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 CUDA, lo que nos permite ejecutar paralelamente funciones en la unidad de procesamiento gráfico o GPU de nuestro ordenador . 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: detección y reconocimiento de caras y gestos, identificación objetos, detección de características 2D y 3D, estimación y seguimiento del movimiento, visión estéreo y calibración de la cámara, eliminación 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...
Como ya hemos dicho, openCV incluye todo tipo de funciones e implementaciones de algoritmos. Dichas funciones se pueden dividir en varios grupos:

- Las relacionadas con el interfaz gráfico: se encargan de crear ventanas,

mostrar imágenes por pantalla, registrar las pulsaciones del teclado, y de crear la interfaz de usuario (botones, barras de desplazamiento, etc). Veremos las más importantes en los ejemplos de la sección siguiente.

- Las estructuras básicas como las matrices y las funciones para trabajar con ellas. En la versión para python esto no es muy importante porque está adaptada para trabajar con arrays de Numpy (una extesión para python que añade funciones para el cálculo científico y sobre la que hablaremos un poco más en detalle en el siguiente capítulo).

- Las funciones de procesado de imágenes. Es la parte fundamental de openCV y contiene fucniones de todo tipo:

  - Filtros de imagen, como desenfoques varios, ampliar o reducir una imagen, calcular el gradiente de una imagen mediante el operador Sobel...
    Veremos algunas de estas funciones en los ejemplos.
  - Transformaciones geométricas. Estas funciones realizan diversas tareas de carácter geométrico sobre las imágenes 2D, como calcular la transformación afín que lleva unos puntos a otros, calcular la matriz de una rotación o aplicarle a una imagen la matriz de una transformación.
  - Funciones relacionadas con el cálculo y manejo de histogramas (comparar dos histogramas, normalizar un histograma...).
  - Funciones de estimación de movimiento y detección de características. La más importantes son usadas y explicadas en el capítulo 4.

- Funciones de calibrado de la cámara (para corregir la distorsión provocada por la lente) y de reconstrucción 3D usando visión estéreo (dos cámaras).

- Funciones para detectar objetos y caras. Destaca la clase `CascadeClassifier` .

- Además, openCV consta de una librería de aprendizaje automático (*machine learning*) que está formada por funciones de clasificación estadística, árboles de decisión, redes neuronales...

Por supuesto openCV es una libería enorme y consta de muchísimas más funciones que las aquí mencionadas

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]*. No obstante en el libro estos ejemplos están programados en C con una versión antigua de la librería y aquí se presentan en Python y actualizados.

Ejemplo 1

En primer lugar veamos como mostrar una imagen en una ventana

```
1  import cv2
2  def load (name):
3      img = cv2.imread(name)
4      cv2.imshow('Example1',img)
5      cv2.waitKey(0)
6
7  if __name__ == '__main__':
8      import sys
9      try: name = sys.argv[1]
10     except:
11         print("Error, introduce nombre del archivo")
12         sys.exit()
13     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. Esta función toma como argumentos el nombre del archivo que se quiere abrir y además podemos especificar con un segundo parámetro opcional el modo de color en que se carga la imagen.

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 de espera en milisegundos 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 video 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 una barra de deslizamiento con el instante del video en que nos encontramos, el cual se puede utilizar para avanzar y retroceder hasta donde deseemos.

```
1  def nothing(*arg):
2      pass
3
4  def onTrackbarSlide(pos):
5      g_capture.set(cv2.cv.CV_CAP_PROP_POS_FRAMES,pos)
6
7  def load (name):
8      g_slider_position=0
9      global g_capture
10     g_capture =  cv2.VideoCapture(name)
11     frames = int(g_capture.get(cv2.cv.CV_CAP_PROP_FRAME_COUNT))
12     if (frames != 0):
13         cv2.namedWindow("Example3")
14         cv2.createTrackbar("Position","Example3",
       g_slider_position,frames,onTrackbarSlide)
15      while (True):
16         ret, frame = g_capture.read()
17         if (not ret): break
18         cv2.imshow("Example3",frame)
19         g_slider_position = int(g_capture.get(cv2.cv.
       CV_CAP_PROP_POS_FRAMES))
20
21         cv2.setTrackbarPos("Position","Example3",
       g_slider_position)
22         c = cv2.waitKey(33)
23         if (c==27): break
```

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 entre los que destacan: `CV_CAP_PROP_POS_FRAMES` (posición del próximo fotograma que va a ser decodificado/capturado), `CV_CAP_PROP_FPS` (fotogramas por segundo) y `CV_CAP_PROP_FRAME_WIDTH`

y `CV_CAP_PROP_FRAME_HEIGHT` (anchura y altura de los fotogramas del video respectivamente). La lista completa de opciones puede ser encontrada fácilmente en la documentacíon de openCV.

También usamos la función `cv2.VideoCapture.set(propId, value)`, usada para fijar el valor de alguna de las propiedades del video. 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 fotogramas 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 el nombre de la barra de deslizamiento creada, el nombre de la ventana en la que queremos la barra,una variable que refleja la posición del deslizador, la posición máxima del deslizador (la posición mínima es 0 siempre) y por último una 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. El desenfoque gaussiano es un filtro usado para suavizar imágenes y desenfocarlas. Funciona de la siguiente manera: para cada píxel de la imagen mezcla su color con los de un ventana a su alrededor (de altura y anchura dadas). Esta mezcla es una media, ponderada con la distribución Gaussiana, de los píxeles que están dentro de la ventana.

