



UNIVERSIDAD DE MURCIA

VISIÓN ARTIFICIAL

Documentación

Norberto García Marín

Junio 2019

Índice

1	Introducción	2
2	Ejercicio 1	2
3	Ejercicio 4	4
4	Ejercicio 6	6
5	Ejercicio 8	11
6	Ejercicio 11	19
7	Ejercicio 12	21
8	Ejercicio 15	24
9	Ejercicio 16	27
10	Ejercicio 17	30
11	Ejercicio 18	31

1 Introducción

Este trabajo trata de la resolución de los ejercicios propuestos para la asignatura de Visión Artificial.

En esta entrega se ha añadido este documento pdf, además de los últimos ejercicios propuestos. También se ha mejorado la explicación del Ejercicio 1 y se ha resuelto el ejercicio 11 que estaba pendiente.

2 Ejercicio 1

Estima de forma aproximada el campo de visión (FOV) y parámetro f de tu cámara. Determina a qué altura habría que ponerla para obtener una vista cenital completa de un campo de baloncesto. Calcula la altura de una pelota sobre el suelo a partir de su tamaño en la imagen.

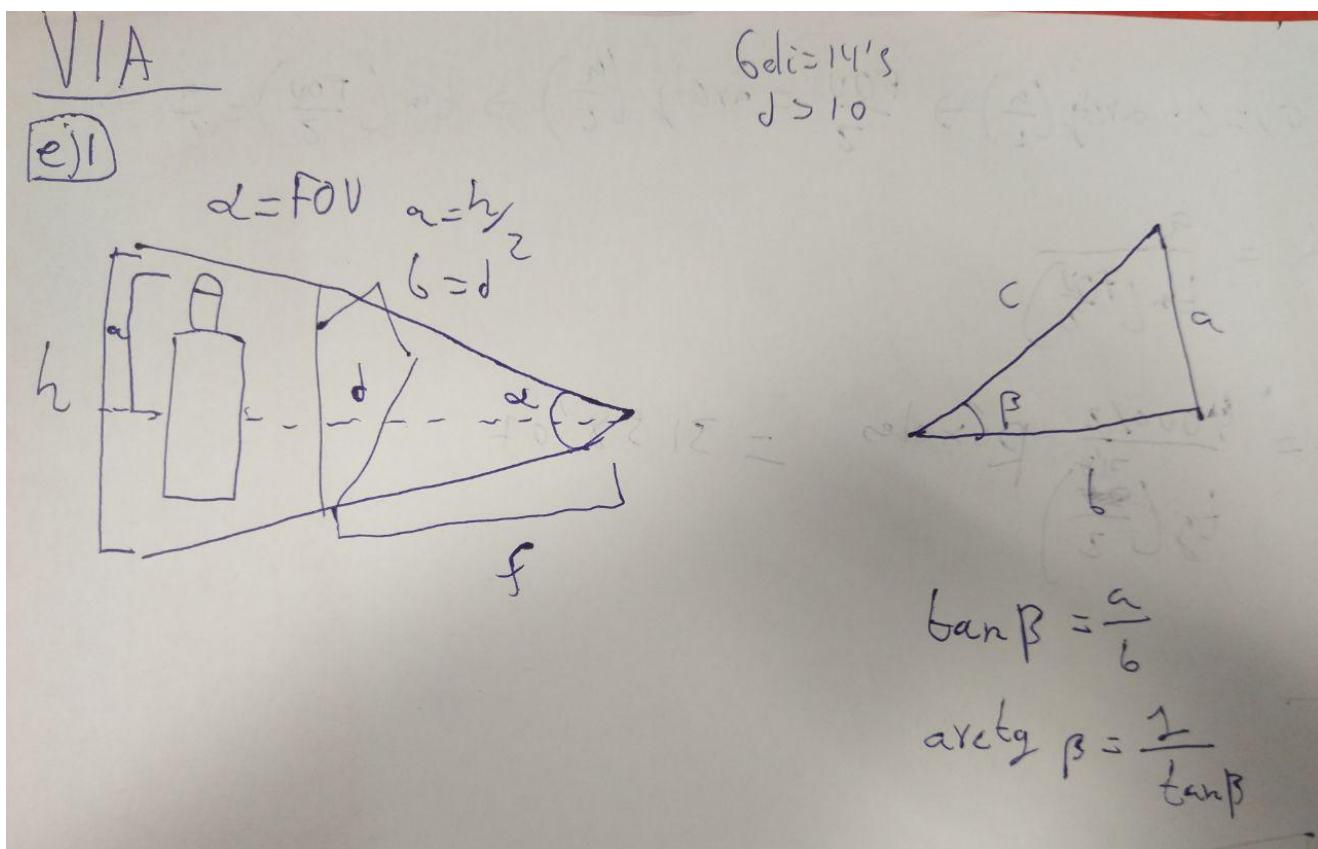


Figure 1. Esquema del Problema

Datos de la imagen:

Ancho = 3000 px

Alto = 4000 px

Alto de la botella = 10 cm

Ancho de la botella = 14.5 cm

Para calcular el FOV tenemos que calcular el valor de β

$$\arctan(\tan \beta) = \arctan\left(\frac{a}{b}\right) \rightarrow \beta = \arctan\left(\frac{a}{b}\right)$$

$$FOV = 2\beta = 71,8842$$

Para calcular el parámetro f es necesario saber la resolución de la imagen, en mi caso de 3000 x 4000 px.

$$71,88 = \arctan\left(\frac{4000/2}{f}\right)$$
$$f = \frac{4000/2}{\tan(71,88)} = 3158,07 \text{ px}$$

Para determinar la altura que habría que poner la cámara en un campo de baloncesto suponemos que el campo mide 28 m de largo y 15 m de ancho, y la pelota 23,87 cm de diámetro.

La altura h sería:

$$h = \frac{a_{FOV}}{\tan \frac{\pi}{2}} = 22.1 \text{ m}$$

Para calcular la altura de la pelota sobre el suelo, sabes que la longitud de la pelota es de 75 cm con lo que supone $\frac{74}{\pi}$ cm de diámetro.

$$i = f * \frac{x}{z}$$
$$z = \frac{f * x}{i} = \frac{3158,07 * 23,87}{i}$$

Si tomamos una foto en la que la pelota de baloncesto mida i píxeles sabemos que la pelota se encuentra a z cm de la cámara. Luego, con el apartado anterior sabemos que la altura a situar la cámara serían de 22.1 m del suelo, entonces la pelota se debe encontrar a $1313-z$ cm del suelo.

3 Ejercicio 4

Construye un detector de movimiento basado en una sencilla diferencia de imágenes sucesivas. Opcionalmente puedes hacer un generador que filtre la secuencia de imágenes dejando pasar solo los frames estáticos (o, alternativamente, los que han sufrido un cambio apreciable. En este caso no hace falta detectar las zonas con movimiento). Puedes apoyarte en deque.py. Opcional: a) Limitar la detección a una región de interés. b) Guardar 2 ó 3 segundos de la secuencia detectada en un archivo de vídeo o imagen gif.

