

Índice

1. CALIBRACIÓN	2
1.1. Apartado A	2
1.2. Apartado B	3
1.3. Apartado C	3
1.4. Apartado D	4
1.5. Apartado E	5
2. ACTIVIDAD	6
2.1. Detector de Movimiento	6
2.2. Grabación de Video	7
2.3. Anulación de Fondo	7
3. COLOR	9
3.1. Modelo de Histogramas	9
3.2. Comparación de Histogramas	10
3.3. Segmentación por reprojeción	11
4. FILTROS	13
5. SIFT	15
6. RECTIF	17
7. PANÓ	18
8. RA	19
9. DLIB	20
10. VROT	21
11. DELOGO	22
12. SILU	23
13. ANON	24
14. CR	25
15. SWAP	26

16.NFACE	27
17.DNI	28
18.SUDO	29
19.STC	30
20.PROBLEMA ORIGINAL	31

1. CALIBRACIÓN

- a) Realiza una calibración precisa de tu cámara mediante múltiples imágenes de un chessboard.
- b) Haz una calibración aproximada con un objeto de tamaño conocido y compara con el resultado anterior.
- c) Determina a qué altura hay que poner la cámara para obtener una vista cenital completa de un campo de baloncesto.
- d) Haz una aplicación para medir el ángulo que definen dos puntos marcados con el ratón en el imagen.
- e) Opcional: determina la posición aproximada desde la que se ha tomado una foto a partir ángulos observados respecto a puntos de referencia conocidos.

Para la realización de este ejercicio se ha hecho uso de los siguientes códigos ejemplo proporcionados por el profesor:

- `calibrate.py`
- `undist.py`
- `medidor.py`

1.1. Apartado A

La calibración precisa de la cámara se realizará mediante el uso de un patron con forma de tablero de ajedrez. Con el objetivo de obtener el valor FOV de la imagen tomada. Todo el proceso se realiza bajo las instrucciones marcadas en la carpeta donde se localiza `calibrate.py`.

El procedimiento es tomar una serie de fotos en las que aparezca el patrón de ajedrez, y pasarlas como entrada al código de `calibrate.py`. Este devuelve tres parámetros:

- Error de ajuste. (RMS)
- La calibración de cámara. (K)
- Coeficientes de distorsión radial.

Dentro de la diagonal de la matriz K se encuentra el valor f . Con todos estos valores se puede obtener la imagen transformada sin distorsión, mediante el uso de `undist.py` pero modificando los valores de K y D .

Tras la ejecucion de los programas se obtiene un valor f de $3,06144885e + 03$ y se puede calcular el valor FOV mediante la siguiente fórmula:

$$\tan\left(\frac{FOV}{2}\right) = \frac{\frac{W}{2}}{f} \rightarrow \frac{FOV}{2} = \arctan(0,4899) \rightarrow FOV = 52,2$$

Asimismo, la información de la imagen se puede obtener a partir de los metadatos de la misma. Obteniendo que la distancia focal de la cámara es de 26 (35 mm), y mediante una tabla de referencia [1] se puede saber que el **FOV es de 54.4**.

1.2. Apartado B

Para la realización de este ejercicio se ha seleccionado como objeto una libreta (*Figura 2*) con un tamaño x de 21 cm y una distancia entre la cámara y el objeto de $z = 35$ cm. Mediante el programa *medidor.py* se obtiene que la medida del objeto en pixeles es de $u = 1904$ pixeles.



Figura 1: Imagen Apartado B

Una vez obtenidos todas las medidas se aplica la siguiente fórmula, para obtener el valor de f :

$$u = f \frac{X}{Z} \rightarrow 1904 = f \frac{21}{35} \rightarrow f = 3173'3 \text{ pixels}$$

Del mismo modo, se calcula el valor de FOV (teniendo en cuenta que las dimensiones de la imagen tomada son: $w = 3000$ y $h = 4000$):

$$\tan\left(\frac{\text{FOV}}{2}\right) = \frac{\frac{w}{2}}{f} \rightarrow \tan\left(\frac{\text{FOV}}{2}\right) = 0,473 \rightarrow \frac{\text{FOV}}{2} = \arctan(0,473) \rightarrow \text{FOV} = 50,63$$

De manera que, el valor FOV (horizontal) obtenido a partir de un objeto con tamaño conocido es de **50.63**, que difiere menos de un 30 % con el valor obtenido en la calibración precisa.

