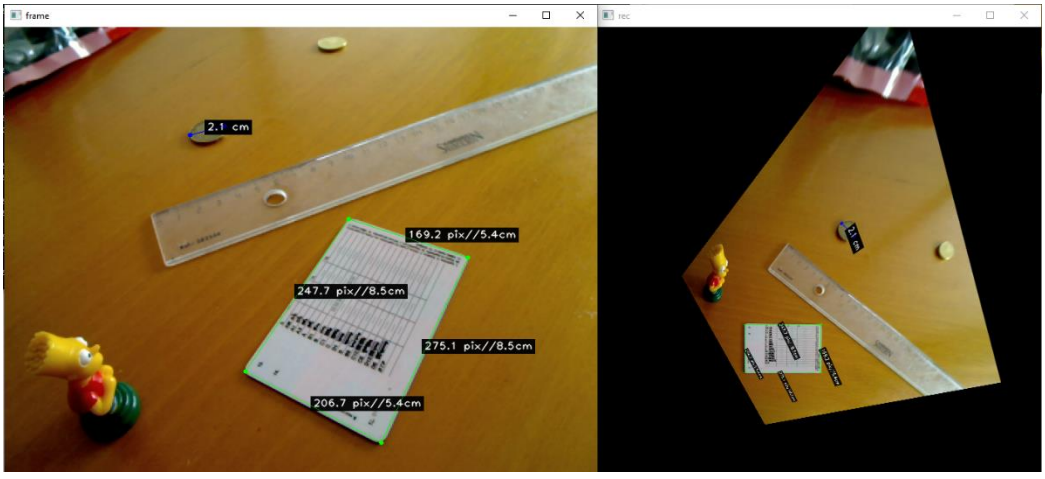
```
for p in medidor3:#Muestro mis puntos de medicion

    cv.circle(rec, p,3,(0,0,255),-1)

    if len(medidor3) == 2:#Cuando son dos dibujo una linea y la informacion de su medida

        cv.line(rec,medidor3[0],medidor3[1],(0,0,255))
        c = np.mean(medidor3, axis=0).astype(int)
        d= np.linalg.norm(np.array(medidor3[0])-medidor3[1])
        putText(rec,f'{d/propr:.1f} cm',c)#Calculo su distancia respecto a la proporcion
```

Esta parte trata la colocación y medición de puntos sobre la imagen original en la que usaré mis puntos para dibujar una línea entre ellos y para calcular con la proporción sacada anteriormente los normalizo para usar la referencia correcta y esa distancia de pixeles podre calcular la distancia total y finalmente mostrare rec siempre.

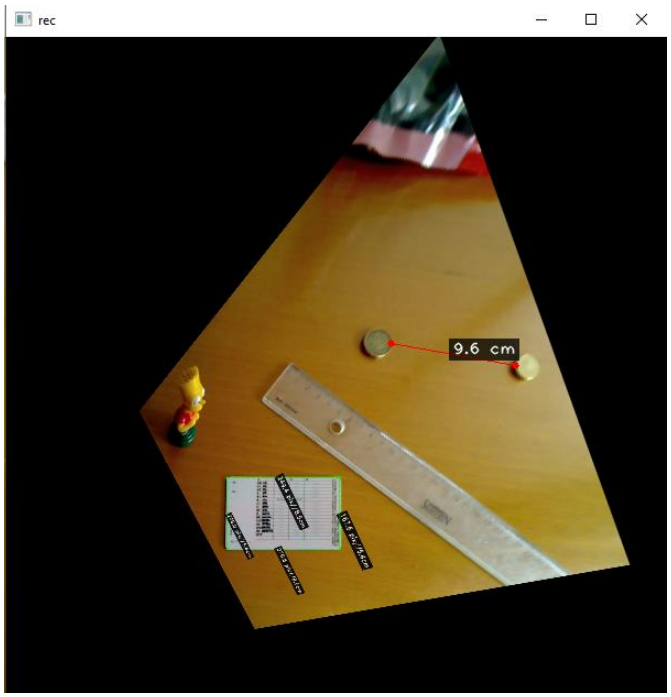
```
for p in medidor:#Muestro mis puntos de medicion

    cv.circle(frame, p,3,(255,0,0),-1)

    if len(medidor) == 2:#Cuando son dos dibujo una linea y la informacion de su medida

        cv.line(frame,medidor[0],medidor[1],(255,0,0))
        c = np.mean(medidor, axis=0).astype(int)
        aux=htrans(H,medidor)#Normalizo mis puntos de medicion
        d= np.linalg.norm(np.array(aux[0])-aux[1])
        putText(frame,f'{d/propr:.1f} cm',c)#Calculo su distancia respecto a la proporcion

cv.imshow('rec',rec)#La muestro de manera que esta bien orientada
```

A continuación, mostraré algunos ejemplos:

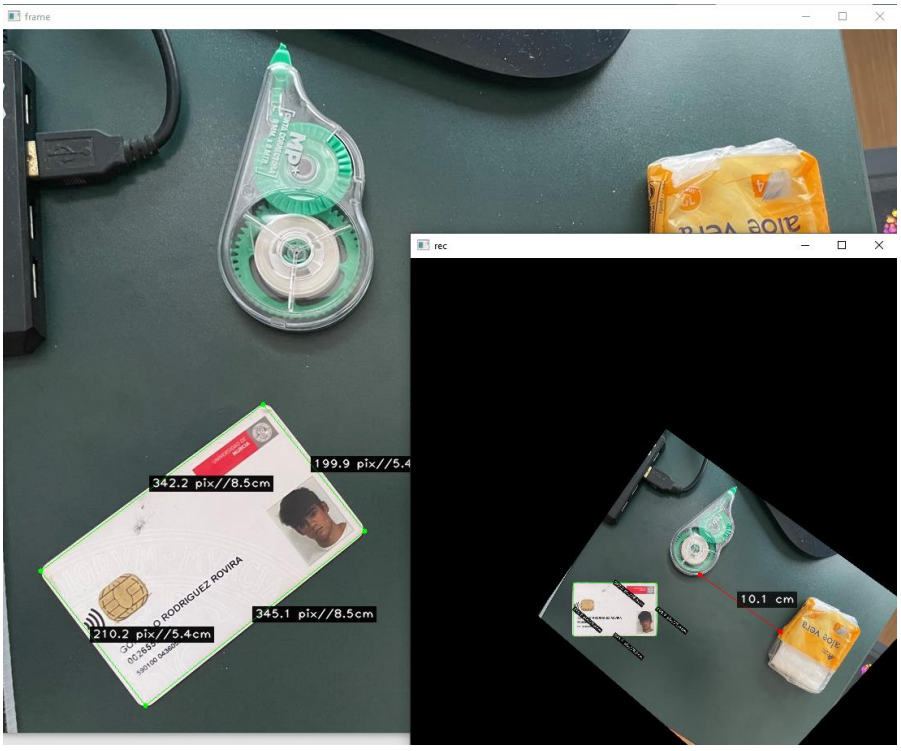
En este ejemplo del archivo images/coins.png podemos ver en primer lugar la medición de la moneda la nos da un total de 2.1cm, sabiendo que la moneda de 20 céntimos (la que parece que es) mide 2.25cm podemos asumir que es esa, esta medición es hecha en la imagen original.