El programa es capaz de detectar movimiento. Cuando esto pasa se imprime por la salida estándar una linea con la fecha del suceso.

```
1 #!/usr/bin/env python
2
3 import cv2    as cv
4 import numpy as np
5 import time
6
7 from datetime import datetime
8 from umucv.stream import autoStream
9
10 # Numero maximo de frames mientras no esta capturando movimiento
11 # Es necesario ya que si los frames son muy parecidos nunca va a detectar
12 # movimiento porque apenas hay diferencia entre ellos.
13 MAX_FRAMES = 10
14
15 # Variables de control para esperar MAX_FRAMES
16 isRecording = False
17 frames = 0
18
19 # Array de frames para detectar diferencias entre ellos
20 static_frames = []
21
22 for key,frame in autoStream():
23     if not isRecording:
24         frames += 1
25         if frames == MAX_FRAMES:
26             frames = 0
27             isRecording = True
28
29             cv.imshow("cam",frame)
30             continue
31
32     # Inicializar motion a 0 (no hay movimiento)
33     motion = 0
34
35     # Convertir imagen a gray_scale
36     gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
```

```

39     # Convertir imagen gray a GaussianBlur
40     # Para poder encontrar cambios facilmente
41     gray = cv.GaussianBlur(gray, (21, 21), 0)
42
43     # Si estamos en la primera iteración inicializamos el frame
44     if not static_frames:
45         static_frames.append(gray)
46         static_frames.append(gray)
47         continue
48
49     # Frame actual
50     static_frames[1] = gray
51
52     # Diferencia entre el frame anterior
53     # y la imagen actual
54     diff_frame = cv.absdiff(static_frames[0], static_frames[1])
55
56
57     thresh_frame = cv.threshold(diff_frame, 30, 255, cv.THRESH_BINARY)[1]
58     thresh_frame = cv.dilate(thresh_frame, None, iterations = 2)
59
60     # Contorno de los objetos en movimiento
61     cnts = cv.findContours(thresh_frame.copy(),
62                           cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)[1]
63
64     # Detectamos si hay diferencia
65     for contour in cnts:
66         if cv.contourArea(contour) >= 1000:
67             motion = 1
68             break
69
70
71     # El frame actual es el frame anterior
72     # para la siguiente iteracion
73     static_frames[0] = static_frames[1]
74
75
76     cv.imshow('cam', frame)
77
78     isRecording = False
79
80     # Si no hay movimiento no hago nada
81     if motion == 0:
82         continue
83
84
85     print('Movimiento')
86     print(datetime.now())
87
88     cv.destroyAllWindows()

```



Figure 2. Imagen Inicial

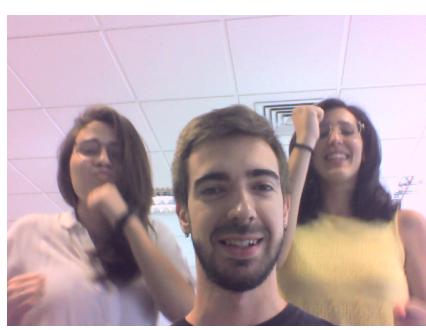


Figure 3. Imagen Movimiento

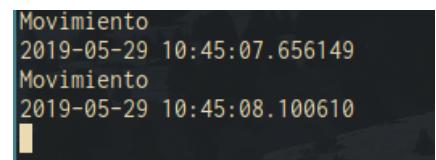


Figure 4. Output

4 Ejercicio 6

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

El programa clasifica objetos que han sido guardados anteriormente a través del ROI.

```
1 #!/usr/bin/env python
2
3 ##### Uso del programa #####
4 #   Detector de objetos similares basado en el histograma. #
5 #   Se usa el raton para marcar el ROI, este aparecerá azul #
6 #   cuando se encuentre un objeto similar, en otro caso      #
7 #   el color será verde.                                     #
8 #####
9
10
11 import cv2 as cv
12 import numpy as np
13 import datetime
14
15
16 # Cordenadas del ROI
17 xi = yi = xf = yf = -1
18
19 #Máxima diferencia permitida entre histogramas
20 MAX_DIF = 0.4
21
22 # Variables de control para el ROI
23 selection = False
24 isDrawing = False
25
26
27 # Almacenamiento de imágenes obtenidas
```

```

28 roiList = list()
29 lastRoi = None
30 frame = None
31 capture = None
32
33 # Colores para dibujar el ROI
34 BLUE = (255,0,0)
35 GREEN = (0,255,0)
36
37
38 # Función para ordenar las coordenadas que contiene el ROI
39 def tratarCordROI():
40     global xi,yi,xf,yf,frame
41
42     # Ancho y alto del frame
43     width = capture.get(3)
44     height = capture.get(4)
45
46
47     # Establecer valores en función de si es mayor o menor
48     if xi > xf:
49         aux = xi
50         xi = xf
51         xf = aux
52
53     if yi > yf:
54         aux = yi
55         yi = yf
56         yf = aux
57
58     # Si la iamgen esta fuera de los margenes, la pongo dentro
59     if xi < 0:
60         xi = 0
61     if yi < 0:
62         yi = 0
63
64     if xf > width:
65         xf = int(width)
66     if yf > height:
67         yf = int(height)
68
69
70     return frame[yi:yf, xi:xf]
71
72
73
74 # Histograma normalizado de una imagen
75 def histogram(image):
76     h = cv.calcHist([image],
77                   [0, 1, 2],

```

```
78                                     None,  
79                                     [8, 8, 8],  
80                                     [0,256]+[0,256]+[0,256])  
81  
82     his = h / np.sum(h)  # normalizacion  
83  
84     return his  
85  
86  
87 # Caputara de eventos del raton para obtener el ROI  
88 def captureMouse(event, x, y, flags, param):  
89     global xi,yi,xf,yf,isDrawing,frame, roiList, selection, capture, lastRoi  
90  
91  
92     # Boton izquierdo raton  
93     if event == cv.EVENT_LBUTTONDOWN:  
94         xi = x  
95         yi = y  
96         xf = x  
97         yf = y  
98         isDrawing = True  
99         selection = True  
100  
101    # Mover raton  
102    elif event == cv.EVENT_MOUSEMOVE:  
103        if isDrawing:  
104            xf = x  
105            yf = y  
106  
107  
108    # Dejar de pulsar boton izquierdo  
109    elif event == cv.EVENT_LBUTTONUP:  
110        if isDrawing:  
111            isDrawing = False  
112            xf = x  
113            yf = y  
114  
115            # Normalizar las coordenadas  
116            lastRoi = tratarCordROI()  
117  
118            #Guardar el ROI en la lista  
119            roiList.append(lastRoi)  
120  
121            #Mostrar el ROI  
122            cv.imshow('ROI', lastRoi)  
123  
124 # Dibujar un rectangulo como si fuera el ROI  
125 def drawRectROI(f, xi, yi, xf, yf, color):  
126     cv.rectangle(f, (xi, yi), (xf, yf), color)
```

```

128 # Diferencia entre los histogramas del frame actual y el del ROI
129 def difHistograms(frameAct, roi):
130     hframeAct = histogram(frameAct)
131     hroi = histogram(roi)
132
133     return cv.compareHist(hroi, hframeAct, cv.HISTCMP_CHISQR)
134
135
136
137
138
139
140 cv.namedWindow("cam")
141 cv.setMouseCallback("cam", captureMouse)
142 capture = cv.VideoCapture(0)
143
144
145 while(True):
146     key = cv.waitKey(1) & 0xFF
147
148     if key == 27: break
149
150     # Pulsar x para eliminar el ROI de la lista
151     if key == ord('x'):
152         selection = False
153         roiList = roiList[:-1]
154
155
156     ret, frame = capture.read()
157
158
159     # Si hay algún ROI seleccionado
160     if selection:
161         # Si se encuentra un objeto igual se va a dibujar un rectangulo azul
162         color = BLUE;
163
164         # Si hay un ROI se obtiene la imagen
165         if lastRoi is not None:
166             fAct = tratarCordROI()
167             diferencia = difHistograms(fAct, lastRoi)
168
169             #Si el objeto no es el del ROI el rectangulo es verde
170             if diferencia >= MAX_DIF:
171                 color = GREEN
172
173             #Dibujar rectangulo
174             drawRectROI(frame, xi, yi, xf, yf, color)
175
176             cv.imshow('cam',frame)
177