1.3. Apartado C

Este apartado es parecido al anterior, pero se busca una distancia entre la cámara y el campo (z) para que este se vea en su totalidad.

Para esto se nombran los siguientes datos conocidos:

- Focal Length = 26 (35mm)
- FOV = 52.2
- $h = 15000$ cm (dim. campo basket)
- $w = 28000$ cm (dim. campo basket)
- $h = 4000$ pixels (dim. imagen cámara)
- $h = 3000$ pixels (dim. imagen cámara)

Por último, el cálculo sería el siguiente:

$$\tan\left(\frac{FOV}{2}\right) = \frac{\frac{W}{2}}{f} \rightarrow \tan\left(\frac{52,2}{2}\right) = \frac{\frac{3000}{2}}{f} \rightarrow f = 3061,88 \text{ pixels}$$
$$u = f \frac{X}{Z} \rightarrow 4000 = 3061,88 * \frac{15000}{z} \rightarrow z = 11,48 \text{ m}$$

El resultado obtenido es que la distancia a la que se debe colocar la cámara para captar la totalidad del campo es de **11,48 m** de altura.

1.4. Apartado D

Para lograr medir el ángulo entre dos puntos, tomo como esqueleto el código de *medidor.py* y un artículo web [5] donde se explica este concepto.

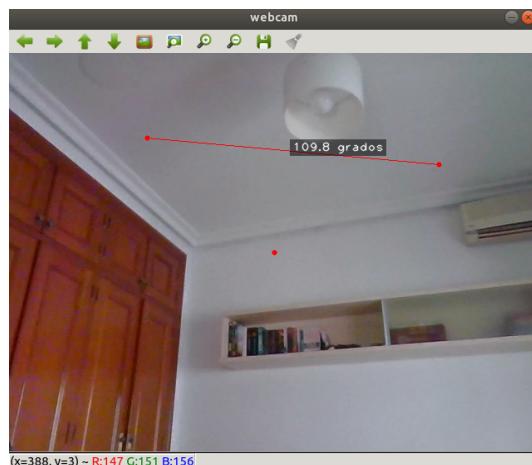


Figura 2: Programa para medir ángulos

La idea es que mediante el punto central de la imagen, obtener dos vectores desde esta a los otros dos puntos y calcular el ángulo entre dichos vectores.

```
def getAngle(a, b, c):
    a = np.array(a)
    b = np.array(b)
    c = np.array(c)

    ba = a - b
    bc = c - b

    cosine_angle = np.dot(ba, bc) / (np.linalg.norm(ba)
                                      * np.linalg.norm(bc))
    angle = np.arccos(cosine_angle))
    return np.degrees(angle)
```

Manual de Usuario

El archivo que contiene el programa de *Medidor de Ángulos* tiene el nombre de *medidor.py*. Y para ejecutarlo, será necesario introducir el siguiente comando en el directorio donde se encuentre dicho archivo:

```
python medidor.py
```

- -dev : Seleccionar dispositivo de entrada.

1.5. Apartado E

...

2. ACTIVIDAD

Construye un detector de movimiento en una región de interés de la imagen marcada manualmente. Guarda 2 ó 3 segundos de la secuencia detectada en un archivo de vídeo. Opcional: muestra el objeto seleccionado anulando el fondo.

Para la realización de este ejercicio se ha hecho uso de los siguientes códigos ejemplo proporcionados por el profesor:

- roi.py
- video_save.py

2.1. Detector de Movimiento

El detector de movimiento se basa en la idea de tomar una instantánea en el momento que se selecciona una región y utilizarla como imagen *background*, donde si se produce algún cambio sobre ella se detectaría una actividad no esperada.

```
frameROI = frame[y1:y2+1, x1:x2+1]
gray = cv.cvtColor(frameROI, cv.COLOR_BGR2GRAY)
gray = cv.GaussianBlur(gray, (21,21), 0)

if background is None:
    background = gray

subtraction = cv.absdiff(background, gray)
threshold = cv.threshold(subtraction, 25, 255, cv.THRESH_BINARY) [1]
threshold = cv.dilate(threshold, None, iterations=2)
```

Cuando se inicia la detección de actividad, si no lo hay, se asigna un nuevo *background* que estará transformado al espacio de color gris. Esto se hace para simplificar la ejecución del programa sin perder demasiada información.