```
1  import cv2
2  def load (name):
3
4      img = cv2.imread(name)
5      cv2.imshow('Example4-in',img)
6
7      out = cv2.GaussianBlur(img,(9,9), 0)
8      cv2.imshow("Example4-out", out)
9      cv2.waitKey(0)
10
11 if __name__ == '__main__':
12     import sys
13     try: name = sys.argv[1]
14     except:
15         print("Error, introduce nombre del archivo")
16         sys.exit()
17     load(name)
```

ejemplos/ejemplo4.py

La única función nueva es `cv2.GaussianBlur()`, que es la que realiza el desenfoque.

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` Este algoritmo funciona de la siguiente manera: en primer lugar utiliza un desenfoque gaussiano a la imagen para reducir el ruido. Posteriormente calcula el gradiente de la imagen. Esto lo hace aplicando dos matrices de convolución para calcular las derivadas en las direcciones x e y:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Para finalmente calcular el gradiente y su dirección:

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \arctan(\frac{G_y}{G_x})$$

Por último, para decidir si los píxeles pertenecen a un borde, se utilizan dos umbrales, uno superior y otro inferior:

- Si el gradiente del píxel es mayor que el umbral superior, se acepta el píxel como borde.
- Si el gradiente del píxel es menor que el umbral inferior, se rechaza el píxel.
- Si el gradiente está entre ambos umbrales, entonces se acepta solo si está conectado a un píxel que está por encima del umbral superior.

A continuación vemos un ejemplo que muestra la imagen de la *webcam* y utiliza el algoritmo de Canny de detección de bordes.

```
import cv2

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__':

    capture =  cv2.VideoCapture(0)

    while (True):
        ret, frame = capture.read()
        if (not ret): break
        edgeimg = doCanny(frame, 30, 80, 3)
        out = 255- edgeimg
        cv2.imshow("Example9-in",frame)
        cv2.imshow("Example9-out",out)
        c = cv2.waitKey(33)
        if (c==27): break
```

ejemplos/ejemplo8.py

Como vemos el código para mostrar una imagen de la *webcam* es exactamente el mismo que el de mostrar un video por pantalla, excepto que a la función `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).

Ejemplo 7

Ahora simplemente vamos a combinar los dos ejemplos anteriores, reduciendo dos veces la imagen original 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 8

```
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/ejemplo9.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 a 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.

Marco de trabajo(*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. Como ya hemos dicho estas aplicaciones se programan en Java, disponiendo para ello del conjunto de herramientas Android SDK.

Estructura de una aplicación Android

Las aplicaciones Android están formadas 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
: Se encarga de compartir información entre distintas aplicaciones.

Broadcast receivers
: Este componente permite el registro de eventos del sistema

Una vez visto esto veamos por encima la estructura de una aplicación Android. Dichas aplicaciones vienen en formato APK, que no es más que un fichero comprimido ZIP en el que se encuentran: el código, los recursos, la firma digital y el fichero de manifiesto.
Los recursos se encuentran en la carpeta `res`. Además de las imágenes y otro tipo de contenido que usemos en la aplicación, esta carpeta contiene otra carpeta, llamada `layouts`, en la que se encuentran archivos XML que describen las distintas pantallas o vistas de cada aplicación, es decir las interfaces gráficas asociadas a cada actividad.
El archivo "`manifest.xml`" es un fichero XML donde se declaran los componentes que conforman la aplicación, así como se indican los permisos necesarios que requiere la aplicación, las funciones de *hardware* o *software* que se necesitan (la cámara, el acelerómetro, los servicios de *bluetooth*...) y algunás características más del programa.

3.2. Numpy

Numpy (NUMerical PYthon) es una extensión para Python que agrega un montón de funcionalidades para el calculo matemático. La más destacable que incorpora es el soporte para arrays N-dimensonales y funciones para operar con ellos. Además incluye numerosas funciones de álgebra lineal,transformadas de fourier, estadística básica...
El núcleo de Numpy es el objeto `ndarray`, que es la estructura de datos que representa los arrays N-dimensionales. Estos arrays, a diferencia de las listas de python, deben estar formados por elementos del mismo tipo. La manera más usual de crear estos arrays es a partir de una lista de Python: `numpy.array([1,2,3,4])`. También pueden usarse las funciones `numpy.zeros(n,m)` y `numpy.ones((n,m))` para crear matrices nxm formadas solo por ceros o por unos, respectivamente.
Los operadores aritméticos pueden usarse con los arrays y se aplican elemento a elemento y devuelven un nuevo array con el resultado. El operador `*` es la multiplicación elemento a elemento, para la multiplicación de matrices se usa la función `numpy.dot(A,B)`. Otros ejemplos de funciones son: `numpy.sum()`,`numpy.min()` y `numpy.max()`.