```

```
178
179 # Guardar ROIs en disco
180     if key == ord('s'):
181         count = 1
182         for r in roiList:
183             fname = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
184             cv.imwrite(fname+' '+str(count)+'.png',r)
185             count += 1
186
187     # Si se guarda, inicializar las variables del ROI
188     selection = False
189     roiList.clear()
190
191
192 cv.destroyAllWindows()
```

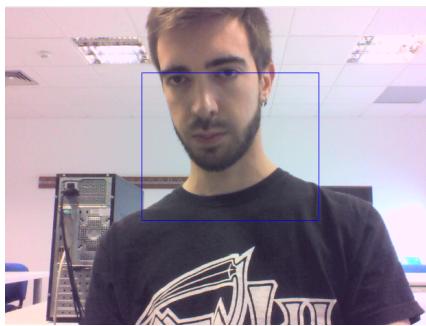


Figure 5. Imagen Inicial

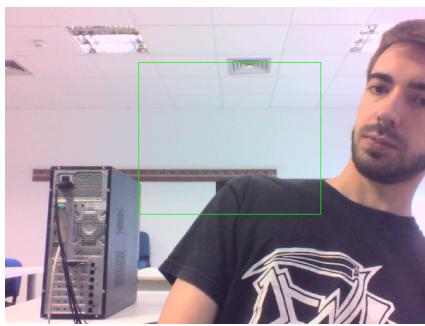


Figure 6. No detecta el objeto

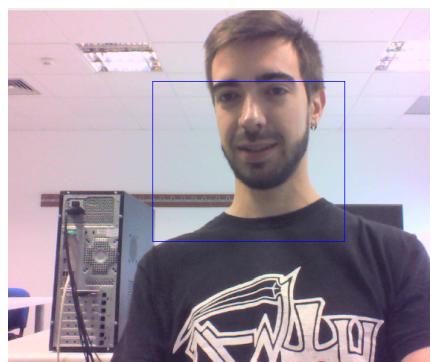


Figure 7. Dectecta el objeto

5 Ejercicio 8

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 de flecha. Es conveniente permitir la selección de un ROI para comparar el resultado del filtro con el resto de la imagen.

```
1 #!/usr/bin/env python
2
3
4 ##### Uso del programa #####
5 # 1 : Aumentar o reducir brillo #
6 # 2 : Imagen doble #
7 # 3 : Bordes horizontales #
8 # 4 : Bordes verticales #
9 # 5 : Suavizado Gaussiano #
10 # x : Elimina el ROI seleccionado #
11 # q : Elimina el filtro del ROI #
12 #       ↑ : Aumenta el valor de filtro
13 #       ↓ : Disminuye el valor de filtro #
14 ##### #####
15
16
17
18 import numpy           as np
19 import cv2              as cv
20 import scipy.signal     as signal
21 import time
22
23 # Teclas
24 BRILLO = '1'
25 DOBLE = '2'
26 BHOR = '3'
27 BVER = '4'
28 SGAU = '5'
29 UP = 82
30 DOWN = 84
31
32 # Color
33 BLUE = (255,0,0)
34
35
36 # Coordenadas del ROI
37 xi = yi = xf = yf = -1
38
39 # Variables de control para el ROI
40 selection = False
41 isDrawing = False
42
43 # Almacenamiento de imágenes obtenidas
```

```

44 frame = None
45 capture = None
46
47 # Variables para el control del filtro
48 filterValue = 0
49 InitFilterValue = True
50
51
52 # Funciones para transformacion inicial de la imagen
53 def rgb2gray(img):
54     return cv.cvtColor(img, cv.COLOR_RGB2GRAY)
55
56 def gray2float(img):
57     return img.astype(float) / 255
58
59 def frame2gray(capture):
60     ret, frame = capture.read()
61     return gray2float(rgb2gray(frame))
62
63
64 # Matriz de convolución
65 def cconv(k,x):
66     return signal.convolve2d(x, k, boundary='symm', mode='same')
67
68
69 # Función para ordenar las coordenadas que contiene el ROI
70 def tratarCordROI():
71     global xi,yi,xf,yf,frame
72
73     # Ancho y alto del frame
74     width = capture.get(3)
75     height = capture.get(4)
76
77
78     # Establecer valores en función de si es mayor o menor
79     if xi > xf:
80         aux = xi
81         xi = xf
82         xf = aux
83
84     if yi > yf:
85         aux = yi
86         yi = yf
87         yf = aux
88
89     # Si la iamgen esta fuera de los margenes, la pongo dentro
90     if xi < 0:
91         xi = 0
92     if yi < 0:
93         yi = 0

