

Memoria Entrega Final

VIA

Alejandro Ramón López Rodríguez
Subgrupo 1.2
Curso 2019/20

Índice

1. Introducción	3
2. FOV	3
2.1 Código	3
2.2 Ejemplos	6
2.3 Calibración	10
3. CHRO.....	10
3.1 Código	10
3.2 Ejemplos	12
4. HISTCOL.....	14
4.1 Código	14
4.2 Ejemplos	18
5. DLIB.....	21
5.1 Código	21
5.2 Ejemplos	24
6. SIFT	27
6.1 Código	27
6.2 Ejemplos	28
7. RECTIF	32
7.1 Código	32
7.2 Ejemplos	34
8. RA	38
8.1 Código	39
8.2 Ejemplos	43
9. SROI	46
9.1 Código	46
9.2 Ejemplos	47
10. MSAT.....	49
10.1 Código.....	49
10.2 Ejemplos	50
11. FIL	53
11.1 Código.....	53
11.2 Ejemplos	55
12. ANON	63
12.1 Código.....	64
12.2 Ejemplos	66
13. Conclusión	71

1. Introducción

En esta memoria se recoge el funcionamiento de los ejercicios implementados para la práctica de la asignatura, tanto los siete obligatorios (FOV, CHRO, HISTCOL, DLIB, SIFT, RECTIF y RA) como otros cuatro opcionales (SROI, MSAT, FIL y ANON). Cada uno tendrá su propio apartado con una explicación general de código y ejemplos de su funcionamiento.

2. FOV

En este ejercicio obtendremos el campo de visión de nuestra Webcam y además calcularemos otros parámetros interesantes.

2.1 Código

```
10 # Parseo de argumentos para pasarlos por terminal
11 import argparse
12 parser = argparse.ArgumentParser()
13 parser.add_argument('--tam', type=float, help='tamaño del objeto referencia')
14 parser.add_argument('--dist', type=float, help='distancia al objeto referencia')
15 parser.add_argument('--img', type=str, help='ruta hacia la imagen')
16 args = parser.parse_args()
17
18 # Función auxiliar para añadir puntos a las colas de las dos ventanas
19 def encolar(event, x, y, flags, queue):
20     if event == cv.EVENT_LBUTTONDOWN:
21         if queue == 0: pointsF.append((x,y))
22         elif queue == 1: pointsB.append((x,y))
```

El fichero comienza con unas cuantas líneas para tratar los argumentos de la terminal. Para este ejercicio he creído conveniente pasar algunos de los parámetros, que más tarde se usarán, mediante la línea de comandos. Se definen tres argumentos:

- Tam: Tamaño del objeto con el que se va a medir la distancia focal (X en la ecuación).
- Dist: Distancia del objetivo a este objeto (Z en la ecuación).
- Img: Imagen de una pelota sobre una pista de baloncesto, para medir su altura sobre la misma.

Tras esto se define una función auxiliar que nos ayudará a añadir puntos a una cola. Con estos puntos se medirá el tamaño en píxeles de los objetos de la imagen.

```

38 # Bucle principal de entrada de video
39 for key, frame in autoStream():
40     # Dibujar los puntos de la ventana FOV
41     for pf in pointsF:
42         cv.circle(frame, pf, 3, (0,0,255), -1)
43
44     # Si hay dos puntos se dibuja una linea entre ellos y se registran los pixeles que ocupa
45     if len(pointsF) == 2:
46         cv.line(frame, pointsF[0], pointsF[1], (0,0,255))
47         c = np.mean(pointsF, axis = 0).astype(int)
48         pixF = np.linalg.norm(np.array(pointsF[0]) - pointsF[1])
49         putText(frame,f'{pixF:.1f} p',orig=c)
50
51     # Cuando se pulsa 'f' se calcula la distancia focal y el FOV
52     if (pixF > 0) & (key == ord('f')):
53         f = pixF / (args.tam/args.dist)
54         fov = 2*math.degrees(math.atan((401.4)/(2*f)))
55
56     # Cuando se pulsa 'b' se calcula la altura de la cámara sobre el campo de baloncesto
57     if (fov > 0) & (key == ord('b')):
58         h = 14 / math.tan(math.radians(fov/2))

```

Una vez llegamos al bucle principal lo primero que hace el script es dibujar los puntos sobre la ventana, si es que los hay. Estos se detectan mediante eventos de click del ratón definidos sobre la ventana. Tras esto se detectan dos eventos más, la pulsación de la tecla *f* y la pulsación de la tecla *b*. Cuando se pulsa *f* se calcula la distancia focal y el FOV, siempre que haya definida una distancia en píxeles del objeto que se ve. Una vez sabemos el FOV podemos pulsar *b* para calcular la altura necesaria en metros que se necesitaría para que la cámara que se usa en el programa pueda ver completamente una pista de baloncesto en vista cenital.

```

60     # Cuando se pulsa 'i'
61     if key == ord('i'):
62         # Si el flag estaba desactivado se crea una nueva ventana
63         if flag == 0:
64             cv.namedWindow("pista")
65             cv.setMouseCallback("pista", encolar, 1)
66             flag = 1
67         # Si estaba activado se destruye esa ventana
68     else:
69         cv.destroyWindow('pista')
70         flag = 0

```

A continuación se detecta la pulsación de la tecla *i*. Con ella se activa un flag, que activará la segunda parte del bucle. El uso de un flag permite el uso de la tecla *i* como un interruptor.

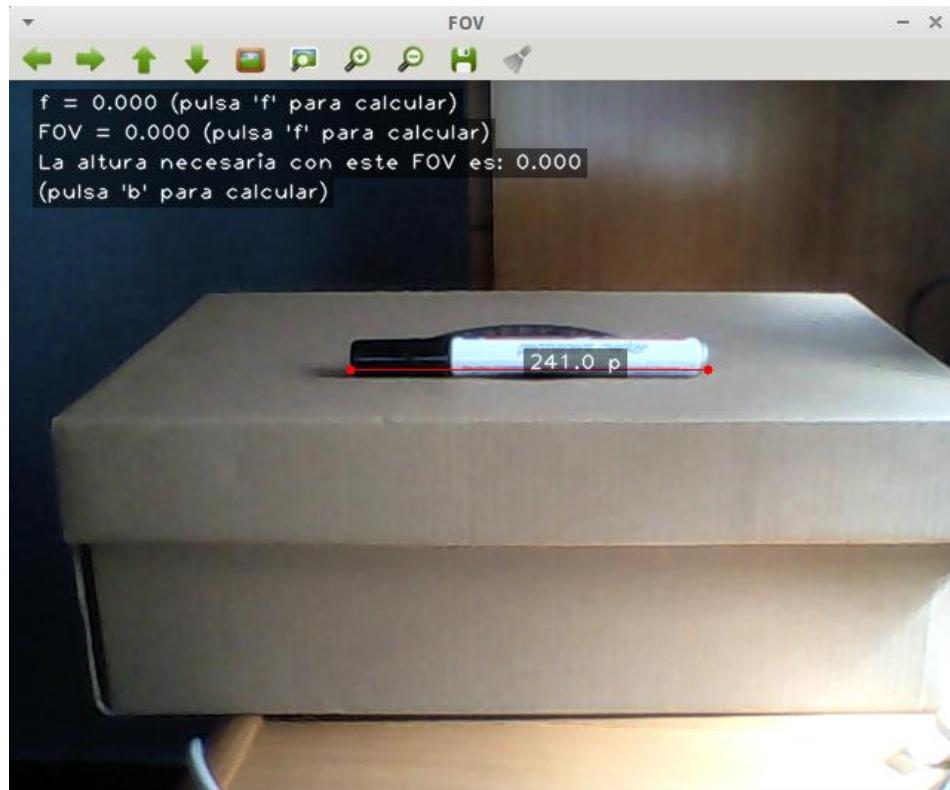
```

72 # Si el flag está activado
73 if flag == 1:
74     # Copiar la imagen original
75     imagen = orig.copy()
76
77     # Dibujar los puntos de la ventana pista
78     for pb in pointsB:
79         cv.circle(imagen, pb, 3, (0,255,255), -1) # Dibujar puntos
80
81     # Si hay dos puntos se dibuja una linea entre ellos y se registran los pixeles que ocupa
82     if len(pointsB) == 2:
83         cv.line(imagen, pointsB[0], pointsB[1], (0,255,255)) # Dibujar linea
84         c = np.mean(pointsB, axis = 0).astype(int) # Calcular media
85         pixB = np.linalg.norm(np.array(pointsB[0]) - pointsB[1]) # Calcular pixeles
86         putText(imagen,f'{pixB:.1f} p',orig=c) # Mostrar longitud
87
88     # Cuando se pulsa 'p' se calcula la distancia desde el suelo a la pelota
89     if (pixB > 0) & (f > 0) & (key == ord('p')):
90         dist = (f*24)/pixB
91         hp = h - ((dist + 24)/100)
92
93     # Imprimir la información y mostrar la imagen en pista
94     putText(imagen,f'La altura de la pelota es de {hp:3.3f} (pulsa \'p\' para calcular)',orig=
95     cv.imshow('pista',imagen)
96
97     # Imprimir la información y mostrar la imagen en FOV
98     putText(frame,f'f = {f:3.3f} (pulsa \'f\' para calcular)',orig=(20,20))
99     putText(frame,f'FOV = {fov:3.3f} (pulsa \'f\' para calcular)',orig=(20,40))
100    putText(frame,f'La altura necesaria con este FOV es: {h:3.3f}',orig=(20,60))
101    putText(frame,f'(pulsa \'b\' para calcular)',orig=(20,80))
102    cv.imshow('FOV',frame)
103
104 # Salir del programa
105 cv.destroyAllWindows()

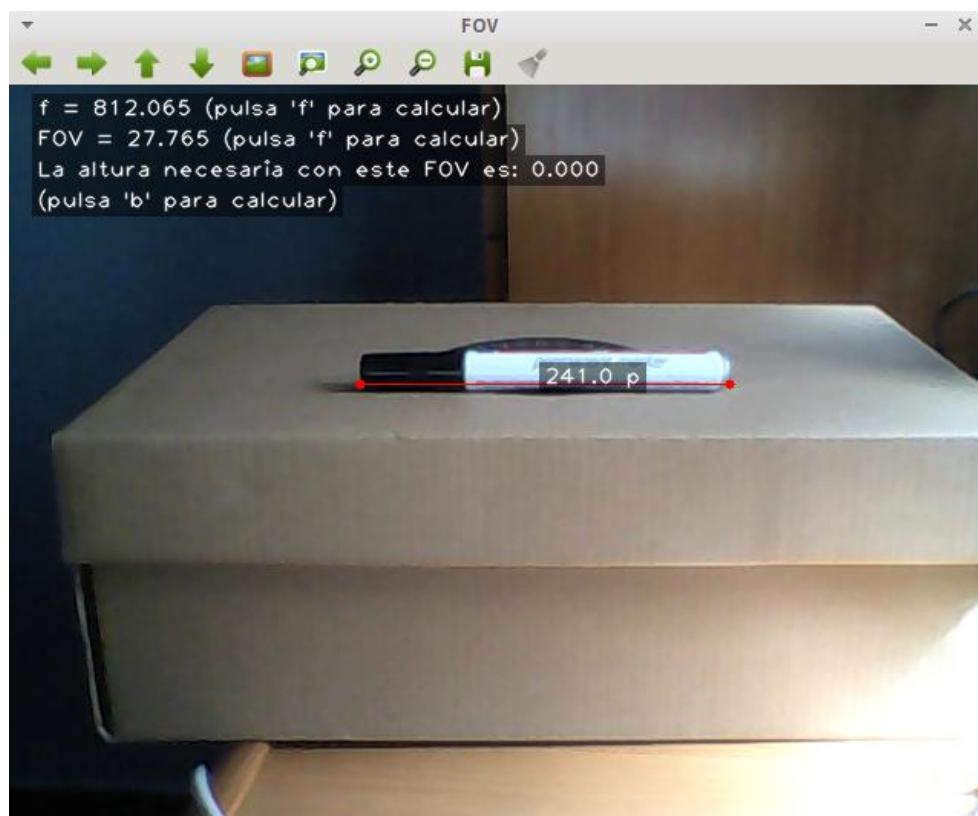
```

Cuando se activa el flag se muestra una nueva ventana. Dentro de ella se comprueba de nuevo si hay puntos para ella y se dibujan. En esta ventana también se detectan los eventos de click de ratón para marcar la distancia en píxeles de la pelota. Cuando se pulsa *p* se calcula la altura de la misma sobre la pista (en metros). Por último se muestra toda la información en pantalla.

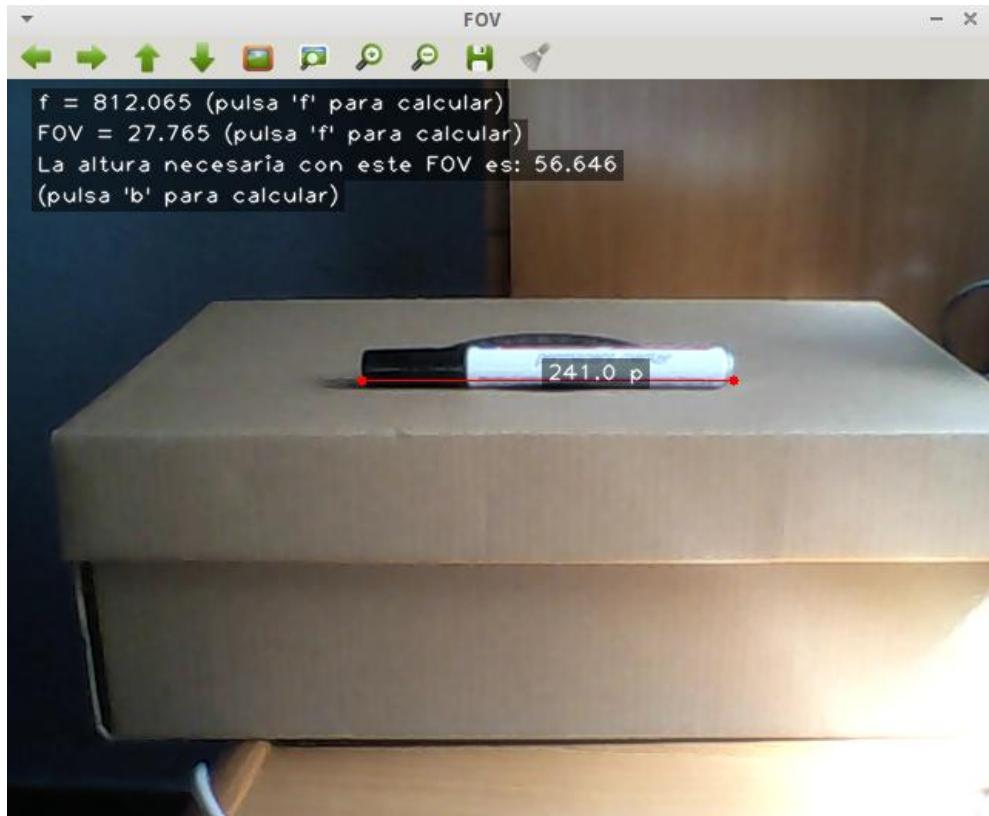
2.2 Ejemplos



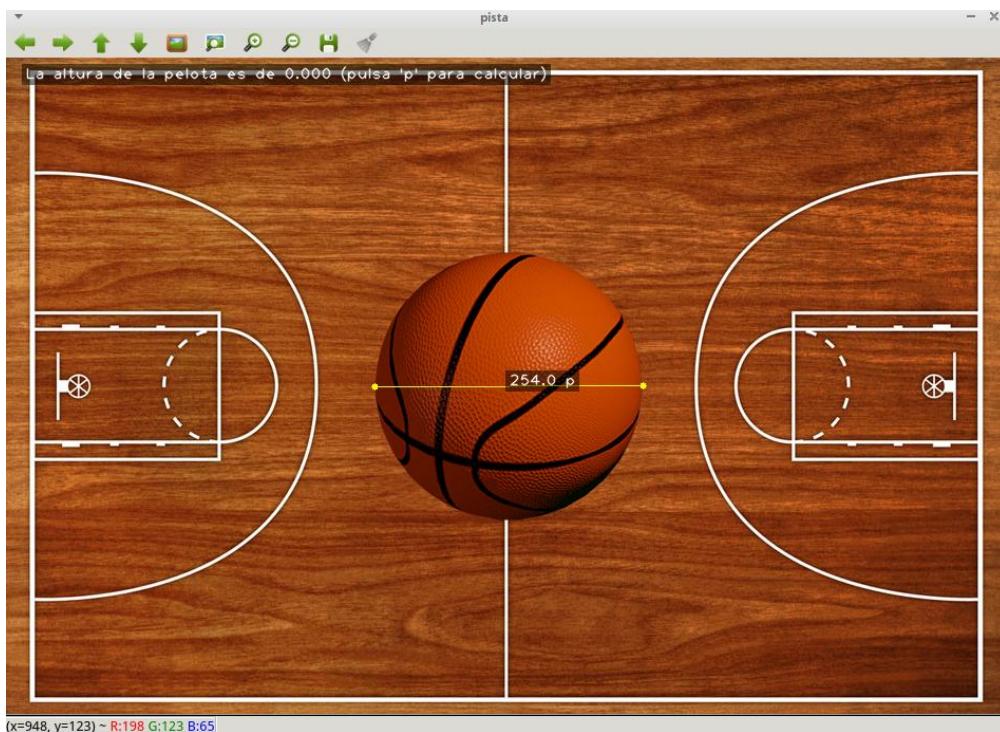
En este primer ejemplo vamos a usar un rotulador de 13'8 cm colocado a una distancia de 46'5 cm. Podemos ver que de primeras no tenemos ningún valor calculado.



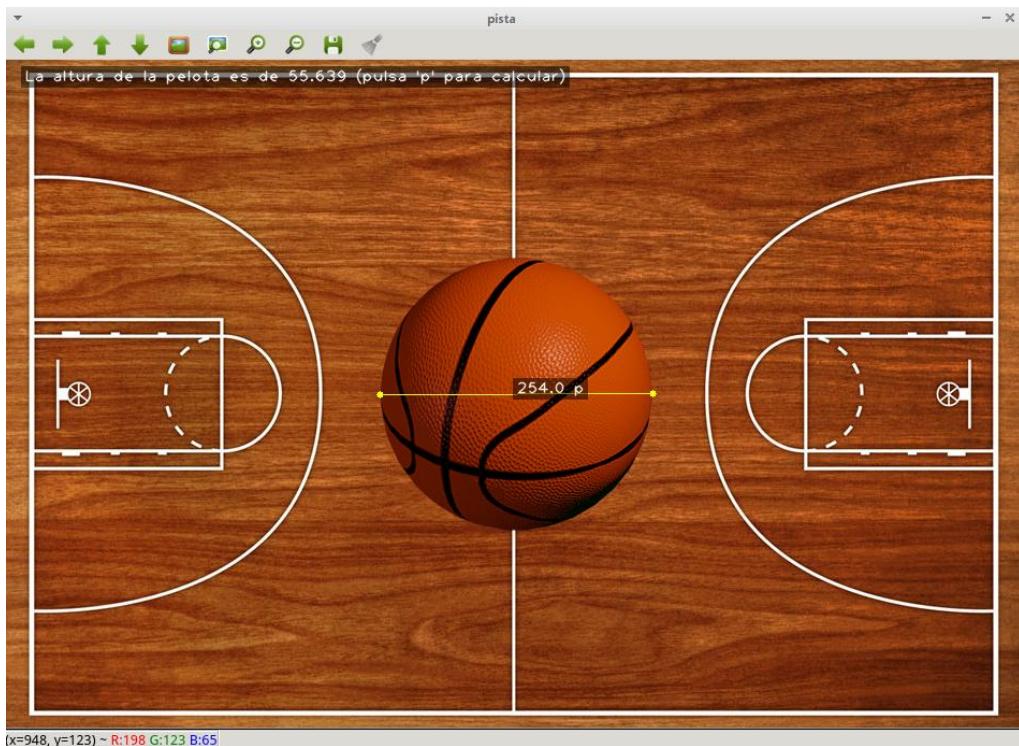
Tras pulsar f obtenemos una distancia focal de 812 aproximadamente y un FOV de 27'7 grados.



En la siguiente imagen tras pulsar la tecla b obtenemos una altura necesaria de 56'6 metros aproximadamente.



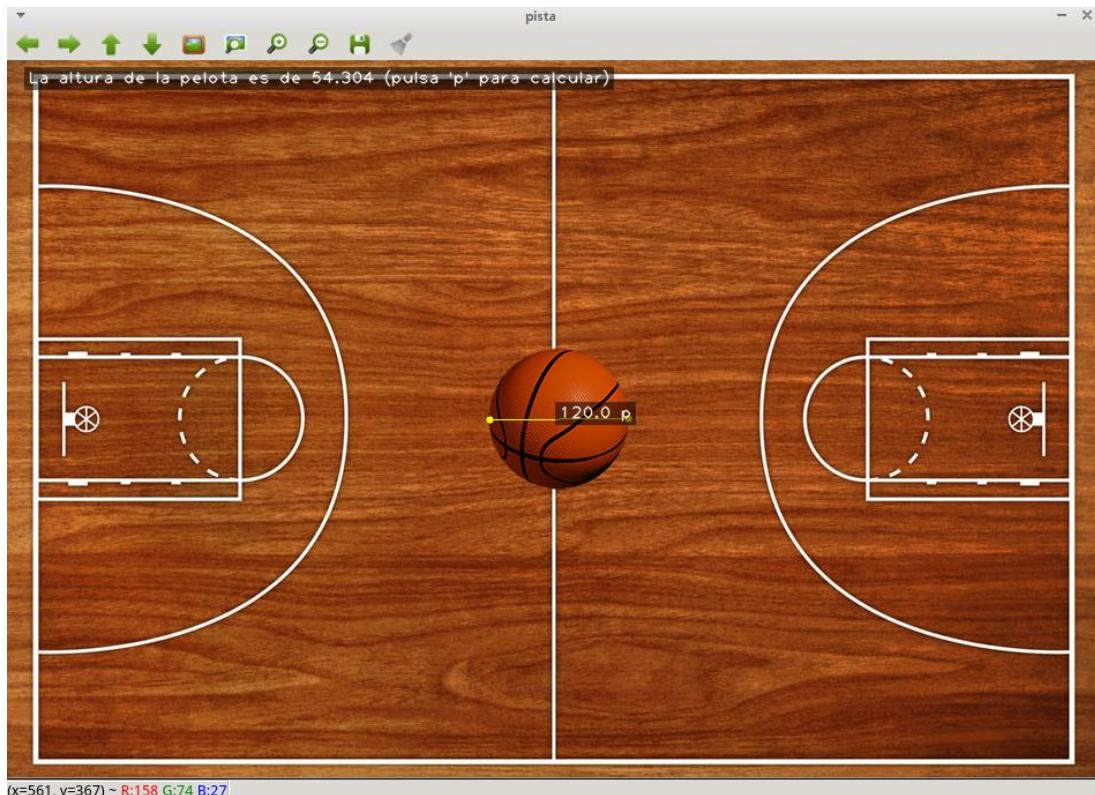
Al pulsar la tecla *i* se muestra en otra ventana la imagen que se pasó por parámetro (en mi caso pista1.jpg). Inicialmente no sabemos la altura de la pelota.



Tras pulsar *i* vemos que la pelota debe estar a unos 55'6 metros del suelo, debido al tamaño de la pelota en la imagen, que indica su proximidad a la cámara. Veamos un segundo ejemplo.



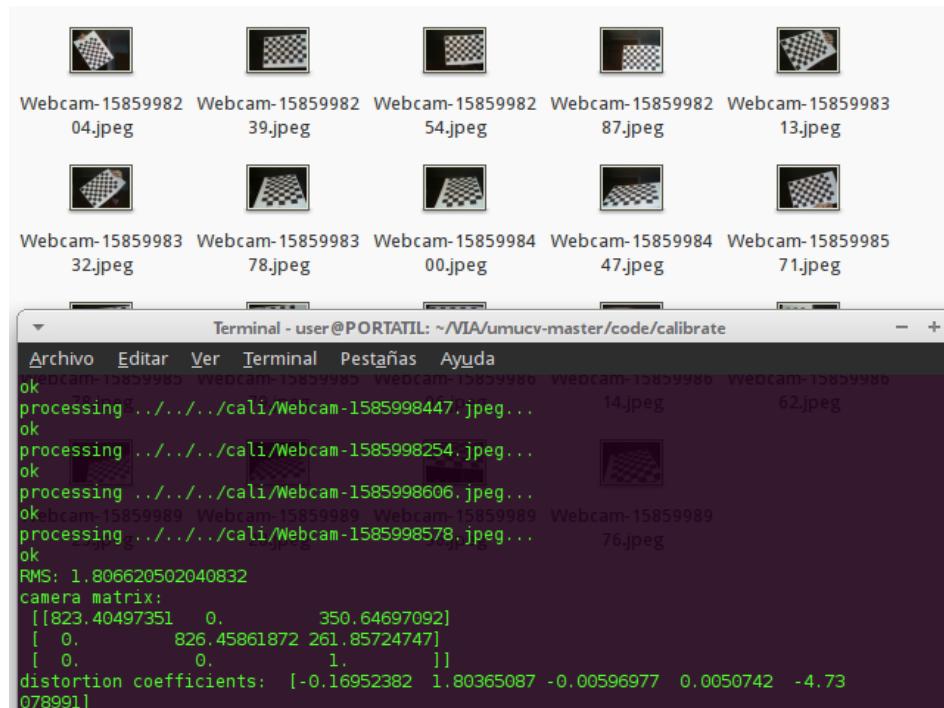
En este caso se ha usado un mando como referencia, ahora el tamaño del objeto es de 22 cm y la distancia al objetivo de 64'4 cm. Podemos ver como la distancia focal ha caído hasta 805 y el FOV ha subido hasta los 27'99 grados. Tras varios ejemplos más he podido comprobar que estos valores, aun que no son exactos, siempre oscilan en torno a estos datos. Al tener un FOV más grande la altura de la cámara es un poco menor (56,15 metros aproximadamente).



Para la imagen introducida (pista2.jpg) obtenemos una altura de 54'3 metros para esta pelota que se ve más lejos del objetivo.

2.3 Calibración

Por último compararemos la distancia focal obtenida por el script de calibrado proporcionado. Para ello he realizado varias capturas desde la Webcam enfocando un patrón concreto similar a un tablero de damas o ajedrez. En la imagen inferior se pueden apreciar algunas de estas capturas y la matriz obtenida del script.



El primer valor de la matriz es la distancia focal. Según este script es de 823'4. Si comparamos este valor con los obtenidos en los ejemplos (con un rango de entre 805 y 812) vemos que es un valor muy aceptable y similar al obtenido en mi script. De hecho en otros ejemplos he llegado a obtener valores para f de 830-840.

3. CHRO

En este ejercicio colocaremos un fondo elegido por el usuario detrás del mismo o de algún objeto.

3.1 Código

```
7 # Parseo de argumentos para pasarlo por terminal
8 import argparse
9 parser = argparse.ArgumentParser()
10 parser.add_argument('--img', type=str, help='ruta hacia la imagen de fondo')
11 args = parser.parse_args()
```

Al comienzo del script se realiza el parseo de los argumentos, en este caso solo se acepta como parámetro una imagen, la que se situará en el fondo de la imagen.