Los principales atributos de `ndarray` son:

- `ndarray.ndim`: el número de dimensiones (o ejes) del array.
- `ndarray.shape`: las dimensiones del array. Consiste en una tupla (de longitud `ndim`) de enteros que indican el tamaño del array en cada dimensión. Por ejemplo, para una matriz de n filas y m columnas, será (n,m).
- `ndarray.size`: el número total de elementos del array. Es el producto de todos los elementos de `shape`.
- `ndarray.dtype`: el tipo de los elementos del array. Se pueden crear o especificar tipos usando los tipos estándar de Python. Además Numpy proporciona algunos tipos más, como `numpy.int32`, `numpy.int16`, y `numpy.float64`
- `ndarray.data`: el búfer que contiene los elementos del array. Generalmente no se usa este atributo ya que se suele acceder a los elementos usando índices.

Capítulo 4

Estabilización de imagen

4.1. Algoritmo básico

Se trata de un algoritmo básico de estabilización de imagen para eliminar el posible movimiento de agitación la cámara. Dicho algoritmo 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 fotogramas consecutivos y buscamos una serie de puntos característicos mediante el algoritmo propuesto por Shi y Tomashi [5], usando la función: `goodFeaturesToTrack`; después vemos a donde se han movido esos puntos en el siguiente fotograma 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 fotograma para colocarlo donde debería estar.

4.1.1. Algoritmo para 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 $\lambda_2 \approx 0$ entonces el punto no tiene interés.
- Si $\lambda_1 \approx 0$ y λ_2 es positivo 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 [5] 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 calcula las segundas derivadas utilizando el operador Sobel (es el método que explicamos en el ejemplo 6 del capítulo 2 para calcular el gradiente de una imagen) 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 [4] 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 píxel 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 píxel se mantiene nos lleva a la siguente igualdad:

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

Como hemos supuesto que el movimiento entre dos fotogramas es relativamente pequeño, podemos aproximar por desarrollo de Taylor la parte derecha de la igualdad anterior, de donde obtenemos:

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

donde $u = \dfrac{dx}{dt}$ y $v = \dfrac{dy}{dt}$ 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 $\begin{pmatrix} u \\ v \end{pmatrix} = (A^t A)^{-1} A^t b$

El problema hasta ahora son las suposiciones de los movimientos pequeños y coherentes. Queremos usar una ventana grande para captar movimientos grandes pero esto puede romper dichas suposiciones. Para solventar esto aplicamos el algoritmo iterativamente a una pirámide de imágenes del fotograma. Primero se aplica el algoritmo a la imagen en la cima de la pirámide para seguir los movimiento grandes y luego según se va bajando en la pirámide se va refinando iterativamente el resultado partiendo del anterior. De esta manera los movimientos grandes se van convirtiendo en movimientos pequeños. Este algoritmo lo realiza openCV mediante la función:
`cv2.calcOpticalFlowPyrLK(prevImg, nextImg, prevPts[, nextPts[, status [, err[, winSize[, maxLevel[, criteria[, flags[, minEigThreshold ]]]]]]]]) → nextPts, status, err`

Donde:

- `prevImg` y `nextImg` son las dos imágenes entre las que queremos detectar movimiento y `prevPts` los puntos que queremos seguir.
- `nextPts` es el vector de salida que contiene las nuevas posiciones de los puntos de entrada. Si se usa el argumento `OPTFLOW_USE_INITIAL_FLOW` , este vector deberá tener el mismo tamaño que el de entrada.
- `status` es un vector de salida en el que cada elemento es 1 si el flujo para el punto correspondiente se ha encontrado y 0 en otro caso.
- `err` es un vector de salida que contiene en cada el elemento el error asociado a cada punto. El tipo de este error se puede fijar en el argumento `flags`. Si no se encuentra flujo el error no está definido.
- `winSize` es el tamaño de la ventana en cada nivel de la pirámide.
- `maxLevel` es el nivel máximo de niveles de la pirámide a usar empezando desde 0. Si es 0 no se usará pirámide; si es 1, se usarán dos niveles, etc.
- `criteria` es un parámetero que indica el criterio para terminar la búsqueda iterativa del algoritmo: después de un número máximo de iteraciones o cuando la ventana de búsqueda se mueve menos que un épsilon dado.

- `flags`:
  - `OPTFLOW_USE_INITIAL_FLOW` utiliza estimaciones iniciales, almacenadas en `nextPst`. Si no, copia `prevPts` a `nextPts` y los considera la estimación inicial.
  - `OPTFLOW_LK_GET_MIN_EIGENVALS` utiliza los autovalores mínimos como medida de error. Si este parámetro no se indica se usará como medida de error la distancia L_1 entre los bloques del punto original y un punto movido, dividido por el número de píxeles en una ventana.
- `minEigThreshold` – El algortimo calcula el menor autovalor de la matriz de las ecuaciones del flujo. Si este valor dividido por el número de píxeles en una ventana es menor que `minEigThreshold`, entonces el punto es filtrado y no se calcula su flujo.