```

```
94
95     if xf > width:
96         xf = int(width)
97     if yf > height:
98         yf = int(height)
99
100
101    return frame[yi:yf, xi:xf]
102
103 # Caputara de eventos del raton para obtener el ROI
104 def captureMouse(event, x, y, flags, param):
105     global xi,yi,xf,yf,isDrawing,frame, selection, capture, lastRoi
106
107
108     # Boton izquierdo raton
109     if event == cv.EVENT_LBUTTONDOWN:
110         xi = xf = x
111         yi = yf = y
112         isDrawing = True
113         selection = True
114
115     # Mover raton
116     elif event == cv.EVENT_MOUSEMOVE:
117         if isDrawing:
118             xf = x
119             yf = y
120
121
122     # Dejar de pulsar boton izquierdo
123     elif event == cv.EVENT_LBUTTONUP:
124         if isDrawing:
125             isDrawing = False
126             xf = x
127             yf = y
128
129         # Normalizar las coordenadas
130         lastRoi = tratarCordROI()
131
132
133         # Trata el ROI
134         lastRoi = tratarCordROI()
135
136         #Mostrar el ROI
137         cv.imshow('ROI', lastRoi)
138
139     # Dibujar un rectangulo como si fuera el ROI
140     def drawRectROI(f, xi, yi, xf, yf, color):
141         cv.rectangle(f, (xi, yi), (xf, yf), color)
142
143     #Imprime el texto en pos
144     def printText(frame, text, pos):
```

```
144             cv.putText(frame, text, pos, fontFace=cv.FONT_HERSHEY_COMPLEX, fontScale=0.55, color=(0, 0, 255))
145
146     # FPS
147     def getFps(nFrames, startTime):
148         fps = nFrames / (time.time() - startTime)
149         return round(fps, 2)
150
151     # Devuelve el valor inicial del filtro
152     def getvInicial(selectedKey):
153         value = 0
154
155         if selectedKey == BRILLO:
156             value = 1.6
157
158         if selectedKey == DOBLE:
159             value = 3
160
161         if selectedKey == BHOR or selectedKey == BVER or selectedKey == SGAU:
162             value = 1
163
164     return value
165
166     # Tratamiento de los usos de las flechas para aumentar o reducir
167     def cambiarFilterValue(selectedKey, key):
168         global filterValue
169
170         # Brillo
171         if selectedKey == BRILLO:
172             if key == UP:
173                 return filterValue + 0.2
174             if key == DOWN:
175                 return filterValue - 0.2
176
177         # Resto de filtros
178         elif selectedKey == DOBLE or selectedKey == BHOR or selectedKey == BVER or selectedKey == SGAU:
179             if key == UP:
180                 return filterValue + 1
181             if key == DOWN:
182                 return filterValue - 1
183
184     return filterValue
185
186     # Matriz de convolución con el filtro
187     def getMatrizConv(selectedKey):
188         global filterValue
189
190         #Brillo
191         if selectedKey == BRILLO:
192             if filterValue <= 0:
193                 filterValue = 0
```

```
194         ker = np.array([[ 0,  0,  0]
195                           ,[ 0, filterValue,  0]
196                           ,[ 0,  0,  0]]))
197
198 #Imagen doble
199 elif selectedKey == DOBLE:
200
201     if filterValue <= 1:
202         filterValue = 1
203
204     ker = np.zeros([filterValue,filterValue])
205     ker[0,0] = 1
206     ker[filterValue-1,filterValue-1] = 1
207     ker = ker/np.sum(ker)
208
209 #Bordes horizontales
210 elif selectedKey == BHOR:
211     ker = np.array([[ 0,  0,  0]
212                               ,[-filterValue,  0, filterValue]
213                               ,[ 0,  0,  0]]))
214
215 #Bordes verticales
216 elif selectedKey == BVER:
217     ker = np.array([[ 0, -filterValue,  0]
218                               ,[ 0,  0,  0]
219                               ,[ 0, filterValue,  0]]))
220
221     return ker
222
223 #Aplica el suavizado Gaussiano a una imagen
224 def gaussianBlur(frameFiltro):
225     global filterValue
226
227     if filterValue <= 1:
228         filterValue = 1
229
230     return cv.GaussianBlur(frameFiltro, (0,0), filterValue)
231
232
233 def numAsFilterName(fnum):
234     filter = ""
235
236     if fnum == BRILLO:
237         filter = "BRILLO"
238     elif fnum == DOBLE:
239         filter = "DOBLE"
240     elif fnum == BHOR:
241         filter = "BORDE HORIZONTAL"
242     elif fnum == BVER:
243         filter = "BORDE VERTICAL"
```

```

244     elif fnum == SGAU:
245         filter = "SUAVIZADO GAUSSIANO"
246
247     return filter
248
249
250 #Aplica el filtro correspondiente al ROI seleccionado
251 def aplicarFiltro(capture, xi, yi, xf, yf, selectedKey, key):
252     global filterValue
253
254     frameFiltro = frame2gray(capture)
255
256     #Dibuja ROI al que aplicar el filtro
257     if selection:
258         drawRectROI(frameFiltro, xi, yi, xf, yf, (255,0,0))
259
260     #Si es la primera vez que se inicia
261     if InitFilterValue:
262         filterValue = getvInicial(selectedKey)
263
264     #Aumenta/reduce el valor del filtro
265     filterValue = cambiarFilterValue(selectedKey, key)
266
267     # Si no se aplica el suavizado Gaussiano
268     # se calcula la matriz de convolucion
269
270     if selectedKey != SGAU:
271         ker = getMatrizConv(selectedKey)
272         frame = cconv(ker, frameFiltro[yi:yf, xi:xf])
273     else:
274         frame = gaussianBlur(frameFiltro[yi:yf, xi:xf])
275
276
277     frameFiltro[yi:yf, xi:xf] = frame
278     text = numAsFilterName(selectedKey) + ": " + str(round(filterValue, 1))
279
280     return frameFiltro, text
281
282
283
284 cv.namedWindow("cam")
285 cv.setMouseCallback("cam", captureMouse)
286
287 capture = cv.VideoCapture(0)
288
289 seg = 1
290 nFrames = 0
291 fps = 0
292 selectedKey = ''
293 startTime = time.time()

```

```
294 while(True):
295     key = cv.waitKey(1) & 0xFF
296
297     if key == 27: break
298
299
300     #Tecla x para hacer desaparecer el ROI y el filtro
301     if key == ord('x'):
302         selection = False
303         selectedKey = ''
304         InitFilterValue= True
305
306     # Q para quitar el ROI pero dejar el filtro
307     if key == ord('q'):
308         selectedKey = ''
309         InitFilterValue= True
310
311     # Obtener frame en tono de grises
312     frame = frame2gray(capture)
313
314     # Si hay ROI se dibuja
315     if selection:
316         drawRectROI(frame, xi, yi, xf, yf, BLUE)
317
318     #Filtros
319     if selectedKey == BRILLO or key == ord(BRILLO):
320         selectedKey = BRILLO
321         frame, text = aplicarFiltro(capture, xi, yi, xf, yf, selectedKey, key)
322
323     elif selectedKey == DOBLE or key == ord(DOBLE):
324         selectedKey = DOBLE
325         frame, text = aplicarFiltro(capture, xi, yi, xf, yf, selectedKey, key)
326
327     elif selectedKey == BHOR or key == ord(BHOR):
328         selectedKey = BHOR
329         frame, text = aplicarFiltro(capture, xi, yi, xf, yf, selectedKey, key)
330
331     elif selectedKey == BVER or key == ord(BVER):
332         selectedKey = BVER
333         frame, text = aplicarFiltro(capture, xi, yi, xf, yf, selectedKey, key)
334
335     elif selectedKey == SGAU or key == ord(SGAU):
336         selectedKey = SGAU
337         frame, text = aplicarFiltro(capture, xi, yi, xf, yf, selectedKey, key)
338
339
340     #Imprime información del filtro
341     if selectedKey != '' and selection:
342         printText(frame, text, pos=(5, 40))
343         InitFilterValue = False
```

```
344
345 #FPS
346 nFrames += 1
347 if ((time.time() - startTime) > seg):
348     fps = getFps(nFrames, startTime)
349     nFrames = 0
350     startTime = time.time()
351
352     printText(frame, "FPS: " + str(fps), pos=(5, 20))
353
354 cv.imshow('cam', frame)
355
356
357 cv.destroyAllWindows()
```