En este otro caso podemos ver la medición de distancia entre las dos monedas realizada en la imagen rectificada.



Finalmente podemos ver un ejemplo tomado de manera manual en el que la diferencia era de 10 cm de distancia.



RA

a) Crea un efecto de realidad aumentada en el que el usuario interactúe con los objetos virtuales. Por ejemplo, haciendo que un objeto se desplace hacia un punto señalado con el ratón.

Este problema lo he resuelto de dos maneras:

Para comenzar a tratar este problema declare dos constantes, una para conocer en todo momento a la posición de mi cubo y otra para conocer el destino de mi cubo, la primera se actualizará de manera continua ya que se obtendrá directamente del cubo cada vez que lo desplace y la otra se obtendrá al marcan con un click en la imagen, ese punto será su destino.

```
destino=()#Variable que contine la posicion a la que debe llegar el cubo
posicion=()#Variable que contine la posicion en la que se situa el cubo
```

A continuación, una vez que obtengo por lo menos una pose buena me quedo con la primera ya que solo realizare el desplazamiento de un cubo ya que por cada pose podré colocar u cubo. Una vez que tengo la pose marco la zona donde voy a situar mi cubo y lo creo y obtengo su posición (de esta manera siempre tendré su posición actualizada en cada ciclo del bucle).

```
if poses:#Para este ejercicio solo necesito una pose de referencia por lo que al tener una ya podré actuar

    M=poses[0]#Me quedo con la primera pose ya que solo necesito actuar sobre una

    x,y = htrans(M, (0.7,0.7,0) ).astype(int)#Capturamos el color de un punto cerca del marcador para borrarlo
    b,g,r = frame[y,x].astype(int)
    cv.drawContours(frame,[htrans(M,marker).astype(int)], -1, (0,0,0) , 3, cv.LINE_AA)#Dibujando un cuadrado encima

    showAxes(frame, M, scale=0.5)# Mostramos el sistema de referencia inducido por el marcador

    cosa = (cube)* (0.5, 0.5, 0.5)#Creo mi cubo

    posicion=[ htrans(M, cosa).astype(int) ][0][0]#Obtengo la posicion de mi cubo
```

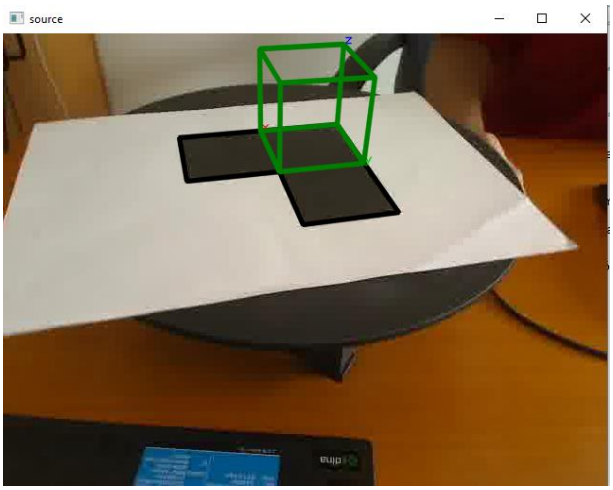
En la primera parte tarto un error en el que muchas veces el cubo no llega a la posición exacta, para solucionar eso hago una aproximación, tanto en el eje x como en el y. Posteriormente, compruebo si debo aumentar mi eje x o disminuirlo en función de donde este situado el destino, después creo un nuevo objeto cubo del cual obtengo la posición del cubo y comprobar que el movimiento se ha realizado en el sentido correcto ya que en ocasiones cuando tratamos con cámaras que rotan y alteran el valor de x produce un fallo por lo que de esta manera se dónde se ha trasladado mi cubo.

