Título

Índice general

Capítulo 1

Introducción

1.1. Motivación

Capítulo 2

OpenCV

2.1. Introducción

OpenCV es una librería de código abierto escrita en C y C++ destinada a la visión artificial y al tratamiento de imágenes. Se trata de una librería multiplataforma con versiones para GNU/Linux, Windows, Mac OS y Android y actualmente cuenta con interfaces para Python, Java y MATLAB/OCTAVE. Las últimas versiones incluyen soporte para GPU usando CUDA. Desarrollada originalmente por Intel, su primera versión alfa se publicó en el año 2000 en el *IEEE Conference on Computer Vision and Pattern Recognition*. OpenCV nació inicialmente como un proyecto para avanzar en las aplicaciones de uso intenso de la CPU y dando gran importancia a las aplicaciones en tiempo real. Hoy en día cuenta con más 2500 algoritmos optimizados que abarcan todo tipo de campos relacionados con la visión artificial. Estos algortimos pueden ser usados para tareas como: detectar y reconocer caras y gestos, identificar objetos, detección de características 2D y 3D, estimación de movimiento, seguimiento del movimiento, visión estéro y calibración de la cámara, eliminar los ojos rojos de las fotografías realizadas con flash...
OpenCV es ampliamente utilizada por todo tipo de empresas (desde grandes empresas como Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota a pequeñas empresas), grupos de investigación y organismos gubernamentales y en sectores de todo tipo como: inspección de los productos en las fábricas, seguridad, usos médicos, robótica...

2.2. Ejemplos

A continuación veremos una serie de ejemplos sencillos de uso de openCV con python, similares a los propuestos en el capítulo 2 de "Learning OpenCV"[1]

Ejemplo 1

En primer lugar veamos como mostrar una imagen en una ventana

```
import cv2
def load (name):
    img = cv2.imread(name)
    cv2.imshow('Example1',img)
    cv2.waitKey(0)

if __name__ == '__main__':
    import sys
    try: name = sys.argv[1]
    except:
        print("Error, introduce nombre del archivo")
        sys.exit()
    load(name)
```

ejemplos/ejemplo1.py

Como vemos se trata de un código muy sencillo. Al ser ejecutado con un argumento, carga una imagen y la muestra en una ventana; luego espera hasta que el usuario pulse una tecla.

La función cv2.imread() es la que se encarga de cargar la imagen desde un archivo.

Según la documentación de openCV, la función cv2.imread(filename[, flags]) toma como parámetros:

- filename: Es el nombre del archivo que se quiere abrir.
- flags: Especifican el modo de color en que se carga la imagen:
  - CV_LOAD_IMAGE_ANYDEPTH: Devuelve una imagen de 16-bit/32-bit cuando la entrada tiene la correspondiente profundidad, en otro caso la convierte a 8-bit.
  - CV_LOAD_IMAGE_COLOR: Convierte la imagen a color
  - CV_LOAD_IMAGE_GRAYSCALE: Convierte la imagen a escala de grises
  - > 0 Devuelve una imagen de 3 canales de color.
  - $= 0$ Devuelve una imagen en escala de grises.

- < 0 Return the loaded image as is (with alpha channel).

La función devuelve la imagen (en forma de array de numpy).
Una vez hemos cargado la imagen, la mostramos en una ventana usando cv2 .imshow(winname, image), donde winname es el nombre de la ventana e image es la imagen que queremos mostrar. Por último, utilizamos cv2.waitKey(0) para esperar a que el usuario pulse una tecla. Esta función recibe un único parámetro que indica el tiempo, en milisegundos, a esperar para la pulsación de una tecla. Si este parámetro es menor o igual que cero, esperará indefinidamente.