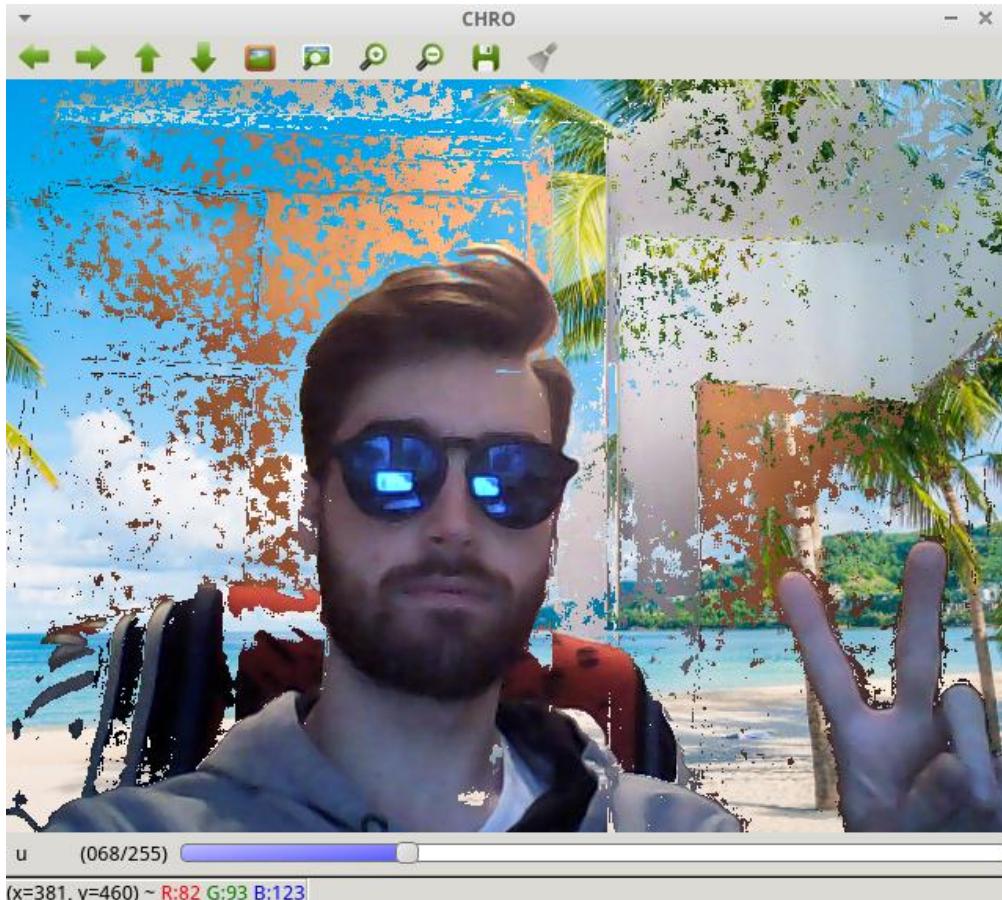
```

27 # Bucle principal de entrada de video
28 for key, frame in autoStream():
29     # Para el primer frame se selecciona este como fondo capturado por defecto
30     if first:
31         prev = frame
32         first = False
33
34     # Cuando se pulsa 'c' se captura el fondo
35     if key == ord('c'): prev = frame
36
37     # Actualizar el umbral con el trackbar
38     u = cv.getTrackbarPos('u','CHRO')
39
40     # Diferencia de imágenes en el espacio RGB
41     diff = np.sum(cv.absdiff(prev,frame), axis=2)
42
43     # Crear la máscara a partir del umbral
44     mask = diff > u
45
46     # Ajustamos el tamaño del fondo elegido para que encaje con la entrada de video
47     r,c = mask.shape
48     result = cv.resize(bg,(c,r))
49
50     # Expandir la máscara a 3 canales para poder copiar RGB
51     mask3 = np.expand_dims(mask, axis=2)
52     np.copyto(result, frame, where = mask3)
53
54     # Mostrar imagen resultante
55     cv.imshow('CHRO',result)
56
57 # Salir del programa
58 cv.destroyAllWindows()

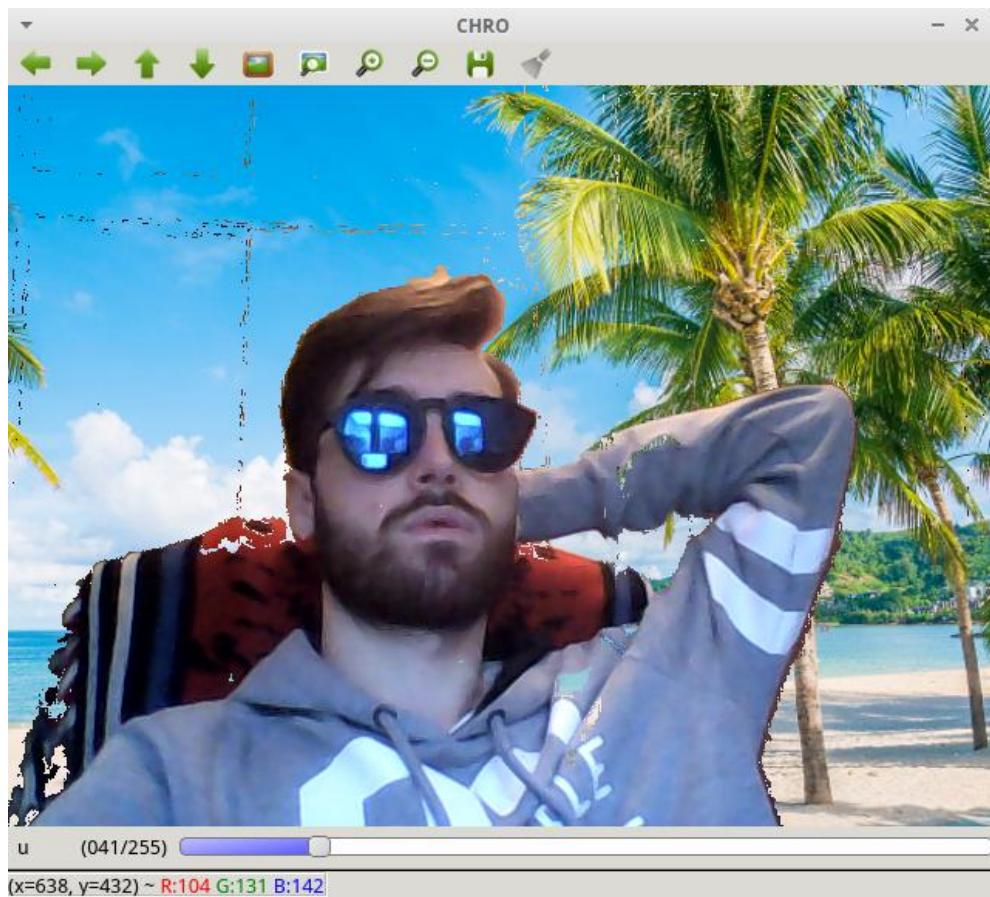
```

Al inicio del bucle principal se elige el primer frame como fondo a borrar, por defecto. En cualquier momento el usuario puede pulsar *c* para capturar uno nuevo. Este fondo capturado se usará para buscar las diferencias entre la imagen actual y esa imagen capturada. La información que varía es nueva, por ejemplo un objeto nuevo en el plano o una persona moviéndose, por lo que esos píxeles deberán permanecer en la imagen final. Tras esto se obtiene el umbral mediante un trackbar colocado en la ventana, a partir de él se enmascara la imagen y se ajusta el fondo, de este modo se eliminan los píxeles que no varíen respecto al fondo y que no superen el umbral.

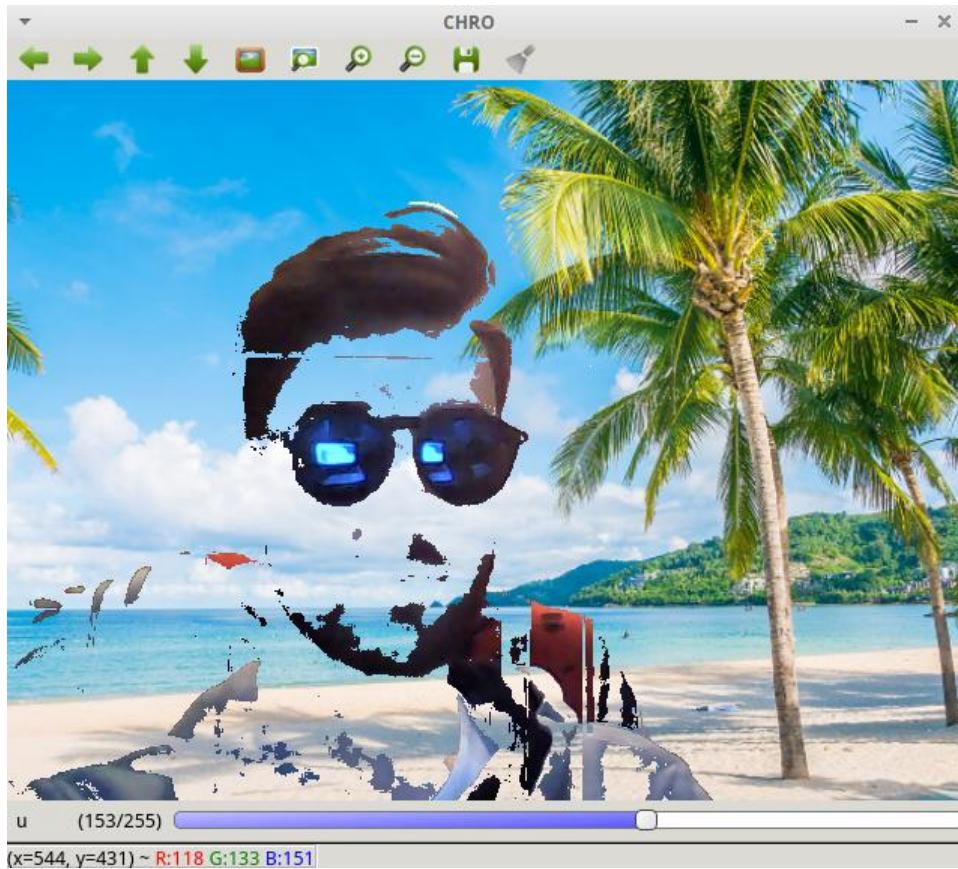
3.2 Ejemplos



Tras capturar el fondo simplemente tenemos que mover la barra inferior para modificar el valor de umbral, esta herramienta permite cambiar el valor en tiempo real para poder ajustarlo visualmente. En esta primera imagen se muestra el resultado obtenido ante un valor de umbral insuficiente. El sujeto se muestra a gran calidad pero el fondo no aparece uniformemente.



En esta imagen se muestra el valor de umbral idóneo. Podemos ver que es inferior al usado en el ejemplo anterior, esto es así porque el umbral afecta de forma específica a cada fondo capturado, por eso mismo no se puede situar en tiempo de compilación un valor de umbral y que funcione adecuadamente en la ejecución.



Por último se muestra el resultado obtenido cuando el valor de umbral es elevado. Se consigue mostrar el fondo de manera uniforme, sin ruido, pero gran parte del sujeto se pierde.

4. HISTCOL

Este ejercicio consiste en un clasificador de objetos basado en los histogramas de cada canal de color de la imagen y los modelos capturados (clases posibles).

4.1 Código

```

9 # Función que calcula el histograma multidimensional de la imagen
10 def hist(x, bins=16):
11     return cv.calcHist([x],      # lista de imágenes
12                        [0,1,2], # canales deseados
13                        None,    # posible máscara
14                        [bins, bins, bins], # número de cajas
15                        [0,256] + [0,256] + [0,256]) # intervalos en cada canal
16

```

En primer lugar se define una función auxiliar que agilizará el cálculo de los histogramas de los canales.

```

27 # Bucle principal de entrada de vídeo
28 for key, frame in autoStream():
29
30     # Si se ha seleccionado una región
31     if roi.roi:
32         # Obtener sus coordenadas
33         [x1,y1,x2,y2] = roi.roi
34
35         # Recortar esa sección de la imagen
36         region = frame[y1:y2+1, x1:x2+1]
37
38         # Obtener sus dimensiones y su tamaño
39         hei, wid = region.shape[0:2]
40         size = hei*wid
41
42         # Calcular el histograma para cada canal de la roi y normalizarlo de dos formas
43         # (para la puntuación de cada modelo y para la representación de los
44         # histogramas de la roi)
45         hb = cv.calcHist([region],[0],None,[256],[0,256])
46         bValue = hb / size
47         cv.normalize(hb,hb,0,255,cv.NORM_MINMAX)
48         hg = cv.calcHist([region],[1],None,[256],[0,256])
49         gValue = hg / size
50         cv.normalize(hg,hg,0,255,cv.NORM_MINMAX)
51         hr = cv.calcHist([region],[2],None,[256],[0,256])
52         rValue = hr / size
53         cv.normalize(hr,hr,0,255,cv.NORM_MINMAX)

```

Al empezar el bucle principal se detecta si se ha seleccionado una región de interés o no. Cuando existe una Roi se trata esta como una imagen por separado. Primero se recorta esta sección, se obtiene su tamaño y se calculan los histogramas normalizados para cada canal. El histograma se normaliza de dos formas, una necesaria para mostrarlo de forma adecuada con la función *polyline()*, la otra para obtener el valor que se usará para elegir el modelo más cercano a la Roi.

```

55     # Si se pulsa la tecla 'm'
56     if key == ord('m'):
57         # Copiar la roi
58         trozo = region.copy()
59
60         # Redimensionar a la altura y ancho definida
61         trozo = cv.resize(trozo,(w,h))
62         if first: # Si es el primer modelo se añade a la lista
63             models.append(trozo)
64             modelsImg = trozo
65             first = False
66         else: # Si no es el primero se crea una imagen que combina cada modelo
67             models.append(trozo)
68             modelsImg = np.hstack((modelsImg, trozo))
69
70         # Calcular el histograma de cada canal del modelo y normalizarlo
71         valsB = cv.calcHist([trozo],[0],None,[256],[0,256]) / size
72         valsG = cv.calcHist([trozo],[1],None,[256],[0,256]) / size
73         valsR = cv.calcHist([trozo],[2],None,[256],[0,256]) / size
74
75         # Añadir los histogramas a la lista
76         modelsH.append(valsB)
77         modelsH.append(valsG)
78         modelsH.append(valsR)
79
80         # Eliminar la roi cuando se registra el modelo
81         roi.roi = []

```

Inmediatamente después se detecta la pulsación de la tecla *m*. Cuando este evento ocurre se captura la Roi seleccionada y se añade como modelo. Para ello se registra la imagen y sus valores del histograma de cada canal. Además se borra la Roi seleccionada.

```

83      # Dibujar el rectangulo de la roi
84      cv.rectangle(frame, (x1,y1), (x2,y2), color=(0,255,255), thickness=2)
85
86      # Crear el indice del histograma [0 .. 255] pero ajustado al tamaño de la roi
87      index = np.int32(np.around(np.arange(256) * ((wid-6)/256) + x1+3))
88
89      # Obtener los valores para cada canal del histograma ajustados al tamaño de la roi
90      vals=np.int32(np.around(abs(hb - 256) * ((hei-5)/256) + y1+2))
91      pts = np.column_stack((index,vals))
92      cv.polyline(frame,[pts],False,(255,0,0),2)
93
94      vals=np.int32(np.around(abs(hg - 256) * ((hei-5)/256) + y1+2))
95      pts = np.column_stack((index,vals))
96      cv.polyline(frame,[pts],False,(0,255,0),2)
97
98      vals=np.int32(np.around(abs(hr - 256) * ((hei-5)/256) + y1+2))
99      pts = np.column_stack((index,vals))
100     cv.polyline(frame,[pts],False,(0,0,255),2)
```

A continuación se procede a dibujar en la Roi. En primer lugar se dibuja en pantalla el rectángulo que define la Roi. Después se dibuja el histograma de cada canal. Para ello se tienen que calcular dos dimensiones de la gráfica:

- La dimensión X: En un histograma estándar es el rango de valores posibles (en nuestro caso [0 .. 255]. Para poder mostrarlo dentro de la Roi esta dimensión no puede ser estática, por lo que a veces la Roi medirá menos de 256 píxeles en la X y se deberán solapar valores para un mismo valor de X, otras veces en cambio se debe estirar el histograma replicando datos para alcanzar un ancho de la Roi superior a 256 píxeles. Además hay que llevar la gráfica hasta la Roi, ya que por defecto se muestra en la coordenada (0,0) como punto inicial. Por este motivo se realiza el cálculo de la variable *index*.
- La dimensión Y: En este caso ocurre algo parecido a lo que ocurre en la dimensión X, de hecho el cálculo es muy similar como se puede ver, pero hay que añadir que los valores de Y por defecto crecen hacia abajo, es decir, un valor de histograma 255 se sitúa por debajo de otro que sea de valor 0, por lo que para que el valor 0 sea el más bajo hay que invertir los valores de la dimensión Y, lo que se consigue restando 256 y calculando el valor absoluto.

```

102     # Encontrar el modelo con menor diferencia a la roi
103     mini = 1000
104     for i in range(len(models)):
105         # Por cada modelo registrado
106         # Calcular la diferencia por cada canal
107         diffB = np.sum(np.abs(cv.subtract(bValue, modelsH[i*3], dtype=cv.CV_64F)), axis=0)
108         diffG = np.sum(np.abs(cv.subtract(gValue, modelsH[(i*3)+1], dtype=cv.CV_64F)), axis=0)
109         diffR = np.sum(np.abs(cv.subtract(rValue, modelsH[(i*3)+2], dtype=cv.CV_64F)), axis=0)
110         # Obtener el máximo de las tres diferencias
111         maxi = np.max([diffR,diffG,diffB])
112         putText(frame,f'{maxi:2.2f}',orig=(42*i,20))
113
114         # Obtener el modelo que lo ha producido
115         if maxi < mini:
116             mini = maxi
117             miniModel = i
118
119         # Si hay algún modelo registrado mostramos el modelo elegido como más parecido a la roi
120         if len(models) > 0:
121             cv.namedWindow('elegido')
122             cv.imshow('elegido',models[miniModel])
123
124         # Cuando hay algún modelo registrado se muestra la ventana con los modelos registrados
125         if not first:
126             cv.namedWindow('modelos')
127             cv.imshow('modelos', modelsImg)
128
129         # Mostrar la imagen original
130         cv.imshow('original',frame)
131
132     # Salir del programa
133     cv.destroyAllWindows()

```

Por último se debe calcular la diferencia de cada modelo respecto a la Roi seleccionada. Para ello se calcula la diferencia absoluta de cada canal y se escoge el mayor valor. De este valor resultante se escoge el modelo que ofrece el menor, ya que será el menos diferente a la Roi. Finalmente se muestra el modelo escogido y todos modelos totales, siempre que existan estos objetos.

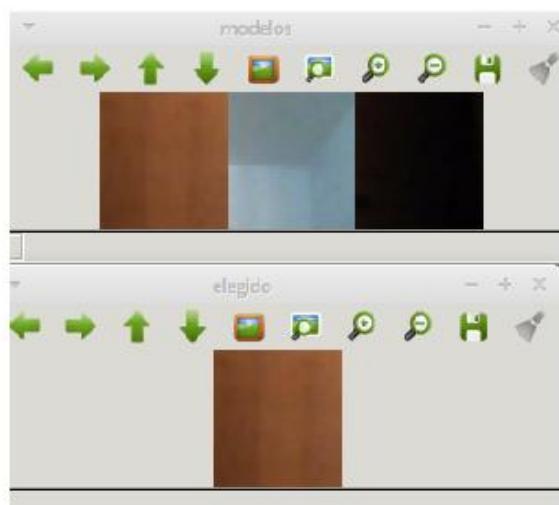
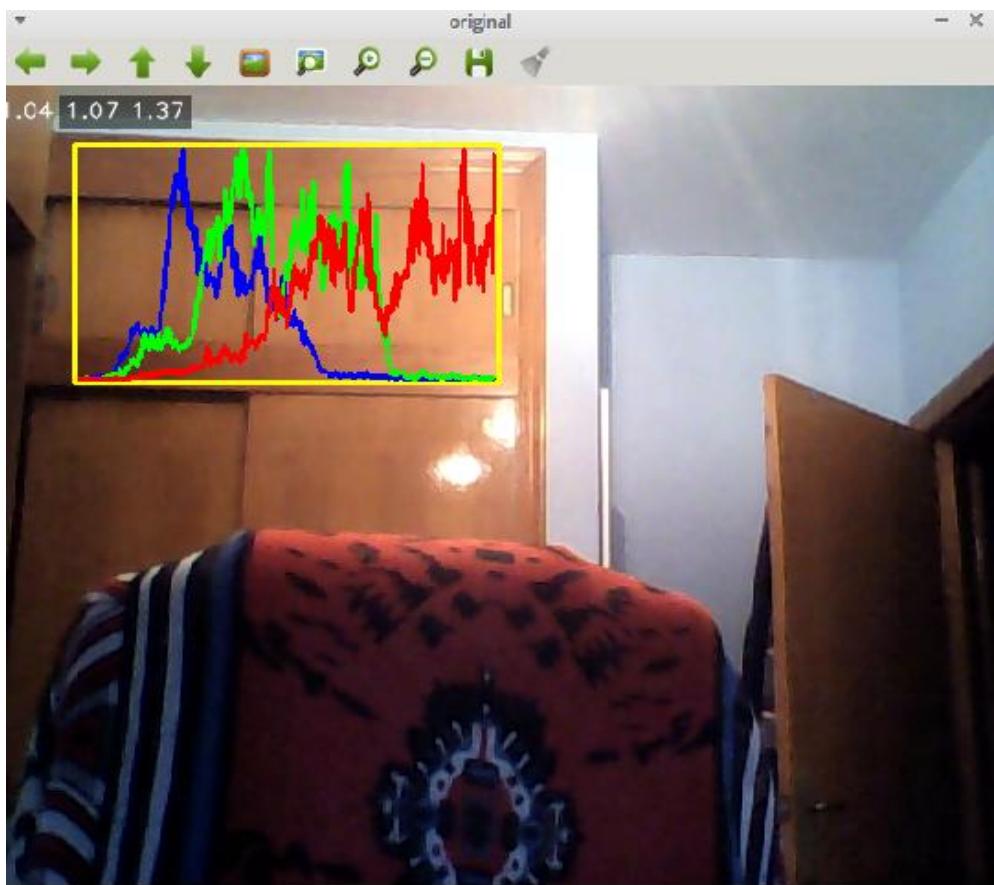
4.2 Ejemplos



Describamos la imagen por partes. Comenzando por la parte superior izquierda podemos ver tres valores, estas son las distancias que la Roi tiene hasta cada modelo. La menor distancia está en la segunda posición, indicando que el segundo modelo es el más similar a la Roi. En la parte superior derecha podemos ver la Roi en cuestión junto con sus histogramas. Debajo de la imagen se muestran los modelos que se seleccionaron para el ejemplo, que aparecen redimensionados para que todos tengan el mismo tamaño. Por último debajo de los modelos vemos el modelo ganador que en este caso es el segundo como ya se anticipó. Visualmente podemos comprobar que realmente es el más parecido a la Roi.



En este ejemplo seleccionamos otra región diferente y vemos como el clasificador también acierta indicando que el modelo más similar es el tercero.



Por último se selecciona otra región para ver como el clasificador también escoge el primer modelo cuando es el más cercano en color a la Roi.

5. DLIB

En este ejercicio se usa el detector de caras de DLIB para aplicar un efecto sobre ellas.

5.1 Código

```
9 # Parseo de argumentos para pasarlo por terminal
10 import argparse
11 parser = argparse.ArgumentParser()
12 parser.add_argument('--predictor', type=str, help='ruta hacia el predictor de la cara')
13 args = parser.parse_args()
```

En primer lugar se realiza el parseo de argumentos. Este script únicamente recibe un parámetro que es la ruta hasta el predictor de caras.

```
23 # Bucle principal de entrada de video
24 for key,frame in autoStream():
25     # Imprimir las instrucciones en la pantalla
26     putText(frame,f'Pulsa \'p\' para mostrar los puntos de la cara',orig=(10,20))
27     putText(frame,f'Pulsa \'e\' para aplicar efectos. Factor actual: {f:d}',orig=(10,40))
28
29     # Activar/Desactivar los flags y actualizar el factor
30     if key == ord('p'):
31         if puntos: puntos = False
32         else: puntos = True
33     if key == ord('e'):
34         if efectos & (f < 4): f = f+1
35         elif efectos & (f == 4):
36             efectos = False
37             f = 0
38         else:
39             efectos = True
40             f = 2
```

Al empezar el bucle principal se muestra la información sobre el uso del script. Inmediatamente después se actualizan los flags mediante la pulsación de *p* y *e*. Estos flags activarán funcionalidades posteriores del programa. Además la pulsación de *e* actualiza el factor de aumento.

```
42     # Detectar la cara de la imagen
43     dets = detector(frame, 0)
44     # Para cada cara detectada
45     for k, d in enumerate(dets):
46         # Obtener los puntos de la cara
47         shape = predictor(frame, d)
48
49         # Crear una lista con las coordenadas de los puntos de la cara
50         L = []
51         for p in range(68):
52             x = shape.part(p).x
53             y = shape.part(p).y
54             L.append(np.array([x,y]))
55         L = np.array(L)
56
57         # Si el flag de puntos está activado se muestran
58         if puntos:
59             cv.rectangle(frame, (d.left(), d.top()), (d.right(), d.bottom()), (255,128,64) )
60             for p in range(68):
61                 x = shape.part(p).x
62                 y = shape.part(p).y
63                 cv.circle(frame, (x,y), 2,(255,0,0), -1)
```

A continuación se detecta la cara y se almacenan en un array los puntos de la misma. Si el flag *puntos* está activado estos puntos se muestran, si no se omiten.

```
65      # Si el flag de efectos está activado se muestra el efecto con el factor f
66  if efectos:
67      # Si hay puntos de cara detectados
68  if len(L) > 0:
69      # Obtener los puntos de la boca
70  mouth = []
71  for p in range(12):
72      mouth.append(L[p+48])
73  mouth = np.array(mouth, dtype=np.int32)
74
75  # Crear una máscara del tamaño de las dimensiones de la imagen
76  mask = np.zeros(frame.shape[:2], np.uint8)
77
78  # Rellenar un polígono en la máscara con los puntos de la boca
79  cv.fillPoly(mask, [mouth], 255)
80
81  # Enmascarar la imagen con la máscara de la boca
82  masked_data = cv.bitwise_and(frame, frame, mask=mask)
83
84  # Obtener el contorno de la máscara y el rectángulo que la contiene
85  _, thresh = cv.threshold(mask, 1, 255, cv.THRESH_BINARY)
86  contours = cv.findContours(thresh, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
87  x,y,w,h = cv.boundingRect(contours[0])
88
89  # Calcular las nuevas dimensiones en función del factor y recortar la boca
90  neww = w*f
91  newh = h*f
92  crop = cv.resize(masked_data[y:y+h,x:x+w], (neww,newh))
93
94  # Calcular los márgenes para centrar la imagen y obtener las dimensiones originales
95  marginw = int(round((neww-w)/2))
96  marginh = int(round((newh-h)/2))
97  height, width = frame.shape[:2]
```

Cuando el flag *efectos* se encuentra activado se realiza el efecto de agrandar la boca. Para ello primero se obtienen los puntos de la boca, si es que hay puntos de la cara generados. Se crea una máscara con las coordenadas de la boca y se recorta solo la zona donde se encuentra el polígono. Tras esto se calcula el nuevo tamaño de la boca y se redimensiona la imagen. A continuación se calculan las coordenadas donde se debe mostrar la imagen para que siga centrada en la cara (para ello debemos saber el margen que debe sobrar por cada borde de la imagen grande respecto de la pequeña).