4.1.3. Cálculo de la homografía

En visión artificial se llama homografía a una transformación proyectiva de un plano a otro.
Tenemos una serie de puntos $q_1 \dots q_n$ que van a parar a $q'_1 \dots q'_n$. Denotaremos $q_i = (x_i, y_i)$ y $q'_i = (x'_i, y'_i)$, Entonces podemos expresar la homografía como:

$$\begin{pmatrix} x'_i \\ y'_i \\ 1 \end{pmatrix} = sH \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \tag{4.7}$$

donde s es un factor de escala y H es la matriz de la transformación. Para encontrar dicha matriz usamos la función `cv.FindHomography()` que en un principio devuelve dicha matriz minimizando el error:

$$\sum_i \left(x'_i - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 + \left(y'_i - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2$$

Pero como no todos los puntos encajan en la transformación ya que puede haber errores u otros movimientos que no son los de la cámara utilizaremos un método robusto llamado RANSAC (RANdom SAmple Consensus). Se trata de un método iterativo para estimar los parámetros de un modelo matemático que contiene valores atípicos. La idea básica de este algoritmo es resolver varias veces el problema con distintos subconjuntos aleatorios de los puntos dados y luego tomar como solución particular la más cercana a la media/mediana de las soluciones.

4.1.4. Código

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

import numpy as np
import cv2
import cv

# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 10000,
                       qualityLevel = 0.001,
                       minDistance = 5,
                       blockSize = 3 )

# Parameters for lucas kanade optical flow
term = ( cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_COUNT, 30,
   0.001 )

lk_params = dict( winSize  = (50,50),
                  maxLevel = 4,
                  #criteria = (cv2.TERM_CRITERIA_EPS , 10,
   0.001)
                  criteria=term
)



def checkedTrace(img0, img1, p0, back_threshold = 1.0):
    p1, st, err = cv2.calcOpticalFlowPyrLK(img0, img1, p0, None
   , **lk_params)
    p0r, st, err = cv2.calcOpticalFlowPyrLK(img1, img0, p1,
   None, **lk_params)
    d = abs(p0-p0r).reshape(-1, 2).max(-1)
    status = d < back_threshold
    return p1, status


if __name__ == '__main__':
    frames = []
    import sys
    try:
```

```
36          name = sys.argv[1]
37          nout = sys.argv[2]
38      except:
39          print("Error, introduce los nombre de los ficheros de
    entrada y salida")
40          sys.exit()
41
42      cap = cv2.VideoCapture(name)
43      fps = cap.get(cv2.cv.CV_CAP_PROP_FPS)/2
44      print fps
45      size = (int(cap.get( cv.CV_CAP_PROP_FRAME_WIDTH)),int(cap.
    get(cv.CV_CAP_PROP_FRAME_HEIGHT)))
46      nframes = int(cap.get(cv.CV_CAP_PROP_FRAME_COUNT))
47      print nframes
48
49      # Create some random colors
50      color = np.random.randint(0,255,(10000,3))
51
52      draw = False
53
54      # Take first frame and find corners in it
55      ret, old_frame = cap.read()
56      old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
57      p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **
    feature_params)
58      cv2.cornerSubPix(old_gray, p0, (5, 5), (-1, -1), term)
59
60
61      # Create a mask image for drawing purposes
62      mask = np.zeros_like(old_frame)
63
64      #writer.write(old_frame)
65      frames.append(old_frame)
66
67      count = 1
68
69      while(True):
70          porcentaje = 100*count/nframes
71          print porcentaje, "% completed....................."
72          ret,frame = cap.read()
73
```

```
74          if not ret:
75              break
76
77          frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
78
79          # calculate optical flow
80          #p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray,
    frame_gray, p0, None, **lk_params)
81
82          p2, trace_status = checkedTrace(old_gray, frame_gray,
    p0)
83
84          #good_new = p1[st==1]
85          #good_old = p0[st==1]
86
87          p1 = p2[trace_status].copy()
88          p0 = p0[trace_status].copy()
89
90          h, w = old_frame.shape[:2]
91
92      try:
93              H, status = cv2.findHomography(p0, p1, cv2.RANSAC)
94      except:
95          pass
96
97          old_frame = cv2.warpPerspective(frame, H, (w, h),
    flags=cv2.INTER_LINEAR + cv2.WARP_INVERSE_MAP)
98          frames.append(old_frame)
99
100         # Now update the previous frame and previous points
101
102         old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
103         p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **
    feature_params)
104         cv2.cornerSubPix(old_gray, p0, (5, 5), (-1, -1), term)
105         count += 1
106
107     writer = cv2.VideoWriter(nout, cv.CV_FOURCC('M','J','P','G'
    ), fps, size)
108     print len(frames)
109     for frm in frames:
```