Ejemplo 2

En el siguiente ejemplo vemos como cargar un archivo de video y mostrarlo en una ventana. El programa termina cuando el video se acaba o cuando el usuario pulsa la tecla Esc

```
import cv2
def load (name):
    capture =  cv2.VideoCapture(name)
    while (True):
        ret, frame = capture.read()
        if (not ret): break
        cv2.imshow("Example2",frame)
        c = cv2.waitKey(33)
        if (c==27): break


if __name__ == '__main__':
    import sys
    try: name = sys.argv[1]
    except:
        print("Error, introduce nombre del archivo")
        sys.exit()
    load(name)
```

ejemplos/ejemplo2.py

En este ejemplo hemos hecho uso de la clase VideoCapture. La función cv2.VideoCapture(filename) recibe como argumento el nombre de un archivo de vídeo o la ID de un dispositivo de video (en este caso un nombre de archivo) y devuelve un objeto de la clase VideoCapture. De esta clase tan solo usamos ahora la función cv2.VideoCapture.read() que toma el siguiente fotograma, lo decodifica y lo devuelve. Además de la imagen, devuelve un booleano, que en el código hemos llamado ret, que es False si ningún fotograma ha sido tomado y True en otro caso. Como ahora ya tenemos una imagen, para mostrarla por pantalla usamos la misma función que en el ejemplo 1. Por último esta vez usamos la función cv2.waitKey de manera un poco distinta al anterior ejemplo. Como ahora estamos en un bucle, esperamos a la pulsación 33 milisegundos. Si alguna tecla es pulsada, su valor ASCII se almacena y lo comparamos con 27 que es el correspondiente a la tecla Esc.

Ejemplo 3

En este ejemplo mejoraremos un poco el reproductor de video que hemos programado en el ejemplo anterior. Se ha añadido un slider con el instante del video en que nos encontramos, el cual se puede utilizar para avanzar y retroceder hasta donde deseemos.

```
import cv2
global g_slider_position

def nothing(*arg):
    pass

def onTrackbarSlide(pos):
    g_capture.set(cv2.cv.CV_CAP_PROP_POS_FRAMES, pos)

def load (name):
    g_slider_position=0
    global g_capture
    g_capture =  cv2.VideoCapture(name)
    frames = int(g_capture.get(cv2.cv.CV_CAP_PROP_FRAME_COUNT))
    if (frames != 0):
        cv2.namedWindow("Example3")
        cv2.createTrackbar("Position","Example3",
    g_slider_position,frames,onTrackbarSlide)
    while (True):
        ret, frame = g_capture.read()
        if (not ret): break
        cv2.imshow("Example3",frame)
        g_slider_position = int(g_capture.get(cv2.cv.
    CV_CAP_PROP_POS_FRAMES))

        cv2.setTrackbarPos("Position","Example3",
    g_slider_position)
        c = cv2.waitKey(33)
        if (c==27): break


if __name__ == '__main__':
    import sys
    try: name = sys.argv[1]
    except:
        print("Error, introduce nombre del archivo")
        sys.exit()
    load(name)
```

ejemplos/ejemplo3.py

Como vemos, en este código hemos introducido unas cuantas funciones nuevas. En primer lugar, hemos usado varias veces la función cv2.VideoCapture.get (propId) con distintos argumentos. Esta función devuelve el valor de distintas propiedades del video con el que estamos trabajando. El argumento es el identificador de la propiedad y puede ser uno de los siguientes según la documentación:

- CV_CAP_PROP_POS_MSEC Posición actual en el video en milisegundos o el "timestamp"del capturador de video.
- CV_CAP_PROP_POS_FRAMES Posición del próximo fotograma que va a ser decodificado/capturado.
- CV_CAP_PROP_POS_AVI_RATIO Posición relativa del video, siendo 0 el inicio del video y 1 el final.
- CV_CAP_PROP_FRAME_WIDTH Anchura de los fotogramas del video.
- CV_CAP_PROP_FRAME_HEIGHT Altura de los fotogramas del video.
- CV_CAP_PROP_FPS Fotogramas por segundo.
- CV_CAP_PROP_FOURCC Código de 4 caracteres del codec usado.
- CV_CAP_PROP_FRAME_COUNT Número de fotogramas en el archivo de video.
- CV_CAP_PROP_FORMAT Format of the Mat objects returned by retrieve() .
- CV_CAP_PROP_MODE Backend-specific value indicating the current capture mode.
- CV_CAP_PROP_BRIGHTNESS Brillo de la imagen (solo para cámaras).
- CV_CAP_PROP_CONTRAST Contraste de la imagen (solo para cámaras).
- CV_CAP_PROP_SATURATION Saturación de la imagen (solo para cámaras).
- CV_CAP_PROP_HUE Tono de la imagen (solo para cámaras).
- CV_CAP_PROP_GAIN Ganancia de la imagen (solo para cámaras)..
- CV_CAP_PROP_EXPOSURE Exposición (solo para cámaras).