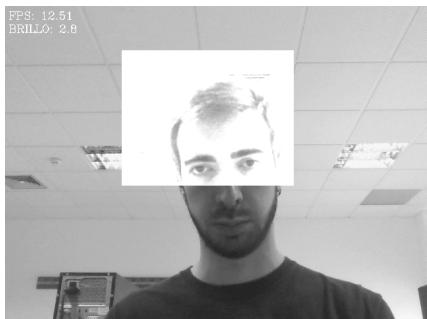


Figure 8. Brillo

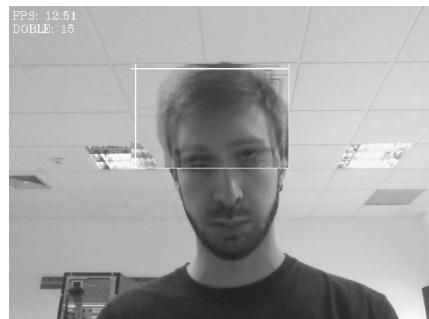


Figure 9. Doble



Figure 10. Borde Horizontal



Figure 11. Borde Vertical



Figure 12. Suavizado Gaussiano

6 Ejercicio 11

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

El programa es capaz de mostrar el sentido y la rotación de la cámara respecto a un frame en concreto.

También, cuando la trayectoria se pierde y no encuentra contornos se reinicia de nuevo.

```
1 #!/usr/bin/env python
2
3 import cv2           as cv
4 from umucv.stream import autoStream
5 from umucv.util import putText
6
7
8 COLOR_LINEAS = (0,255,255)
9
10
11 # Para las coincidencias
12 sX = 0
13 sY = 0
14
15 stream = autoStream()
16
17 # Para calcular el centro de la imagen
18 height, width = next(stream)[1].shape[:2]
19 centroImg = (height//2,width//2)
20
21 medir = True
22 for key,frame in stream:
23     gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
24
25     # Obtener las esquinas
26     # Si hay mucha perdida (lost) tambien se recalculan
27     if medir or key == ord('c') or len(corners)/2<=lost:
28         corners = cv.goodFeaturesToTrack(gray, 50, 0.1, 10).reshape(-1,2)
29         nextPts = corners
30         prevgray = gray
31         medir = False
32
33     # Flujo optico del frame actual y la imagen obtenida
34     nextPts, status, err = cv.calcOpticalFlowPyrLK(prevgray, gray, nextPts, None)
35     lost=0
36     prevgray = gray
37
38     # Sumar vectores en el caso de que las coincidencias sean buenas
39     # Si no aumentar la perdida
40     for (x,y), ok, (x0,y0) in zip(nextPts, status, corners):
41         if ok:
```

```
42             sX += x - x0
43             sY += y - y0
44     else:
45         lost +=1
46
47 # Calcular distancia media
48 tam = len(corners)
49 distMedia = (int(-sX/tam),int(sY*-1/tam))
50
51 #Longitud de la flecha a dibujar
52 finArrow = (distMedia[0] + centroImg[0], distMedia[1] + centroImg[1])
53
54 # Circulo en el centro de la imagen
55 cv.circle(frame, centroImg, 2, COLOR_LINEAS, -1, cv.LINE_AA)
56
57 # Flecha indicando la direccion de la imagen
58 cv.arrowedLine(frame, centroImg, finArrow, COLOR_LINEAS, 1, cv.LINE_AA)
59
60 # Reinciar las sumas de las coincidencias
61 sX = 0
62 sY = 0
63
64 putText(frame, 'Esquinas : {}'.format(len(corners)), (5,20), color=COLOR_LINEAS)
65 cv.imshow('Rotacion',frame)
66
67
68 cv.destroyAllWindows()
```