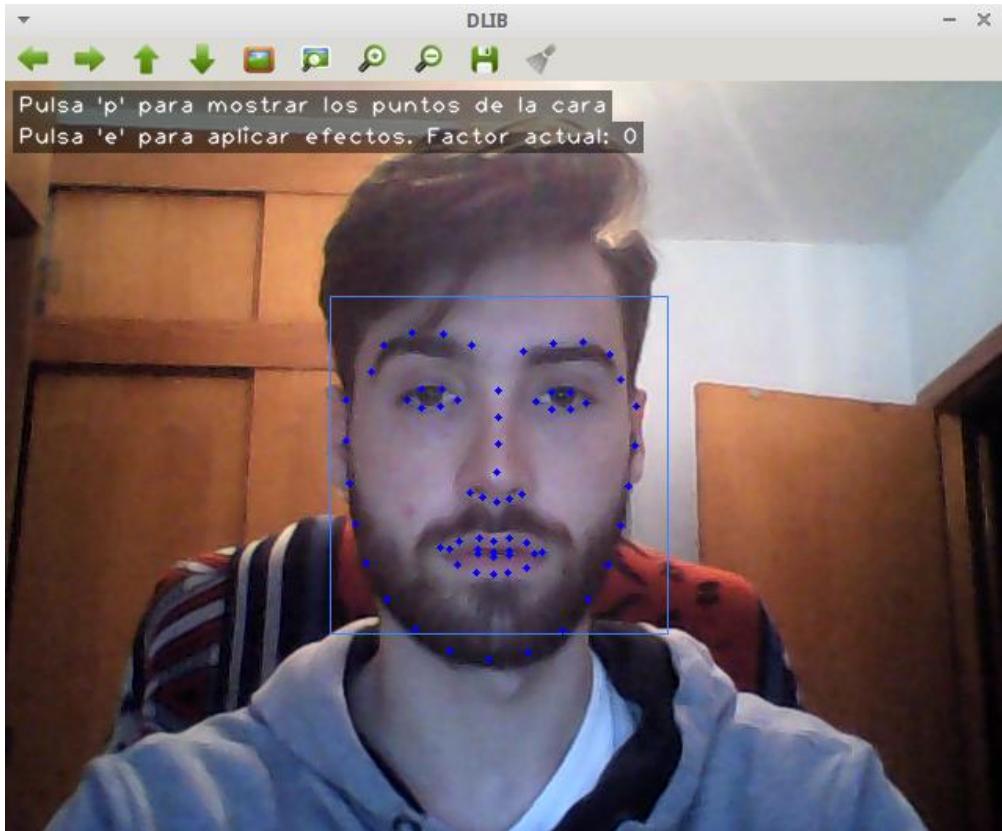
```

99      # Calcular los nuevos puntos de inicio y fin para la imagen original
100     newx = x - marginw
101     newy = y - marginh
102     finx = newx+neww
103     finy = newy+newh
104
105     # Puntos de inicio y fin de la boca recortada
106     cropx = 0
107     cropy = 0
108     cropfinx = neww
109     cropfiny = newh
110
111     # Comprobaciones para los casos en los que la imagen sale del
112     # rango de la cámara (derecha y abajo)
113     if (newx+neww) > width:
114         cropfinx = cropfinx - (finx - width)
115         finx = width
116     if (newy+newh) > height:
117         cropfiny = cropfiny - (finy - height)
118         finy = height
119
120     # Comprobaciones para los casos en los que la imagen sale del
121     # rango de la cámara (izquierda y arriba)
122     if newx < 0:
123         cropx = -newx
124     if newy < 0:
125         cropy = -newy
126
127     # Bucle que muestra la boca aumentada sobre la real. Sigue las coordenadas
128     # descriptas e ignora los valores negros de la máscara
129     for j in range(cropy,cropfiny):
130         for i in range(cropx,cropfinx):
131             if np.any(crop[j,i]): frame[newy+j,newx+i] = crop[j,i]

```

Cuando sabemos las coordenadas donde se dibujará la boca se comprueban algunos casos especiales relacionados con salirnos de las dimensiones de la ventana original. Una vez tenemos las coordenadas corregidas tan solo recorremos la dimensión X para cada coordenada de Y colocando en ella el píxel correspondiente de nuestra máscara calculada, en caso de que este no sea un pixel negro, ya que esos valores corresponden a valores restantes de la máscara, obtenidos al recortar el polígono de la boca mediante un rectángulo. De este modo solo se dibuja la boca en pantalla.

5.2 Ejemplos



En este primer ejemplo se muestra la distribución de los puntos por la cara.



Tras una pulsación de *e* el factor de aumento se sitúa en 2.

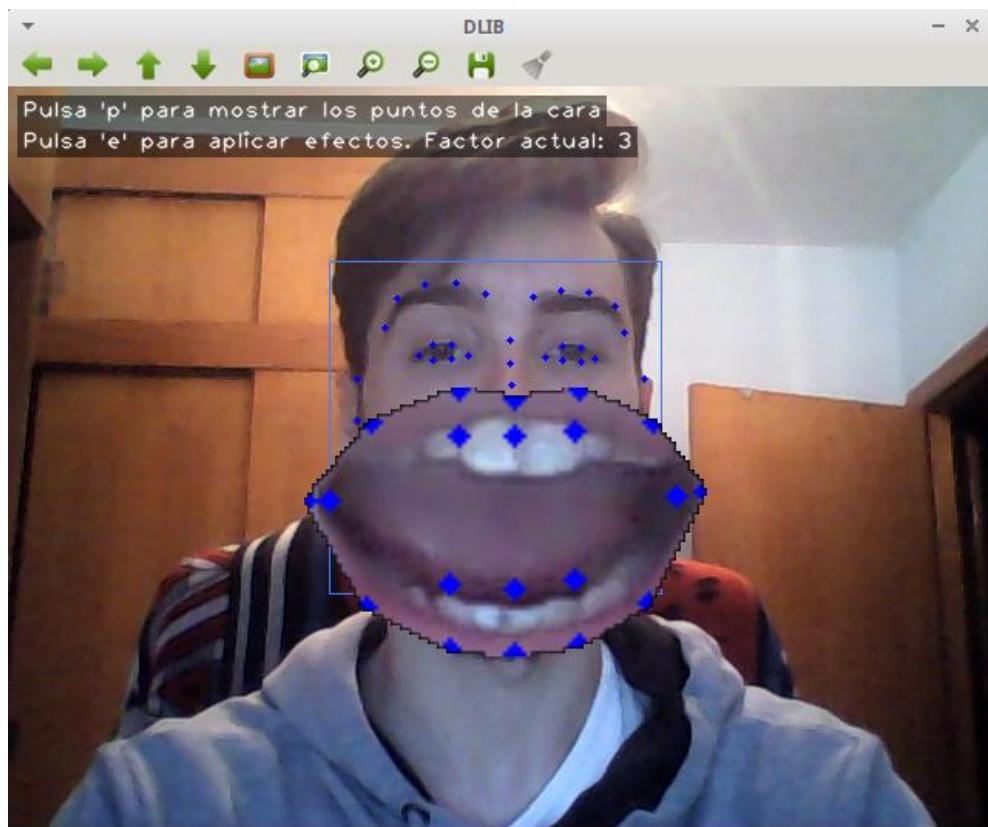


Tras otra pulsación de *e* el factor aumenta a 3.

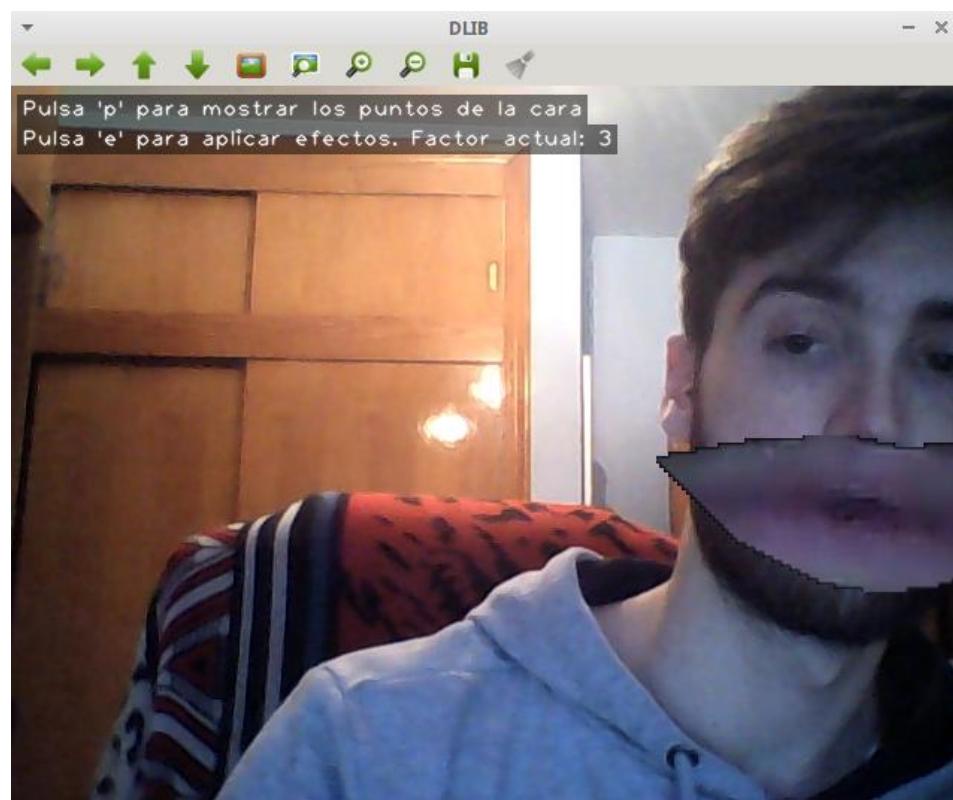


Tras una pulsación más de *e* el factor llega a su máximo (4). Hay que destacar que en este punto el rendimiento del programa cae considerablemente. Si se pulsase una vez

más la tecla *e* el factor se resetearía a cero.



También podemos ver los puntos mientras se está aplicando el efecto.



Por último vemos el correcto funcionamiento del efecto cuando sobrepasa las

dimensiones de la ventana.

6. SIFT

Este ejercicio consiste en implementar un clasificador de objetos mediante detección de puntos claves.

6.1 Código

```
9 # Crear una instancia del método de selección de características, el matcher y la ventana
10 sift = cv.xfeatures2d.SIFT_create(nfeatures=500)
11 matcher = cv.BFMatcher()
12 cv.namedWindow('SIFT')
```

En primer lugar debemos declarar el método de selección de características y Matcher.

```
22 # Bucle principal de entrada de vídeo
23 for key, frame in autoStream():
24
25     # Detectar las características del frame actual y registrar el tiempo
26     t0 = time.time()
27     keypoints, descriptors = sift.detectAndCompute(frame, mask=None)
28     t1 = time.time()
29
30     # Cuando se pulsa 'm' se captura un modelo
31     if key == ord('m'):
32         # Redimensionar la imagen para mostrarla
33         model = cv.resize(frame,(w,h))
34         if first: # Si es el primer modelo la imagen de la ventana será solo este
35             modelsImg = model
36             first = False
37         else: # Si no es el primero se crea una imagen que combina cada modelo
38             modelsImg = np.hstack((modelsImg, model))
39
40     # En cualquier caso se añade a la lista de modelos y se registran sus características
41     models.append(frame.copy())
42     kpImages.append(keypoints)
43     descImages.append(descriptors)
44
45     # Mostrar la información en pantalla
46     putText(frame, f'Pulsa \'m\' para capturar nuevos modelos',orig=(10,20))
47     putText(frame, f'TC: {len(keypoints)} pts = {1000*(t1-t0):.0f} ms',orig=(10,40))
48
49     # Dibujar los keypoints
50     flag = cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
51     cv.drawKeypoints(frame, keypoints, frame, color=(100,150,255), flags=flag)
```

Nada más comenzar el bucle principal se detectan los puntos clave del frame actual, además se registra el tiempo que se tarda en este proceso para mostrar la información por pantalla. A continuación se detecta la pulsación de *m*. Cuando ocurre se registra el modelo, por un lado se guarda la imagen para mostrarla y por otro se almacenan sus puntos clave y sus detectores para poder hacer más tarde la comparación. Tras imprimir la información por pantalla se dibujan los puntos clave del frame actual.

```

53     # Para cada modelo de la lista
54     bestMatch = 0
55     for m in range(len(models)):
56         # Usar el matcher para calcular las coincidencias
57         matches = matcher.knnMatch(descriptors, descImages[m], k=2)
58
59         # Guardar solo las coincidencias prometedoras, las que son mucho mejores que
60         # la segunda opción
61         good = []
62         for mat in matches:
63             if len(mat) >= 2:
64                 best,second = mat
65                 if best.distance < 0.75*second.distance:
66                     good.append(best)
67
68         # Obtener el porcentaje de coincidencia (matches/keypoints totales del modelo)
69         percent = (len(good)/len(kpImages[m]))*100
70
71         # Si el porcentaje es mejor que el mejor que se había encontrado, se guarda
72         if percent > bestMatch:
73             bestMatch = percent
74             bestImg = m
75
76         # Solo cuando se supere el umbral se muestra el modelo elegido (para esto debe haber modelos)
77         if bestMatch > umbral:
78             putText(frame ,f'Modelo {bestImg:d} con un {bestMatch:.2f}%',orig=(10,60))
79             frame[70:70+h,10:10+w] = cv.resize(models[bestImg],(w,h))
80
81         # Cuando hay algún modelo registrado se muestra la ventana con los modelos registrados
82         if not first:
83             cv.namedWindow('modelos')
84             cv.imshow('modelos', modelsImg)

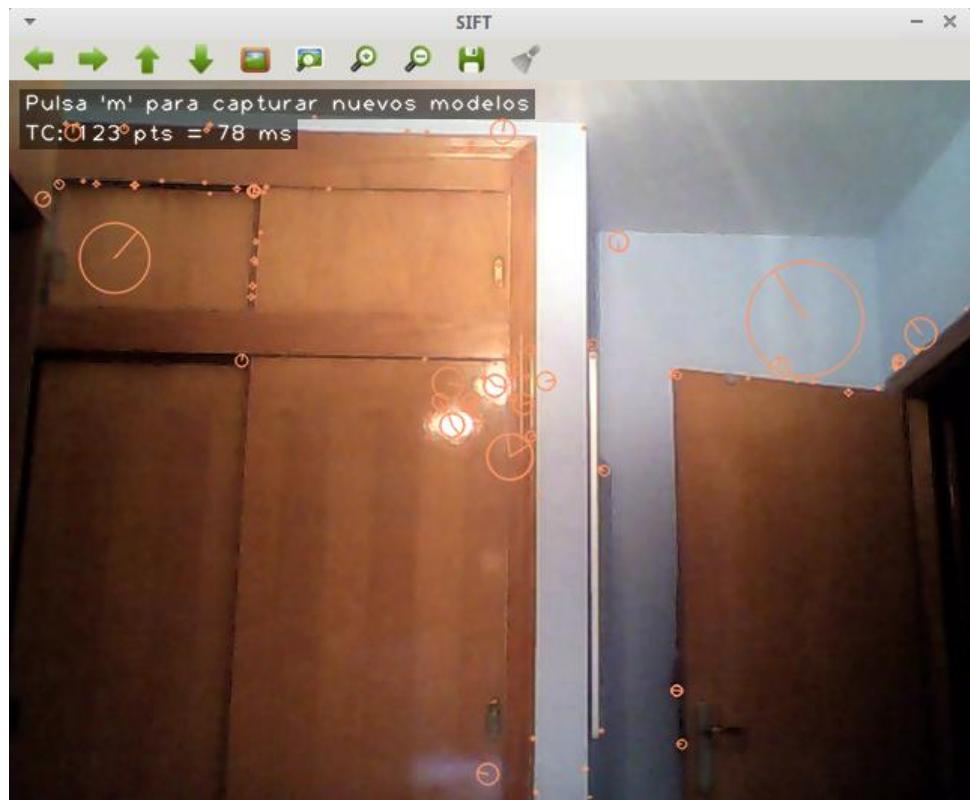
```

Tras esto se comprueba como de parecido es cada modelo a la imagen actual. Para ello se recorren los modelos y se comprueban las coincidencias. De las coincidencias se guardan solo las que sean muy buenas, es decir, mucho mejores que el segundo punto posible con el que podían coincidir. Cuando se sabe el número de enlaces podemos obtener una proporción que nos indique como de similar es la imagen al modelo. El modelo ganador será el que obtenga un mayor porcentaje. Por último se muestra el modelo si es que supera el umbral escogido, para la práctica a mi me ha ido muy bien con un umbral del 20%. Si el modelo es similar en un porcentaje inferior no se ofrece ningún ganador. Un umbral del 20% permite reconocer las imágenes cuando no se muestran totalmente pero aun así no confunde objetos clasificándolos como clases erróneas.

6.2 Ejemplos



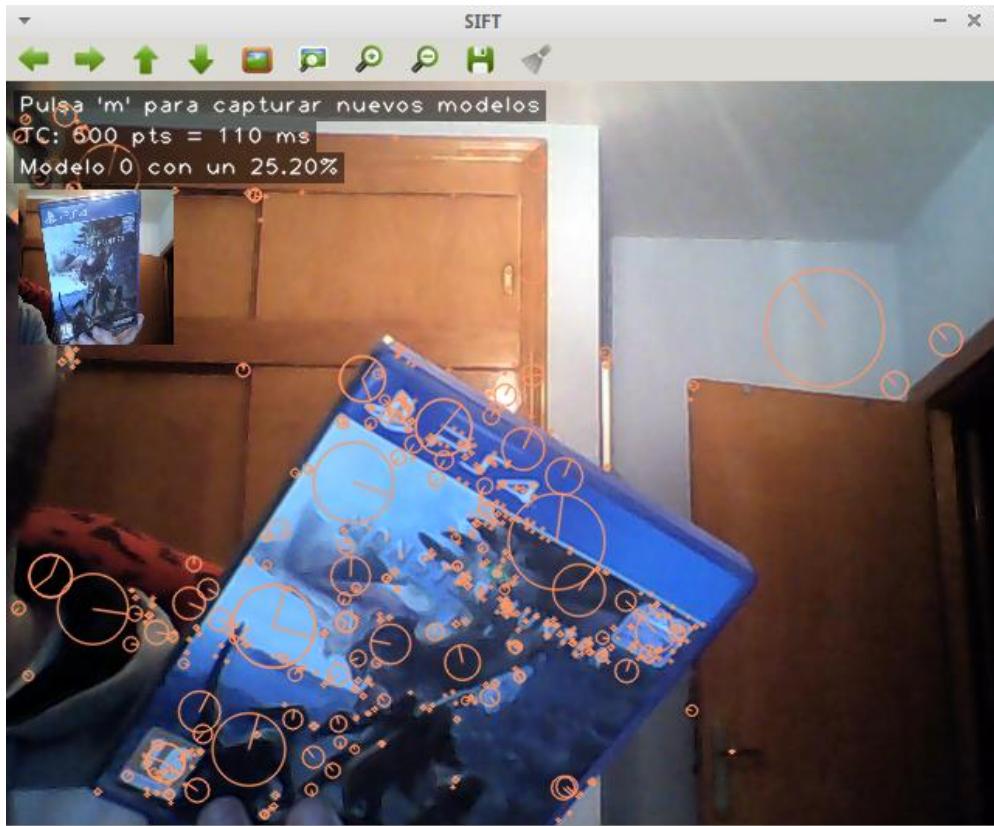
En primer lugar se registran X modelos, en este caso he usado estos videojuegos por tener muchas características a reconocer.



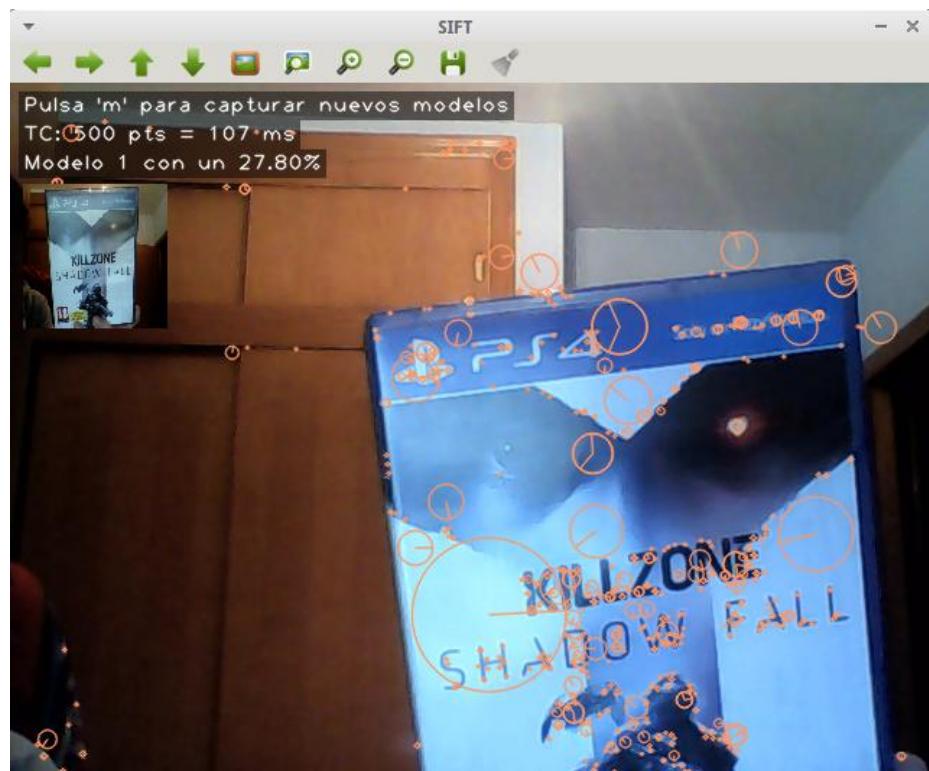
En este primer ejemplo no se encuentra ninguna coincidencia con ninguna clase, pero se sigue mostrando la información del tiempo de cómputo.



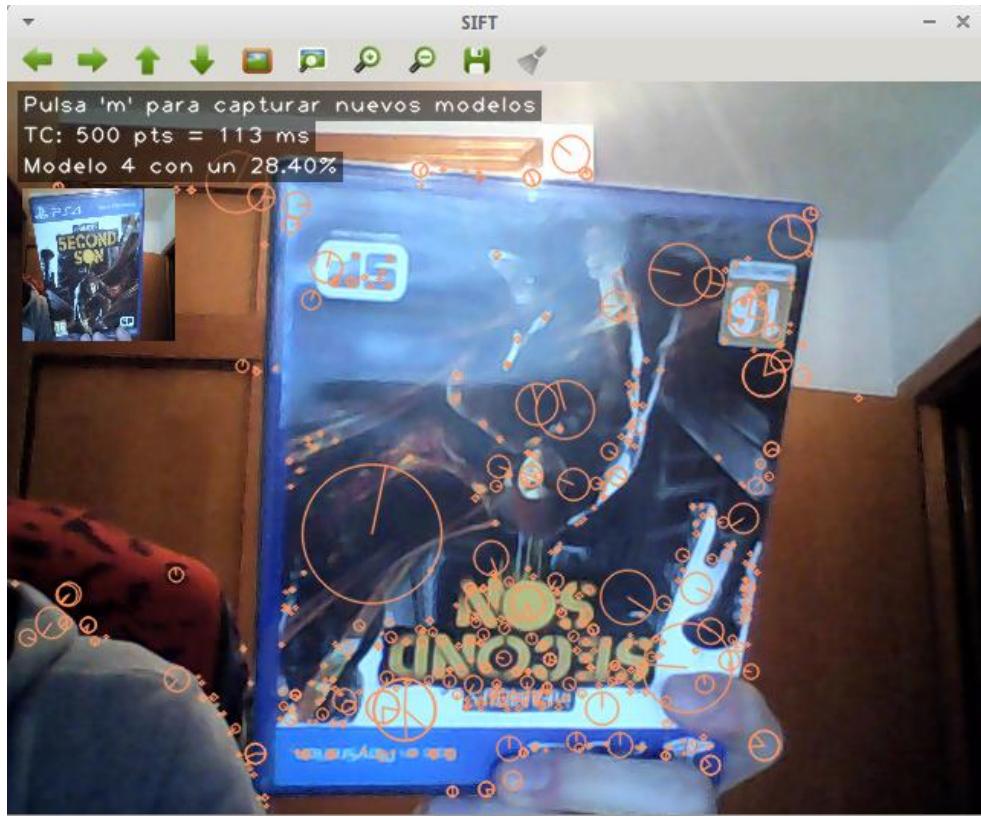
En este ejemplo vemos una gran cantidad de coincidencia al mostrarse el objeto tal cual se capturó.



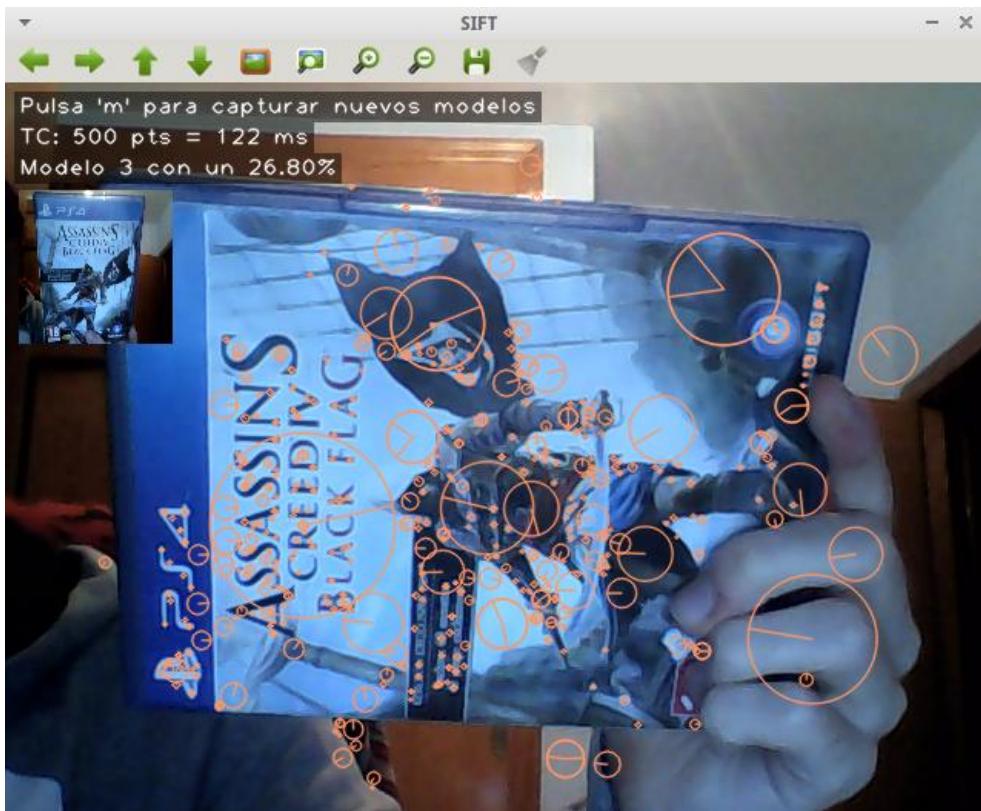
Cuando el objeto se inclina el porcentaje disminuye notablemente pero se sigue reconociendo el objeto con su clase adecuada.