Posteriormente, se realiza una resta, con la operación *absdiff* de *OpenCV*, entre la imagen de fondo y la imagen actual en busca de la existencia de alguna diferencia entre ambas. Si esta diferencia existe, significa que ha habido un cambio en la imagen, y por tanto una actividad inesperada. Asimismo, este cambio podría no ser tan significativo y solamente se trate de una diferencia mínima entre el color de la imagen, por cambios de luz o similar.

Para solucionar este problema se hace uso de la operación *threshold* que aporta *OpenCV*. Creando un filtro para ese tipo de condiciones, específicamente se indica un threshold mínimo de 25. [4] Finalmente, se realiza la

operación **dilate** para podemos detectar de manera más facil los objetos/personas que han activado la actividad.

```
im, outlines, hierarchy = cv.findContours(countorimg, cv.RETR_TREE,  
                                         cv.CHAIN_APPROX_SIMPLE)  
  
for c in outlines:  
    if cv.contourArea(c) < 500:  
        continue  
  
    (x,y,w,h) = cv.boundingRect(c)  
    (x,y,w,h) = (x+x1,y+y1,w+x2,h+y2)  
    cv.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 2)
```

Con esta porción de código se detectan los contornos que componen el ROI de la actividad detectada y se muestra por pantalla dibujando un cuadrado alrededor del objeto detectado. A continuación, se muestra una imagen con la demostracion del programa: mostrando el estado antes de la actividad y cuando se produce alguna actividad.

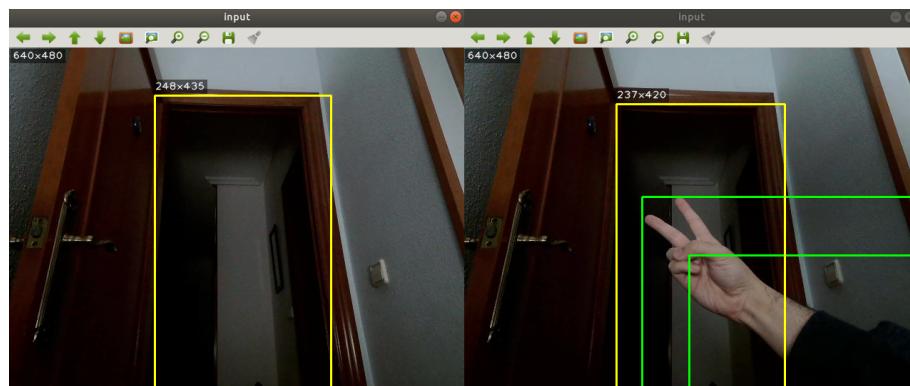


Figura 3: Detección de Actividad

2.2. Grabación de Video

Para la realización de este apartado se hace una implementación del código proporcionado por el profesor, *video_save.py*. El video se va a almacenar en un archivo de salida llamado **output.mp4** que contendrá, concatenadas, todas las actividades que se hayan reconocido durante el tiempo de ejecución del programa. En cuanto una actividad finaliza, se graban un tres segundos más.

2.3. Anulación de Fondo

Mediante el uso de la variable **background** que se define en el apartado 2.1 de este ejercicio se puede realizar una anulación del fondo. Y para esto, solo es necesario realizar una resta entre este background y la imagen actual, mostrando la diferencia entre ambas. El resultado es el siguiente:

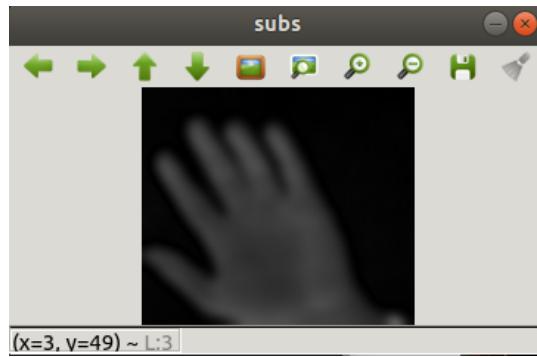


Figura 4: Anulación de Fondo

Manual de Usuario

El archivo que contiene el programa de *Actividad* tiene el nombre de *act.py*. Y para ejecutarlo, será necesario introducir el siguiente comando en el directorio donde se encuentre dicho archivo:

```
python act.py
```

- -dev : Seleccionar dispositivo de entrada.

Operaciones durante la ejecución:

Tecla	Acción
c	Guardar zona ROI
x	Eliminar zona ROI
d	Comenzar detección de actividad

3. COLOR

Construye un clasificador de objetos en base a la similitud de los histogramas de color del ROI (de los 3 canales por separado). Opcional: Segmentación por reprojeción.

Para la realización de este ejercicio se ha hecho uso de los siguientes códigos ejemplo proporcionados por el profesor:

- roi.py
- histogram.py
- histogram2.py

3.1. Modelo de Histogramas

Para la obtención de los histogramas que representan la imagen incluida en la región ROI se ha creado una función, llamada *calcBRGHist*, que devuelve tres histogramas (uno por cada canal de color RGB). Para ello se sigue el siguiente procedimiento:

```
def calcBRGHist(bgr_hist , hist_h):  
    histSize = 256  
    histRange = (0,256)  
    accumulate = False  
    hist_h = y2 - y1  
  
    b_hist = cv.calcHist(bgr_hist , [0] , None , [histSize] ,  
                         histRange , accumulate=accumulate)  
    g_hist = cv.calcHist(bgr_hist , [1] , None , [histSize] ,  
                         histRange , accumulate=accumulate)  
    r_hist = cv.calcHist(bgr_hist , [2] , None , [histSize] ,  
                         histRange , accumulate=accumulate)  
  
    cv.normalize(b_hist , b_hist , alpha=0, beta=hist_h ,  
                norm_type=cv.NORM_MINMAX)  
    cv.normalize(g_hist , g_hist , alpha=0, beta=hist_h ,  
                norm_type=cv.NORM_MINMAX)  
    cv.normalize(r_hist , r_hist , alpha=0, beta=hist_h ,  
                norm_type=cv.NORM_MINMAX)  
  
    return b_hist , g_hist , r_hist
```

1. Inicialización de los parámetros necesarios para que solamente se hagan los histogramas de la región ROI.
2. Función ***calcHist***: Calcula los histogramas para cada uno de los canales.
3. Función ***normalize***: Se utiliza para poder comparar los histogramas de una manera más realista, ya que cada uno de los ROIs que se seleccionen no van a tener las mismas dimensiones.

Toda esta función está basada en el tutorial de la documentación oficial de OpenCV. [2]

A continuación, se almacenan los tres histogramas en un array llamado ***modeloHist***, que contendrá los histogramas de las tres regiones ROI que componen el modelo. Una vez almacenados, se puede proceder a la comparación con ROIs futuros de la imagen.

3.2. Comparación de Histogramas

Para la comparación entre histogramas, se hará uso de la documentación oficial de OpenCV sobre dicho tema. [3] Y se crea una función llamada ***compararHistogramas***, que realizará todo el proceso de cálculo de los valores de relación entre el histograma del ROI actual con los del modelo.

```
def compararHistogramas(modelo , trozo , hist_h):
    # Histograma del trozo a comparar
    bgr_planes = cv.split(trozo)
    histSize = 256
    b_hist , g_hist , r_hist = calcBRGHist(bgr_planes , hist_h)

    valores = []
    # Comparar con los histogramas del modelo
    for [b_hist2 , g_hist2 , r_hist2] in modeloHist:
        b_comp = cv.compareHist(b_hist , b_hist2 , cv.HISTCMP_CORREL)
        g_comp = cv.compareHist(g_hist , g_hist2 , cv.HISTCMP_CORREL)
        r_comp = cv.compareHist(r_hist , r_hist2 , cv.HISTCMP_CORREL)

        valores.append(b_comp + g_comp + r_comp)
    return valores
```

El procedimiento que se sigue para realizar esta función es el siguiente:

1. Se obtienen los histogramas de la región ROI actual.
2. Se crea un array que contendrá los valores de comparación de cada uno de los histogramas de las imágenes del modelo.
3. Función ***compareHist***: Se utiliza para comparar dos histogramas mediante una medida. En este caso, se utiliza la medida ***HISTCMP_CORREL***, que es la correlación. Cuanto mayor sea su valor mas relación habrá entre los histogramas de ambas imágenes.