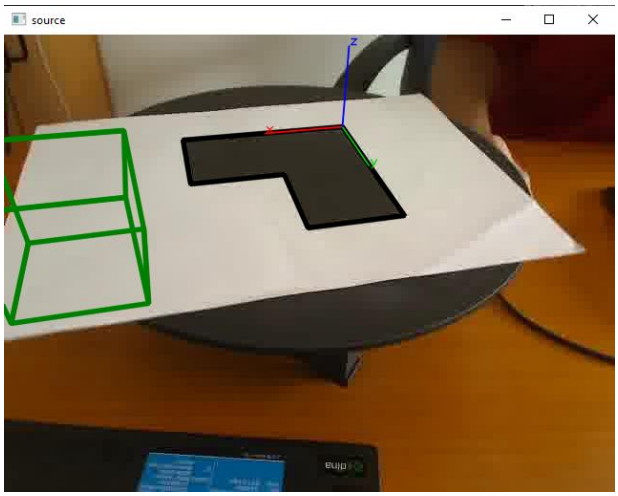
```
        if destino:#Si se ha marcado un punto al que ir me desplazo

            if destino[0]==posicion[0]+2 or destino[0]==posicion[0]-2:#Muchas veces el cubo no llega a la posicion exacta, para solucionar eso hago una aproximacion
                posicion[0]=destino[0]

            if destino[1]==posicion[1]+2 or destino[1]==posicion[1]-2:
                posicion[1]=destino[1]

            if destino[0]!=posicion[0]:#Trato si no se encuentra el cubo en su eje x

                if(posicion[0]<destino[0]):#En caso de ser mi x menor a la del destino la aumento y creo un nuevo objeto con el cubo

                    cube=cube+(0.05,0,0)
                    cosa2 = (cube)* (0.5, 0.5, 0.55 + 0.2 * np.sin(n / 50))
                    posicion2=[ htrans(M, cosa2).astype(int) ][0][0]#Creo esta variable para obtener de nuevo la posicion del cubo y comprobar que el movimiento se ha
realizado en el sentido correcto ya que en ocasiones cuando tratamos con camaras que rotan y alteran el valor de x produce un fallo por lo que de esta manera se donde se
ha trasladado mi cubo

                    if(posicion2[0]<posicion[0]):#Compruebo que mi cubo se ha trasladado en el sentido correcto

                        cube=cube-(0.1,0,0)#Si no es asi lo desplazo el doble en el otro sentido para compensar el desplazamiento erroneo anterior

                elif(posicion[0]>destino[0]):#En caso de ser mi x mayor a la del destino la reduzco y creo un nuevo objeto con el cubo

                    cube=cube-(0.05,0,0)
                    cosa2 = (cube)* (0.5, 0.5, 0.55 + 0.2 * np.sin(n / 50))
                    posicion2=[ htrans(M, cosa2).astype(int) ][0][0]#Creo esta variable para obtener de nuevo la posicion del cubo y comprobar que el movimiento se ha
realizado en el sentido correcto ya que en ocasiones cuando tratamos con camaras que rotan y alteran el valor de x produce un fallo por lo que de esta manera se donde se
ha trasladado mi cubo

                    if(posicion2[0]>posicion[0]):#Compruebo que mi cubo se ha trasladado en el sentido correcto

                        cube=cube+(0.1,0,0)#Si no es asi lo desplazo el doble en el otro sentido para compensar el desplazamiento erroneo anterior
```

He mostrado como se realizaría para el eje x, para el y seria lo mismo, pero usando la coordenada y.

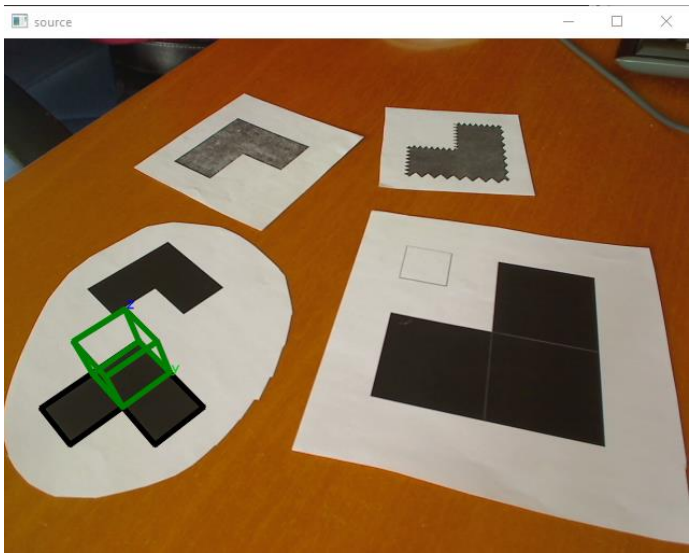
A continuación, mostrare algunas pruebas de la ejecución de mi programa:

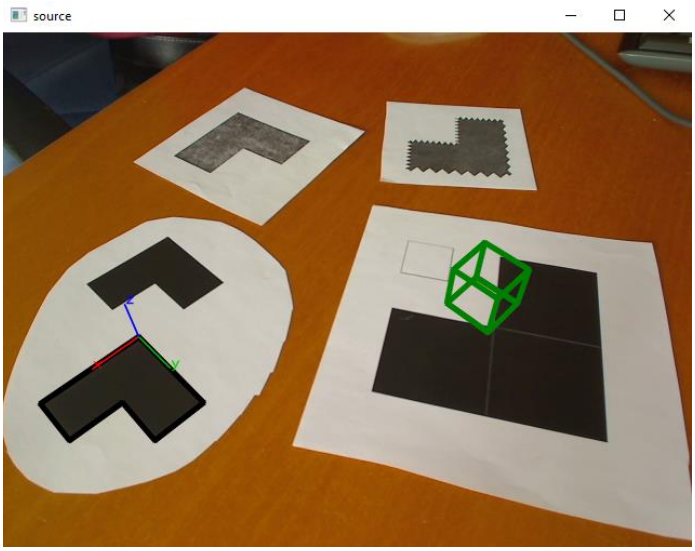
En esta primera prueba se puede ver como en un video el cubo se desplaza de un punto a otro, en la primera imagen se puede ver el cubo nada mas iniciar el programa y antes de marcar el destino, en la segunda se puede ver como el cubo ha llegado a la posición destino.





En este otro ejemplo se puede ver lo mismo pero realizado en una imagen estática.





Otra forma, más correcta:

Para comenzar a tratar este problema declare dos constantes, una para conocer en todo momento a la posición de mi cubo y otra para conocer el destino de mi cubo, la primera se actualizará de manera continua ya que se obtendrá directamente del cubo cada vez que lo desplace y la otra se obtendrá al marcan con un click en la imagen, ese punto será su destino.

```
destino=()#Variable que contine la posicion a la que debe llegar el cubo
posicion=()#Variable que contine la posicion en la que se situa el cubo
```

Posteriormente, he creado una función para calcular el vector de movimiento y el de rotación para saber cómo llegar de un punto a otro, esta función recibirá el destino al cual quiere llegar el cubo el cual ha sido marcado de manera manual mediante un click, por lo tanto debo hacerle htrans con la matriz H (sacada de ok[0]), después ya podré usar su valor junto con el valor de posición del cubo para poder calcular los dos vectores.

```
def cacularVectores(des,posicion):#Funcion para calcular el vector de movimiento y el de rotacion para saber como llegar de un punto a otro
    global vectorM
    global vectorR
    destino=htrans(H,des)
    vectorM=[destino[0]-posicion[0],destino[1]-posicion[1]]
    magnitud = np.linalg.norm(vectorM)
    vectorR = vectorM / magnitud
```