También se reconocen los objetos cuando no se muestran completamente.



Además los objetos se pueden reconocer incluso en orientaciones diferentes a las que tenían cuando se capturaron como modelos. Por ejemplo volteados 180 grados.



O volteados solo 90 grados.

7. RECTIF

En este ejercicio se usarán nociones de geometría visual para rectificar la proyección obtenida en una imagen para medir el tamaño real que tienen los objetos de la imagen.

7.1 Código

```
8 # Parseo de argumentos para pasarlo por terminal
9 import argparse
10 parser = argparse.ArgumentParser()
11 parser.add_argument('--img', type=str, help='ruta hacia la imagen')
12 args = parser.parse_args()
13
14 # Función auxiliar para añadir puntos a las colas de las dos ventanas
15 def encolar(event, x, y, flags, queue):
16     if event == cv.EVENT_LBUTTONDOWN:
17         if queue == 0: pointsR.append((x,y))
18         elif queue == 1: pointsL.append((x,y))
19
20 # Definir la ventana para la imagen original
21 cv.namedWindow("Original")
22 cv.setMouseCallback("Original", encolar, 0)
23
24 # Leer la imagen pasada por parámetro
25 imagen = cv.imread(args.img, 1)
26
27 # Colas de los puntos
28 pointsR = deque(maxlen=4) # Rectángulo
29 pointsL = deque(maxlen=2) # Línea
30 hCard = 5.5 # Altura en cm de una tarjeta estándar
31 wCard = 8.5 # Ancho en cm de una tarjeta estándar
32 proportion = wCard/hCard # Proporción entre los lados
33 factor = 20 # Factor en 20 = 10(cm a mm)*2 pixeles
34 hReal = hCard * factor # Calcular la altura en pixeles con el factor
35 wReal = hReal * proportion # Calcular el ancho en pixeles con la proporción
36 x0 = 400. # Coordenadas de origen elegidas por prueba y error
37 y0 = 300. # Flag para detectar la primera pulsación de 'r'
```

Al comienzo del script se realiza el parseo de argumentos, en este caso para recibir la ruta hacia la imagen a rectificar. Tras esto se define la función para manejar las colas de puntos, tal y como se vio en otros ejercicios. Después se crea la ventana, se lee la imagen y se inicializan las variables auxiliares.

```

40 # Bucle infinito donde se mostrarán las ventanas hasta pulsar 'esc'
41 while (1):
42     # Copiar la imagen original
43     img = imagen.copy()
44
45     # Dibujar los puntos sobre la ventana original
46     for pf in pointsR:
47         cv.circle(img, pf, 2, (0,255,255), -1)
48
49     # Si hay cuatro puntos se dibuja el rectángulo
50     if len(pointsR) == 4:
51         pts = np.array([pointsR[0],pointsR[1],pointsR[2],pointsR[3]], np.int32)
52         pts = pts.reshape((-1,1,2))
53         cv.polyline(img,[pts],True,(0,255,255), 2)
54
55     # Si está seleccionado el rectángulo y se pulsa 'r'
56     if ((cv.waitKey(10) == ord('r')) & (len(pointsR) == 4)):
57         # Si el flag estaba desactivado se crea una nueva ventana
58         if flag == 0:
59             cv.namedWindow("Rectificación")
60             cv.setMouseCallback("Rectificación", encolar, 1)
61             flag = 1
62
63         # Crear las coordenadas sobre las que se colocará el rectángulo rectificado
64         real = np.array([[x0, y0], [x0+hReal, y0],
65                         [x0+hReal, y0+wReal], [x0, y0+wReal]])
66
67         # Obtener la homografía
68         H,_ = cv.findHomography(pts, real)
69
70         # Crear la rectificación
71         rectif = cv.warpPerspective(imagen.copy(),H,(650,650))

```

En un bucle infinito se mostrarán la imagen original y la rectificada. Primero se copia la imagen para poder eliminar los cambios producidos en iteraciones anteriores del bucle, tras esto se muestran los puntos y el rectángulo que selecciona el área de la tarjeta. Después se detecta la pulsación de la tecla *r* y si el rectángulo ya existe se realiza la rectificación. Para ello se crea una nueva ventana (solo la primera vez que se pulsa *r*), se eligen las coordenadas donde se representará la tarjeta (en base al factor que relaciona los píxeles con los milímetros), se obtiene la homografía y se genera la rectificación.

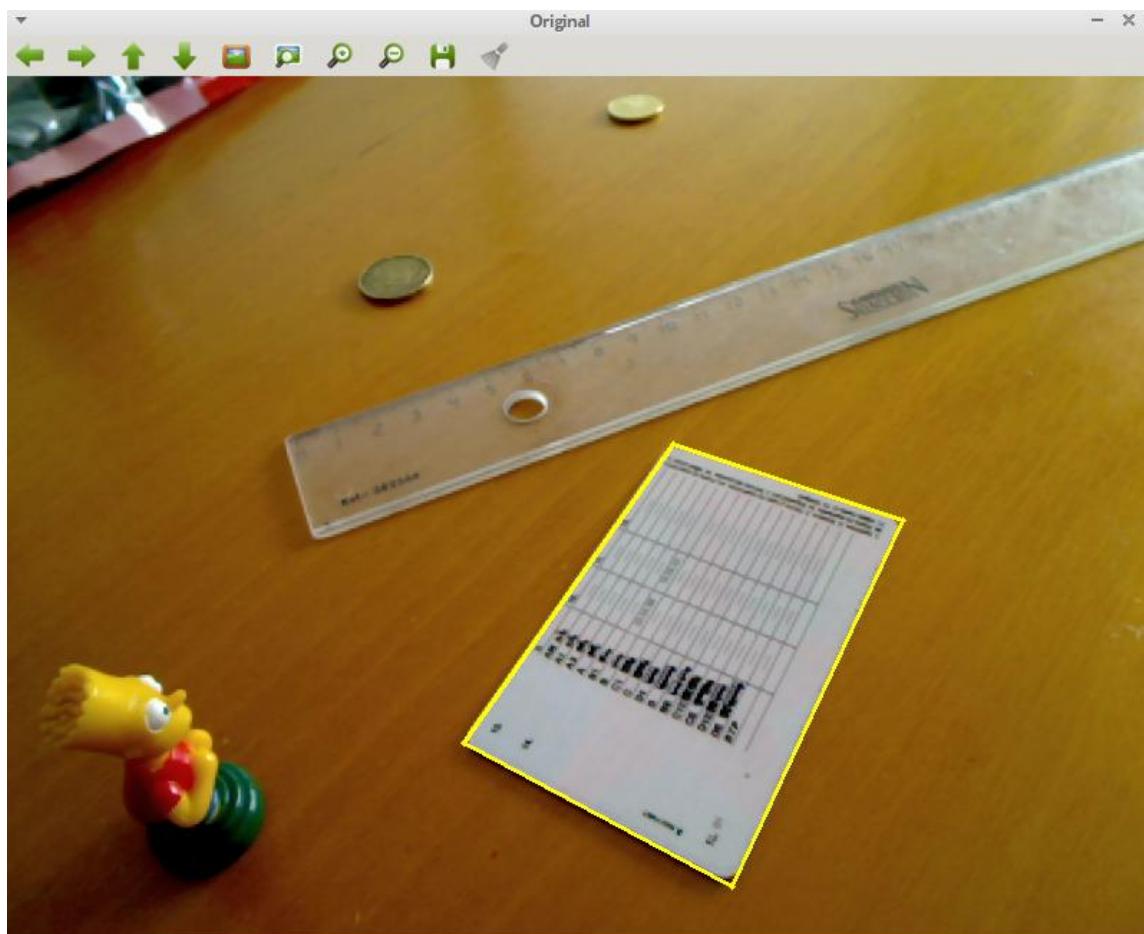
```

73     # Si el flag está activado
74     if flag == 1:
75         # Copiar la rectificación
76         rec = rectif.copy()
77
78         # Dibujar los puntos sobre la ventana de rectificación
79         for pf in pointsL:
80             cv.circle(rec, pf, 2, (255,0,0), -1)
81
82         # Si hay dos puntos se dibuja una línea entre ellos y se calculan los cm que ocupa
83         if len(pointsL) == 2:
84             cv.line(rec, pointsL[0], pointsL[1], (255,0,0))
85             c = np.mean(pointsL, axis = 0).astype(int)
86             pix = np.linalg.norm(np.array(pointsL[0]) - pointsL[1])
87             cm = pix/factor
88             putText(rec,f'{cm:.1f} cm',orig=c)
89
90         # Mostrar la imagen rectificada
91         cv.imshow('Rectificación',rec)
92
93         # Mostrar la imagen original
94         cv.imshow('Original',img)
95
96         # Si se pulsa 'esc' salir del bucle
97         if (cv.waitKey(10) == 27): break
98
99 # Salir del programa
100 cv.destroyAllWindows()

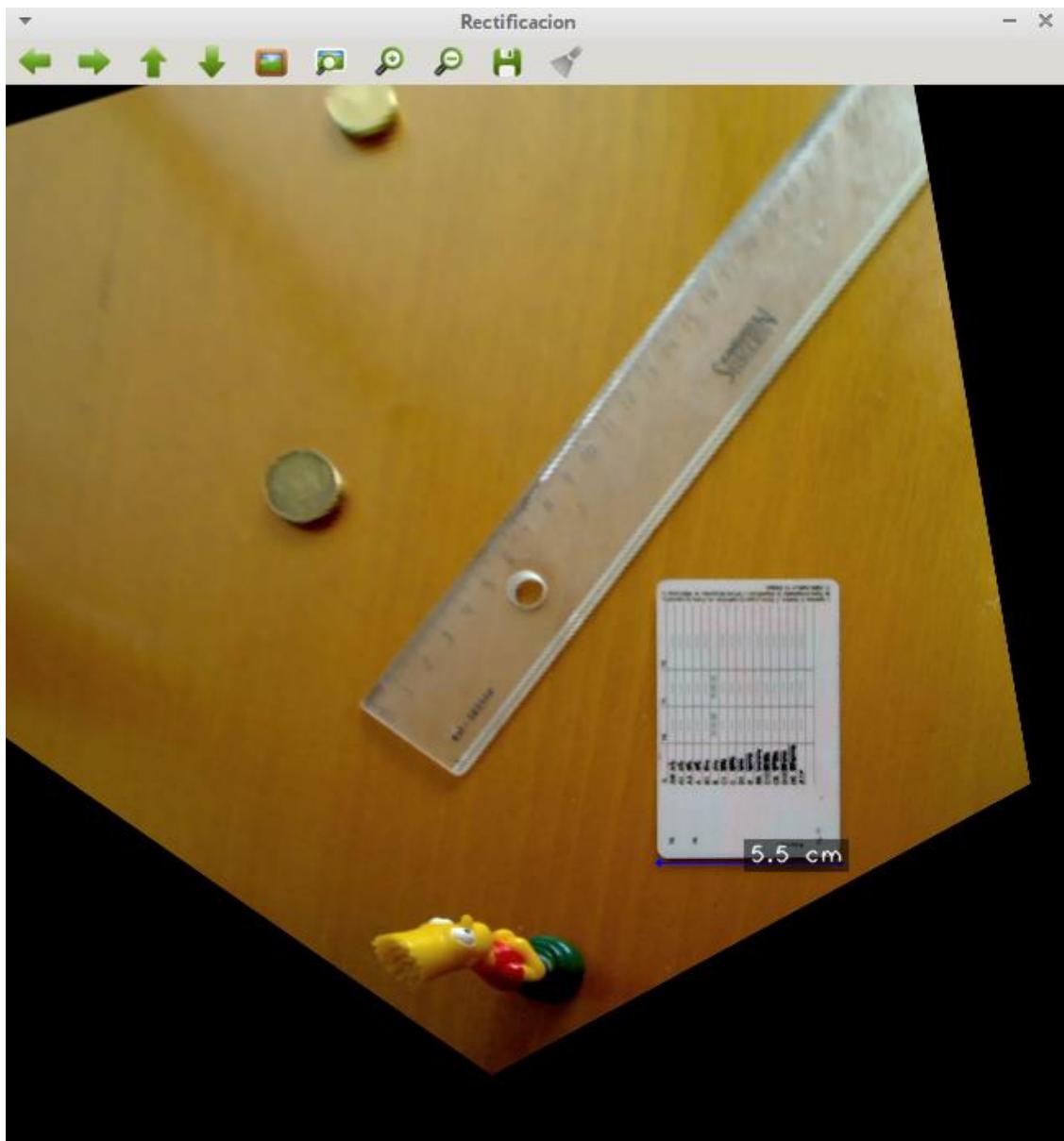
```

Cuando la ventana ha sido creada se repite el proceso anterior pero ahora para generar puntos en la nueva ventana, de este modo se dibuja una línea y se calcula su tamaño en función de los píxeles que mide y el factor dado. Por último se muestran ambas imágenes. Se puede salir del bucle gracias a la detección de la tecla *esc*, y el programa acaba.

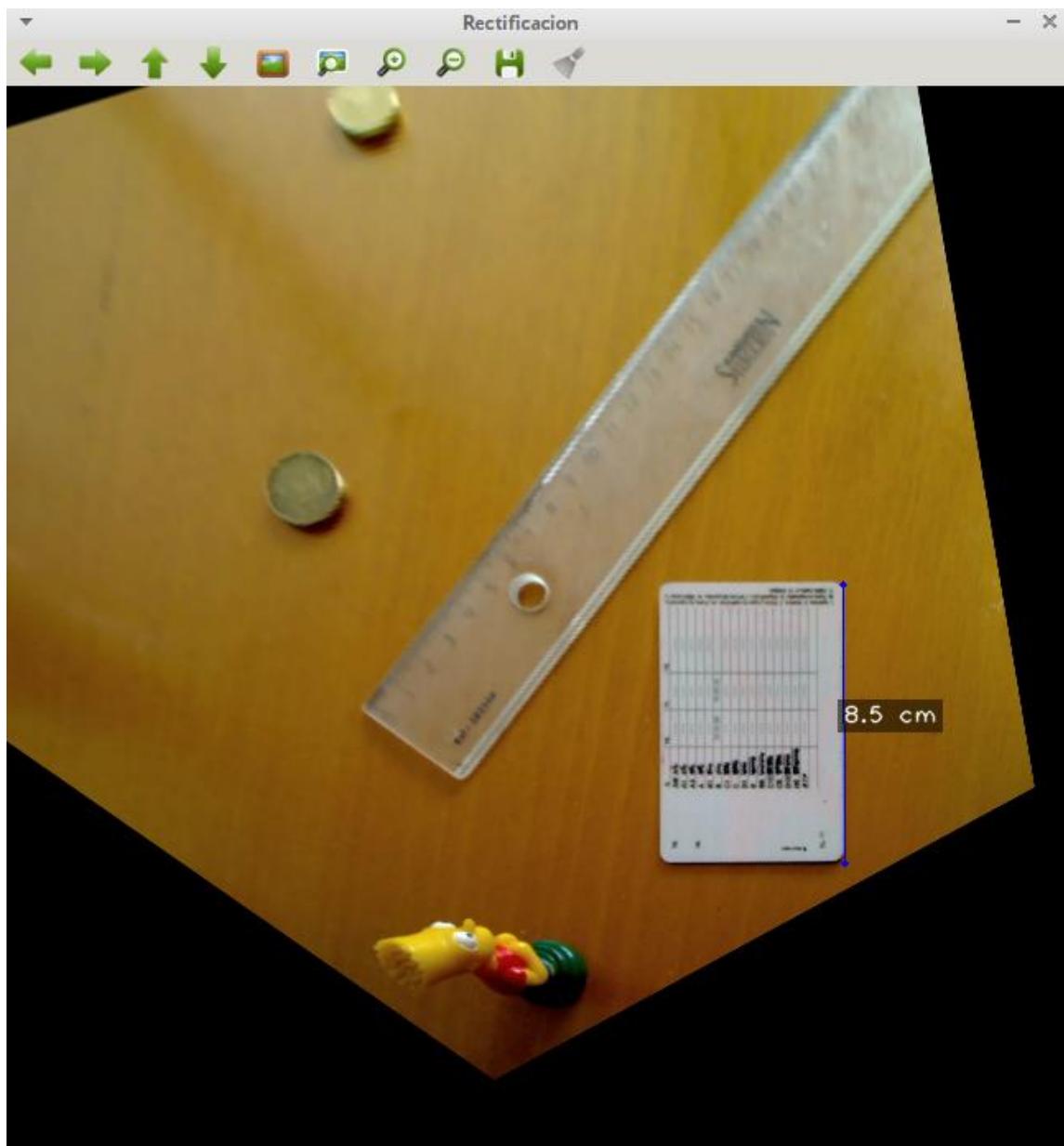
7.2 Ejemplos



En la primera imagen se muestra como se selecciona el área de la tarjeta manualmente.



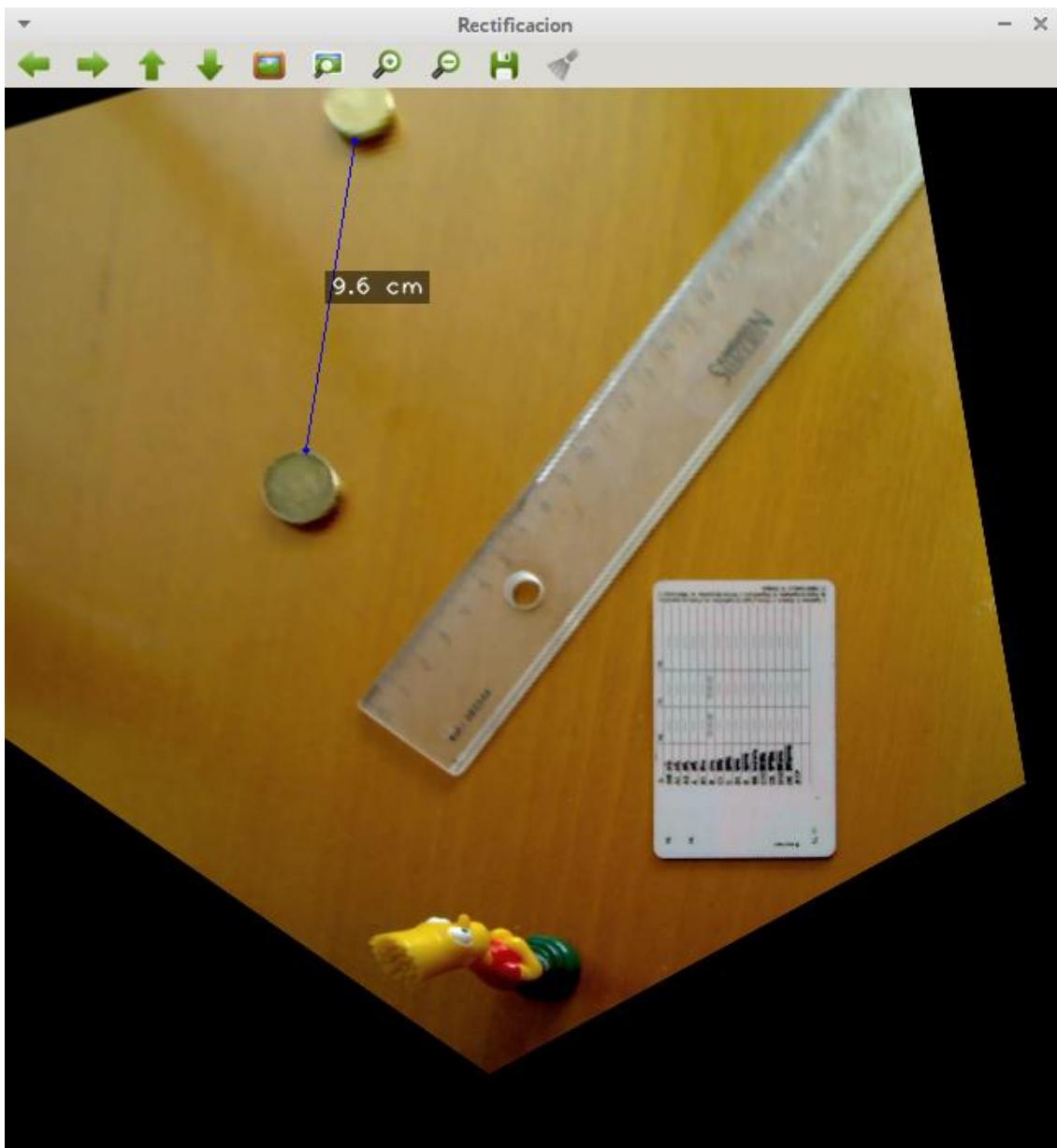
Comprobamos que realmente las medidas obtenidas son las que el objeto original produce. 5'5 cm de ancho.



Y 8'5 cm de largo.



Vemos esta comprobación en otro objeto externo a la tarjeta y que no se sitúa en el origen de la rectificación. La medida sobre la regla también es correcta ya que realmente existen 4 cm entre el 1 y el 5.



Por último se mide la distancia entre las monedas, obteniendo 9'6 cm. Esta medida puede no ser exacta en la realidad ya que las dimensiones se distorsionan cuanto más lejos de la tarjeta nos encontramos.

8. RA

En este ejercicio se representan objetos tridimensionales que pueden cambiar dinámicamente y que se ubican en tiempo real mediante la posición de unos marcadores establecidos.

8.1 Código

Para la representación del objeto sobre las referencias se ha usado el software proporcionado para la práctica, este proporciona funciones para detectar polígonos sobre las imágenes en tiempo real, obtener la matriz de calibración, etc. Así que no las comentaré por no haber sido implementadas por mí.

```
8 # Utilidades de extracción de contornos (disponibles en umucv)
9
10 # area, con signo positivo si el contorno se recorre "counterclockwise"
11 def orientation(x):
12     return cv.contourArea(x.astype(np.float32),oriented=True)
13
14 # ratio area/perímetro^2, normalizado para que 100 (el arg es %) = círculo
15 def redondez(c):
16     p = cv.arcLength(c.astype(np.float32),closed=True)
17     oa = orientation(c)
18     if p>0:
19         return oa, 100*4*np.pi*abs(oa)/p**2
20     else:
21         return 0,0
22
23 def boundingBox(c):
24     (x1, y1), (x2, y2) = c.min(0), c.max(0)
25     return (x1, y1), (x2, y2)
26
27 # comprobar que el contorno no se sale de la imagen
28 def internal(c,h,w):
29     (x1, y1), (x2, y2) = boundingBox(c)
30     return x1>1 and x2 < w-2 and y1 > 1 and y2 < h-2
31
32 # reducción de nodos
33 def redu(c,eps=0.5):
34     red = cv.approxPolyDP(c,eps,True)
35     return red.reshape(-1,2)
36
37 # intenta detectar polígonos de n lados
38 def polygons(cs,n,prec=2):
39     rs = [ redu(c,prec) for c in cs ]
40     return [ r for r in rs if r.shape[0] == n ]
```

```

42 # detecta siluetas oscuras que no sean muy pequeñas ni demasiado alargadas
43 def extractContours(g, minarea=10, minredon=25, reduprec=1):
44     ret, gt = cv.threshold(g,189,255, cv.THRESH_BINARY+cv.THRESH_OTSU)
45
46     contours = cv.findContours(gt, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)[-2]
47
48     h,w = g.shape
49
50     tharea = (min(h,w)*minarea/100.)**2
51
52     def good(c):
53         oa,r = redondez(c)
54         black = oa > 0 # and positive orientation
55         return black and abs(oa) >= tharea and r > minredon
56
57     ok = [redu(c.reshape(-1,2),reduprec) for c in contours if good(c)]
58     return [c for c in ok if internal(c,h,w) ]
59
60 # añadimos una coordenada cero a todas las filas
61 # para convertir un polígono del plano en un polígono
62 # en el espacio, a altura z=0
63 def addzerocol(x):
64     return np.hstack([x,np.zeros([len(x),1])])
65
66 # matriz de calibración sencilla dada la
67 # resolución de la imagen y el fov horizontal en grados
68 def Kfov(sz,hfov):
69     hfov = np.radians(hfov)
70     f = 1/np.tan(hfov/2)
71     # print(f)
72     w,h = sz
73     w2 = w / 2
74     h2 = h / 2
75     return np.array([[f*w2, 0,      w2],
76                     [0,      f*w2, h2],
77
78
79 # convierte un conjunto de puntos ordinarios (almacenados como filas de la matriz de entrada)
80 # en coordenadas homogéneas (añadimos una columna de 1)
81 def homog(x):
82     ax = np.array(x)
83     uc = np.ones(ax.shape[:-1]+(1,))
84     return np.append(ax,uc, axis=-1)
85
86 # convierte en coordenadas tradicionales
87 def inhomog(x):
88     ax = np.array(x)
89     return ax[...,-1] / ax[...,-1]
90
91 # aplica una transformación homogénea h a un conjunto
92 # de puntos ordinarios, almacenados como filas
93 def htrans(h,x):
94     return inhomog(homog(x) @ h.T)
95
96 # juntar columnas
97 def jc(*args):
98     return np.hstack(args)
99
100 # mide el error de una transformación (p.ej. una cámara)
101 # rms = root mean squared error
102 # "reprojection error"
103 def rmsreproj(view, model, transf):
104     err = view - htrans(transf, model)
105     return np.sqrt(np.mean(err.flatten()**2))
106
107 def pose(K, image, model):
108     ok,rvec,tvec = cv.solvePnP(model, image, K, (0,0,0,0))
109     if not ok:
110         return 1e6, None
111     R,_ = cv.Rodrigues(rvec)
112     M = K @ jc(R,tvec)
113     rms = rmsreproj(image,model,M)

```

```

116 def rrots(c):
117     return [np.roll(c,k,0) for k in range(len(c))]
118
119 # probamos todas las asociaciones de puntos imagen con modelo
120 # y nos quedamos con la que produzca menos error
121 def bestPose(K,view,model):
122     poses = [ pose(K, v.astype(float), model) for v in rrots(view) ]
123     return sorted(poses,key=lambda p: p[0])[0]
124
125 # Definir la forma del marcador de referencia
126 ref = (np.array(
127     [[0, 0],
128      [0, 1],
129      [0.5, 1],
130      [0.5, 0.5],
131      [1, 0.5],
132      [1, 0],
133      [0.5, 0]]))
134
135 ref3d = addzerocol(ref)
136 marker = ref3d[:-1]
137
138 # Obtener la matriz de calibración
139 K = Kfov((640,480), 64)

```

En primer lugar se define la estructura del objeto de referencia, que en este caso será una L formada por tres cuadrados. Obtenemos el marcador a partir de esta referencia y obtenemos la matriz de calibración.

```

140
141 # Objeto tridimensional a mostrar
142 cube = np.array([
143     [0,0,0],
144     [1,0,0],
145     [1,1,0],
146     [0,1,0],
147     [0,0,0],
148
149     [0,0,1],
150     [1,0,1],
151     [1,1,1],
152     [0,1,1],
153     [0,0,1],
154
155     [1,0,1],
156     [1,0,0],
157     [1,1,0],
158     [1,1,1],
159     [0,1,1],
160     [0,1,0],
161 ])
162
163 # Definir la ventana para el cálculo del FOV
164 cv.namedWindow("RA")
165
166 # Flags e inicialización de los valores de las características del objeto tridimensional
167 flag_color = 0
168 color = (255,0,0)
169 grosor = 1
170 tam = 1.