- CV_CAP_PROP_CONVERT_RGB Boolean flags indicating whether images should be converted to RGB.

También usamos la función cv2.VideoCapture.set(propId, value), usada para fijar el valor de alguna de las propiedades del vídeo. Por tanto los valores que puede tomar propId son los mismos que para la función cv2.VideoCapture.get() y el argumento value indica el nuevo valor que tomará la propiedad indicada.

Una vez cargamos el video como ya sabemos y obtenemos el número total de frames con la función get como acabamos de ver, procedemos a crear una ventana con la función cv2.namedWindow(winname[, flags]). Hasta ahora no había hecho falta ya que cv2.imshow creaba una nueva ventana si no existía una con ese nombre. Ahora le vamos a añadir algo (una trackbar) a nuestra ventana, por lo que necesitamos que ya esté creada. Para esto usamos cv2.createTrackbar (trackbarName, windowName, value, count, onChange), siendo estos parámetros:

- trackbarname – El nombre de la trackbar creada.
- winname – Nombre de la ventana en la que queremos la trackbar.
- value – Variable que refleja la posición del slider.
- count – La posición máxima del slider. (La posición mñinima es 0 siempre)
- onChange – Función que será llamada cada vez que el slider cambie de posición.
  ******

Ejemplo 4

Ahora vamos a ver un ejemplo de una transformación muy sencilla, vamos a abrir una imagen y a aplicarle un desenfoque gaussiano y a mostrar tanto la imagen original como la resultante por pantalla

```
import cv2
def load (name):

    img = cv2.imread(name)
    cv2.imshow('Example4-in',img)

    out = cv2.GaussianBlur(img,(9,9), 0)
    cv2.imshow("Example4-out", out)
    cv2.waitKey(0)

if __name__ == '__main__':
    import sys
    try: name = sys.argv[1]
    except:
        print("Error, introduce nombre del archivo")
        sys.exit()
    load(name)
```

ejemplos/ejemplo4.py

La única función nueva es:
cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]), cuyos argumentos son:

Ejemplo 5

En este ejemplo realizamos otra transformación, en este caso reducimos la imagen a la mitad. Esta tarea la realiza la función cv2.pyrDown

```
import cv2

def doPyrDown(im):
    h, w = im.shape[:2]
    assert (w%2 == 0 and h%2 == 0)
    out= cv2.pyrDown(im)
    return out


if __name__ == '__main__':
    import sys
    try: name = sys.argv[1]
    except:
        print("Error, introduce nombre del archivo")
        sys.exit()

    img = cv2.imread(name)
    out = doPyrDown(img)
    cv2.imshow('Example5-in',img)
    cv2.imshow("Example5-out", out)
    cv2.waitKey(0)
```

ejemplos/ejemplo5.py

Como vemos, en el resto del código no hay nada nuevo, a excepción de la función ya nombrada.

Ejemplo 6

En este ejemplo utilizamos el algoritmo de Canny [2] de detección de bordes, usando para ello la función cv2.Canny

```
import cv2
import numpy as np

def doCanny(im, lowThresh, highThresh, aperture):
    if (len(im.shape) != 2):
        aux =  cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
        out = cv2.Canny(aux, lowThresh, highThresh, aperture)
    else:
        out = cv2.Canny(im, lowThresh, highThresh, aperture)
    return out

if __name__ == '__main__':
    import sys
    try: name = sys.argv[1]
    except:
        print("Error, introduce nombre del archivo")
        sys.exit()

    img = cv2.imread(name)
    out = doCanny(img, 10, 100, 3)
    cv2.imshow('Example6-in',img)
    cv2.imshow("Example6-out", out)
    cv2.waitKey(0)
```

ejemplos/ejemplo6.py

Ejemplo 7

Ahora simplemente vamos a combinar los dos ejemplos anteriores, reduciendo la imagen original dos veces con pyrDown y luego encontrando los bordes con Canny

```
import cv2
import numpy as np

def doCanny(im, lowThresh, highThresh, aperture):
    if (len(im.shape) != 2):
        aux =  cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
        out = cv2.Canny(aux, lowThresh, highThresh, aperture)
    else:
        out = cv2.Canny(im, lowThresh, highThresh, aperture)
    return out

def doPyrDown(im):
    h, w = im.shape[:2]
    assert (w%2 == 0 and h%2 == 0)
    out= cv2.pyrDown(im)
    return out

if __name__ == '__main__':
    import sys
    try: name = sys.argv[1]
    except:
        print("Error, introduce nombre del archivo")
        sys.exit()

    img = cv2.imread(name)
    img1 = doPyrDown(img)
    img2 = doPyrDown(img1)
    out = doCanny(img2, 10, 100, 3)
    cv2.imshow('Example7-in',img)
    cv2.imshow("Example7-out", out)
    cv2.waitKey(0)
```