Por último, se selecciona la imagen del modelo más semejante a la región ROI actual recorriendo la salida de la función anterior y obteniendo el valor más alto.

3.3. Segmentación por reprojeción

Para el desarrollo de esta parte se crea un nuevo programa python (*segcolor.py*) semejante al programa de color pero solamente centrada en la explicación que se realiza en el notebook *colorseg*.

La implementación realizada se basa en la clasificación probabilística, donde se intenta clasificar los pixeles de la imagen dependiendo de que a que clase pertenece, dentro de los colores almacenados en el modelo.

Para obtener estos colores del modelo, se sigue el siguiente procedimiento:

- Se calcula el histograma de la region ROI seleccionada. (Dentro de los canales UY)
- Obtener la verosimilitud de cada pixel y se le aplica un filtro *Gaussian* para que los pixeles vecinos influyan en casos donde la verosimilitud no es tan clara.
- Se normaliza las verosimilitudes y se obtiene la probabilidad.
- Se imprime el resultado en una imagen donde se presenta el color ganador en cada uno de los pixeles de la imagen.

El resultado obtenido es una segmentación de la imagen por color, mediante el uso de los histogramas.

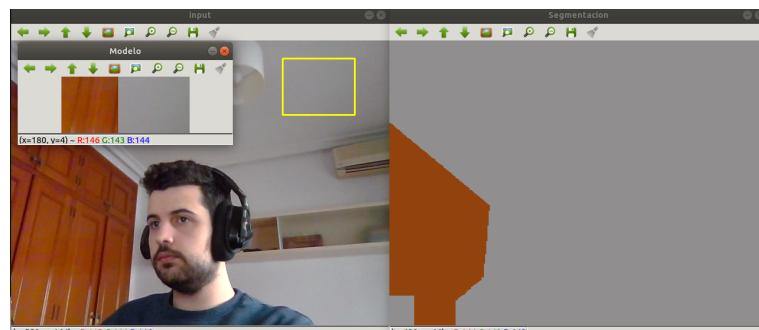


Figura 5: Ejecución de la segmentación de color

Manual de Usuario

El archivo que contiene el programa de Color tiene el nombre de *color.py*. Y para ejecutarlo, será necesario introducir el siguiente comando en el directorio donde se encuentre dicho archivo:

```
python color.py
```

- -dev : Seleccionar dispositivo de entrada.

Operaciones durante la ejecución:

Tecla	Acción
c	Guardar zona ROI
x	Eliminar zona ROI
m	Almacenar histogramas del ROI en el modelo

Ejemplo de ejecución del programa:

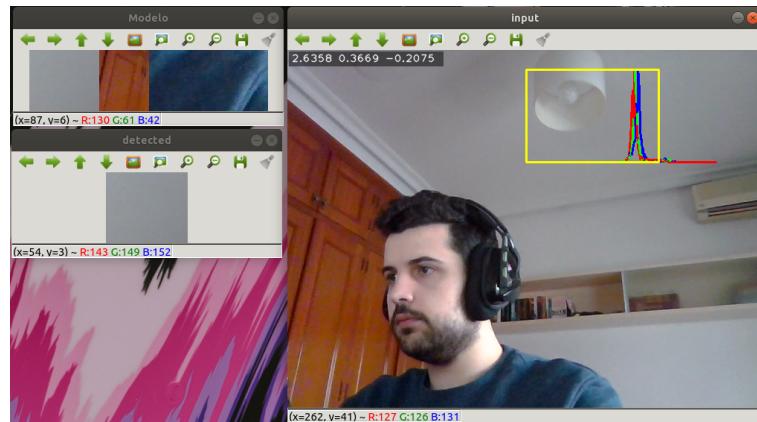


Figura 6: Comparación de histogramas

El archivo que contiene el programa de *Segmentación de color* tiene el nombre de *segcolor.py*.

```
python segcolor.py  
- -dev : Seleccionar dispositivo de entrada.
```