A continuación, una vez que obtengo por lo menos una pose buena me quedo con la primera ya que solo realizare el desplazamiento de un cubo ya que por cada pose podré colocar u cubo. Una vez que tengo la pose marco la zona donde voy a situar mi cubo y lo creo y obtengo su posición (de esta manera siempre tendré su posición actualizada en cada ciclo del bucle).

```
if poses:#Para este ejercicio solo necesito una pose de referencia por lo que al tener una ya podré actuar

    M=poses[0]#Me quedo con la primera pose ya que solo necesito actuar sobre una

    x,y = htrans(M, (0.7,0.7,0) ).astype(int)#Capturamos el color de un punto cerca del marcador para borrarlo
    b,g,r = frame[y,x].astype(int)
    cv.drawContours(frame,[htrans(M,marker2).astype(int)], -1, (0,0,0) , 3, cv.LINE_AA)#Dibujando un cuadrado encima

    showAxes(frame, M, scale=0.5)# Mostramos el sistema de referencia inducido por el marcador

    cosa = (cube)* (0.5, 0.5, 0.5)#Creo mi cubo

    posicion=[cosa[0][0], cosa[0][1]]#Obtengo la posicion de mi cubo

    cv.drawContours(frame, [ htrans(M, cosa).astype(int) ], -1, (0,128,0), 3, cv.LINE_AA)#Dibujo el cubo
```

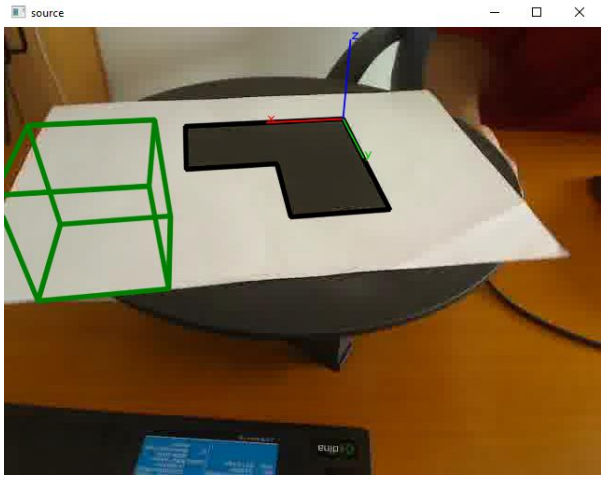
En primer lugar, compruebo que tenga ya un destino seleccionado, posteriormente gracias a mi destino y mi posición puedo generar mediante la función cacularVectores mi vector de rotación y mi vector de movimiento para saber desplazarme en el punto en el que se encuentra mi cubo hasta mi punto destino. A continuación, sé que mi vector de movimiento será 0 si ya he llegado a mi posición, por lo que comprobare tanto para el eje x como para el eje y si estoy en esa posición, en caso de no estarlo usaré mi vector de rotación para saber en qué dirección debo desplazarme y aumentar en consecuencia el valor de mi eje, tanto x como y.

```
if len(destino)>0:#En caso de ya tener destino

    cacularVectores(destino,posicion)#LLamo a mi funcion para calcular el vector de movimiento y le de rotacion

    if vectorM[0]!=0:#Trato si no se encuentra el cubo en su eje x

        if(vectorR[0]>0):#En caso de ser mi x menor a la del destino la aumento y creo un nuevo objeto con el cubo

            cube=cube+(0.05,0,0)

        elif(vectorR[0]<0):#En caso de ser mi x mayor a la del destino la reduzco y creo un nuevo objeto con el cubo

            cube=cube-(0.05,0,0)
```

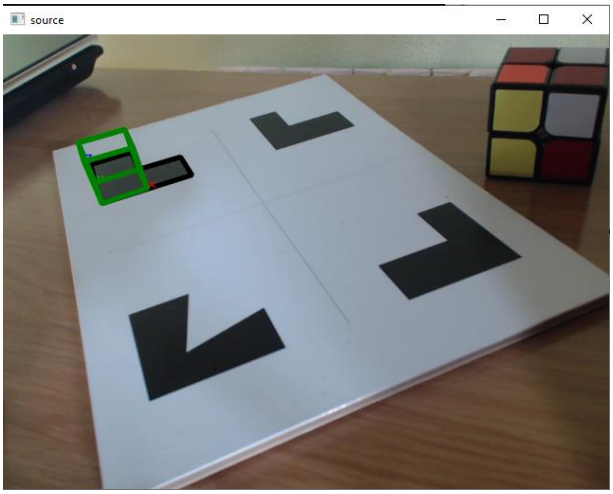
He mostrado como se realizaría para el eje x, para el y seria lo mismo, pero usando la coordenada y.

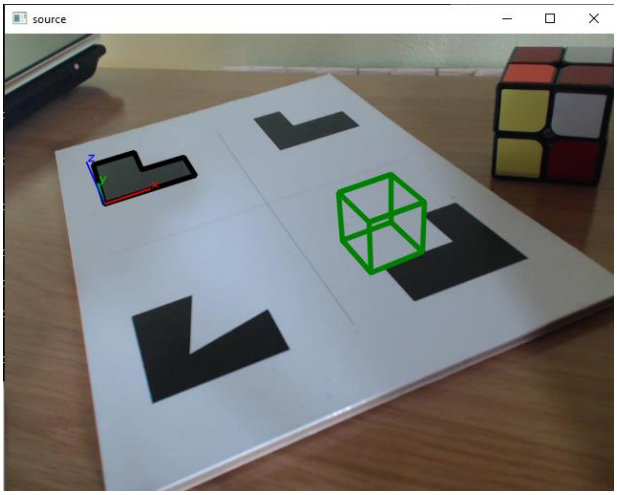
A continuación, mostrare algunas pruebas de la ejecución de mi programa:

En esta primera prueba se puede ver como en un video el cubo se desplaza de un punto a otro, en la primera imagen se puede ver el cubo nada más iniciar el programa y antes de marcar el destino, en la segunda se puede ver como el cubo ha llegado a la posición destino.



En este otro ejemplo se puede ver lo mismo pero realizado en una imagen estática.





VROT

a) Amplía el tracker de Lucas-Kanade (ejemplo LK/lk_track.py) para a) determinar en qué dirección se mueve la cámara (UP,DOWN,LEFT,RIGHT, [FORWARD, BACKWARD]); b) estimar de forma aproximada la velocidad angular de rotación de la cámara (grados/segundo).