```
110         writer.write(frm)
111     cv2.destroyAllWindows()
112     cap.release()
```

estabilizacion.py

4.1.5. Comentarios y variaciones del algoritmo

Este algoritmo funciona bastante bien en la mayoría de los casos; sin embargo cuando el movimiento (ya sea el de la imagen o el de lo que estamos grabando) es muy brusco, puede producir efectos no deseados y ser contraproducente. Si asumimos que la camara no sufre rotaciones y solo hay traslación, en lugar de estimar la homografía podemos usar una traslación de razón la media o la mediana de lo que e ha movido cada píxel detectado. La media no es muy recomendable sin algún filtro, ya que es muy sensible a los valores atípicos y si ha objetos moviéndose se producirá traslación aun cuando la cámara este completamente quieta. Sin embargo la media funciona bastante bien y es más rápida que el algoritmo original. Sin embargo si no requerimos rapidez no supone una gran mejora.

4.2. Algoritmo basado en la correlación de fase

Otro algoritmo bastante utilizado y más rápido que el anterior está basado en la correlación de fase. Supone que la cámara no sufre rotaciones y que el único movimiento no deseado es de traslación y emplea el nombrado método para calcular esta traslación.
La correlación de fase es un método de registro de imagen que utiliza el dominio de la frecuencia para estimar la traslación entre dos imágenes similares. Lo hace de la siguiente forma:

Dadas dos imágenes (g_a y g_b), se les aplica una función ventana y se calcula la transformada discreta de Fourier de ambas imágenes:

$$\mathbf{G}_a = \mathcal{F}\{g_a\},\ \mathbf{G}_b = \mathcal{F}\{g_b\}$$

Calculamos la densidad espectral cruzada y normalizamos por su producto.

$$R = \frac{\mathbf{G}_a \circ \mathbf{G}_b^*}{|\mathbf{G}_a \mathbf{G}_b^*|}$$

Donde $\circ$ es el producto de Hadamard.

Obtenemos la correlación cruzada normalizada aplicando la inversa de la transformada de Fourier

$$r = \mathcal{F}^{-1}\{R\}$$

Por último, localizamos el máximo r.

$$(\Delta x, \Delta y) = \arg\max_{(x,y)}\{r\}$$

OpenCV dispone de la función `cv2.phaseCorrelate` que se encarga de todo esto.

4.3. Uso del acelerómetro 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 estabilizar la imagen utilizando estos datos. Para ello en primer lugar hacemos un programa para Android que se encargue 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.

4.3.1. Programa para Android

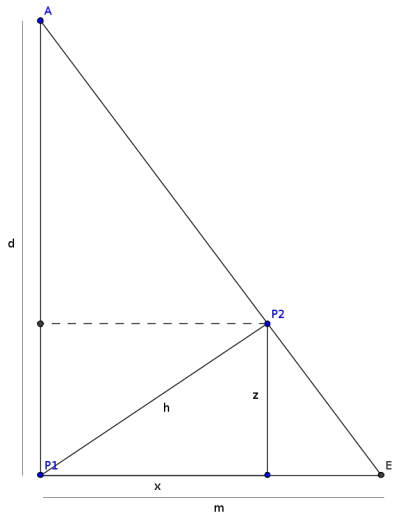
Se trata de un programa bastante simple. Al abrirlo desde el móvil vemos la pantalla en negro (aunque con un marco blanco alrededor). Si pulsamos el boton de subir volumen, empezamos a ver vídeo por pantalla y el dispositivo empieza a grabar. Cuando le volvemos a dar al botón de subir volumen, el programa termina la grabación y guarda dos archivos: uno de vídeo y otro de texto con la siguiente información: en la primera línea el instante en el que comenzamos a grabar y en las siguientes lineas, separados por espacios las aceleraciones lineales medidas en cada eje, así como el instante en que han sido medidas. Ahora bien, el acelerómetro mide la aceleración que sufre el dispositivo, lo cual incluye la aceleración de la gravedad y, por tanto, tenemos que elminarla. Hay varias maneras de hacer esto, pero por fortuna, la mayoría de móviles andorid actuales incluyen un tipo de sensor llamado `TYPE_LINEAR_ACCELERATION` que realiza estos cálculos por nosotros. Suele ser una fusión de sensores que, utilizando el acelerómetro y el giroscopio, resta la aceleración de la gravedad a la medida por el acelerómetro. La parte de grabación de video es muy básica y hace uso de la clase `MediaRecorder`. El código completo se incluye al final de la memoría como apéndice.

4.3.2. Programa de procesado de los datos

Este programa recibe el video grabado y el fichero con los datos del acelerómetro,es decir tenemos una serie de instantes t_i y las aceleraciones medidas en dichos instantes a_{xi}, a_{yi}, a_{zi}. Como lo que queremos en realidad es el desplazamiento en cada instante lo aproximamos usando la siguiente fórmula:

$$\begin{cases} v_{xi} = v_{x,i-1} + a_{xi}t_i \\ \\ x_i = x_{i-1} + \dfrac{v_{x,i-1} + v_{xi}}{2}t_i \end{cases} \tag{4.8}$$

tomando $v_{x0} = 0$ y $x_0 = 0$ Para calcular y_i y z_i se realiza de manera equivalente.
Para conocer el desplazamiento en el instante de cada fotograma utilizamos `numpy.interp()` para calcular la interpolación lineal del conjunto de datos que tenemos en el instante deseado. Ya sabemos el desplazamiento dela cámara en un instante dado, pero queremos saber el movimiento relativo en la imagen. Para esto, suponemos que estamos grabando a un objeto a una distancia d conocida y que dicho objeto está centrado en la imagen. Veamos desde arriba la situación, siendo A el objeto, $P1$ la posición inicial de la cámara y $P2$ a donde se ha movido.

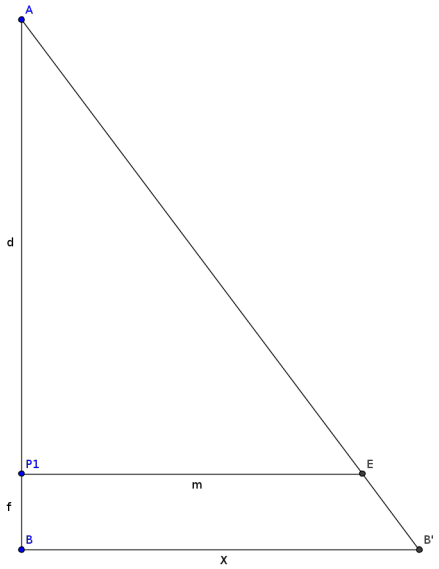


Tenemos, usando el teorema de Pitágoras:

$$h = \sqrt{x^2 + z^2}$$

Y por el Teorema de Tales:

$$m = \frac{h}{d - z} d$$



Por último, de nuevo por Tales y por lo anterior, obtenemos que

$$X = \frac{m}{d}(d + f)$$

siendo f la distancia focal de la cámara. Para obtener el desplazamiento en la imagen en el eje y, los cálculos son análogos.