```

Creamos un objeto tridimensional para representar sobre las coordenadas del marcador, en este caso un cubo geométrico.

```

172 # Bucle principal de entrada de vídeo
173 for key, frame in autoStream():
174     # Copiar el frame antes de modificarlo
175     img = frame.copy()
176
177     # Obtener los contornos interesantes
178     g = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
179     cnts = extractContours(g, reduprec=3)
180     good = polygons(cnts, 6)
181
182     # Si se pulsa la tecla 'c'
183     if (key == ord('c')):
184         # Actualizar el color en función del flag
185         if flag_color == 0:
186             color = (0,255,0)
187             flag_color = 1
188         elif flag_color == 1:
189             color = (0,0,255)
190             flag_color = 2
191         elif flag_color == 2:
192             color = (255,0,0)
193             flag_color = 0
194
195     # Si se pulsa la tecla 'g' aumentar el grosor
196     if (key == ord('g')):
197         if grosor <= 5: grosor = grosor+1
198
199     # Si se pulsa la tecla 'f' disminuir el grosor
200     if (key == ord('f')):
201         if grosor > 2: grosor = grosor-1
202
203     # Si se pulsa la tecla 'g' aumentar el tamaño
204     if (key == ord('b')):
205         if tam < 1.6: tam = tam+0.1

```

Entramos en el bucle de entrada de imágenes desde la webcam. Primero se copia el frame actual antes de tratarlo, tras esto se generan los contornos de los polígonos encontrados que tienen 6 lados. El siguiente fragmento de código se encarga de la detección de las teclas que configurarán las características del objeto. Cuando se pulsa *c* se cambia el color en función del flag, cuando se pulsa *g* se incrementa el grosor, con *f* se decrementa, lo mismo ocurre para el tamaño, con *b* se aumenta y con *p* se disminuye.

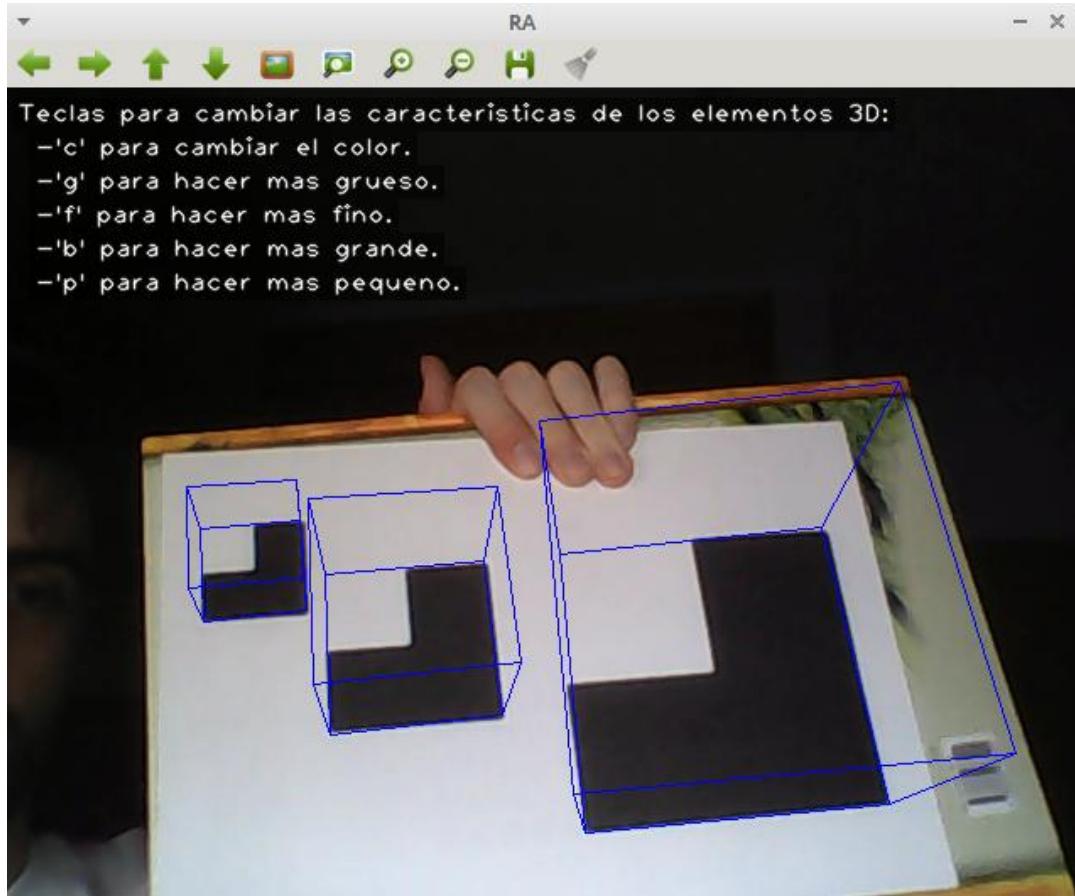
```

207 # Si se pulsa la tecla 'g' disminuir el tamaño
208 if (key == ord('p')):
209     if tam > 0.2: tam = tam-0.1
210
211 # Para cada polígono seleccionado mostrar el objeto si el error es menor que 2
212 for g in good:
213     err,Me = bestPose(K,g,marker)
214     if err < 2:
215         pts = htrans(Me,cube*tam)
216         cv.polyline(img,np.int32([pts]),True,color,grosor)
217
218 # Mostrar la información en pantalla
219 putText(img,f'Teclas para cambiar las características de los elementos 3D:',orig=(10,20))
220 putText(img,f'-\'c\' para cambiar el color.',orig=(20,40))
221 putText(img,f'-\'g\' para hacer mas grueso.',orig=(20,60))
222 putText(img,f'-\'f\' para hacer mas fino.',orig=(20,80))
223 putText(img,f'-\'b\' para hacer mas grande.',orig=(20,100))
224 putText(img,f'-\'p\' para hacer mas pequeno.',orig=(20,120))
225
226 # Mostrar la imagen resultante
227 cv.imshow('RA',img)
228
229 # Salir del programa
230 cv.destroyAllWindows()

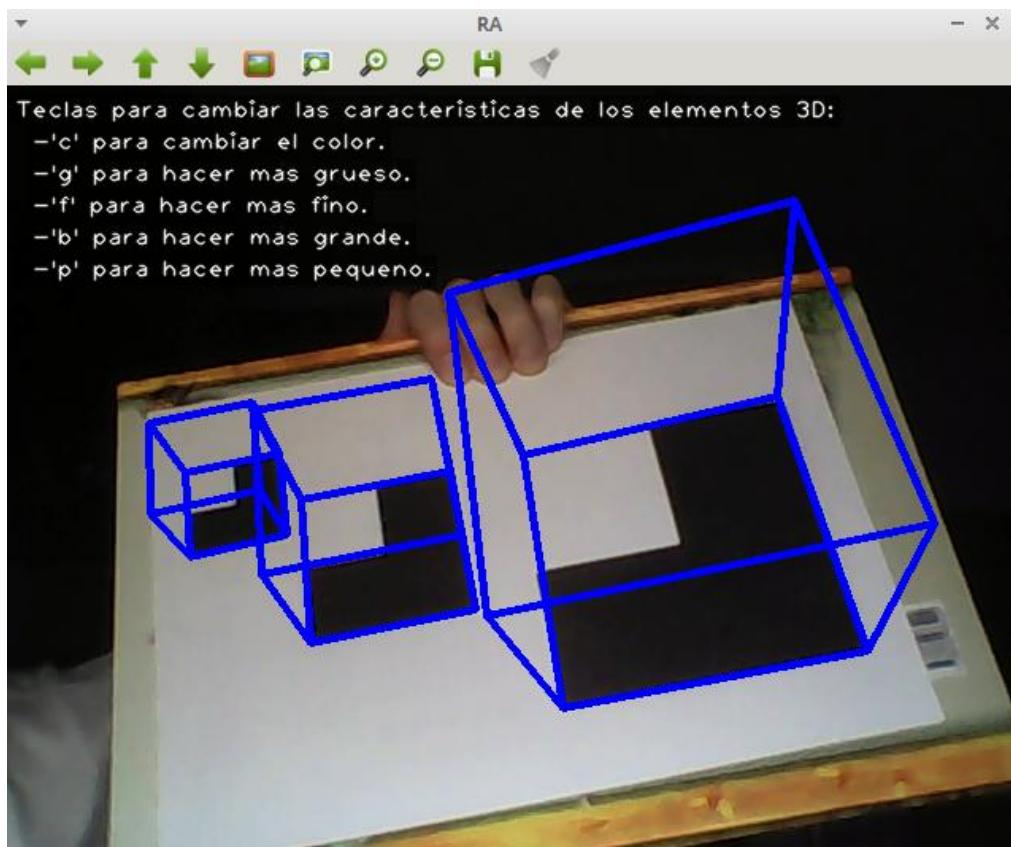
```

Por último se recorren los polígonos encontrados, escogemos los buenos y mostramos el objeto. Tras esto mostramos la información por pantalla e imprimimos la imagen resultante.

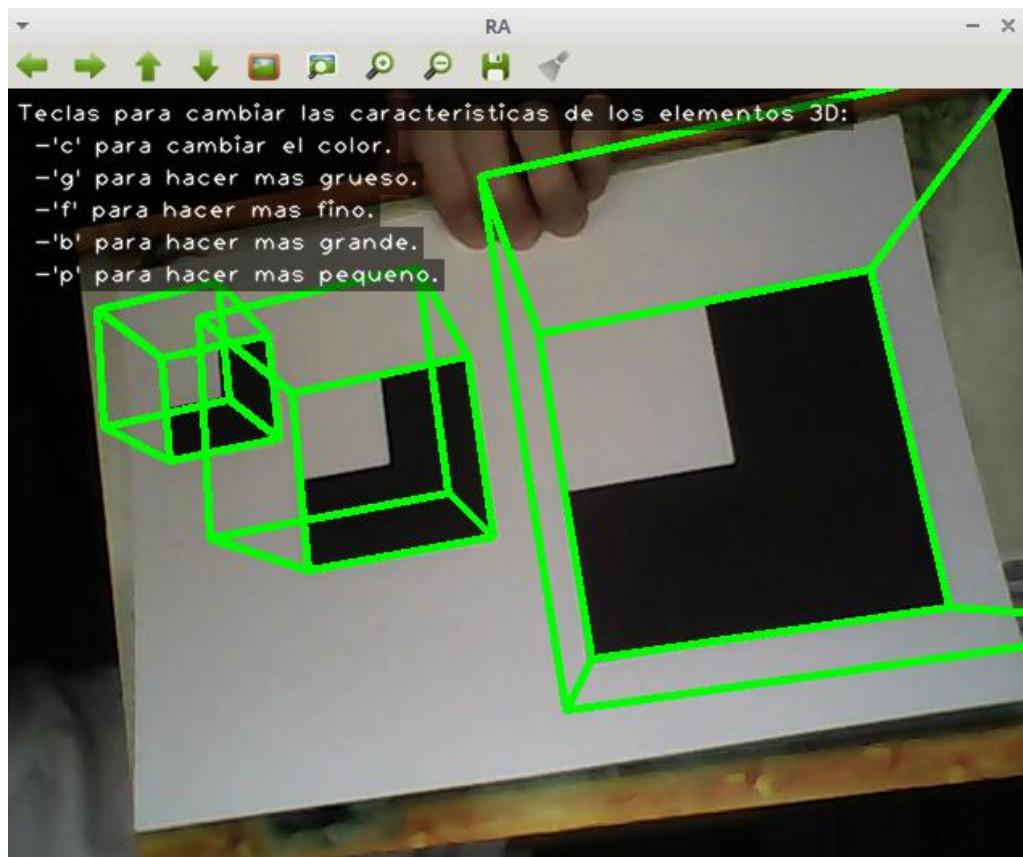
8.2 Ejemplos



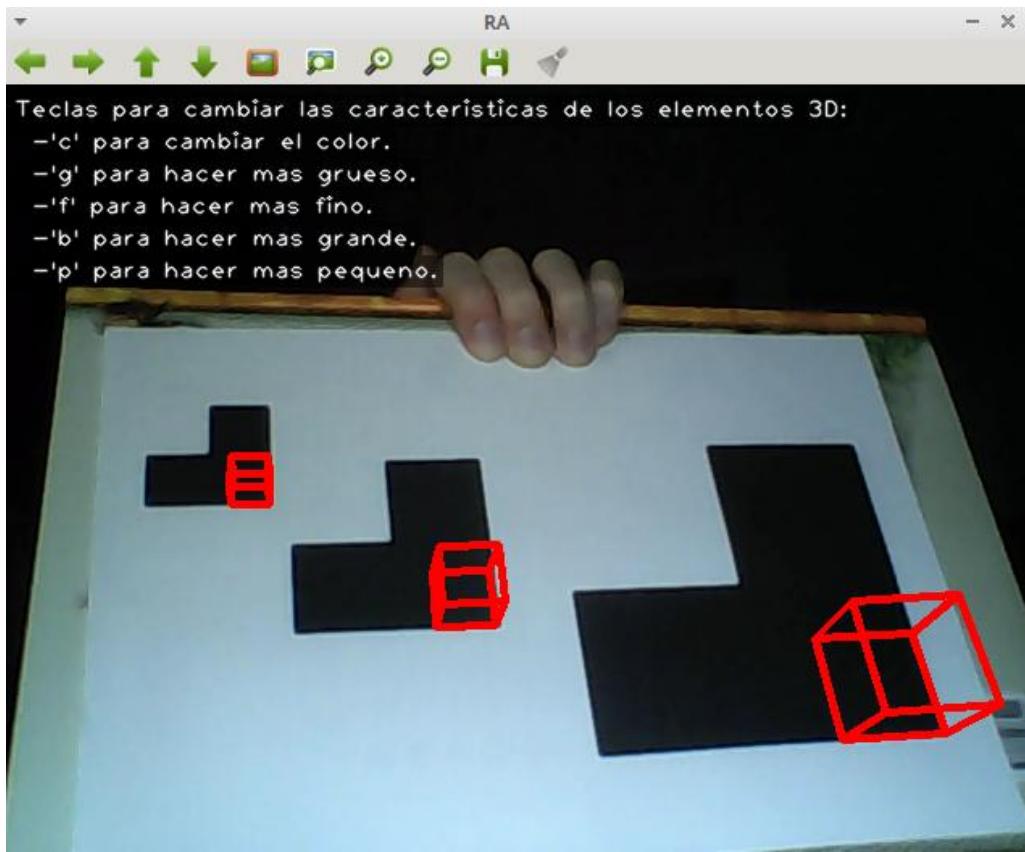
Situación inicial sin alterar los parámetros.



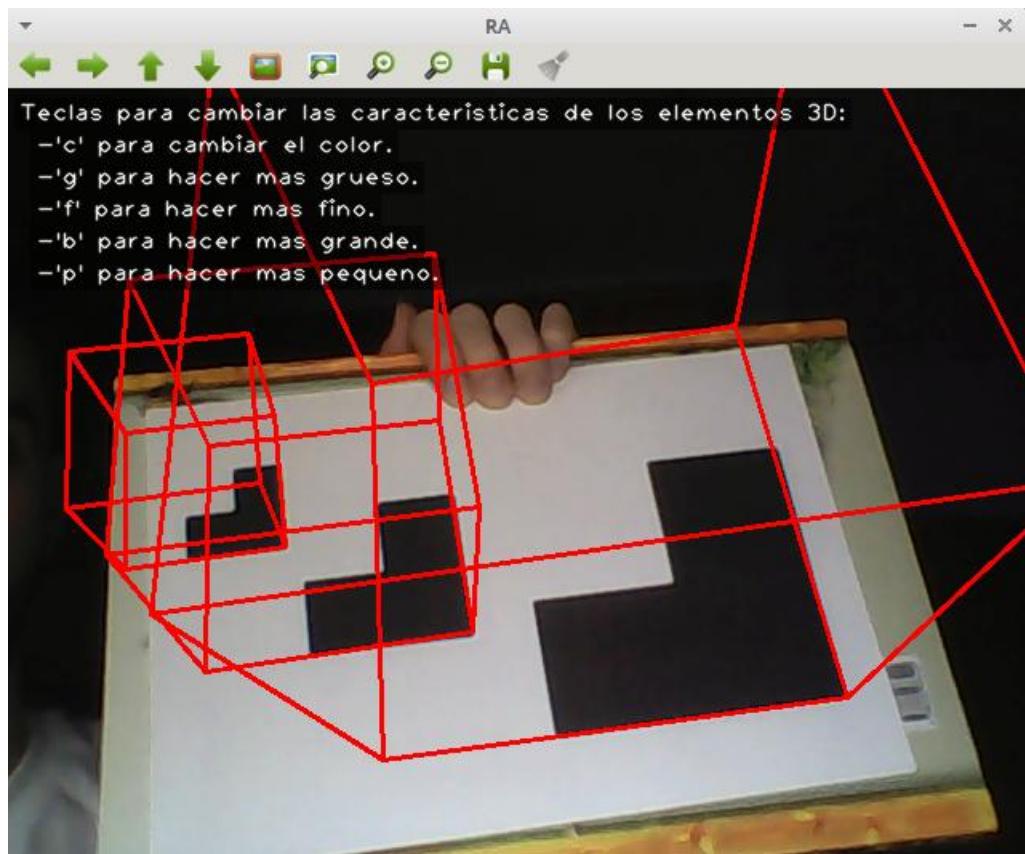
Efecto de pulsar la tecla *g*.



Efecto de pulsar la tecla *c*.



Situación tras pulsar una vez *c* y repetidas veces *p*.



Situación final tras pulsar *f* y repetidas veces *b*.

9. SROI

Este ejercicio es más bien un ejercicio auxiliar para otros ejercicios más complejos. En él se implementa la funcionalidad de seleccionar y almacenar diferentes regiones de interés (ROIs)

9.1 Código

```
# Crear la ventana y añadir una región de interés a ella
cv.namedWindow('SROI')
roi = ROI('SROI')

# Variables auxiliares
h = w = 50      # Alto y ancho de las rois a mostrar
rois = []        # Lista de las regiones de interés
first = True     # Booleano para detectar la primera roi que se registre

# Bucle principal de entrada de video
for key, frame in autoStream():

    # Si se ha seleccionado una región
    if roi.roi:
        # Obtener sus coordenadas
        [x1,y1,x2,y2] = roi.roi

        # Recortar esa sección de la imagen
        region = frame[y1:y2+1, x1:x2+1]

        # Si se pulsa la tecla 'r'
        if key == ord('r'):
            # Copiar la roi
            trozo = region.copy()

            # Redimensionar a la altura y ancho definida
            trozo = cv.resize(trozo,(w,h))
            if first: # Si es la primera roi se añade a la lista
                rois.append(trozo)
                roisImg = trozo
                first = False
            else: # Si no es la primera se crea una imagen que combina cada roi
                rois.append(trozo)
                roisImg = np.hstack((roisImg, trozo))
```

En primer lugar se crea la ventana del programa y se inicializan otros parámetros como el tamaño de los iconos de las ROIs, la lista de regiones y un flag para encontrar la primer ROI que se registre. A continuación comienza el bucle principal, si se selecciona una región y se pulsa la tecla *r*, la sección elegida se redimensiona para mostrarla al usuario y se añade a la lista de regiones. Si no es la primera región habrá, además, que combinar las diferentes imágenes de las regiones para obtener un combinado que se mostrará al usuario.

```

# Eliminar la roi cuando se registra
roi.roi = []

# Dibujar el rectangulo de la roi
cv.rectangle(frame, (x1,y1), (x2,y2), color=(0,255,255), thickness=2)

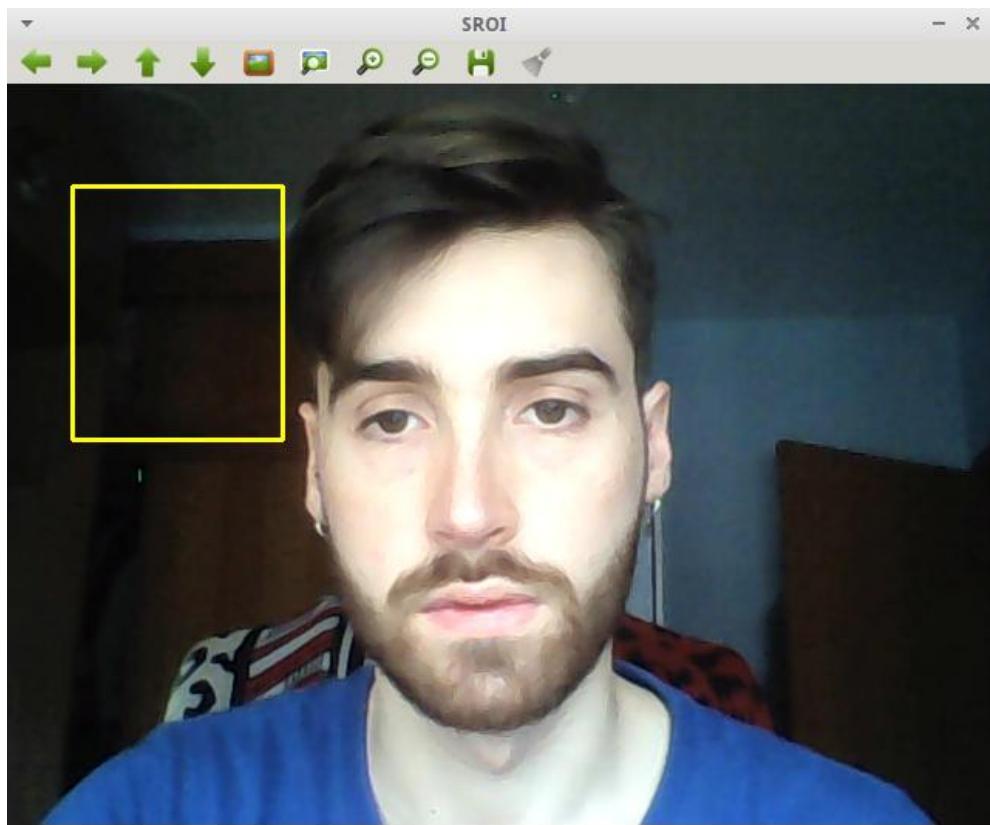
# Cuando hay alguna roi registrada se muestra la ventana con las rois registradas
if not first:
    cv.namedWindow('Rois')
    cv.imshow('Rois', roisImg)
# Mostrar la imagen original
cv.imshow('SROI',frame)

# Salir del programa
cv.destroyAllWindows()

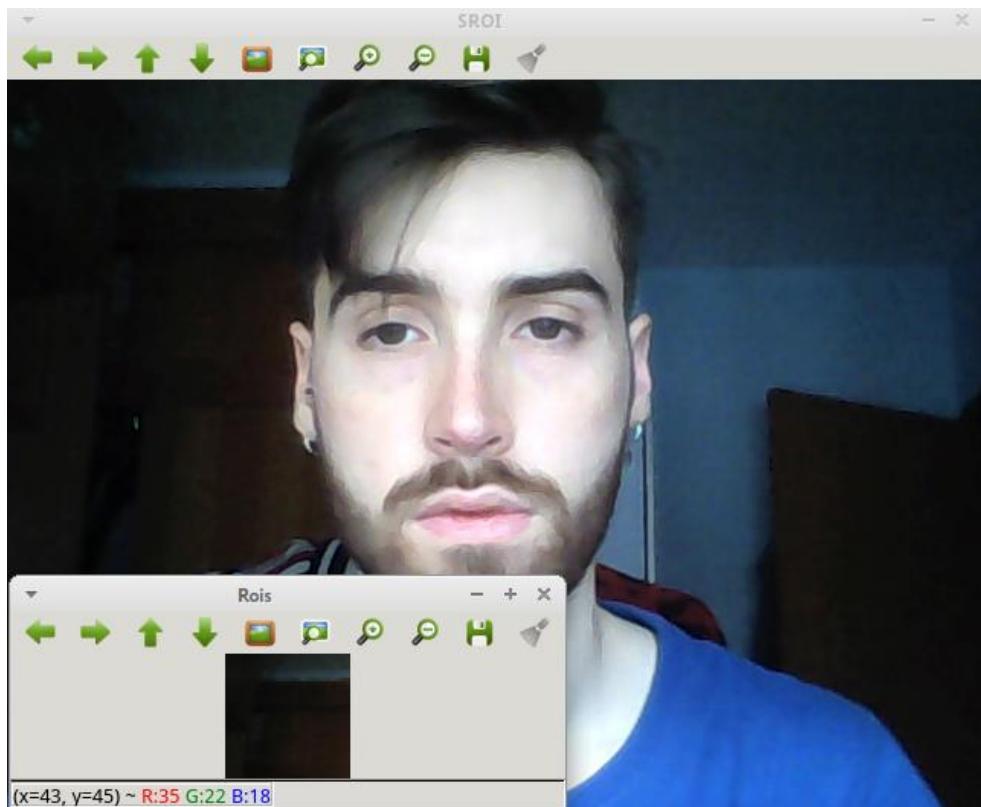
```

Tras registrarse se elimina la ROI de la selección. Hasta que se registra la ROI esta permanece seleccionada y se indica con un rectángulo amarillo. Cuando existe alguna ROI seleccionada se muestra otra ventana que muestra todas las regiones escogidas. Por último al salir del bucle se cierran las ventanas.

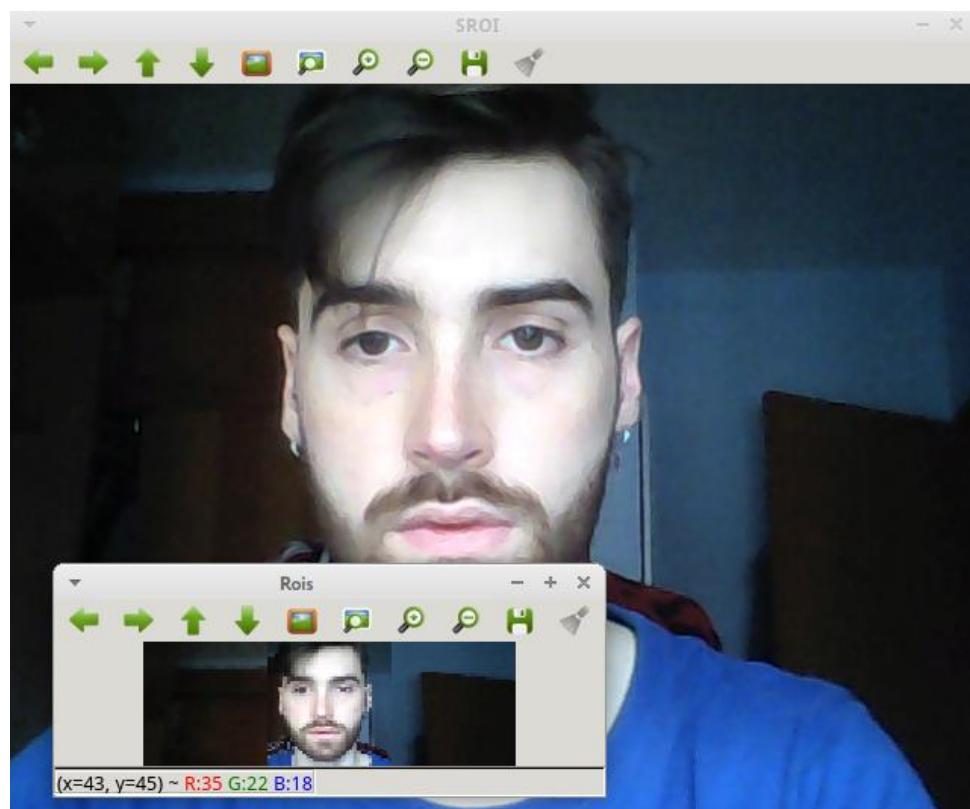
9.2 Ejemplos



En esta primera imagen se muestra el estado del programa al seleccionar con el ratón una región de la imagen.