Operaciones durante la ejecución:

Tecla	Acción
c	Guardar zona ROI
x	Eliminar zona ROI
m	Almacenar histogramas del ROI en el modelo

4. FILTROS

Muestra el efecto de diferentes filtros sobre la imagen en vivo de la webcam. Selecciona con el teclado el filtro deseado y modifica sus posibles parámetros (p.ej. el nivel de suavizado) con las teclas o con trackbars. Aplica el filtro en un ROI para comparar el resultado con el resto de la imagen. Opcional: implementa en Python o C "desde cero".

Para la realización de este ejercicio se ha hecho uso de los siguientes códigos ejemplo proporcionados por el profesor:

- roi.py
- trackbar.py

OpenCV dispone de una serie de filtros por defecto, entre estos se implementan *boxFilter* y *GaussianBlur*. El primero se trata de un filtro de suavizado, mientras que el segundo aplica un filtro gausiano. Asimismo, se implementa un filtro desde cero mediante el uso de la función *filter2D* de openCV y la creación de kernel, que ira recorriendo la imagen aplicando dicho filtro.

```
size = cv.getTrackbarPos("Size", "input")
if size == 0:
    size = 1

kernel = np.array([[-1.0, -1.0],
                  [2.0, 2.0],
                  [-1.0, -1.0]])

kernel = kernel/(np.sum(kernel) if np.sum(kernel)!=0 else 1)
kernel = kernel * size

frame[y1:y2+1, x1:x2+1] = cv.filter2D(frame[y1:y2+1, x1:x2+1], -1, kernel)
```

Ademas, el programa dispone de un trackbar para poder modificar los tres filtros que estan implementados.

Manual de Usuario

El archivo que contiene el programa de *Filtros* tiene el nombre de *filtros.py*. Y para ejecutarlo, será necesario introducir el siguiente comando en el directorio donde se encuentre dicho archivo:

```
python filtros.py
```

- dev : Seleccionar dispositivo de entrada.

Operaciones durante la ejecución:

Tecla	Acción
c	Guardar zona ROI
x	Eliminar zona ROI
b	Mantener para aplicar filtro <i>boxFilter</i>
g	Mantener para aplicar filtro <i>GaussianBlur</i>
v	Mantener para aplicar filtro <i>custom</i>

Ejemplo de ejecución del programa:



Figura 7: Funcionamiento programa *filtros.py*

5. SIFT

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.

Para la realización de este ejercicio se ha hecho uso de los siguientes códigos ejemplo proporcionados por el profesor:

- sift-0.py
- sift-1.py
- sift.py

Este ejercicio trata de encontrar los puntos de interés más importantes de un objeto y crear un modelo con ellos para poder detectarlos en cualquier imagen futura. Para esto, primero habrá que preparar una carpeta con imágenes de los objetos con los que se quiere formar el modelo. En este caso, se creará con imágenes de caratulas de películas, y son las siguientes:



Figura 8: Imágenes del modelo

La obtención de los puntos de interés de las imágenes del modelo se realiza la función `getKeypoints`. Este realiza uso de una característica de *OpenCV*, llamada SIFT, y se crea con un número de *features* de 500. Estas serán los puntos de interés totales que se tomarán de una imagen. Asimismo, el programa permite la inclusión de nuevas imágenes en el modelo durante la ejecución de este. El lado malo, es que las imágenes deben estar tomadas en condiciones donde solamente se vea la carátula, sino se crearán puntos que se pueden considerar como *ruido*.

```
def getKeypoints(frame):  
    keypoints , descriptors = sift.detectAndCompute(frame , mask=None)  
    return keypoints , descriptors
```