ejemplos/ejemplo7.py

Ejemplo 9

A continuación vemos un ejemplo que muestra la imagen de la webcam.

```
import cv2

if __name__ == '__main__':

    capture =  cv2.VideoCapture(0)

    while (True):
        ret, frame = capture.read()
        if (not ret): break
        cv2.imshow("Example9",frame)
        c = cv2.waitKey(33)
        if (c==27): break
```

ejemplos/ejemplo9.py

Como vemos el código es exactamente el mismo que el de mostrar un vñideo por pantalla, excepto que a la funcion cv2.VideoCapture le pasamos como argumento, en lugar del nombre del archivo, un entero que representa el índice de la cámara (si solo hay una cámara, debe ser 0).

```
import cv2
global r
def call(pos):
    global r
    r = pos
    return
def main ():
    global r
    r = 3
    cv2.namedWindow("Blur")
    capture =  cv2.VideoCapture(0)
    cv2.createTrackbar("Radio","Blur",r,30,call)
    while (True):
        ret, frame = capture.read()
        if (not ret): break
        if ( r!=0):
            out = cv2.blur(frame, (r,r))
            cv2.imshow("Blur",out)
        else:
            cv2.imshow("Blur",frame)
        c = cv2.waitKey(33)
        if (c==27): break
    cv2.destroyAllWindows()
main()
```

ejemplos/BlurCam.py

Capítulo 3

Otras librerías usadas

3.1. Android

Android es un sistema operativo desarrollado por Google y orientado a móviles y tablets. Está basado en linux y es de código abierto. Actualmente se estima que en torno un 80 % de los dispositívos móviles usan el sistema operativo Android. Su núcleo está programado en C, pero la aplicaciones y toda la interfaz de usuario se programan en Java. Este sistema operativo está estructurado en 4 capas:

Núcleo linux
: Se encarga de las funcionalidades básicas del sistema, como manejo de procesos, de la memoria y de los dispositivos como la cámara, la pantalla, etc. Asimismo funciona como una capa de abstracción entre el hardware y el resto del software.

Librerías y Runtime de Android
: Por encima del núcleo, hay una serie de librerías usadas por componentes del sistema, entre ellas destacan Surface Manager, Media Framework, SQLite, WebKit, SGL y Open GL
El runtime de Android proporciona un componente clave,la máquina virtual Dalvik. Cada aplicación Android corre su propio proceso, con su propia instancia de esta máquina virtual. Además, el runtime de android proporciona librerías básicas que proporcionan la mayor parte de las funciones del lenguaje de programación Java.

Framework de aplicaciones
: Esta capa proporciona a las aplicaciones muchos servicios en forma de clases de Java

Aplicaciones
: En esta capa se encuentran tanto las aplicaciones base como las que instalemos y desarrollemos ...

Estructura de una aplicación Android

Las aplicaciones android están por los llamados componentes de aplicación. Hay cuatro tipo de componentes:

Activities
: Representa una única pantalla de la aplicación con su interfaz de usuario.

Services
: Son componentes que se ejecutan en segundo plano y que no constan de interfaz gráfica.

Content providers

Broadcast receivers

Capítulo 4

Estabilización de imagen

4.1. Algoritmo

Se trata de un algoritmo básico de estabilización de imagen que se basa en la suposición de que el único movimiento de la cámara es el que queremos eliminar; es decir, excepto por ligeros movimientos no deseados, la cámara está estática. La idea del algoritmo es muy sencilla: Tomamos dos frames consecutivos y buscamos una serie de puntos característicos mediante el algoritmo propuesto por Shi y Tomashi [4], usando la función: goodFeaturesToTrack; después vemos a donde se han movido esos puntos en el siguiente frame mediante el algoritmo de flujo óptico de Lucas-Kanade. Finalmente calculamos la homografía que lleva los puntos originales a donde hemos calculado que se han movido y le aplicamos la inversa de esa transformación al segundo frame para colocarlo donde debería estar.