```
#!/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[3]
            aux_date = date_str.split(":")
            #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]),float(aux[2])
   , 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
```

```
38      vy0 = 0.0
39      z0 = 0.0
40      vz0 = 0.0
41      dx = [(0.0,0.0)]
42      dy = [(0.0,0.0)]
43      dz = [(0.0,0.0)]
44      for line in array:
45          ax = line[0]
46          ay = line[1]
47          az = line[2]
48          t1 = line[3]
49          vx1 = vx0 + ax*(t1-t0)
50          vy1 = vy0 + ay*(t1-t0)
51          vz1 = vz0 + az*(t1-t0)
52          x1 = x0 + (vx0 + vx1)*(t1-t0)/2
53          y1 = y0 + (vy0 + vy1)*(t1-t0)/2
54          z1 = z0 + (vz0 + vz1)*(t1-t0)/2
55          dx.append((t1, x1))
56          dy.append((t1, y1))
57          dz.append((t1, z1))
58          x0 = x1
59          y0 = y1
60          z0 = z1
61          t0 = t1
62      return dx, dy, dz
63
64  def recalculate_pos(x, z, d, f):
65      h = np.sqrt(x*x+z*z)
66      m = h*d/(d-z)
67      return (m/d)*(d+f)
68
69  if __name__ == '__main__':
70
71      import sys
72      try:
73          name = sys.argv[1]
74          accname = sys.argv[2]
75      except:
76          print("Error, introduce nombre del archivo de entrada")
77          sys.exit()
78      cap = cv2.VideoCapture(name)
```

```
79      x, y, z = calculate_movement(accname)
80
81
82      #esto hay que cambiarlo
83      fps = 30.0
84
85      t = 0.0
86      f = 0.004
87      d = 10
88
89      # Create some random colors
90      color = np.random.randint(0,255,(10000,3))
91
92      # Take first frame and find corners in it
93      ret, old_frame = cap.read()
94      old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
95
96
97      # Create a mask image for drawing purposes
98      mask = np.zeros_like(old_frame)
99
100     cv2.imshow('original',old_frame)
101     cv2.imshow('stabilized',old_frame)
102
103     while(1):
104         ret,frame = cap.read()
105         cv2.imshow('original',frame)
106         frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
107
108
109         h, w = old_frame.shape[:2]
110
111         #calculamos la traslacion
112         t = t + 1/fps
113         dx = np.interp(t,[d[0] for d in x], [d[1] for d in x])
114         dy = np.interp(t,[d[0] for d in y], [d[1] for d in y])
115         dz = np.interp(t,[d[0] for d in z], [d[1] for d in z])
116
117         print dx,dy
118         #dx = dx*(13800.0)
119         #dy = dy*(13800.0)
```

```
120
121         dx = recalculate_pos(dx, dz, d, f)
122         dy = recalculate_pos(dy, dz, d, f)
123
124         M = np.array([[1, 0, dx],[0, 1, dy]])
125
126
127         overlay = cv2.warpAffine(frame, M, (w,h), flags=cv2.
    INTER_LINEAR + cv2.WARP_INVERSE_MAP)
128
129
130         old_frame = overlay.copy()
131
132
133         cv2.imshow('stabilized',overlay)
134         k = cv2.waitKey(30) & 0xff
135         if k == 27:
136             break
137
138         # Now update the previous frame and previous points
139
140         old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
141
142     cv2.destroyAllWindows()
143     cap.release()
```

estabilizacion_acelerometro.py

4.3.3. Resultados

Apéndice A

Código de Android

A continuación se incluyen las partes más importantes del código de la aplicación de Android.

MainActivity