Una vez este cargado el modelo, se podrá hacer la detección de la imagen actual mediante la función `bestMatch` que obtiene el número de puntos de interés que tiene en común con cada una de las imágenes del modelo y devuelve la imagen de este que más coincidencias tiene con la actual.

```

def bestMatch(descriptores , modelo):
    mayorM = 0
    a = 0
    index = 0
    for [d, img] in modelo:
        nMatch = getMatches(descriptores , d)
        #print("Index: ", index, " , Matches: ", nMatch)
        if nMatch > mayorM:
            mayorM = nMatch
            a = index
        index = index + 1
    return a

```

Manual de Usuario

El archivo que contiene el programa de *SIFT* tiene el nombre de *sift.py*. Y para ejecutarlo, será necesario introducir el siguiente comando en el directorio donde se encuentre dicho archivo:

python sift.py

- -dev : Seleccionar dispositivo de entrada.

Operaciones durante la ejecución:

Tecla	Acción
c	Añadir imagen al modelo
m	Cargar modelo, con las imágenes de la carpeta <i>pelis</i>
d	Mantener la tecla para realizar la detección

Ejemplo de ejecución del programa:



Figura 9: Ejecución de *sift.py*

6. RECTIF

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. Verifica los resultados con imágenes originales tomadas por ti.

7. PANO

Crea automáticamente un mosaico a partir de las imágenes en una carpeta. Las imágenes no tienen por qué estar ordenadas ni formar una cadena lineal y no sabemos el espacio que ocupa el resultado. El usuario debe intervenir lo menos posible. Recuerda que debe tratarse de una escena plana o de una escena cualquiera vista desde el mismo centro de proyección. Debes usar homografías. Compara el resultado con el que obtiene la utilidad de stitching de OpenCV.

8. RA

Crea un efecto de realidad aumentada interactivo: esto significa que a) los objetos virtuales deben cambiar de forma, posición o tamaño siguiendo alguna lógica; b) el usuario puede observar la escena cambiante desde cualquier punto de vista moviendo la cámara alrededor del marcador; y c) el usuario puede marcar con el ratón en la imagen puntos del plano de la escena para interactuar con los objetos virtuales.

9. DLIB

Escribe una aplicación de tu invención que utilice los marcadores de cara obtenidos por el shape detector disponible en dlib (ejemplo /hog/facelandermarks.py).

10. VROT

Estima de forma aproximada la velocidad angular de rotación de la cámara (grados/segundo) analizando las trayectorias obtenidas por el tracker de Lucas-Kanade (ejemplo LK/lk-track.py).

11. DELOGO

Detecta el logotipo de una cadena de TV en una emisión por streaming y suprime los cortes publicitarios.

12. SILU

Escribe una aplicación de reconocimiento de siluetas con la webcam basado en descriptores frecuenciales.

13. ANON

Modifica el ejemplo de reconocimiento de caras DL/facerec para seleccionar caras pinchando con el ratón (o tomándolas de un directorio) para que cuando se reconozcan en las imágenes se oculten (emborronándolas o pixelizándolas).

14. CR

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.

15. SWAP

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

16. NFACE

Implementa la normalización de caras explicada en la sección de transformaciones afines.

17. DNI

Sustituye la foto de un carnet que se observa en una imagen en vivo.

18. SUDO

Haz un programa capaz de llenar un sudoku que se observa en una imagen en vivo.

19. STC

Resuelve los problemas planteados en el notebook stereo-challenge.

20. PROBLEMA ORIGINAL

Problema Original.

Referencias

- [1] Bill Claff. Field-of-view of lenses by focal length. <https://www.nikonians.org/reviews/fov-tables>, June 2013.
- [2] Open CV. Histogram calculation. *OpenCV*, 2020. https://docs.opencv.org/master/d8/dbc/tutorial_histogram_calculation.html.
- [3] Open CV. Histogram comparison. *OpenCV*, 2020. https://docs.opencv.org/3.4/d8/dc8/tutorial_histogram_comparison.html.
- [4] Open CV. Image thresholding. *OpenCV*, 2020. https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html.
- [5] Manivannan Murugavel. Find the angle between three points from 2d using python. <https://manivannan-ai.medium.com/find-the-angle-between-three-points-from-2d-using-python-348c513e2cd>, March 2019.