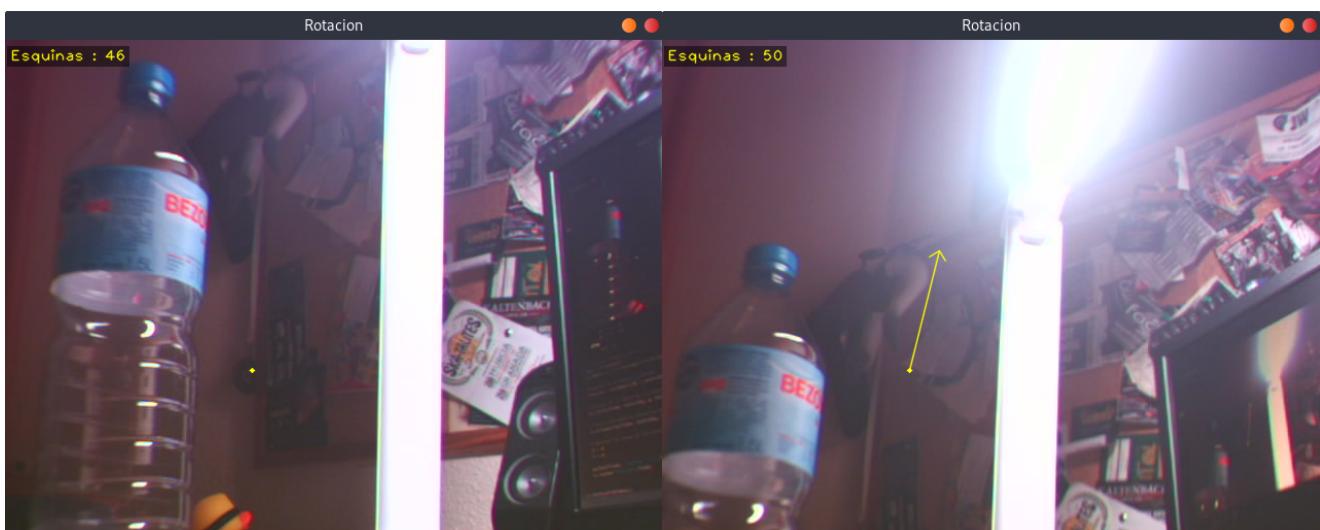


Figure 13. Imagen Incial

Figure 14. Movimiento hacia arriba

7 Ejercicio 12

Escribe una aplicación de reconocimiento de objetos con la webcam basada en el número de coincidencias de keypoints. Pulsando una tecla se pueden ir guardando modelos (p. ej. carátulas de CD, portadas de libros, cuadros de pintores, etc.). Cuando detectamos que la imagen está bastante quieta, o cuando se pulse otra tecla, calculamos los puntos de interés de la imagen actual y sus descriptores y comprobamos si hay suficientes coincidencias con los de algún modelo.

```
1 #!/usr/bin/env python3
2
3 # Reconocimiento de objetos usando sus "descriptores"
4 # Uso de SIFT, método para extraer puntos de interés
5
6 # Uso del programa:
7 # c : añade la iamgen a la lista de modelos a comparar
8 # v : comprueba si el frame actual se corresponde con algun modelo previamente
9 # guardado, si es asi imprime por pantalla el frame y su modelo
10 # x : limpia la lista de modelos a comparar
11
12
13
14 import cv2 as cv
15 import time
16
17 from umucv.stream import autoStream
18 from umucv.util import putText
19
20 # Filtro para SIFT en regiones de bajo contraste
21 # cuando mayor es el filtro más "features" son detectadas
22 CONTRAST_THRESHOLD = 0.05
23
24
25 # Scale invariant feature transfrom
26 sift = cv.xfeatures2d.SIFT_create(nfeatures=0,
27                                     contrastThreshold=CONTRAST_THRESHOLD)
28
29 # buscador de coincidencias por fuerza bruta
30 matcher = cv.BFMatcher()
31
32 # Lista para guardar los modelos de la forma (keypoints,descriptors,image)
33 models = []
34
35
36 def cleanModelWindows():
37     for window in range (0,len(models)+1):
38         cv.destroyWindow("Model" + str(window))
39
40
41 for key, x in autoStream():
```

```
43     # Vaciar la lista de modelos
44     if key == ord('x'):
45         cleanModelWindows()
46         models = []
47
48
49     t0 = time.time()
50     keypoints , descriptors = sift.detectAndCompute(x, mask=None)
51     t1 = time.time()
52     putText(x, '{} {:.0f}ms'.format(len(keypoints), 1000*(t1-t0)))
53
54     # añadimos una imagen de referencia, con sus puntos y descriptores
55     if key == ord('c'):
56         models.append((keypoints, descriptors, x));
57
58
59     # Mostramos por pantalla la camara
60     flag = cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
61     cv.drawKeypoints(x,keypoints,x, color=(100,150,255), flags=flag)
62     cv.imshow('CAM', x)
63
64
65
66     if key == ord('v'):
67         cleanModelWindows()
68
69     # Si hay modelos para comparar
70     if models:
71
72         # Numero de modelo
73         modelCount = 0
74
75         # Mejor modelo encontrado
76         bestModelTest = []
77         bestModel = None
78         bestModelN = 0
79
80         # Para cada modelo comprobamos si se ajusta a la imagen actual
81         t2 = time.time() # Para calcular el tiempo
82         for m in models:
83
84             # metodo knn (k-nearest neigbors)
85             matches = matcher.knnMatch(descriptors, m[1], k=2) # dame las dos mejores coincidencias
86
87             good = []
88             for mat in matches:
89                 if len(mat) >= 2:
90                     best,second = mat
91                     # ratio test, se descartan los puntos cuya mejor coincidencia es parecida
92                     # a la segunda mejor
```

```
93             if best.distance < 0.75*second.distance:  
94                 good.append(best)  
95  
96             # Nos quedamos con el mejor modelo  
97             if len(good) > len(bestModelTest):  
98                 bestModel = m  
99                 bestModelTest = good  
100                bestModelN = modelCount  
101  
102                modelCount +=1  
103  
104            t3 = time.time() # Fin calculo del tiempo  
105  
106            # Mostrar por pantalla el modelo que más se ajusta a la imagen  
107            # si lo hubiese  
108            if bestModel != None:  
109                imgm = cv.drawMatches(x, keypoints, bestModel[2], bestModel[0], bestModelTest,  
110                                flags=0,  
111                                matchColor=(128,255,128),  
112                                singlePointColor = (128,128,128),  
113                                outImg=None)  
114  
115  
116            print("El frame se ha clasificado con el modelo " + str(bestModelN))  
117            putText(imgm , '{} {:.0f}ms'.format(len(bestModelTest),1000*(t3-t2)), (150,16), (20,16))  
118            cv.imshow("Model" + str(bestModelN),imgm)
```



Figure 15. Objeto a clasificar



Figure 16. Otro frame de la camara



Figure 17. Objeto clasificado

8 Ejercicio 15

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.

Para realizar este ejercicio se hace uso de la imagen "coins.png", la cual hay que introducir su ruta como parámetro.

A la hora de ejecutar hay que marcar las esquinas del DNI y luego se rectifica la imagen, después se pueden medir dos puntos en la imagen.

```
1 #!/usr/bin/env python
2
3 import numpy as np
4 import cv2 as cv
5 import sys
6
7
8 # Uso: python E15.py coins.png
9 # Tecla c para obtener medir de nuevo
10 # Ejercicio 15 : Rectifica la imagen de un plano para medir distancias usando coins.png
11
12
13 COLOR_LINEAS = (0,0,255)
14
15 # Medidas del carnet
16 ancho = 8.6
17 alto = 5.4
18 ratio = 1.6
19
20 # En la imagen
21 xv = 150
22 yv = 550
23 lado = 100
24
25 # Para leer el input
26 carnet = []
27 puntos = []
28
29 # leer coins.png
30 img = cv.imread(sys.argv[1], cv.IMREAD_COLOR)
31
32
33 def medirCarnet(event, x, y, flags, param):
34     if event == cv.EVENT_LBUTTONDOWN:
35         punto = [x, y]
36         carnet.append(punto)
37
38         # Dibujar lineas en la imagen que forman el contorno del carnet
```

```
39         cv.circle(img, (x, y), 5, COLOR_LINEAS, -1)
40     if (len(carnet) == 2):
41         cv.line(img, (carnet[0][0], carnet[0][1]), (carnet[1][0], carnet[1][1]), COLOR_LINEAS)
42     elif (len(carnet) == 3):
43         cv.line(img, (carnet[1][0], carnet[1][1]), (carnet[2][0], carnet[2][1]), COLOR_LINEAS)
44     elif (len(carnet) == 4):
45         cv.line(img, (carnet[2][0], carnet[2][1]), (carnet[3][0], carnet[3][1]), COLOR_LINEAS)
46         cv.line(img, (carnet[3][0], carnet[3][1]), (carnet[0][0], carnet[0][1]), COLOR_LINEAS)
47
48 def medirPuntos(event, x, y, flags, param):
49     if event == cv.EVENT_LBUTTONDOWN:
50         punto = [x, y]
51         if len(puntos)< 2:
52             puntos.append(punto)
53
54         #Dibujar los puntos
55         cv.circle(rec, (x, y), 5, COLOR_LINEAS, -1)
56
57     if (len(puntos) == 2):
58         cv.line(rec, (puntos[0][0], puntos[0][1]), (puntos[1][0], puntos[1][1]), COLOR_LINEAS)
59
60         # Teorema de pitagoras
61         if (puntos[0][0] >= puntos[1][0]):
62             c1 = puntos[0][0] - puntos[1][0]
63         else:
64             c1 = puntos[1][0] - puntos[0][0]
65
66         if (puntos[0][1] >= puntos[1][1]):
67             c2 = puntos[0][1] - puntos[1][1]
68         else:
69             c2 = puntos[1][1] - puntos[0][1]
70
71         dstPixel = np.sqrt(c1 * c1 + c2 * c2)
72
73         cv.putText(rec, repr(dstPixel) + ' pixeles', (0, 50), cv.FONT_HERSHEY_SIMPLEX, 1, COLOR_LINEAS)
74         # Pixel a centimetro
75         dstCm = dstPixel * alto / lado
76         cv.putText(rec, repr(dstCm) + ' centimetros', (0, 75), cv.FONT_HERSHEY_SIMPLEX, 1, COLOR_LINEAS)
77
78
79
80
81
82 real = np.array([
83     [float(xv), float(yv)],
84     [float(xv+lado*ratio), float(yv)],
85     [float(xv+lado*ratio), float(yv+lado)],
86     [float(xv), float(yv+lado)]])
```

```
89
90
91 dist = False
92 while True:
93     key = cv.waitKey(1) & 0xFF
94     if key == 27:
95         break
96     if key == ord('c') and dist:
97         puntos = []
98         rec = original.copy()
99
100
101     cv.setMouseCallback("coins", medirCarnet)
102     cv.imshow('coins', img)
103
104     if not dist and len(carnet) == 4:
105         view = np.array([
106             [float(carnet[0][0]), float(carnet[0][1])],
107             [float(carnet[1][0]), float(carnet[1][1])],
108             [float(carnet[2][0]), float(carnet[2][1])],
109             [float(carnet[3][0]), float(carnet[3][1])]])
110
111
112     H = cv.findHomography(view, real)[0]
113     original = cv.warpPerspective(img, H, (800, 800))
114     rec = original.copy()
115     dist = True
116
117 elif dist :
118     cv.setMouseCallback("rectificada", medirPuntos)
119     cv.imshow('rectificada', rec)
120
121
122 cv.destroyAllWindows()
```