```
1  package com.mobvacc.videorecorder;
2
3  import android.os.Bundle;
4  import android.app.Activity;
5  import android.view.Menu;
6
7  import java.io.File;
8  import java.io.FileNotFoundException;
9  import java.io.FileOutputStream;
10 import java.io.IOException;
11 import java.text.SimpleDateFormat;
12 import java.util.Date;
13 import com.mobvacc.videorecorder.R;
14 import com.mobvacc.videorecorder.R.id;
15 import com.mobvacc.videorecorder.R.layout;
16
17
18 import android.content.Context;
19 import android.content.pm.ActivityInfo;
20 import android.hardware.Sensor;
21 import android.hardware.SensorEvent;
22 import android.hardware.SensorEventListener;
23 import android.hardware.SensorManager;
```

```
24 import android.os.Environment;
25 import android.util.Log;
26 import android.view.KeyEvent;
27 import android.view.Window;
28
29 public class MainActivity extends Activity implements
    SensorEventListener{
30
31     private SensorManager mSensorManager;
32     private Sensor mAccelerometer;
33     private vcorderView camcorderView;
34     private boolean recording = false;
35     FileOutputStream outputAcc;
36
37     @Override
38     public void onCreate(Bundle savedInstanceState) {
39           super.onCreate(savedInstanceState);
40           requestWindowFeature(Window.FEATURE_NO_TITLE);
41           setRequestedOrientation(ActivityInfo.
    SCREEN_ORIENTATION_LANDSCAPE);
42           setContentView(R.layout.activity_main);
43           setRequestedOrientation(ActivityInfo.
    SCREEN_ORIENTATION_LANDSCAPE);
44           Date date=new Date();
45           String dateString =date.toString().replace(" ", "
    _").replace(":", "_");
46           String filename="/sdcard/rec"+dateString+".mp4";
47           String Accfilename="acc"+dateString+".txt";
48           //camera init
49           //
50           camcorderView = (vcorderView) findViewById(R.id.
    vcorderView1);
51           camcorderView.setOutputFile(filename);
52
53           //accelerometer init
54           mSensorManager = (SensorManager) getSystemService
    (Context.SENSOR_SERVICE);
55           mAccelerometer = mSensorManager.getDefaultSensor(
    Sensor.TYPE_LINEAR_ACCELERATION);
56
57           //open accelerometer file
```

```
58        File file = new File(Environment.
    getExternalStorageDirectory() + File.separator +
    Accfilename);
59
60        //write the bytes in file
61        if(!file.exists())
62        {
63             try {
64               file.createNewFile();
65           } catch (IOException e) {
66               e.printStackTrace();
67           }
68        }
69        try {
70           outputAcc = new FileOutputStream(file);
71       } catch (FileNotFoundException e) {
72           e.printStackTrace();
73       }
74
75     }
76
77
78     @Override
79     public boolean onKeyDown(int keyCode, KeyEvent event)
80     {
81         if (keyCode == KeyEvent.KEYCODE_VOLUME_UP)
82         {
83           if (recording) {
84                 camcorderView.stopRecording();
85                 mSensorManager.unregisterListener(this);
86                 try {
87                     outputAcc.close();
88                 } catch (IOException e) {
89                     e.printStackTrace();
90                 }
91                finish();
92            } else {
93                recording = true;
94                 mSensorManager.registerListener(this,
    mAccelerometer , SensorManager.SENSOR_DELAY_FASTEST);
95                camcorderView.startRecording();
```

```
96
97          SimpleDateFormat fecha = new SimpleDateFormat(
    "HH:mm:ss.SSS");
98          String fecha_str = fecha.format(new Date());
99          String output = fecha_str+"\n";
100         try {
101             outputAcc.write(output.getBytes());
102         } catch (IOException e) {
103             e.printStackTrace();
104         }
105
106        }
107         return true;
108     }
109     return super.onKeyDown(keyCode, event);
110 }
111
112 @Override
113 public void onAccuracyChanged(Sensor sensor, int
    accuracy) {
114
115 }
116
117 @Override
118 public void onSensorChanged(SensorEvent event) {
119
120     SimpleDateFormat fecha = new SimpleDateFormat("HH:
    mm:ss.SSS");
121     String fecha_str = fecha.format(new Date());
122
123     //esto es asi porque la pantalla esta girada
124     float x = event.values[1];
125     float y = event.values[0];
126     float z = event.values[2];
127     String output = Float.toString(x)+" "+Float.
    toString(y)+" "+" "+fecha_str+"\n";
128     try {
129         outputAcc.write(output.getBytes());
130     } catch (IOException e) {
131         e.printStackTrace();
132     }
```

```
133     }
134
135     @Override
136     public boolean onCreateOptionsMenu(Menu menu) {
137         // Inflate the menu; this adds items to the action
        bar if it is present.
138         getMenuInflater().inflate(R.menu.main, menu);
139         return true;
140     }
141
142 }
```

Android/MainActivity.java

Clase vcorderView

A continuación incluimos el código de la clase `vcorderView` que es la clase que se encarga de grabar de la cámara.