4.1.1. Encontrar puntos característicos

como ya hemos dicho, queremos encontrar una serie de puntos característicos que puedan ser localizados bien para poder encontrarlos en el siguiente fotograma. Es evidente que no vale cualquier punto, por ejemplo, si tenemos una pared completamente blanca, será muy difícil saber a donde se ha movido un punto cualquiera de la pared de un fotograma a otro. Por este motivo se introduce el concepto de buenos puntos característicos (good features to track). Un primer acercamiento a definir qué puntos eran buenos lo dieron Chris Harris y Mike Stephens en "A Combined Corner and Edge Detector"[3]. La idea básica es encontrar la diferencia de la intensidad de la

imagen en un rectángulo centrado en (x, y) . Esto se expresa con la función:

$$E(x,y) = \sum_{u,v} w(u,v)[I(u+x, v+y) - I(u,v)]^2 \tag{4.1}$$

Donde $w(x, y)$ es una función ventana, que asocia un peso a cada punto, e $I(x, y)$ es la intensidad de la imagen en cada punto. Por tanto si E(x,y) es suficientemente grande, (x,y) será un buen punto. Aplicando el desarrollo en serie de Taylor a $I(u + x, v + y)$ obtenemos:

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y \tag{4.2}$$

Donde I_x e I_y son las derivadas parciales en x e y respectivamente. Esto nos lleva a la aproximación:

$$E(x,y) \approx \sum_{u,v} w(u,v)[I_x(u,v)x + I_y(u,v)y]^2$$

que matricialmente se expresa como:

$$E(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} M \begin{pmatrix} x \\ y \end{pmatrix} \tag{4.3}$$

, siendo M:

$$M = \sum_{u,v} w(u,v) \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$$

Como nuestro objetivo era que E(x,y) fuese lo suficientemente grande, si estudiamos los autovalores de M, lo que querremos es que ambos sean suficientemente grandes. Sean λ_1 y λ_2 dichos autovalores, entonces:

- Si $\lambda_1 \approx 0$ y $\lambda_2 \approx 0$ entones el punto no quiere interés.
- Si $\lambda_1 \approx 0$ y λ_2 es positivos y suficientemente grande entonces hemos encontrado un borde.
- Si λ_1 y λ_2 son positivos y suficientemen grandes entonces hemos encontrado un punto de interés. Este punto es la intersección de dos bordes,por ello, estos puntos también son llamados esquinas .

Como función para valorar la calidad de los autovalores, Shi y Tomasi propusieron:

$$R = min(\lambda_1, \lambda_2)$$

en lugar de la propuesta por Harris:

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

La función propuesta por Shi y Tomasi proporciona mejores resultados que la original. De hacer todo esto se encarga la función cv2.goodFeaturesToTrack(), que calculas las segundas derivadas utilizando el operador sobel y luego calcula los autovalores. Sus argumentos son:
cv2.goodFeaturesToTrack(image, maxCorners, qualityLevel, minDistance[, corners[, mask [, blockSize[, useHarrisDetector[, k]]]]])

- image: La imagen en la que queremos buscar los puntos (en escala de grises).
- maxCorners: el máximo número de puntos característicos que queremos.
- qualityLevel: La calidad mínima que deben tener los puntos. Es un valor entre 0 y 1. Este parámetro es multiplicado por el mayor valor de calidad encontrado y sólo las esquinas que superen dicho valor serán devueltas.
- minDistance: distancia (euclídea) mínima que tiene que existir entre los puntos devueltos.
- corners (opcional): la lista de salida con los puntos encontrados.
- mask (opcional):Región de interés en la que se buscarán los puntos.
- blockSize (opcional):Tamaño del bloque para calcular la matriz M.
- useHarrisDetector (opcional): Parámetro que indica si queremos usar el detector de Harris (en lugar de el de Shi y Tomasi).
- k: El parámetro para el detector de Harris.