Se registra esta primera ROI.



En esta imagen final se muestra el estado del programa tras varias capturas de regiones.

10. MSAT

En este ejercicio se pretende trabajar con los diferentes canales de la imagen en el espacio de color HSV para modificar diferentes características de la imagen en tiempo de ejecución, para ello me he apoyado en la ayuda de trackbars.

10.1 Código

```
# Crear ventana
cv.namedWindow("MSAT", cv.WINDOW_NORMAL)

# Definir una función para no hacer nada
def nothing(x): pass

# Crear los trackbars para controlar las características del espacio de color
cv.createTrackbar("h", "MSAT", 0, 179, nothing)
cv.createTrackbar("s", "MSAT", 0, 255, nothing)
cv.createTrackbar("v", "MSAT", 0, 255, nothing)

# Bucle principal de entrada de video
for key, frame in autoStream():
    # Transformar el espacio BGR en HSV que facilita el acceso a ciertas características
    hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)

    # Obtener la posición de los trackbars
    h = cv.getTrackbarPos('h','MSAT')
    s = cv.getTrackbarPos('s','MSAT')
    v = cv.getTrackbarPos('v','MSAT')

    # Modificar las características de la imagen con el trackbar
    hsv[:, :, 0] = hsv[:, :, 0]+h # Matiz
    hsv[:, :, 1] = hsv[:, :, 1]+s # Saturación
    hsv[:, :, 2] = hsv[:, :, 2]+v # Valor (Luminosidad)

    # Convertir el espacio a BGR de nuevo para mostrarlo
    out = cv.cvtColor(hsv, cv.COLOR_HSV2BGR)
    cv.imshow('MSAT',out)

# Al salir del bucle cerrar las ventanas
cv.destroyAllWindows()
```

En primer lugar se crea la ventana del programa y los diferentes trackbars que nos ayudarán a modificar los valores de los canales de forma cómoda. En el bucle principal se convierte el espacio de color a HSV, tras esto, se obtiene el valor actual del trackbar y se modifica cada canal de la imagen con este valor. Se puede observar que este valor simplemente se suma al que ya había en cada pixel del canal, cabe preguntarse qué ocurrirá cuando esta suma supere el valor máximo. Bueno, realmente esta comprobación se produce de manera implícita por lo que no obtendremos ningún error. La forma en la que este error se maneja es tratando el rango de valores como una cola circular, por lo que al superar el valor máximo se vuelve al valor inicial. En el programa el resultado no es del todo favorable al crear discrepancias entre regiones, lo ideal sería que al llegar al valor máximo se quedase en este, se acotasen los valores que lo superan, dando una especie de saturación a la imagen final. He intentado tratar cada pixel por separado para producir este efecto pero el rendimiento del programa caía estrepitosamente, por lo que he optado por dejarlo como se muestra actualmente, aun que el resultado final no sea perfecto.

10.2 Ejemplos

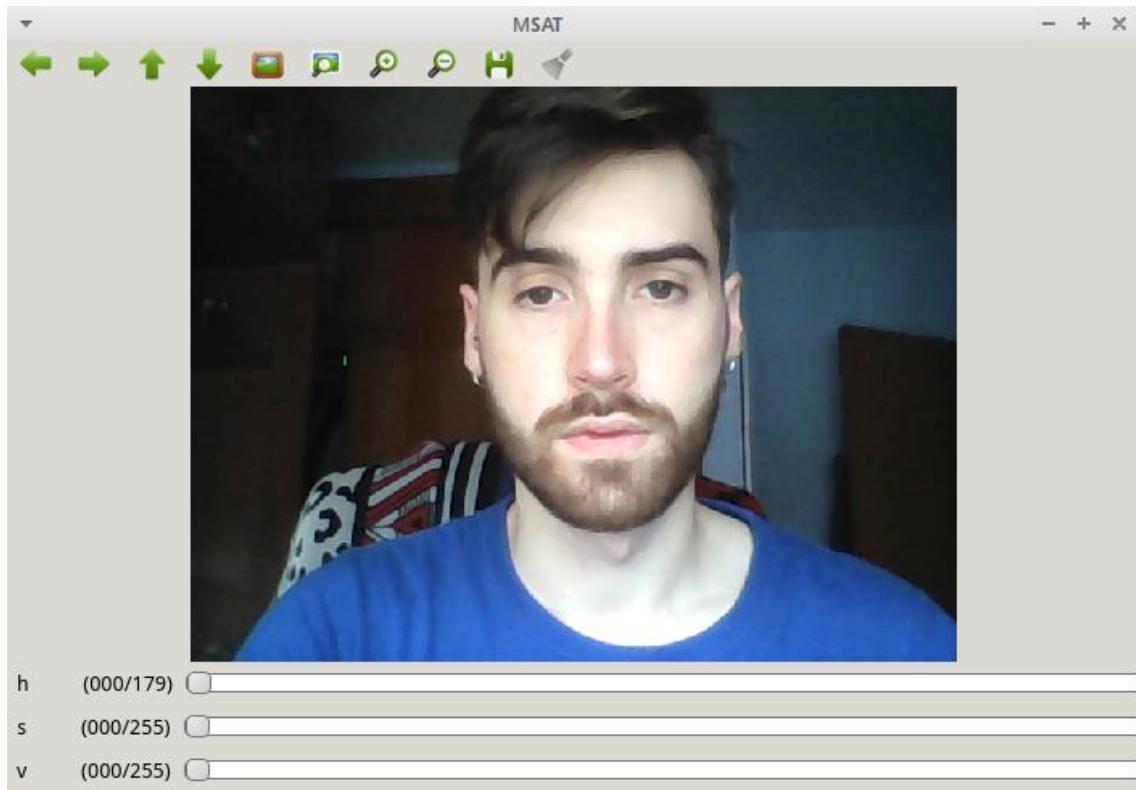
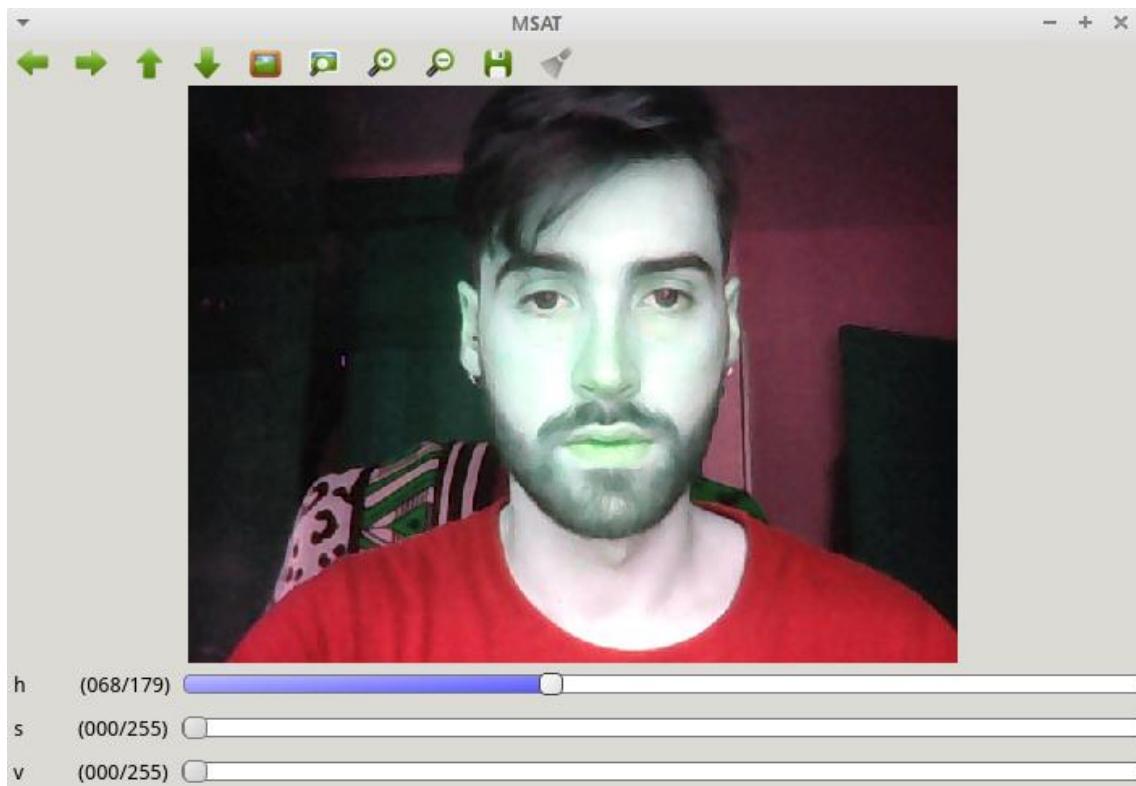


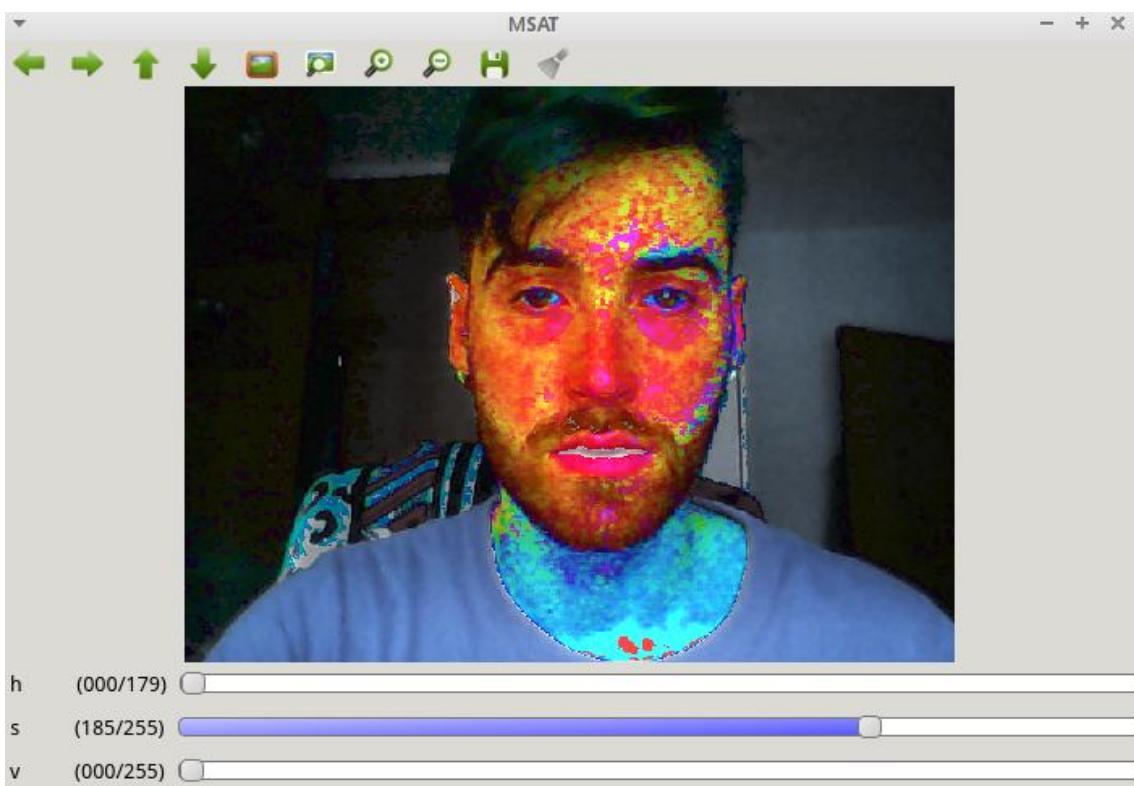
Imagen inicial con los trackbars a 0.



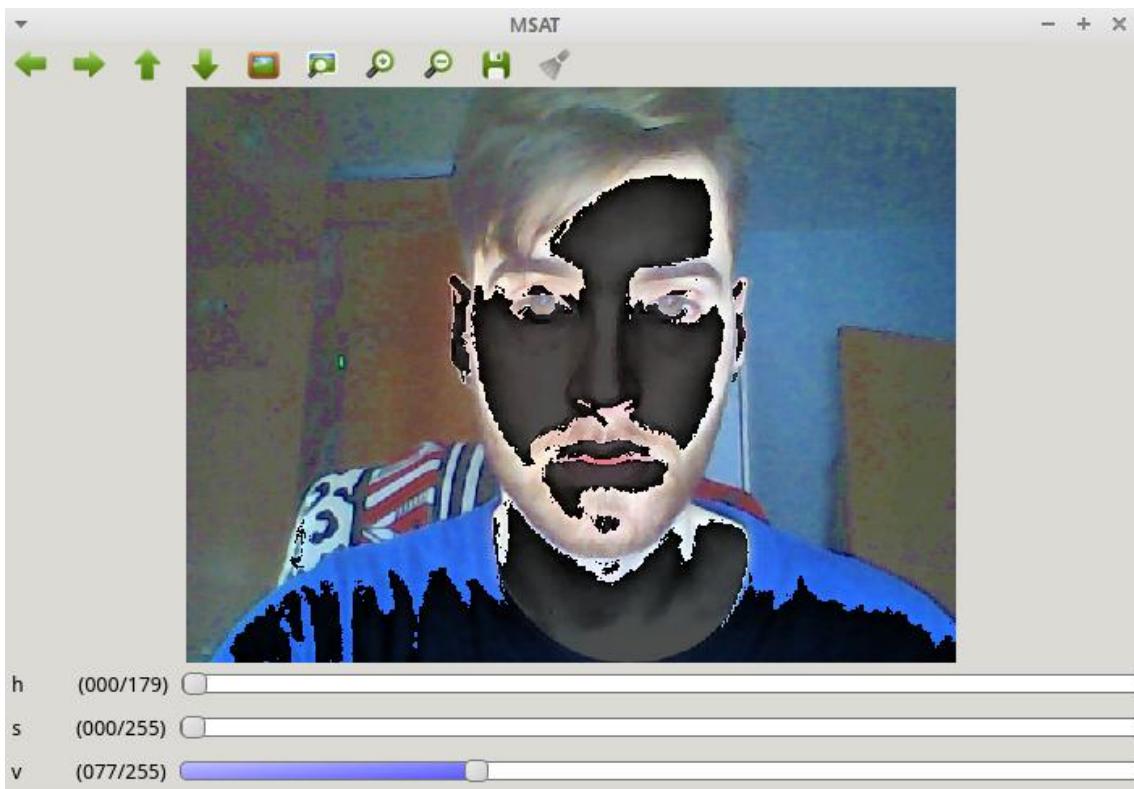
Efecto de incrementar el canal H (Matiz).



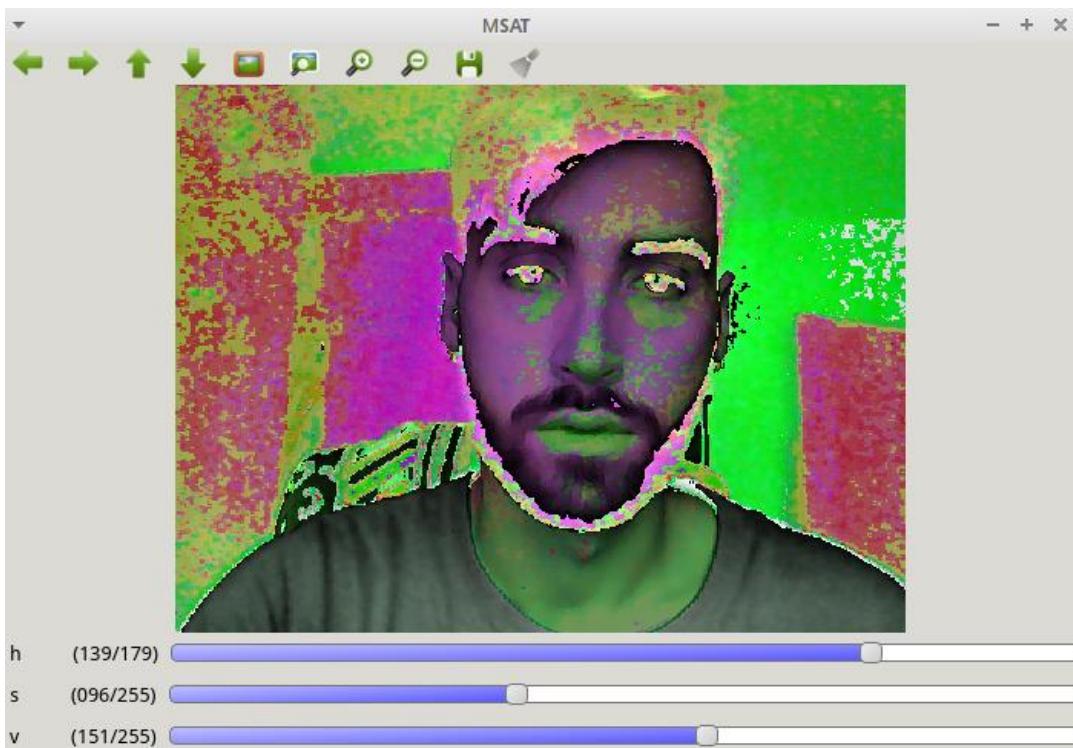
Efecto de seguir incrementando el valor de H.



Efecto de incrementar el valor del canal S (Saturación).



Efecto de incrementar el valor de V (Luminosidad), en este caso podemos apreciar claramente el problema anteriormente mencionado, las zonas blancas de la imagen ya han llegado al máximo y han empezado de nuevo, haciéndose mucho más oscuras, mientras que el resto de la imagen aumenta su brillo de forma natural.



Por último vemos el efecto de combinar un incremento de los diferentes canales.

11. FIL

Este ejercicio trata de aplicar diferentes filtros o efectos a zonas seleccionadas previamente de la imagen.

11.1 Código

```
# Crear la ventana y añadir una región de interés a ella
cv.namedWindow('FIL')
roi = ROI('FIL')

# Definir función para no hacer nada
def nothing(x): pass

# Crear los trackbars para elegir los valores de los filtros
cv.createTrackbar("Gaussian Blur", "FIL", 1, 30, nothing)
cv.createTrackbar("Laplace", "FIL", 0, 50, nothing)
cv.createTrackbar("Threshold", "FIL", 0, 255, nothing)

# Flags para activar los filtros
gaussian = False
laplace = False
threshold = False

# Bucle principal de entrada de video
for key, frame in autoStream():

    # Si se ha seleccionado una región
    if roi.roi:
        # Obtener sus coordenadas
        [x1,y1,x2,y2] = roi.roi

        # Recortar esa sección de la imagen
        region = frame[y1:y2+1, x1:x2+1]

        # Si se pulsa la tecla 'g' activar el filtro gaussiano
        if key == ord('g'):
            gaussian = True
            laplace = False
            threshold = False
```

Para empezar se crea la ventana y los trackbars que nos ayudarán a potenciar el efecto de los filtros implementados, en mi caso un filtro de suavizado gaussiano, un filtro laplaciano y un filtro de umbralización. Durante la ejecución del bucle principal se detecta la selección de una región de interés (sobre la que se aplicará el filtro). A continuación se aplica un filtro u otro en función del flag activo. Si se pulsa la tecla *g* se activa el flag para el filtro de suavizado gaussiano.

```

# Si se pulsa la tecla 'l' activar el filtro laplaciano
if key == ord('l'):
    laplace = True
    gaussian = False
    threshold = False

# Si se pulsa la tecla 't' activar el filtro de threshold
if key == ord('t'):
    threshold = True
    gaussian = False
    laplace = False

# Si se ha activado el filtro gaussiano
if gaussian:
    # Obtener el valor del trackbar
    g = cv.getTrackbarPos('Gaussian Blur','FIL')

    # Asegurar que g es mayor que cero e impar
    if (g == 0): g = g+1
    elif (g % 2 == 0): g = g-1

    # Aplicar el suavizado
    region = cv.GaussianBlur(region,(g,g),cv.BORDER_DEFAULT)

    # Actualizar la región
    frame[y1:y2+1, x1:x2+1] = region

```

Si se pulsa la tecla *l* se activa el flag para el filtro laplaciano y si se pulsa *t* se activa el flag para la umbralización. Después de esto se comprueba el flag activo (solo habrá uno activo en cada momento como máximo). En caso de estar activado el flag gaussiano se obtiene, primero, el valor de su trackbar. Después se trata para que cumpla los requerimientos de la función de OpenCV, que necesita un valor superior a 0 e impar, ya que si el valor es 0 usará otro por defecto que no queremos. Por último se aplica el filtro a la ROI y se actualiza en el frame actual.

```

# Si se ha activado el filtro laplaciano
if laplace:
    # Obtener el valor del trackbar
    l = cv.getTrackbarPos('Laplace','FIL')

    # Aplicar el ruido laplaciano
    region = cv.Laplacian(region,-1,scale=l)

    # Actualizar la región
    frame[y1:y2+1, x1:x2+1] = region

# Si se ha activado el filtro de threshold
if threshold:
    # Obtener el valor del trackbar
    t = cv.getTrackbarPos('Threshold','FIL')

    # Aplicar el umbralizado
    _,region = cv.threshold(region,t,255,cv.THRESH_BINARY)

    # Actualizar la región
    frame[y1:y2+1, x1:x2+1] = region

    # Dibujar el rectángulo de la roi
    cv.rectangle(frame, (x1,y1), (x2,y2), color=(0,255,255), thickness=2)

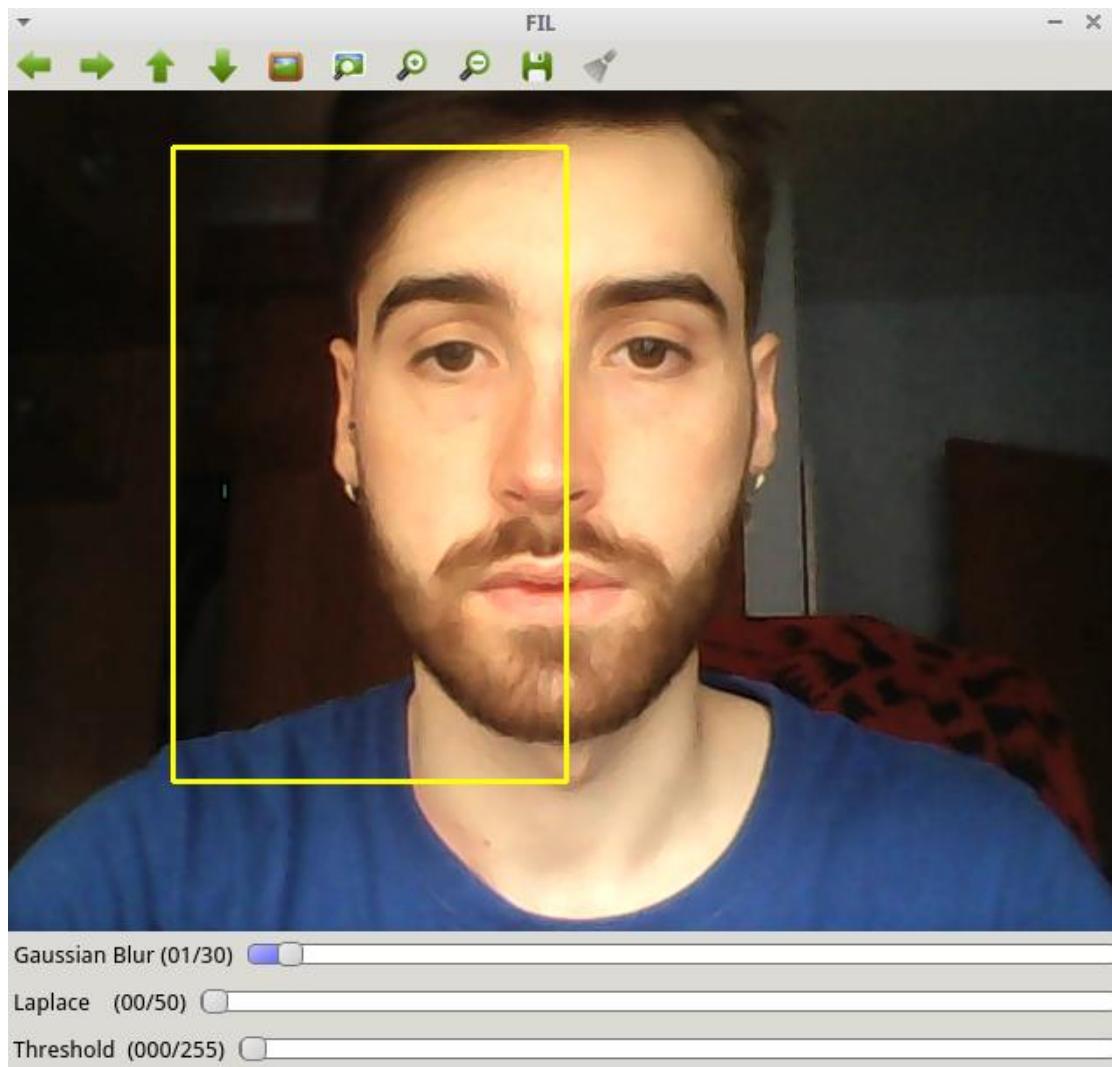
# Mostrar la imagen resultante
cv.imshow('FIL',frame)

# Salir del programa
cv.destroyAllWindows()

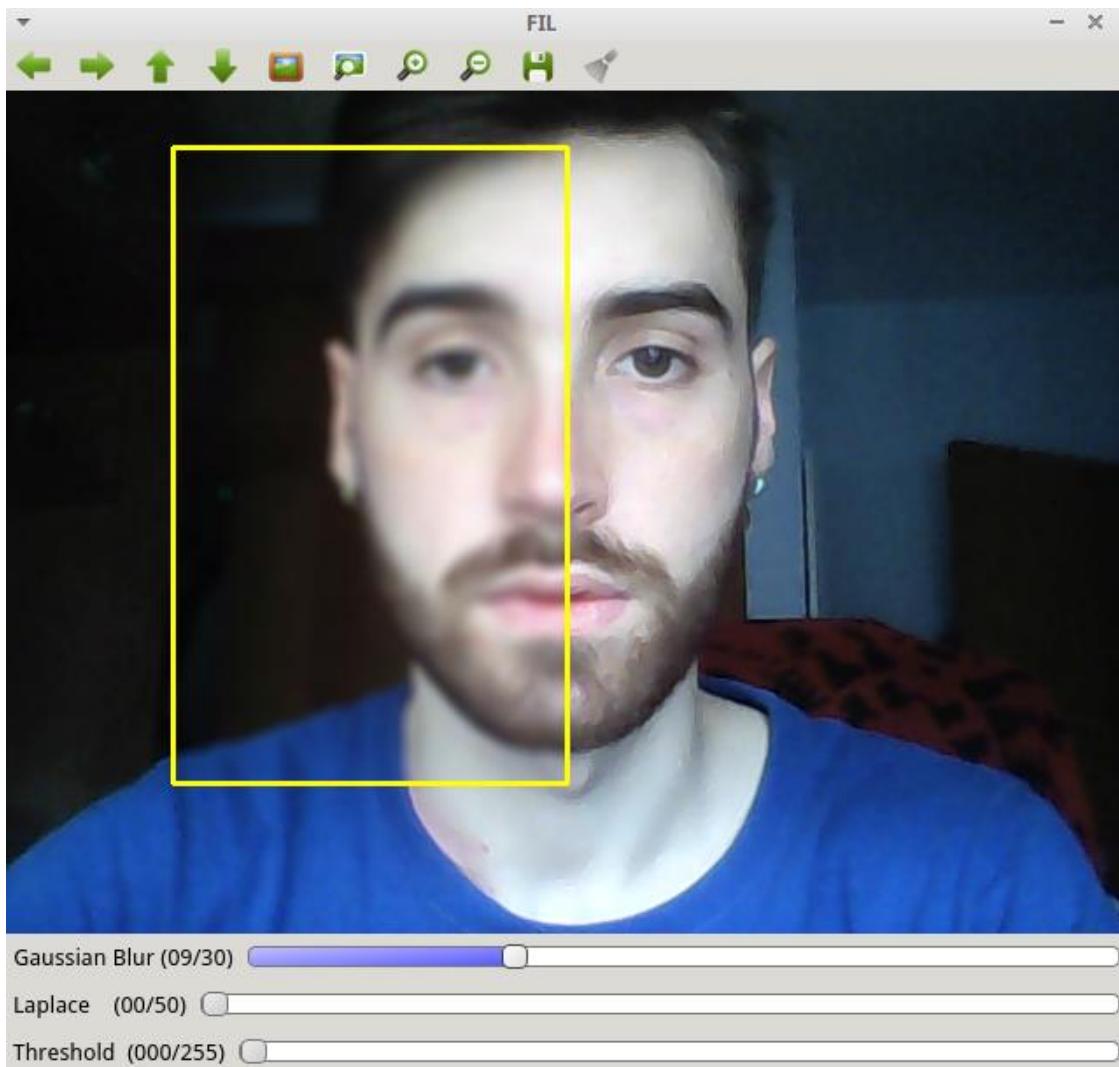
```

En caso de que el activo sea el laplaciano, se obtiene su valor del trackbar, se aplica el filtro y se actualiza la región en el frame. Para el flag de umbralización se sigue este mismo proceso. Por último se dibuja un rectángulo para señalizar la ROI seleccionada y se imprime la imagen resultante. Al acabar el bucle se cierran las ventanas del programa.