```
1  package com.mobvacc.videorecorder;
2
3  import java.io.IOException;
4
5  import android.content.Context;
6  import android.hardware.Camera;
7  import android.media.CamcorderProfile;
8  import android.media.MediaRecorder;
9  import android.util.AttributeSet;
10 import android.util.Log;
11 import android.view.SurfaceHolder;
12 import android.view.SurfaceView;
13
14 public class vcorderView extends SurfaceView implements
15 SurfaceHolder.Callback{
16
17     MediaRecorder recorder;
18     Camera camera;
19     SurfaceHolder holder;
20     String outputFile = "/sdcard/default.mp4";
21
22     public vcorderView(Context context)
23     {
24         super(context);
25         init();
26     }
27
28     public vcorderView(Context context, AttributeSet attrs
       ) {
29         super(context, attrs);
30         init();
31     }
32
33     public vcorderView(Context context, AttributeSet attrs
       , int defStyle) {
34         super(context, attrs, defStyle);
35         init();
```

```
36      }
37
38      public void init(){
39          holder = getHolder();
40          holder.addCallback(this);
41          holder.setType(SurfaceHolder.
    SURFACE_TYPE_PUSH_BUFFERS);
42
43
44
45          recorder = new MediaRecorder();
46
47
48          recorder.setVideoSource(MediaRecorder.VideoSource.
    DEFAULT);
49          recorder.setOutputFormat(MediaRecorder.
    OutputFormat.MPEG_4);
50          recorder.setVideoSize(720,480);
51          recorder.setVideoEncoder(MediaRecorder.
    VideoEncoder.DEFAULT);
52
53      }
54
55      public void surfaceCreated(SurfaceHolder holder) {
56          recorder.setOutputFile(outputFile);
57          recorder.setPreviewDisplay(holder.getSurface());
58          if (recorder != null) {
59              try {
60                  recorder.prepare();
61              } catch (IllegalStateException e) {
62                  Log.e("IllegalStateException", e.toString
    ());
63              } catch (IOException e) {
64                  Log.e("IOException", e.toString());
65              }
66          }
67      }
68
69      public void surfaceChanged(SurfaceHolder holder, int
    format, int width,
70              int height) {
```

```
71      }
72
73      public void surfaceDestroyed(SurfaceHolder holder) {
74      }
75
76      public void setOutputFile(String filename)
77      {
78          outputFile = filename;
79          recorder.setOutputFile(filename);
80      }
81
82      public void startRecording()
83      {
84          recorder.start();
85      }
86
87      public void stopRecording()
88      {
89          recorder.stop();
90          recorder.release();
91      }
92
93  }
```

Android/vcorderView.java

Android manifest

Ahora vemos el archivo `AndroidManifest.xml` en el que cabe destacar las líneas en las que hemos indicado los permisos especiales que necesitamos (el uso de la cámara y de poder escribir en el almacenamiento del móvil).

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/
       res/android"
3       package="com.mobvacc.videorecorder"
4       android:versionCode="1"
5       android:versionName="1.0" >
6       <uses-permission android:name="android.permission.
       CAMERA"></uses-permission>
7       <uses-permission android:name="android.permission.
       RECORD_AUDIO"></uses-permission>
```

```
8     <uses-permission android:name="android.permission.
    WRITE_EXTERNAL_STORAGE"></uses-permission>
9     <uses-feature android:name="android.hardware.camera" /
    >
10    <uses-feature android:name="android.hardware.camera.
    autofocus" />
11
12    <uses-sdk
13        android:minSdkVersion="9"
14        android:targetSdkVersion="18" />
15
16    <application
17        android:allowBackup="true"
18        android:icon="@drawable/ic_launcher"
19        android:label="@string/app_name"
20        android:theme="@style/AppTheme" >
21        <activity
22            android:name="com.mobvacc.videorecorder.
    MainActivity"
23            android:label="@string/app_name" >
24            <intent-filter>
25                <action android:name="android.intent.
    action.MAIN" />
26
27                <category android:name="android.intent.
    category.LAUNCHER" />
28            </intent-filter>
29        </activity>
30    </application>
31</manifest>
```

Android/AndroidManifest.xml

Activity layout

Por último vemos el archivo XML que describe la interfaz de nuestra única actividad (y que tan solo es un marco y la zona en la que mostraremos la imagen mientras estemos grabando).

```
1 <RelativeLayout xmlns:android="http://schemas.android.com/
    apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools"
3     android:orientation="vertical"
```

```
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     android:paddingBottom="@dimen/activity_vertical_margin
      "
7     android:paddingLeft="@dimen/activity_horizontal_margin
      "
8     android:paddingRight="@dimen/
      activity_horizontal_margin"
9     android:paddingTop="@dimen/activity_vertical_margin"
10    tools:context=".MainActivity" >
11
12    <com.mobvacc.videorecorder.vcorderView
13        android:id="@+id/vcorderView1"
14        android:layout_width="fill_parent"
15        android:layout_height="fill_parent"
16        android:enabled="false"
17        android:focusable="true"
18        android:clickable="true"/>
19
20 </RelativeLayout>
```

Android/res/layout/activity_main.xml

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] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of Imaging Understanding Workshop*, pages 121–130, 1981.

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