Para resolver este problema en primer lugar definí las constantes del FOVv y FOVh obtenidas en el ejercicio de calibración, además definí dos variables para contabilizar el total de rotación tanto horizontal como vertical y también defino un np.array de floats.

```
FOVH = 52.47 #Constantes de FOV del ejercicio1
FOVV = 66.575

tracks = []
track_len = 20
detect_interval = 5
tH=0#Variables para ir actualizando el total de angulo desplazado horizontal y vertical
tV=0
vectorP = np.array([0, 0], dtype=np.float32) #Inicializo un vector bidimensional para cada uno de mis ejes
```

A continuación en caso de que tracks no sea vacío inicializo otro np.array para llevar el total desplazado de mis dos ejes y creo una variable para controlar el total de desplazados.

```
if len(tracks):

    # el criterio para considerar bueno un punto siguiente es que si lo proyectamos
    # hacia el pasado, vuelva muy cerca del punto incial, es decir:
    # "back-tracking for match verification between frames"

    p0 = np.float32([t[-1] for t in tracks])
    p1, _, _ = cv.calcOpticalFlowPyrLK(prevgray, gray, p0, None, **lk_params)
    p0r, _, _ = cv.calcOpticalFlowPyrLK(gray, prevgray, p1, None, **lk_params)
    d = abs(p0 - p0r).reshape(-1, 2).max(axis=1)
    good = d < 1
    new_tracks = []
    totalDesplazado = np.array([0, 0], dtype=np.float32)#Inicializo mi varaible como un vector de dos dimensiones para llevar el total desplazado de mis dos ejes
    numDesplazados = 0 #Inicializo el total de desplazados
```

Posteriormente por cada punto de seguimiento válido actualizo el desplazamiento y incremento mi variable para contabilizar el número de desplazados asi como reinicio mis tarcks.

```
    for t, (x, y), ok in zip(tracks, p1.reshape(-1, 2), good):

        if not ok:

            continue

        t.append([x, y])

        if len(t) > track_len:

            del t[0]
        new_tracks.append(t)
        totalDesplazado += np.array([x, y], dtype=np.float32) - np.array(t[-2], dtype=np.float32)#Por cada punto de seguimiento válido actualizo el desplazamient
sumandolo
        numDesplazados+=1 #Lo aumento en uno por cada punto de seguimiento válido

    tracks = new_tracks
```

En caso de haber algún desplazamiento después de procesar los puntos, calculo el vector medio de desplazamiento.

```
if numDesplazados > 0:#Despues de procesar todos los puntos de seguimiento calculo el vector medio de desplazamiento

    vectorP = totalDesplazado / numDesplazados
```

Reinicio el tracking y me quedo la diferencia de tiempos ya que la necesito para poder calcular la velocidad y los ángulos de la cámara.

```
t1 = time.time()#Reseteo el tracking
dt = t1 - t0#Me quedo la diferencia de tiempos ya que la necesito para poder calcular la velocidad y los angulos de la camara
```

Para dibujar mi flecha en sentido opuesto al movimiento primero calculo el centro de la imagen, amplifico mi vector medio de desplazamiento, calculo la longitud de mi flecha que es hasta donde debe llegar en función del desplazamiento de la imagen, la dibujo y creo una variable que almacena el vector medio de desplazamiento en grados.

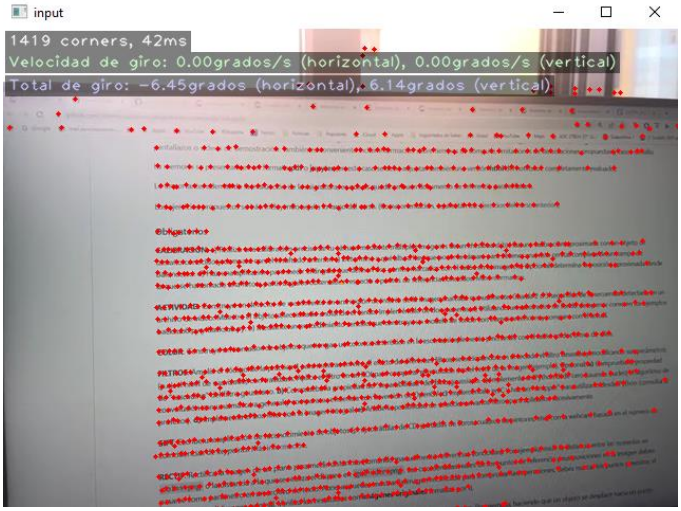
```
putText(frame, f'{len(tracks)} corners, {(t1 - t0) * 1000:.0f}ms')
centro = (frame.shape[1] // 2, frame.shape[0] // 2)#Calculo el centro de mi frame
vectorAumentado = vectorP * 5  # Ampliado y en dirección opuesta
final = tuple(np.int32(np.array(centro) + vectorAumentado))#Calculo hasta donde debe llegar
cv.arrowedLine(frame, centro, final, (0, 165, 255), 2) #Creo mi flecha que ira en sentido contrario al movimiento
width, height = frame.shape[1], frame.shape[0] #Saco el ancho y la altura de mi frame
vectorPDeg = vectorP * np.array([FOVH / width, FOVV / height], dtype=np.float32)
```