Se puede observar que la distancia entre las dos monedas es de 11 centímetros aproximadamente.

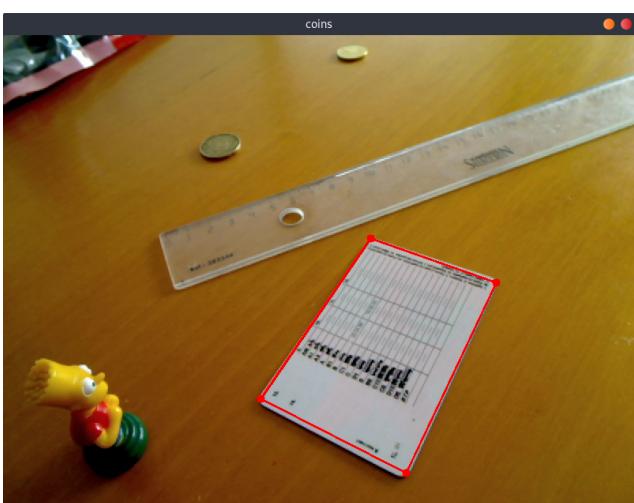


Figure 18. Medimos el dni

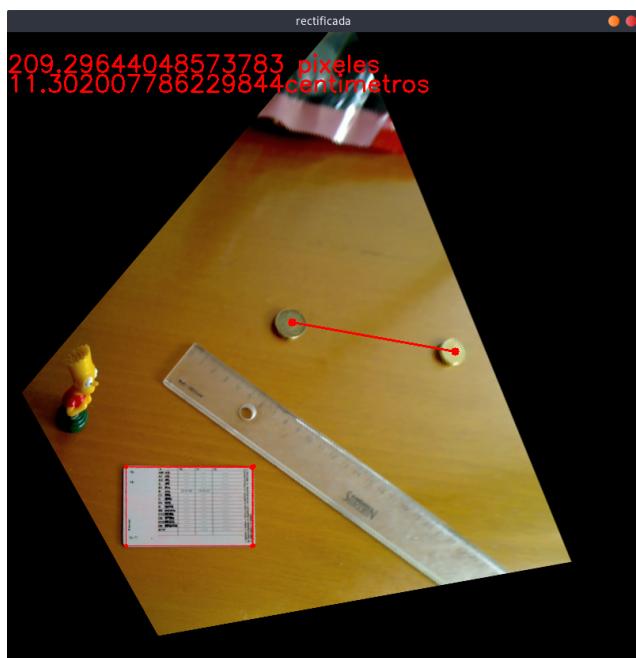


Figure 19. Medimos la moneda

9 Ejercicio 16

Crea automáticamente un mosaico panorámico ajustando varias imágenes (>2). Recuerda que debe tratarse de una escena plana o de una escena cualquiera vista desde el mismo centro de proyección.

```
1 #!/usr/bin/env python
2
3 import cv2 as cv
4 import numpy as np
5 import sys
6
7 # Uso: python E16.py img0 img1 imgN
8 # img es una imagen que se encuentra en la ruta relativa images/img
9
10 # Crea una imagen paronamica atraves de otras imagenes
11 # es necesario introducir las imagenes en orden (izquierda a derecha)
12
13
14 #lista para las imagenes
15 args = []
16
17 # Puntos de interes y descriptores
18 sift = cv.xfeatures2d.SIFT_create(nfeatures=0, contrastThreshold=0.1)
19
20 # Matcher (fuerza bruta)
21 matcher = cv.BFMatcher()
22
23
24 # Lee las imagenes y las transforma a tonos de grises
25 def tratInicial(file):
```

```
26         img = cv.cvtColor( cv.imread('images/'+file), cv.COLOR_BGR2RGB)
27         return cv.cvtColor(img, cv.COLOR_RGB2GRAY)
28
29
30 # Transformación h a la imagen img
31 def t(h,img):
32     return cv.warpPerspective(img, desp((50,150)) @ h,(1800,600))
33
34
35 # Desplazamiento
36 def desp(desp):
37     dx,dy = desp
38     return np.array([
39             [1,0,dx],
40             [0,1,dy],
41             [0,0,1]])
42
43
44 # Mejores coincidencias de los descriptores de la imagen
45 def ratioTest(des1, des0):
46     #coincidencias
47     good = []
48
49     matches = matcher.knnMatch(des1, des0, k=2)
50
51     #Mejor coincidencia
52     for m in matches:
53         best, second = m
54         if best.distance < 0.75 * second.distance:
55             good.append(best)
56
57     return good
58
59 #Combina dos imagenes
60 def union(img1, img2, H):
61     return np.maximum(t(np.eye(3),img2), t(H,img1))
62
63 #Crea una imagen panorámica a partir de la lista de imágenes pasada como parámetro
64 def panoramica(imagenes):
65     # Primera imagen
66     img1 = tratInicial(imagenes[0])
67
68
69     # Para todas las demás imágenes, formar la panorámica
70     for i in range(1, len(imagenes)):
71         img2 = tratInicial(imagenes[i])
72
73         #Obtenemos los vectores de descriptores
74         (kps, des0) = sift.detectAndCompute(img1, None)
75         (kps2, des1) = sift.detectAndCompute(img2, None)
```

```
76
77
78     #lista de las mejores coincidencias de las imágenes
79     good = ratioTest(des1, des0)
80
81     #Arrays que necesita findHomography
82     src_pts = np.array([ kps [m.trainIdx].pt for m in good ]).astype(np.float32).reshape(-1,1,2)
83     dst_pts = np.array([ kps2[m.queryIdx].pt for m in good ]).astype(np.float32).reshape(-1,1,2)
84
85     H, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 3)
86
87     img1 = union(img1, img2, H)
88
89     return img1
90
91
92
93
94
95 #Argumentos (imagenes)
96 args = sys.argv
97
98 # Elimino el primero que no corresponde a ninguna imagen
99 args.pop(0)
100
101 # Realizar panoramica
102 img = panoramica(args)
103
104
105 cv.namedWindow("Panoramica")
106
107 # Mostrar panoramica
108 while(True):
109     key = cv.waitKey(1) & 0xFF
110
111     if key == 27: break
112
113     cv.imshow('Panoramica',img)
114
115
116 cv.destroyAllWindows()
```