4.1.2. Algoritmo de Lucas-Kanade

El algortimo de Lucas-Kanade es un método que sirve para calcular el flujo óptico. El flujo óptico es el patrón de movimiento aparente de los objetos entre un fotograma y el siguiente causado por un movimiento de la cámara o de los objetos. Se trata de un campo vectorial de dos dimensiones donde cada vector es un vector de desplazamiento de un punto entre ambos fotogramas. Este algoritmo está basado en tres suposiciones:

1. Brillo constante: La apariencia de un pixel no cambia entre un frame y otro. Para una imagen en escala de grises esto es equivalente a que su brillo permanece constante

2. Los objetos no se mueven muy rápido.

3. Coherencia espacial:Los píxeles vecinos tienen movimiento similar.

El hecho de que el brillo no varie, es decir, que la intensidad de un pixel se mantiene no lleva a la siguente igualdad:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{4.4}$$

De donde obtenemos:

$$I_x u + I_y v + I_t = 0 \tag{4.5}$$

, donde $u = \frac{dx}{dt}$ y $v = \frac{dy}{dt}$ y siendo I_x e I_y los gradientes de la imagen inicial y análogamente I_t el gradiente sobre el tiempo. La ecuación 4.5 se conoce como la ecuación del flujo óptico y no puede ser resuelta para un solo punto ya que hay dos incógnitas. Para resolverla utilizamos la suposición de la coherencia espacial, entonces podemos tomar una ventana en torno al punto y obtener un sistema de ecuaciones de la forma $Aw = -b$:

$$\begin{pmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_n) \end{pmatrix} \tag{4.6}$$

El algoritmo de Lucas-Kanade obtiene una solución utlizando el ajuste por mínimos cuadrados. Resuelve el sistema:

$$A^t A w = A^t b$$

es decir:

$$\begin{pmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix}$$

y la solución a esta ecuación es $\begin{pmatrix} u \\ v \end{pmatrix} = (A^t A)^{-1} A^t b$

4.1.3. Código

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

import numpy as np
import cv2
if __name__ == '__main__':

    import sys
    try: name = sys.argv[1]
    except:
        print("Error, introduce nombre del archivo de entrada")
        sys.exit()
    cap = cv2.VideoCapture(name)

    # params for ShiTomasi corner detection
    feature_params = dict( maxCorners = 10000,
                           qualityLevel = 0.001,
                           minDistance = 50,
                           blockSize = 30 )

    # Parameters for lucas kanade optical flow
    lk_params = dict( winSize  = (30,30),
                      maxLevel = 4,
                      criteria = (cv2.TERM_CRITERIA_EPS | cv2.
   TERM_CRITERIA_COUNT, 10, 0.03))

    # Create some random colors
    color = np.random.randint(0,255,(10000,3))

    draw = False

    # Take first frame and find corners in it
    ret, old_frame = cap.read()
    old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
    p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **
   feature_params)

    # Create a mask image for drawing purposes
    mask = np.zeros_like(old_frame)

    cv2.imshow('original',old_frame)
    cv2.imshow('stabilized',old_frame)

    while(1):
        ret,frame = cap.read()
        cv2.imshow('original',frame)
        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # calculate optical flow
```

```
        p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray,
    frame_gray, p0, None, **lk_params)

        # Select good points
        good_new = p1[st==1]
        good_old = p0[st==1]

        h, w = old_frame.shape[:2]

        H, status = cv2.findHomography(p0, p1, cv2.RANSAC, 0)

        overlay = cv2.warpPerspective(frame, H, (w, h),  flags=
    cv2.INTER_LINEAR + cv2.WARP_INVERSE_MAP)


        # draw the tracks
        old_frame = overlay.copy()
        if draw:
            for i,(new,old) in enumerate(zip(good_new,good_old))
    :
                a,b = new.ravel()
                c,d = old.ravel()
                cv2.circle(overlay,(a,b),5,color[i].tolist(),-1)


        cv2.imshow('stabilized',overlay)
        k = cv2.waitKey(30) & 0xff
        if k == 27:
            break
        if k == ord('d'):
            draw = not draw