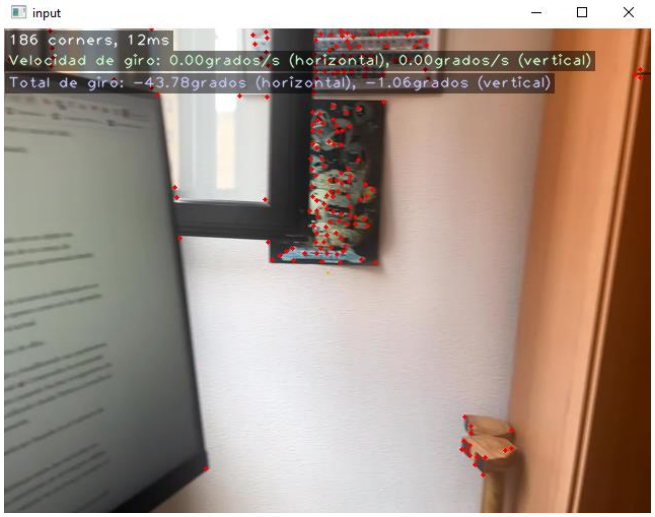
Finalmente compruebo que mis valores no sean nulos, creo una variable que almacena el vector medio de desplazamiento en grados por segundo, saco los valores de cada eje, tarto un error que por algún motivo en algunos giros daba valores NaN que rompe toda la acumulación así que debo tratarlo y acumulo los giros en ambos ejes y lo imprimo

```
if not np.isnan(vectorPDeg).any()and not np.isnan(dt):#Compruebo que mis valores no son nulos

    vectorPDegS = vectorPDeg / dt#Variable que almacena el vector medio de desplazamiento en grados por segundo
    vH=vectorPDegS[0]#Saco la velocidad correspondiente a cada eje
    vV=vectorPDegS[1]
    if not np.isnan(vectorPDegS[0]*dt) and not np.isnan(vectorPDegS[1]*dt): #Por algun motivo en algunos giros daba valores NaN que rompia toda la acumulación asi que debo tratarlo
        tH+=vectorPDegS[0]*dt#Acumulo para cada eje el total desplazado
        tV+=vectorPDegS[1]*dt
        putText(frame,f'Velocidad de giro: {vH:.2f}grados/s (horizontal), {vV:.2f}grados/s (vertical)',orig=(5,36), color=(200,255,200))#Coloco en el frame mi velocidad de giro en ambos ejers
        putText(frame,f'Total de giro: {tH:.2f}grados (horizontal), {tV:.2f}grados (vertical)',orig=(5,58), color=(255,200,200))#Coloco en el frame el total de rotacion acumulado en ambos ejers
```

A continuación, mostrare algunos ejemplos de la ejecución de mi programa:

En la primera imagen podemos ver la imagen inicial sin realizar movimiento y en la segunda una rotación hacia la derecha.





CR

a) Reproduce la demostración del cross-ratio de 4 puntos en una recta del tema de transformaciones del plano. Marca tres puntos con el ratón y añade automáticamente tres más y el punto de fuga, suponiendo que en el mundo real los puntos están alineados y a la misma distancia.

Para resolver este problema en primer lugar declare las siguientes variables siendo la primera para guardar los puntos seleccionados de los postes de manera manual y la segunda para poner la posición de los postes autogenerados (los tres siguientes puntos), por otro lado el primer valor de P es 1/4 ya que para este problema me baso en la formula del cross-ratio para la cual sé que la distancia en la realidad entre los postes es siempre la misma aunque en la imagen se vea cada vez menor, por lo que puedo suponer que P siempre será 1/4 para las distancia a, b y c siendo c el valor a despejar de la formula del cross-ratio P=a/(a+b)*c/(c+b). Por otro lado, el segundo valor de P, P2 es igual a 1/2 ya que este valor de P se usa para obtener el punto de fuga por lo que usando la formula del cross-ratio supongo que a=b=1 y c=infinito por lo que resolviendo la formula como un limine se puede despreciar en la segunda para b y se quedaría tal que P2=a/(a+b)*1, por lo que se estimaría que sea 1/2, finalmente la ultima variable es para contener mi punto de fuga que obtendré más adelante.

```
points = deque(maxlen=3)#Variable que contendra mis puntos colocados de manera manual
p2=deque()#Varaible que contendra mis puntos generados nuevos
P=1/4#Constante que se usara para el cross-ratio ya que si sabemos que la distanci entre a=b=c P sera siempre 1/4
P2=1/2#Constante que se usara para el cross-ratio para calcular el punto de fuga ya que si sabemos a la larga si a=b=1 y c=infinto al despejar la formula como limite dara
1/2
puntoFuga=deque()#Variable que contendrá el punto de fuga
```

Para este problema he creado una función la cual dado un valor numérico y suponiendo que ya hallan sido marcados los tres puntos que deben ser colocados de manera manual (se llama después de esta acción en el bucle principal) en primer lugar comprobará si ya se han añadido los puntos, si no es así los añadirá, en primer lugar, creo una recta que corta por mis puntos 1 y 3 marcados manualmente y la cual usare de manera visual además de obtener de ella los valores de y dado un punto x. Posteriormente trataré la forma de crear los puntos en función de con que otros puntos se generan los valores a y b, y aplicaré esos valores en mi formula cross-ratio transformada para despejar c.

```
def puntos(numero):

    pts=np.array(points)#Para poder saber la altura estimada de mis puntos dado x trazo una recta entre mis puntos 0 y 2
    m = (pts[2][1] - pts[0][1]) / (pts[2][0] -  pts[0][0])
    b = pts[0][1] - m *  pts[0][0]
    height, width, _ = frame.shape
    x1 = 0
    y1 = int(m * x1 + b)
    x2 = width - 1
    y2 = int(m * x2 + b)
    cv.line(frame, (x1, y1), (x2, y2), (128, 128, 128), 1)#Diubujo la recta

    if len(p2)!=numero:#Solo se ejecutará en caso de no tener ya los puntos
```

```
        for i in range(numero):

            if len(p2)==0:
                #Al sabes que p=1/4 y que  p=(a/(a+b))*(c/(c+b)) puedo despejar la formula c = (P * (A+B)/A) * B / (1 - (P * (A+B)/A)) despejando c que seria la distancia
desde mi punto c al punto d
                A=points[1][0]-points[0][0]
                B=points[2][0]-points[1][0]
                c = (P * (A+B)/A) * B / (1 - (P * (A+B)/A))
                x=(math.ceil(c)+points[2][0])#Al sumar a mi ultimo punto c obtengo el nuevo punto d
                y=int(m*x + b)#saco la y dado el valor x de mi recta ya que y = m * x + b
                p2.append((x,y))

            else:
                if len(p2)==1:
                  #Al sabes que p=1/4 y que  p=(a/(a+b))*(c/(c+b)) puedo despejar la formula c = (P * (A+B)/A) * B / (1 - (P * (A+B)/A)) despejando c que seria la
distancia desde mi punto c al punto d
                    A=points[2][0]-points[1][0]
                    B=p2[0][0]-points[2][0]
                    c = (P * (A+B)/A) * B / (1 - (P * (A+B)/A))
                    x=(math.ceil(c)+p2[0][0])#Al sumar a mi ultimo punto c obtengo el nuevo punto d
                    y=int(m * (int(c)+p2[0][0]) + b)#saco la y dado el valor x de mi recta ya que y = m * x + b
                    p2.append((x,y))

                else:
                    if len(p2)==2:
                      #Al sabes que p=1/4 y que  p=(a/(a+b))*(c/(c+b)) puedo despejar la formula c = (P * (A+B)/A) * B / (1 - (P * (A+B)/A)) despejando c que seria la
distancia desde mi punto c al punto d
                        A=p2[i-2][0]-points[2][0]
                        B=p2[i-1][0]-p2[i-2][0]
                        c = (P * (A+B)/A) * B / (1 - (P * (A+B)/A))
                        x=(math.ceil(c)+p2[i-1][0])#Al sumar a mi ultimo punto c obtengo el nuevo punto d
                        y=int(m * (int(c)+p2[i-1][0]) + b)#saco la y dado el valor x de mi recta ya que y = m * x + b
                        p2.append((x,y))

                    else:
                        if len(p2)>2:
                            A=p2[i-2][0]-p2[i-3][0]
                            B=p2[i-1][0]-p2[i-2][0]
                            c = (P * (A+B)/A) * B / (1 - (P * (A+B)/A))
                            x=(math.ceil(c)+p2[i-1][0])#Al sumar a mi ultimo punto c obtengo el nuevo punto d
                            y=int(m * (int(c)+p2[i-1][0]) + b)#saco la y dado el valor x de mi recta ya que y = m * x + b
                            p2.append((x,y))
```

