UNIVERSIDAD COMPLUTENSE
MADRID

FACULTAD DE MATEMÁTICAS

VISIÓN ARTIFICIAL CON OPENCV: ESTABILIZACIÓN DE IMAGEN Y LUZ

Andrés A. Sánchez González

Índice general

Motivación

El interés inicial y motivo principal para la realización de este trabajo fue el acercamiento al interesante mundo de la visión artificial. Los campos relacionados con la visión artificial son amplísimos, abarcando casi la totalidad de las actuales actividades que incorporan algún tipo de imagen a su tecnología . El verdadero problema inicial fue elegir en qué centrar el interés de las numerosísimas actividades y tareas relacionadas con la visión artificial. Por acotar un poco el terreno, mi interés dentro de la visión artificial estaba en algo que tuviese una aplicación real en el mundo. Acudí a la presentación de una aplicación para móviles llamada Mememtum [1]. Esta aplicación, que se encuentra aún en una primera versión, pretende proporcionar una manera de controlar la evolución del Parkinson y otros trastornos del movimiento a los pacientes desde su casa a través del móvil. Por supuesto, esta aplicación usa visión artificial; por ello, hablé con uno de los responsables del proyecto. Me dijo que algunos de los problemas que tenían en esta versión estaban relacionados con la estabilización de la imagen y la iluminación y, por ello, me sugirió que podía orientar mi trabajo en esta dirección.

Esta aplicación para el control del Parkinson consiste en que una segunda persona graba al enfermo de Parkinson y tras procesar el vídeo, devuelve un número que representa la cantidad de movimiento del paciente y que se puede utilizar para hacer un seguimiento del mismo. Podría considerarse como un "termómetro del Parkinson". El problema que surge con una aplicación para el móvil es que al movimiento del enfermo, hay que sumarle el posible movimiento involuntario de la cámara producido por la persona que está grabando. El reto, por tanto, está en intentar eliminar dicho movimiento involuntario. Otro de los problemas eran los posibles cambios de iluminación, que dificultan encontrar al paciente de manerfa eficaz. Por tanto hay que encontrar una manera de minimizar el efecto de los cambios de iluminación. Una idea que me sugirieron era, ya que estaban trabajando con móviles, intentar aprovechar los sensores de que estos disponen, como el acelerómetro y el sensor de luz, para corregir este movimiento.

[1] `http://mememtum.com`

Finalmente decidí realizar el trabajo en esta dirección, y por tanto centrarme en las tareas de: estabilización de la imagen con algoritmos de visión artificial, intentar estabilizar la imagen con ayuda de los datos del acelerómetro y estabilización de la iluminación, al igual que antes primero con algoritmos de visión artificial y luego con los datos extra del sensor de luz en este caso. En primer lugar, para poder realizar los algoritmos de visión artificial tuve que aprender a utilizar la librería OpenCV[3] y familiarizarme con algunos algoritmos típicos de visión artificial. Además como iba a intentar hacer uso del acelerómetro, tuve que aprender a realizar aplicaciones para Android y cómo interactuar con el hardware de los móviles. Tras todo este trabajo previo ya estaba en condiciones de centrarme en las tareas anteriormente mencionadas.

En resumen el trabajo ha consistido en:

- El estudio de la librería openCV.
- Una introducción a la programación en Android.
- Búsqueda de algoritmos para la estabilización de la imagen.
- Programar un algoritmo de estabilización basado en acelerómetros.
- Búsqueda de algoritmos para la estabilización de la luz.
- Programar un algoritmo para estabilizar la luz usando el sensor.

Por tanto, el trabajo está estructurado de la siguiente manera: en primer lugar un capítulo dando una breve visión de conjunto de OpenCV y algunos códigos sencillos de ejemplo que hice durante el proceso de aprendizaje. El siguiente capítulo está dedicado a otras librerías que, sin jugar un papel an protagonista como OpenCV, he tenido que aprender y usar durante el trabajo. La parte central de este capítulo es Android y una pequeña noción sobree cómo se estructuran este sistema operativo y sus aplicaciones. El siguiente capítulo trata sobre la estabilización de la imagen y abarca tanto los algoritmos puramente de visión artificial para estabilizar la imagen, como los intentos de usar el acelerómetro con este objeto. Finalmente, el último capítulo está dedicado a la estabilización de la luz.