        # Now update the previous frame and previous points

        old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
        p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **
    feature_params)

    cv2.destroyAllWindows()
    cap.release()
```

estabilizacion.py

4.2. Uso del acelerometro para estabilizar

Se trata de intentar utilizar los datos extra de los que disponemos, es decir, los que nos proporciona el acelerómetro del móvil para intentar esta-

bilizar la imagen utilizando estos datos. Para ello en primer lugar hacemos un programa para android que se encarga de recoger estos datos, grabando el vídeo a la vez que registra los datos del acelerómetro y los guarda en un archivo junto con el momento exacto en el que se han registrado los datos. Luego, un segundo programa en el ordenador procesa los archivos para intentar estabilizar la imagen: recoge los datos de aceleración en cada instante de media y aproxima el movimiento en dichos instantes. Luego calcula mediante interpolación cuanto se ha movido la cámara en el instante del fotograma y recoloca el fotograma de acuerdo con lo obtenido.

4.2.1. Programa para Android

4.2.2. Programa de procesado de los datos

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

import numpy as np
import cv2
import cv2.cv as cv


def get_array(filename):
    acc_file = open( filename, "r" )
    array = []
    first = True
    old_date = 0.0
    for line in acc_file:
        if first:
            aux_date = line.split(":")
            t0 = ((int(aux_date[0])*60 + int(aux_date[1]))*60 +
   float(aux_date[2]))
            first = False
        else:
            aux = line.split()
            date_str = aux[2]
            aux_date = date_str.split(":")
            #date = [int(aux_date[0]), int(aux_date[1]), float(
   aux_date[2])]
            #get the time in seconds
            date = ((int(aux_date[0])*60 + int(aux_date[1]))*60
   + float(aux_date[2])) - t0
            fline = [float(aux[0]), float(aux[1]), date]
            #esto es un poco cutre habria que cambiarlo
            if date != old_date:
                array.append( fline )
                old_date = date
```

```
    acc_file.close()
    return array

def calculate_movement(filename):
    array = get_array(filename)
    t0 = 0.0
    x0 = 0.0
    vx0 = 0.0
    y0 = 0.0
    vy0 = 0.0
    dx = [(0.0,0.0)]
    dy = [(0.0,0.0)]
    for line in array:
        ax = line[0]
        ay = line[1]
        t1 = line[2]
        vx1 = vx0 + ax*(t1-t0)
        vy1 = vy0 + ay*(t1-t0)
        x1 = x0 + (vx0 + vx1)*(t1-t0)/2
        y1 = y0 + (vy0 + vy1)*(t1-t0)/2
        dx.append((t1, x1))
        dy.append((t1, y1))
        x0 = x1
        y0 = y1
        t0 = t1
    return dx, dy


if __name__ == '__main__':

    import sys
    try:
        name = sys.argv[1]
        accname = sys.argv[2]
    except:
        print("Error, introduce nombre del archivo de entrada")
        sys.exit()
    cap = cv2.VideoCapture(name)
    x, y = calculate_movement(accname)
    print x,y
    fps = cap.get(cv2.cv.CV_CAP_PROP_FPS)
    size = (int(cap.get( cv.CV_CAP_PROP_FRAME_WIDTH)),int(cap.
   get(cv.CV_CAP_PROP_FRAME_HEIGHT)))
    print fps
    fps = 30
    t = 0.0

    # Create some random colors
    color = np.random.randint(0,255,(10000,3))
```

```
# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)


# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)

cv2.imshow('original',old_frame)
cv2.imshow('stabilized',old_frame)

while(1):
    ret,frame = cap.read()
    cv2.imshow('original',frame)
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)


    h, w = old_frame.shape[:2]

    #calculamos la traslacion
    t = t + 1.0/fps
    dx = np.interp(t,[d[0] for d in x], [d[1] for d in x])
    dy = np.interp(t,[d[0] for d in y], [d[1] for d in y])

    dx = dx*(138000.0)
    dy = dy*(138000.0)

    M = np.array([[1, 0, dx],[0, 1, dy]])


    overlay = cv2.warpAffine(frame, M, (w,h), flags=cv2.
INTER_LINEAR + cv2.WARP_INVERSE_MAP)


    old_frame = overlay.copy()


    cv2.imshow('stabilized',overlay)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

    # Now update the previous frame and previous points

    old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)

cv2.destroyAllWindows()
cap.release()
```

estabilizacion_acelerometro.py

4.2.3. Resultados

Bibliografía

[1] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly, 2008.

[2] John Canny. A computational aproach to edge detection. In *IEEE Trans. On Pattern Analysis and Machine Inteligence*, pages 679–698, 1986.

[3] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, page 147–151, 1988.

[4] Jianbo Shi and Carlo Tomasi. Good features to track. In Springer, editor, *9th IEEE Conference on Computer Vision and Pattern Recognition*, 1994.