Además, como los puntos se deben de mostrar de manera continua he creado otra función la cual recibe un valor numérico que indica hasta qué punto quiero que se muestre en la imagen.

```
def mostrarPuntos(numero):#Funcion para monstrar mis puntos, debe hacerse de manera constante

    if numero>0:
        for i in range(numero):
            cv.circle(frame, p2[i],3,(0,0,255),-1)
```

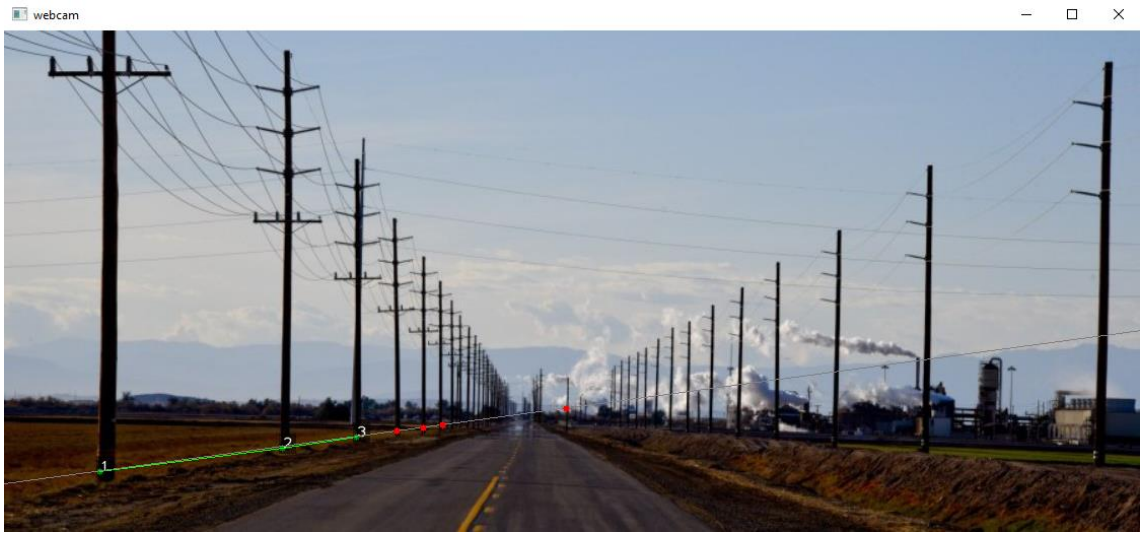
La última función, cree otra función para crear mi punto de fuga y en caso de estar ya creado lo muestra, la forma de crearlo es la misma que en la función de puntos, pero en este caso no recibe parámetros y la se usa P2 en ves de P en la fórmula de cross-ratio.

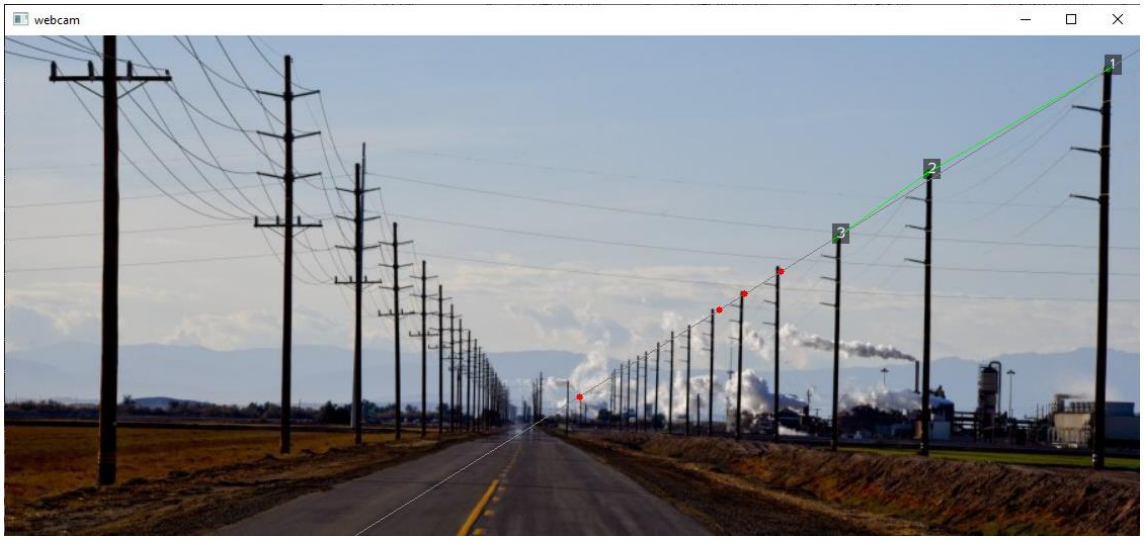
```
def Fuga():

    if len(puntoFuga) ==0:

        pts=np.array(points)#Saco mi m y b de la recta
        m = (pts[2][1] - pts[0][1]) / (pts[2][0] -  pts[0][0])
        b = pts[0][1] - m *  pts[0][0]
        A=p2[1][0]-p2[0][0]#Obtengo mis ultimos puntos(este caso es para solo 3 artificiales)
        B=p2[2][0]-p2[1][0]
        c = (P2 * (A+B)/A) * B / (1 - (P2 * (A+B)/A))
        x=(int(c)+p2[2][0])#Al sumar a mi ultimo punto c obtengo el nuevo punto d
        y=int(m*x + b)#saco la y dado el valor x de mi recta ya que y = m * x + b
        puntoFuga.append((x,y))
        cv.circle(frame, puntoFuga[0],3,(0,0,255),-1)

    else:

        cv.circle(frame, puntoFuga[0],3,(0,0,255),-1)
```