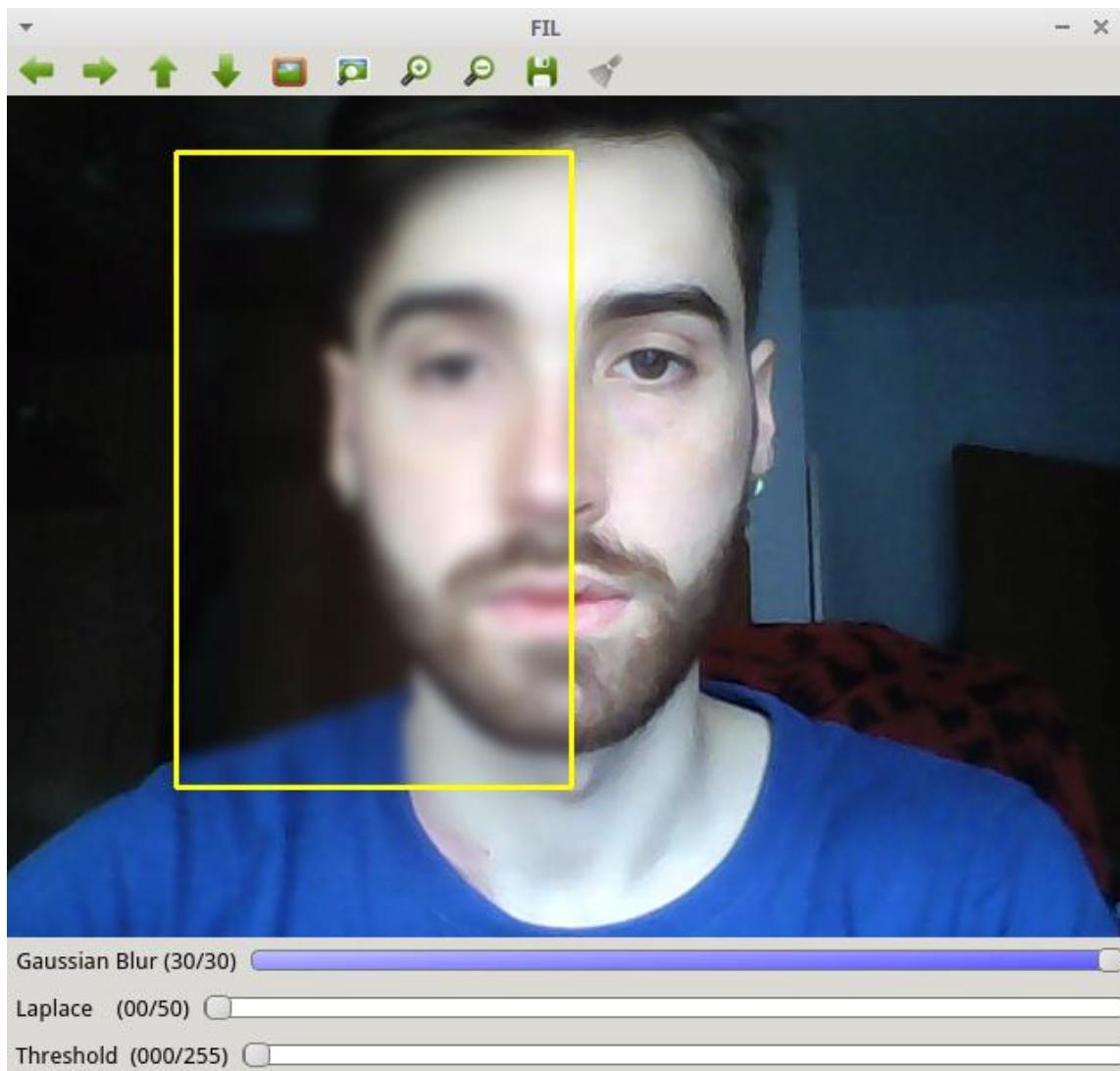
11.2 Ejemplos



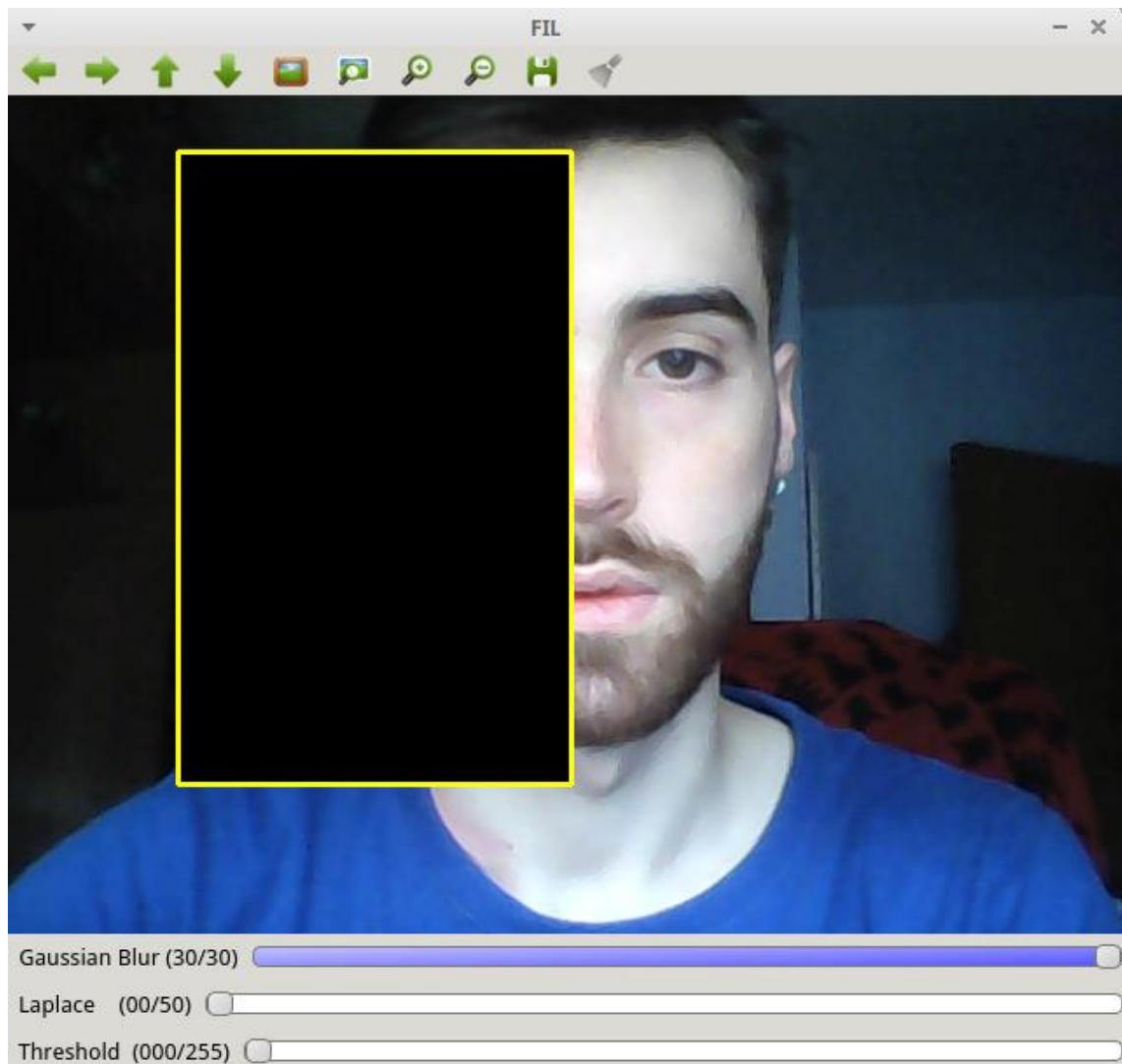
En primer lugar seleccionamos una ROI y vemos como se señalaiza en la imagen.



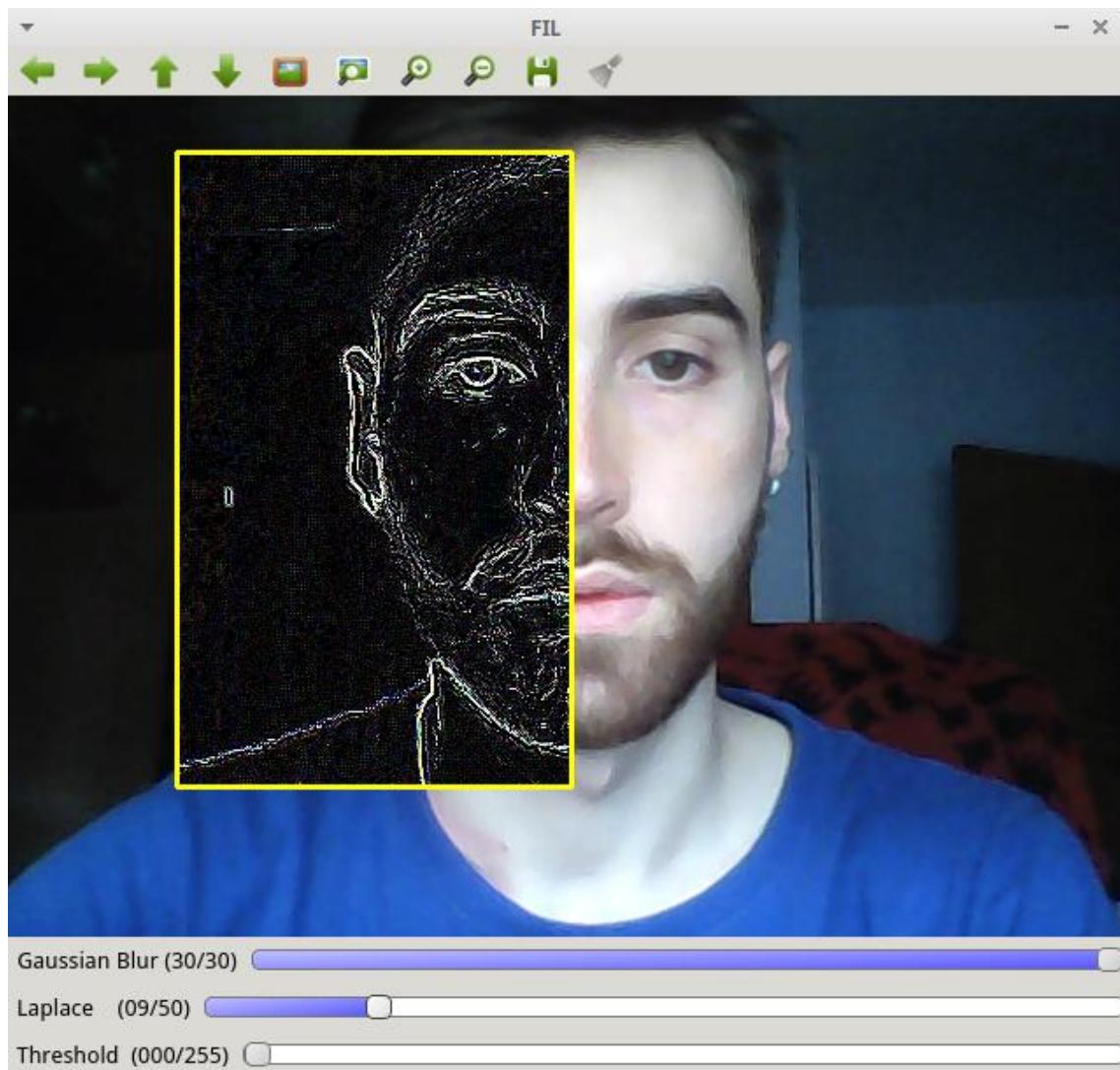
En esta imagen se pulsa la tecla *g* y se incrementa el valor del trackbar correspondiente al suavizado gaussiano. Vemos como se aplica el filtro a la ROI.



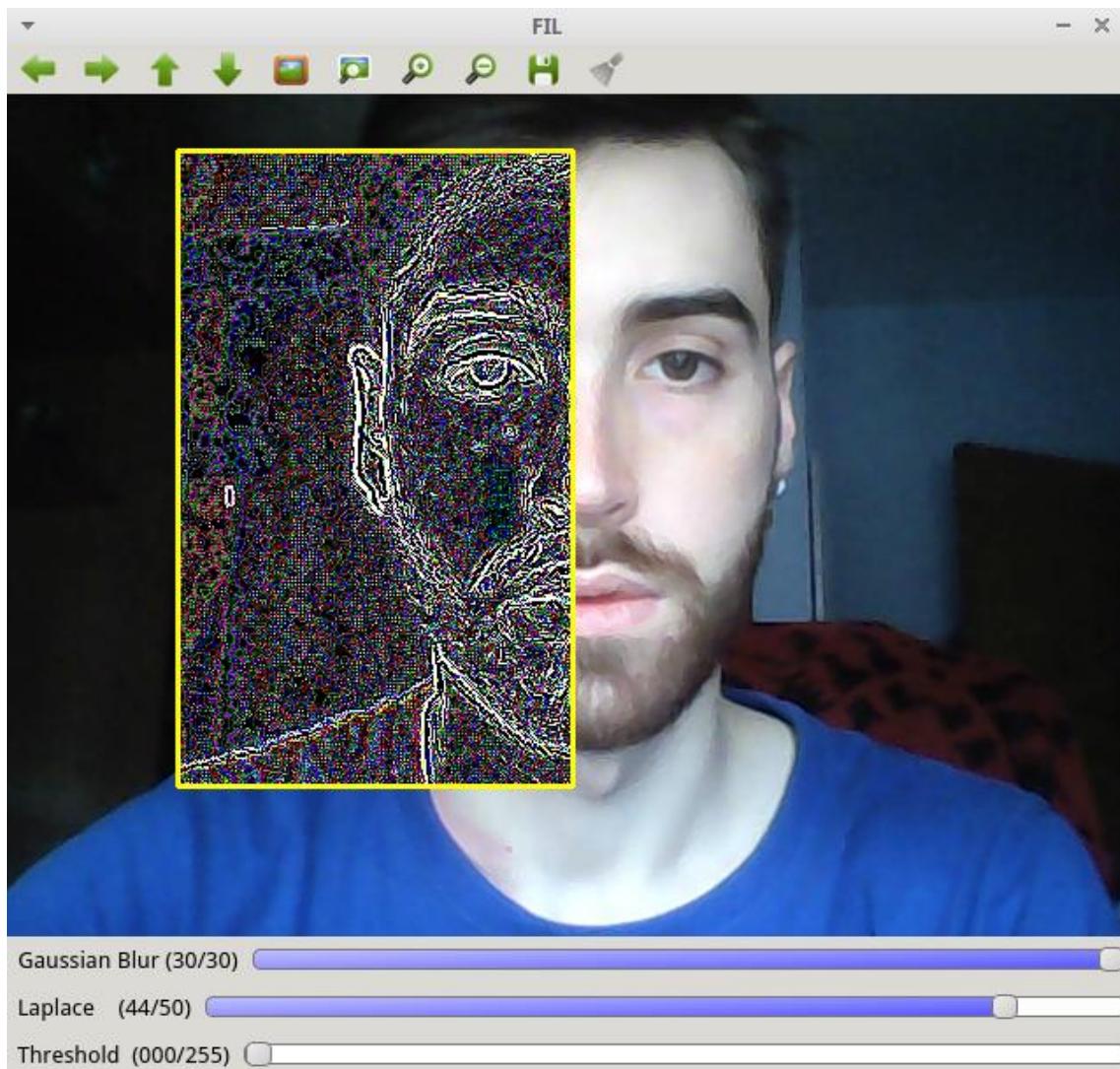
Efecto de aumentar al máximo valor el trackbar del suavizado gaussiano.



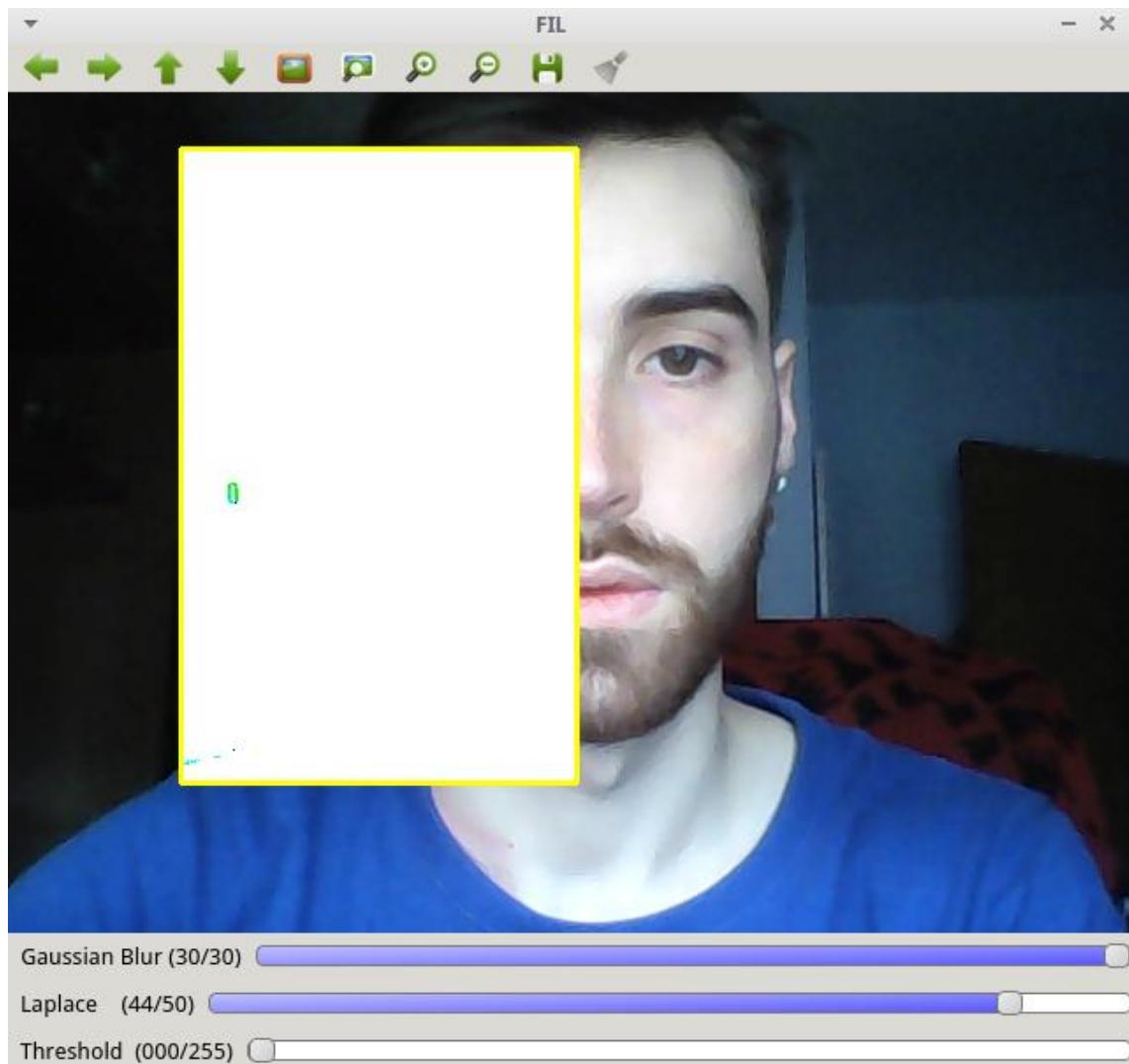
A continuación se pulsa la tecla / y se muestra el efecto del filtro laplaciano a valor 0.



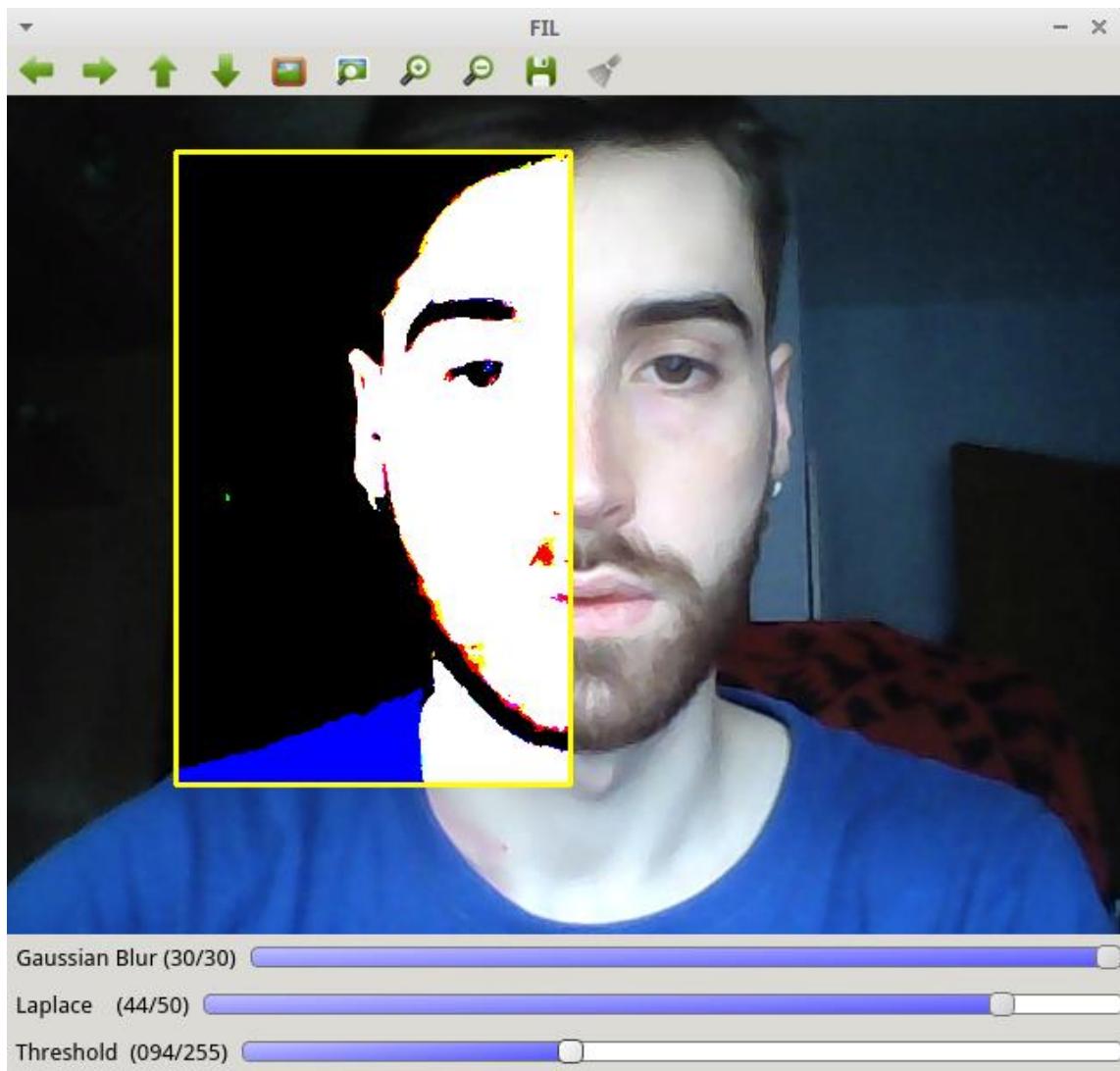
Efecto al incrementar este valor.



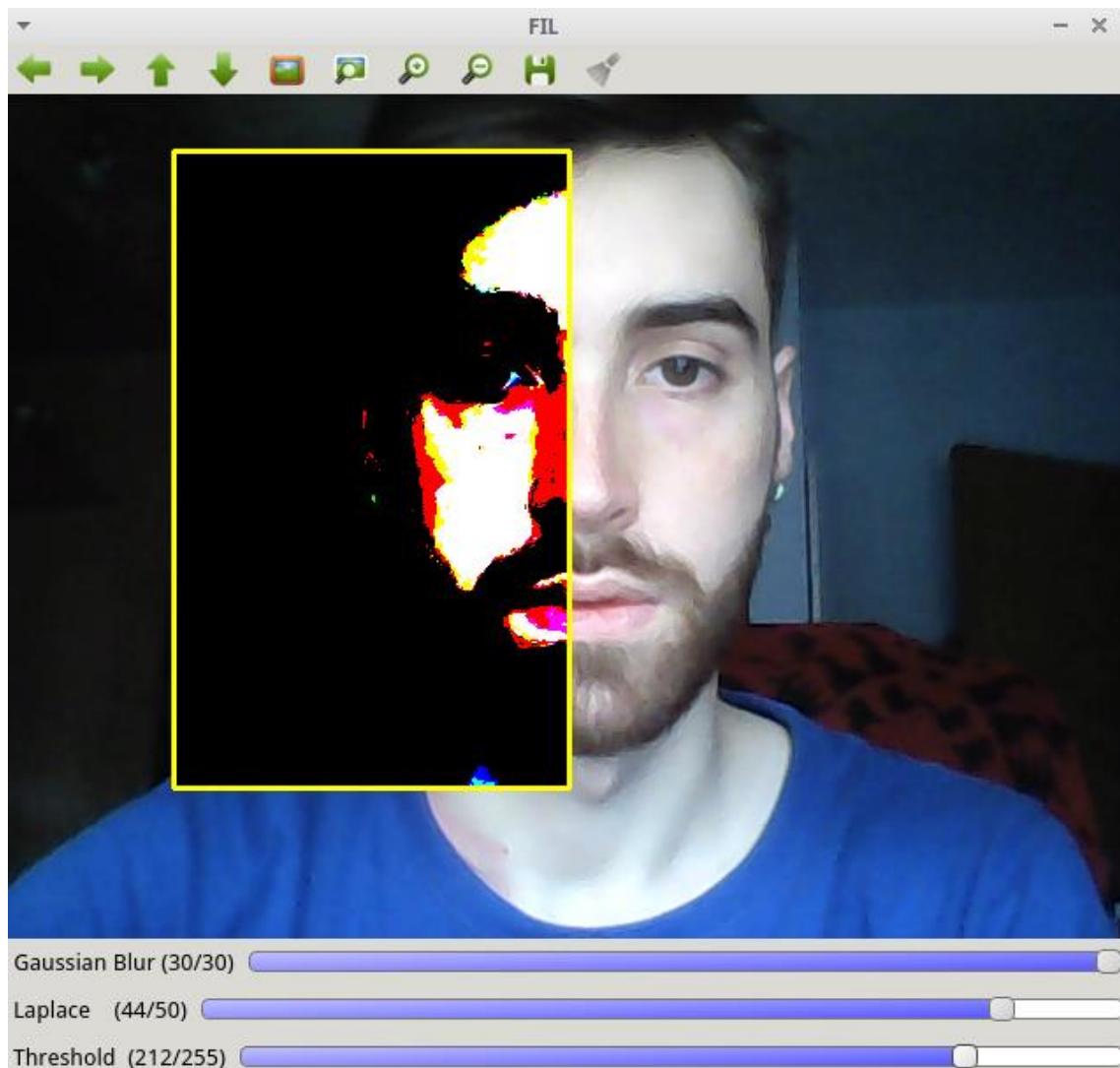
Efecto tras incrementar el valor del trackbar hasta acercarnos al máximo.