Figure 20. Imagen 0



Figure 21. Imagen 1



Figure 22. Imagen 2



Figure 23. Imagen panorámica obtenidas

10 Ejercicio 17

Realiza una calibración precisa de tu cámara mediante múltiples imágenes de un chessboard. Para realizar este ejercicio he hecho uso del script 'calibrate.py', usando unas imágenes de un tablero de ajedrez realizadas por la cámara de mi teléfono, la cual, use en el ejercicio 1.

La ejecución del programa devuelve esta salida:

```
RMS: 0.9912456894745283
camera matrix:
[[1.27548487e+03 0.00000000e+00 3.77424122e+02]
 [0.00000000e+00 1.39915468e+03 6.70875465e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
distortion coefficients: [-3.92208459e-01 1.67425865e+01 -2.01594303e-02 3.69
 557170e-03
 -1.32562766e+02]
```

Figure 24. Output calibrate.py

Sabemos que la f de mi cámara es de 3158,07 gracias al ejercicio 1 y la f obtenida con el script es de 1399. Se puede apreciar una notable diferencia lo que habría que realizar una calibración mejor.

11 Ejercicio 18

Crea un efecto de realidad aumentada dinámico.

Para este ejercicio se ha creado un muñeco azul que irá desplazándose dentro de un cubo y así obtener sensación de movimiento. Para ello se usa un marcador en forma de L donde estará situado el cubo.

```
1 #!/usr/bin/env python
2
3 # python E18.py --dev=file:../images/rot4.jpg
4
5 import cv2          as cv
6 import numpy        as np
7
8 from umucv.stream   import autoStream
9 from umucv.htrans   import htrans, Pose, kgen
10 from umucv.util     import lineType, cube
11 from umucv.contours import extractContours, redu
12
13 # Movimiento del muñeco
14 # Mas incremento mas velocidad
15 incrementoMov = 0.05
16
17 persona = np.array([
18     [0.5,0.25,0],
19     [0.5,0.25,0.5],
20     [1,0.25,0],
21     [0.5,0.25,0.5],
22
23     [0.5,0.25,0.75],
24     [0,0.25,0.55],
25     [0.5,0.25,0.75],
26     [1,0.25,0.75],
27     [0.5,0.25,0.75],
28
29     [0.5,0.25,1],
30     [0.25,0.25,1.15],
31     [0.5,0.25,1.25],
32     [0.75,0.25,1.15],
33     [0.5,0.25,1],
34     [0.5,0.25,0.75],
35
36 ])
37
38
39
40 stream = autoStream()
41 HEIGHT, WIDTH = next(stream)[1].shape[:2]
42 size = WIDTH,HEIGHT
```

```
43
44 K = kgen(size,1.7) # fov aprox 60 degree
45
46 marker = np.array(
47     [[0,    0,    0],
48      [0,    1,    0],
49      [0.5, 1,    0],
50      [0.5, 0.5, 0],
51      [1,    0.5, 0],
52      [1,    0,    0]])
53
54
55
56 def polygons(cs,n,prec=2):
57     rs = [ redu(c,prec) for c in cs ]
58     return [ r for r in rs if len(r) == n ]
59
60
61 def rots(c):
62     return [np.roll(c,k,0) for k in range(len(c))]
63
64
65 def bestPose(K,view,model):
66     poses = [ Pose(K, v.astype(float), model) for v in rots(view) ]
67     return sorted(poses,key=lambda p: p.rms)[0]
68
69 # Dirección en la que se mueve el muñeco
70 reverse = False
71
72 for key,frame in stream:
73     key = cv.waitKey(1) & 0xFF
74     if key == 27: break
75
76
77 g = cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
78 cs = extractContours(g, minarea=5, reduprec=2)
79
80 good = polygons(cs,6,3)
81
82 poses = []
83 for g in good:
84     pos = bestPose(K,g,marker)
85     if pos.rms < 2:
86         poses += [pos.M]
87
88 cv.drawContours(frame,[htrans(M,marker).astype(int) for M in poses], -1, (0,255,255), 1, lineType)
89 cv.drawContours(frame,[htrans(M,cube/2).astype(int) for M in poses], -1, (0,255,0), 1, lineType)
90
91 cv.drawContours(frame, [htrans(M,persona/3).astype(int) for M in poses], -1, (255,0,0), 3, lineType)
```

```
93     # Dirección del muñeco
94     if not reverse:
95         persona = persona+ [0,incrementoMov,0]
96         if persona[0][1] >= 1:
97             reverse = True
98     else:
99         persona = persona - [0,incrementoMov,0]
100        if persona[0][1] <= 0:
101            reverse = False
102
103
104    cv.imshow('Realidad Aumentada',frame)
```

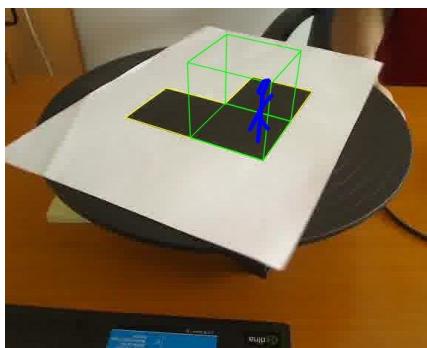


Figure 25. Muestra 1

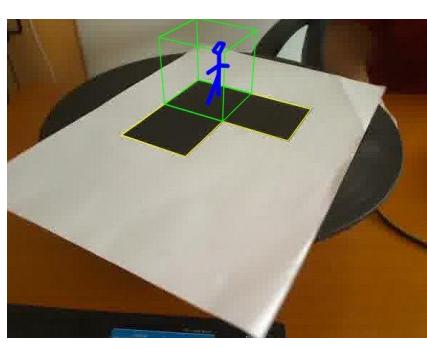


Figure 26. Muestra 2

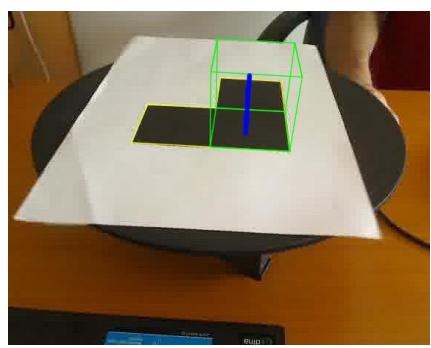


Figure 27. Muestra 3