Finalmente, en el bucle principal de programa me encargo de dibujar todos los puntos marcados de manera manual y espero a que tenga mis tres puntos manuales para comenzar a llamar a las funciones ya explicadas.

```
for key, frame in autoStream():

    for p in points:#Dibujo mis puntos manuales

        cv.circle(frame, p,3,(0,255,0),-1)
        putText(frame,f'{points.index(p)+1}',p)

    if len(points) == 3:#En caso de ya tener y tres manuales obtengo y dibujo todo lo demas

        cv.line(frame, points[0],points[1],(0,255,0))
        cv.line(frame, points[1],points[2],(0,255,0))
        puntos(3)
        mostrarPuntos(3)
        Fuga()

    cv.imshow('webcam',frame)
```

A continuación, mostrare algunos ejemplos de la ejecución del programa:





SWAP

a) Intercambia dos cuadriláteros en una escena marcando manualmente los puntos de referencia.

Para resolver este problema he definido en primer lugar una variable de tamaño máximo 8 la cual se ocupará de tener mis puntos, 4 por cada click ya que he decido para que siempre los clicks tengan las mismas medidas al hacer click se obtengas 4 a su alrededor formando un cuadrado, es decir por cada click guardare 4 puntos en torno al punto clicado. También he definido una variable de control ya que he decidido que el intercambio se haga cuando el usuario tome esa decisión al pulsar una tecla y finalmente una constante para desplazar en x y en y los valores de mi punto clicado para obtener los 4 a su alrededor.

```
points2=deque(maxlen=8)#Lista donde almacenare los puntos que delimitan mis dos secciones a intercambiar
activado=False#Variable para controlar si quiere que se esten intercambiando mis dos secciones
TAMAÑO_SECCION=50#Constnte que determina el tamaño de las secciones a intercambiar
```

Posteriormente he definido la función para coger mis puntos alrededor de mi click usando la constante ya definida.

```
def fun(event, x, y, flags, param):#Funcion que al detetar el click añade puntos al rededor de donde se hizo el click para determinar la seccion a intercambiar

    if event == cv.EVENT_LBUTTONDOWN:

        points2.append((x-TAMAÑO_SECCION,y+TAMAÑO_SECCION))
        points2.append((x+TAMAÑO_SECCION,y+TAMAÑO_SECCION))
        points2.append((x+TAMAÑO_SECCION,y-TAMAÑO_SECCION))
        points2.append((x-TAMAÑO_SECCION,y-TAMAÑO_SECCION))
```

A continuación, dibujo todos mis puntos y en caso de haber algún punto marcado, es decir hice un click y habrá mínimo 4 alrededor de mi click, dibujo el cuadrado que forman esos puntos, por otro lado, si tengo ya 8 puntos dibujaré los puntos y las rectas entre ellos que forman mi segundo punto clicado.

```
    for key, frame in autoStream():

        for p in points2:

            cv.circle(frame, p,3,(0,0,255),-1)

        if points2:#Cuando tengo puntos, es decir hice por lo menos un click se dibuja la seccion (puntos y lineas)

            cv.line(frame, points2[0],points2[1],(0,0,255))
            cv.line(frame, points2[1],points2[2],(0,0,255))
            cv.line(frame, points2[2],points2[3],(0,0,255))
            cv.line(frame, points2[3],points2[0],(0,0,255))

        if len(points2) == 8:#En el caso de qwue tenga 8 puntos quiere decir de que tengo mis dos secciones por lo que dibujo la segunda

            cv.line(frame, points2[4],points2[5],(0,0,255))
            cv.line(frame, points2[5],points2[6],(0,0,255))
            cv.line(frame, points2[6],points2[7],(0,0,255))
            cv.line(frame, points2[7],points2[4],(0,0,255))
```

Finalmente, en esta parte se ve que la tecla i es la encargada de activar y desactivar el intercambio, además podemos ver el intercambio que realizo guardando una copia de las

zonas de la imagen para no perderlas al sustituir una por otra y al realizar una copia me aseguro de que no tenga fallos por aliasing.

```
if key == ord('i'):#Si pulso la tecla i quiere decir que quiero intercambiar mis dos secciones, si vuelve a ser pulsada volvera a la imagen original

    activado= not activado

if activado:#Si quiero intercambiar mis secciones

    #Guardo mis regiones para intercambiar y cloco el copy para crea copias independientes de las áreas del frame evitando que se sobrescriban accidentalmente
    part1 = frame[points2[2][1]:points2[0][1], points2[0][0]:points2[2][0]].copy()
    part2 = frame[points2[6][1]:points2[4][1], points2[4][0]:points2[6][0]].copy()
    #Realizo el intercambio de secciones con las variables donde tenia guardada la otra seccion
    frame[points2[6][1]:points2[4][1], points2[4][0]:points2[6][0]] = part1
    frame[points2[2][1]:points2[0][1], points2[0][0]:points2[2][0]] = part2
```

A continuación, mostrare algunos ejemplos de la ejecución del programa:

En la primera imagen podemos ver que solo se han seleccionado las zonas y no se ha realizado el intercambio, mientras que en la segunda podemos ver como ya si ha sido realizado el intercambio