A continuación se pulsa la tecla *t* y se muestra el efecto de aplicar la umbralización a los píxeles por encima del valor 0 (todos).



Efecto de aplicar la umbralización a un nivel medio.



Por último en esta imagen se muestra el efecto de umbralizar solo por encima de los píxeles con valor 212.

12. ANON

Este ejercicio trata de detectar caras concretas ya registradas y preservar su anonimato mediante la pixelización de las mismas.

12.1 Código

```
# Crear la ventana y añadir una región de interés a ella
cv.namedWindow('ANON')
roi = ROI('ANON')

# Variables auxiliares
h = w = 50 # Alto y ancho de los modelos a mostrar
models = [] # Lista de los modelos
names = [] # Lista de nombres de las caras
n = 0 # Número de nombres
flagModels = False # Flag para saber si hay modelos
flagFaces = False # Flag para la activación del detector de caras

# Bucle principal de entrada de vídeo
for key, frame in autoStream():

    # Si se ha seleccionado una región
    if roi.roi:
        # Obtener sus coordenadas
        [x1,y1,x2,y2] = roi.roi

        # Recortar esa sección de la imagen
        region = frame[y1:y2+1, x1:x2+1]

        # Si se pulsa la tecla 'r'
        if key == ord('r'):
            # Copiar la roi
            trozo = region.copy()
```

En primer lugar se crea la ventana del programa. Después se definen las variables auxiliares: tamaño de los modelos, lista de modelos, lista de sus nombres, número de nombres y algunos flags de utilidad. Al comienzo del bucle principal se comprueba si se ha seleccionado una región de interés, en su caso se almacena esta sección por si el usuario quiere registrarla.

```
# Redimensionar a la altura y ancho definida
trozo = cv.resize(trozo,(w,h))
models.append(trozo)
names.append("face"+str(n))
n = n+1
if not flagModels: # Si es el primer modelo se añade a la lista
    modelsImg = trozo
    flagModels = True
else: # Si no es el primero se crea una imagen que combina cada modelo
    modelsImg = np.hstack((modelsImg, trozo))

# Eliminar la roi cuando se registra
roi.roi = []

# Dibujar el rectángulo de la roi
cv.rectangle(frame, (x1,y1), (x2,y2), color=(0,255,255), thickness=2)

# Si se pulsa 'f' se activa el detector de caras
if key == ord('f'): flagFaces = not flagFaces

# Cuando hay algún modelo registrado se muestra la ventana con los modelos registrados
if flagModels:
    cv.namedWindow('Models')
    cv.imshow('Models', modelsImg)

# Si se activó el flag de detección de caras
if flagFaces:
    # Se inicializa una lista de encodings vacía
    encodings = []

    # Para cada modelo se intenta reconocer una cara en él y solo se guardan los modelos que almacenan caras
    for m in models:
        if face_recognition.face_encodings(m):
            encodings.append(face_recognition.face_encodings(m)[0])
```

Al pulsar la tecla *r* se registra el modelo seleccionado por la ROI, para ello se crea la imagen de este modelo (o de la combinación de varios) escalándola y se crea un nuevo nombre para el modelo como “faceN”. En el momento en el que se registra el modelo se elimina la ROI, pero mientras esta esté activa se indicará con un rectángulo. Pulsando la tecla *f* podemos activar y desactivar la detección de caras. He dado esta opción porque tener la detección activa degradó mucho el rendimiento (por lo menos en mi ordenador) así que era preferible poder tenerla activa en momentos que realmente lo requieran. Tras esto se comprueba que haya modelos para mostrarlos en una nueva ventana. En el momento en el que el flag de la detección de caras está activado se crea una lista vacía de encodings y se almacenan en ella los modelos en los que se reconoce una cara (para evitar problemas derivados de modelos que el usuario registre en los que no haya ninguna cara o sea irreconocible).

```
# Se registran los tiempos t0,t1 y t2 para mostrar estadísticas sobre el rendimiento del programa
t0 = time.time()

# Localización de las caras del frame actual
face_locations = face_recognition.face_locations(frame)
t1 = time.time()

# Encodings de las caras del frame actual
face_encodings = face_recognition.face_encodings(frame, face_locations)
t2 = time.time()

# Para cada cara obtener su posición
for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
    # Comparar la cara del frame con la del modelo
    match = face_recognition.compare_faces( encodings, face_encoding)

    # Para cada nombre y ls lista de matches
    name = "Unknown"
    for n, m in zip(names, match):
        # Si se encuentra a verdadero ha habido una coincidencia
        if m:
            # Almacenar el nombre
            name = n

    #Recortar la posición de la cara de la imagen
    region = frame[top:bottom, left:right]

    # Obtener su forma
    height, width = region.shape[:2]

    # Pixelar la región disminuyéndola a 16x16 pixeles
    region = cv.resize(region, (16, 16), interpolation=cv.INTER_LINEAR)
```

A continuación se registran diferentes marcas de tiempo *t0*, *t1* y *t2* para mostrar estadísticas sobre el rendimiento. En medio de estas marcas se obtienen las localizaciones y los encodings de las caras en tiempo de ejecución. Mediante estas localizaciones obtenemos el cuadrado que las contiene, para cada encoding obtenido en tiempo de ejecución se comprueba si enlaza con alguno de los producidos por los modelos registrados. Si este *match* se produce se obtiene el nombre de su modelo y se realiza la pixelización de la cara. Para ello se obtiene la sección de la cara y se redimensiona a menor resolución.

```

# Devolverla a su forma original
region = cv.resize(region, (width, height), interpolation=cv.INTER_NEAREST)

# Actualizar la región en el frame
frame[top:bottom, left:right] = region

# Dibujar el rectángulo de la cara y escribir su nombre
cv.rectangle(frame, (left, top), (right, bottom), (255, 0, 0), 2)
putText(frame, name, orig=(left+3,bottom+16))

# Mostrar las estadísticas de rendimiento
putText(frame, f'{(t1-t0)*1000:.0f} ms {(t2-t1)*1000:.0f} ms')

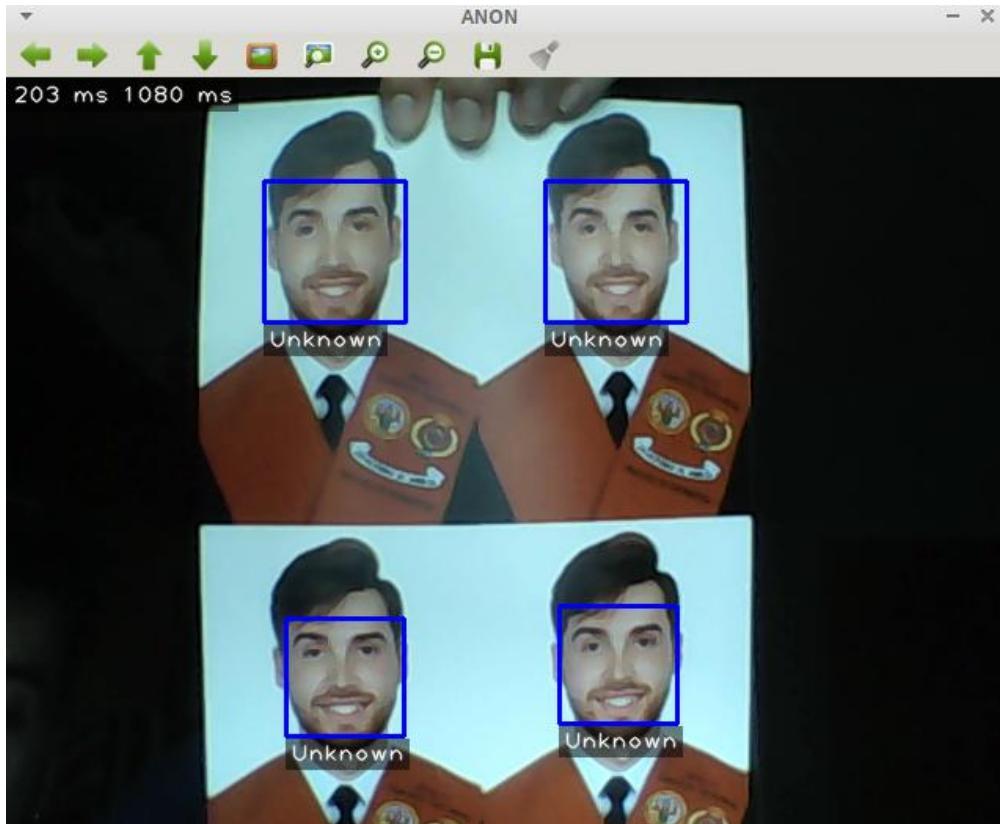
# Mostrar la imagen resultante
cv.imshow('ANON',frame)

# Salir del programa al finalizar el bucle
cv.destroyAllWindows()

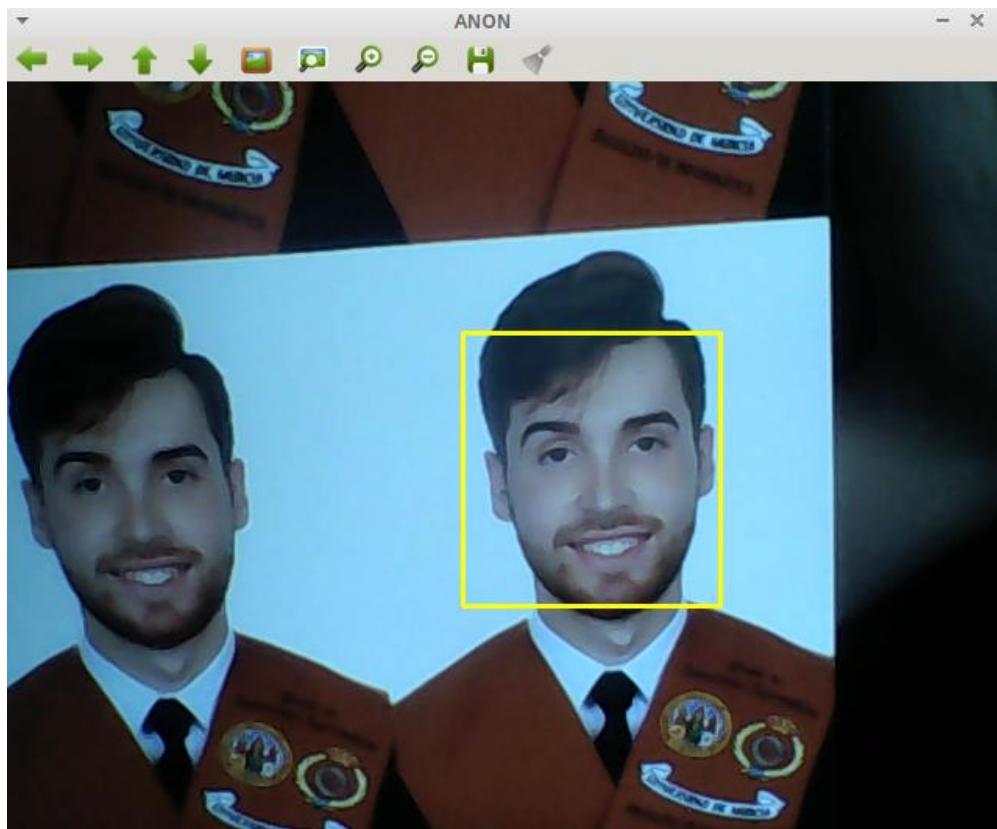
```

Cuando se ha modificado la resolución se devuelve la sección al tamaño original y se actualiza en el frame actual. A continuación se dibuja un rectángulo donde se detecta la cara y se imprime su nombre, esto ocurre siempre que el flag de detección esté activo y se encuentre alguna cara. Por último se muestran las estadísticas de rendimiento, se muestra la imagen resultante y se destruyen las ventanas al acabar el bucle principal.

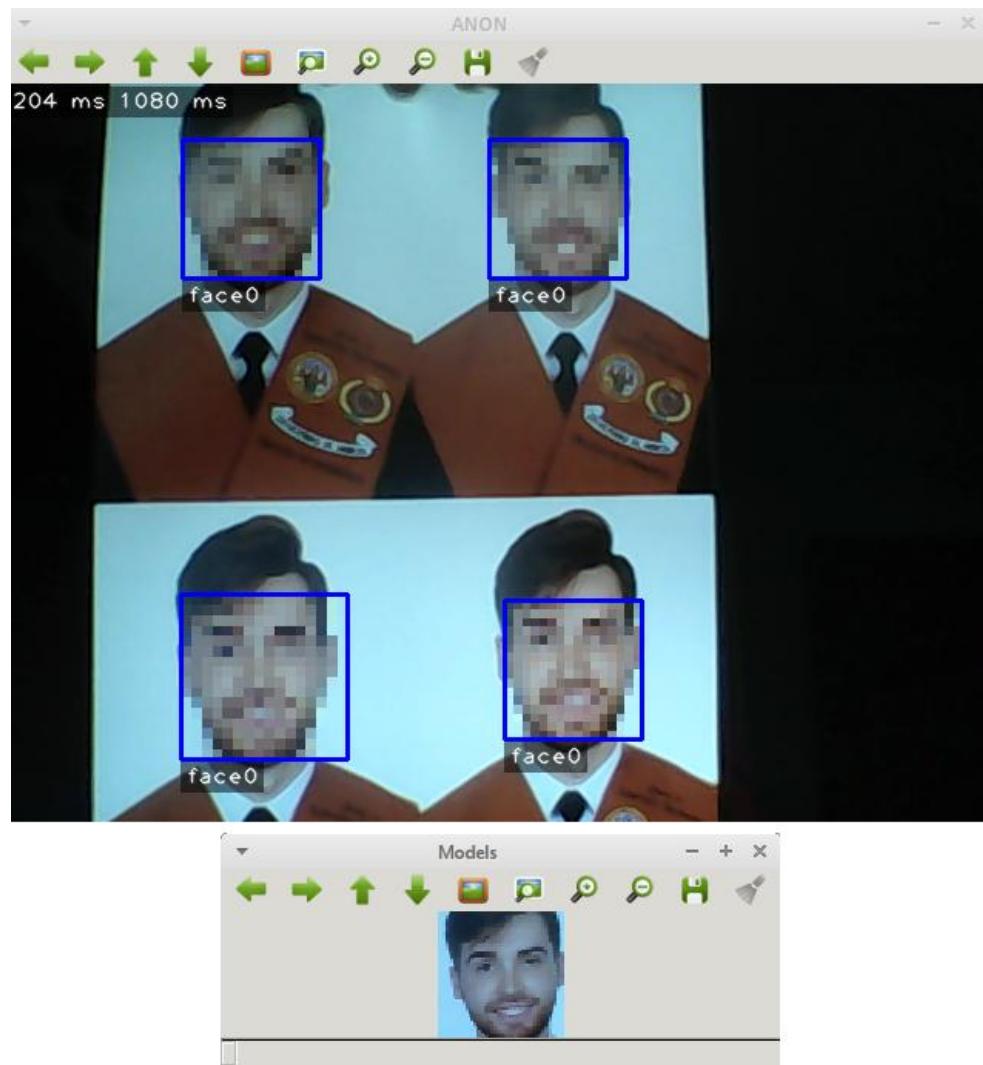
12.2 Ejemplos



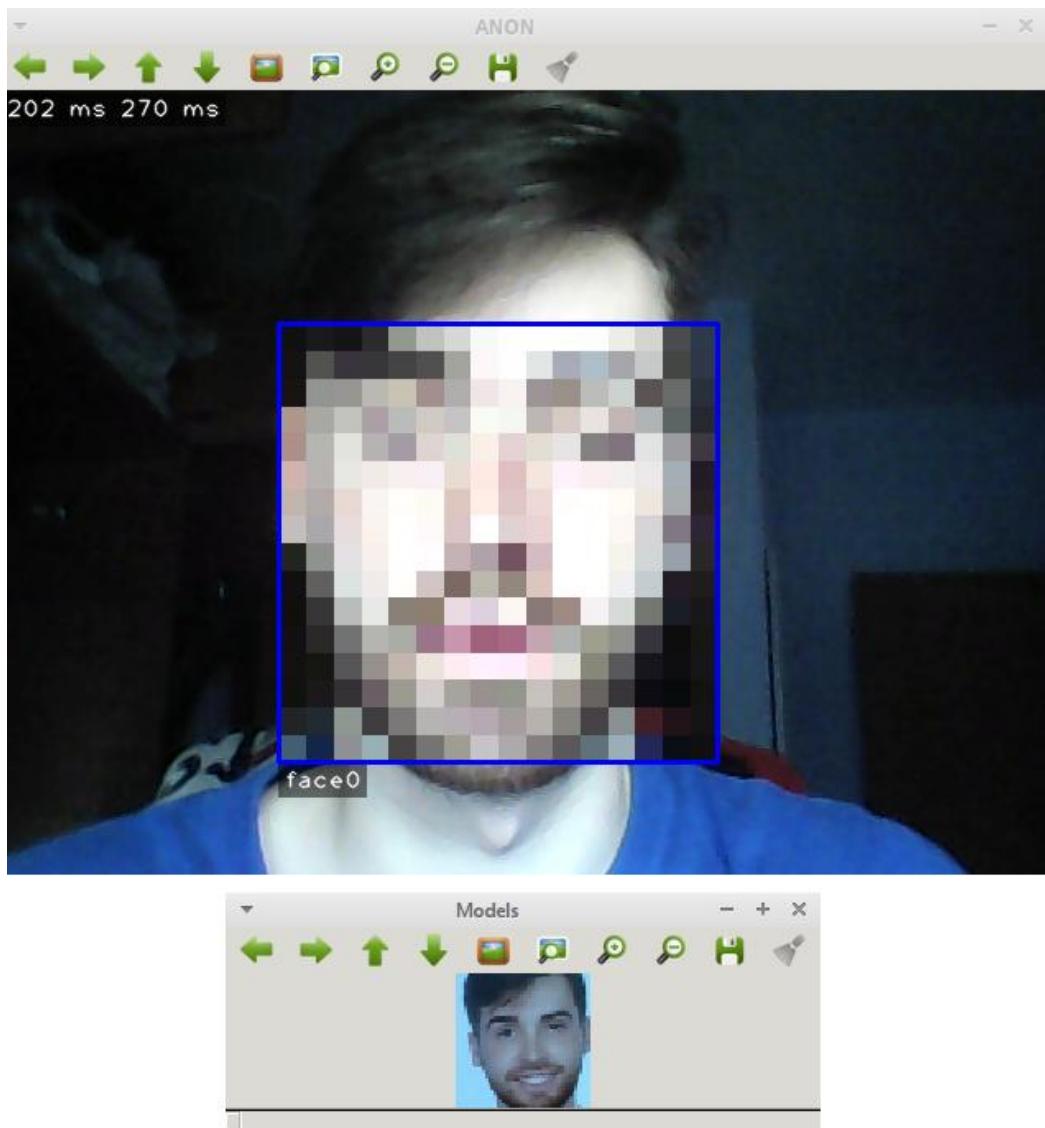
En la primera imagen comprobamos el efecto de pulsar *f* para activar la detección de caras cuando aún no hay caras registradas como modelos. Por lo tanto todas son desconocidas.



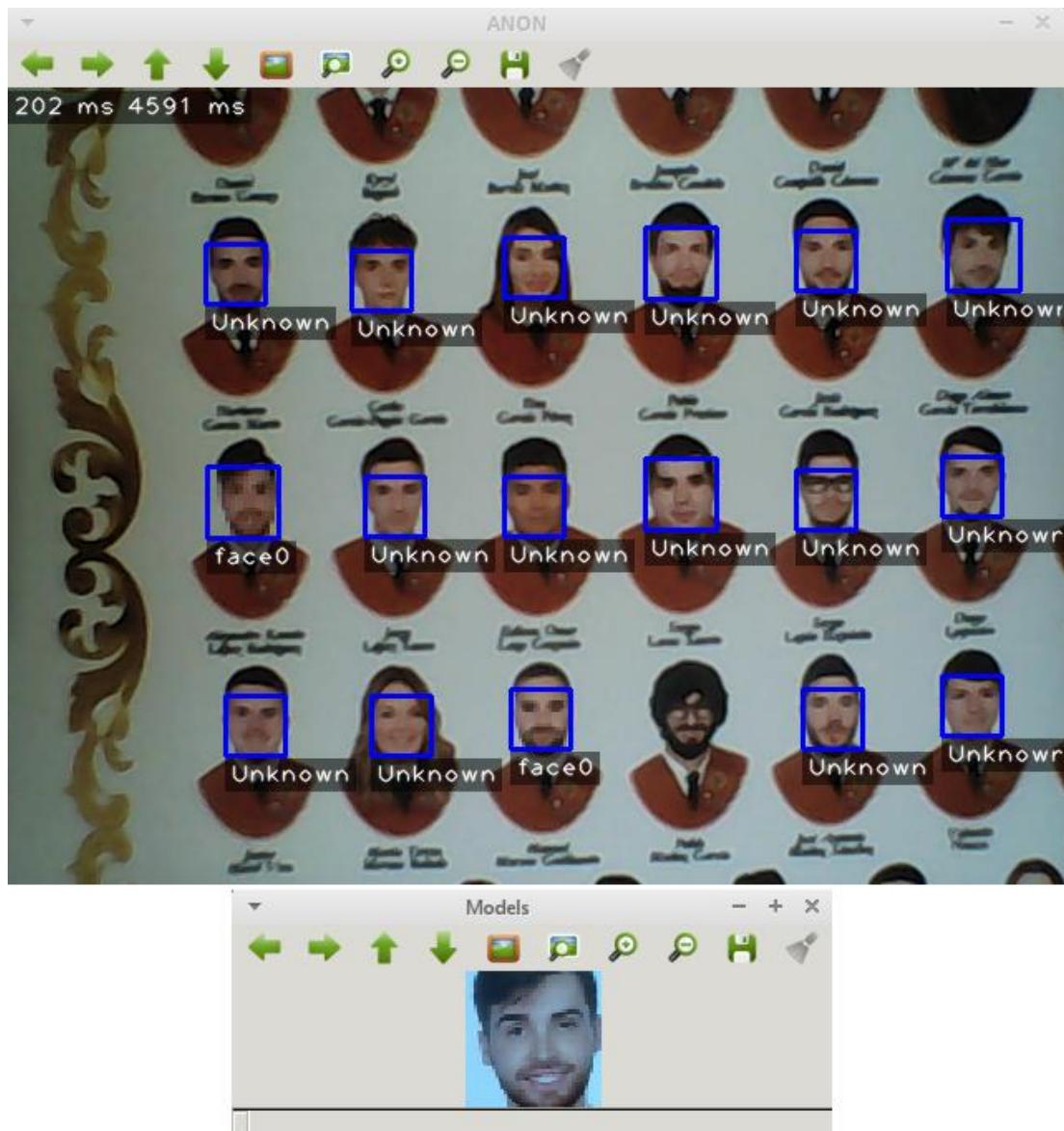
Seleccionamos una cara mediante una ROI y pulsamos *r* para registrarla.



Ahora tenemos una ventana adicional donde se muestran los modelos registrados. Podemos ver como al pulsar *f* de nuevo las caras de la primera imagen se reconocen y se pixelan. El nombre de la cara será el de la primera posible, *face0*.



Comprobamos que también funciona con una cara que no se encuentra estática.



Por último vemos la detección de caras en un conjunto grande de personas, de todas ellas reconoce adecuadamente mi cara aun que se confunde con otro alumno de la orla de la fila de abajo. He cogido esta imagen porque la tenía a mano y comparaba mi cara (fácil de capturar) con muchas otras, espero que no haya ningún problema por usar la imagen de otros alumnos de la carrera para este ejemplo.

13. Conclusión

Para terminar quería dar un poco mi opinión sobre la experiencia que he tenido realizando los ejercicios. Me ha parecido bastante interesante trabajar con imágenes de esta forma y en general con Numpy, OpenCV y Python, lenguaje sobre el que no tenía ninguna experiencia y que me ha parecido muy cómodo e intuitivo. Creo que he aprendido bastante al practicar haciendo los ejercicios y me ha gustado hacerlo. Sin duda uno de los pros a favor de esta asignatura, según mi criterio, es el hecho de poder trabajar de forma autónoma sin que suponga un verdadero infierno conseguir resultados. En mi caso he hecho los siete ejercicios obligatorios y cuatro másopcionales y no me han llevado un tiempo excesivo en ello, la clave es que aun que los ejercicios en sí no son tan fáciles, tenemos guías y estructuras para saber cómo empezar el ejercicio y que la creatividad de los alumnos se centré en crear cosas nuevas a partir del material ya existente. Esta situación, en mi opinión, hace que la asignatura no sea agobiante, en muchos ejercicios yo sabía que podía parar en cierto punto y ya sería válido, lo que me daba la libertad de intentar implementar alguna idea nueva, añadir unos trackbars, algunos efectos más, sin la presión de que obligatoriamente me tenía que funcionar para entregarlo, de hecho hay cosas que intenté y tuve que descartar. A la larga eso me ha hecho practicar y ensayar más con OpenCV y Python, y realmente creo que he aprendido mucho, sobre todo las funcionalidades comunes a varios ejercicios como seleccionar y registrar una ROI, que ya hago de forma automática. A esta flexibilidad de la asignatura se le suma el hecho de que no conste de una evaluación teórica, que estoy seguro de que sería mucho más complicada y que, a pesar de estudiarla, no aprendería tanto como he aprendido ahora. Pienso que practicando es como se aprende y que la teoría que he aprendido de forma práctica es la que voy a preservar como conocimiento, no la que estudie para hacer un examen y que quizás ni comprenda. Tanto es así que creo que otras asignaturas deberían de amoldarse a este modelo de evaluación, sobre todo por ser teóricas en casi su totalidad, y transmitiré esta idea en futuros feedbacks a estas asignaturas.