Capítulo 1

OpenCV

1.1. Introducción

OpenCV[3] 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. Actualmente cuenta con interfaces para Python, Java y MATLAB/OCTAVE. Además 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 algoritmos 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[2] (una extensión para Python que añade funciones para el cálculo científico y sobre la que hablaremos un poco más en detalle en el siguiente capítulo en la sección 2.2).

- Las funciones de procesado de imágenes. Es la parte fundamental de openCV y contiene funciones 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. La clase más importante para esto es `CascadeClassifier` que puede ser entrenada mediante muestras para reconocer objetos.

- 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 librería enorme y consta de muchísimas más funciones que las aquí mencionadas

1.2. 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"[4]*. No obstante en el libro estos ejemplos están programados en C con una versión antigua de la librería (1.0) y aquí se presentan en Python y actualizados a la versión 2. Al tratarse de una librería tan extensa, es imposible cubrirla o resumirla en unos ejemplos. Por tanto, los códigos que vienen a continuación pretenden tan solo dar una idea de algunos funciones básicas a hora de trabajar con la librería como abrir y mostrar vídeos e imágenes y algunas transformaciones sencillas.

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  if __name__ == '__main__':
7      import sys
8      try: name = sys.argv[1]
9      except:
10         print("Error, introduce nombre del archivo")
11         sys.exit()
12     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 vídeo y mostrarlo en una ventana. El programa termina cuando el vídeo se acaba o cuando el usuario pulsa la tecla "Esc"

```
1  import cv2
2  def load (name):
3      capture =  cv2.VideoCapture(name)
4      ret = True
5      while (ret):
6          ret, frame = capture.read()
7          if (ret):
8              cv2.imshow("Example2",frame)
9              c = cv2.waitKey(33)
10             if (c==27):
11                 ret = False
12 if __name__ == '__main__':
13     import sys
14     try: name = sys.argv[1]
15     except:
16         print("Error, introduce nombre del archivo")
17         sys.exit()
18     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 vídeo que hemos programado en el ejemplo anterior. Se ha añadido una barra de deslizamiento con el instante del vídeo en que nos encontramos, el cual se puede utilizar para avanzar y retroceder hasta donde deseemos.

```
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))
    ret = True
    if (frames != 0):
        cv2.namedWindow("Example3")
        cv2.createTrackbar("Position","Example3",
                           g_slider_position,frames,
                           onTrackbarSlide)
    while (ret):
        ret, frame = g_capture.read()
        if (ret):
            cv2.imshow("Example3",frame)
            g_slider_position = int(g_capture.get(
                cv2.cv.CV_CAP_PROP_POS_FRAMES))
            cv2.setTrackbarPos("Position","Example3",
```

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 fácil-

mente en la documentación de OpenCV[1].

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 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 la barra de deslizamiento cambie de posición.

[1] `http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html#videocapture-get`

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: 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 de los píxeles que están dentro de la ventana ponderada con un núcleo gaussiano (es decir la media de los píxeles multiplicando cada uno de ellos por un peso, que en este caso lo da el núcleo gaussiano). Un núcleo gaussiano (de una dimensión) de tamaño `ksize` es una matriz de dimensiones `ksize` $\times 1$ con coeficientes: $G_i = \alpha \cdot e^{-(i-(\texttt{ksize}-1)/2)^2/(2*\sigma)^2}$ [2] donde i va de 0 a `ksize` -1 y α es tal que $\sum_i G_i = 1$. Como las imágenes son en dos dimensiones el núcleo gaussiano que se aplica es de dos dimensiones (esto es equivalente a aplicar un núcleo gaussiano en cada dirección), por lo que habrá dos parámetros σ, que llamaremos σ_1 y σ_2.

```
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. Sus parámetros obligatorios son la imagen y el tamaño del núcleo.

[2] Está fórmula proviene de la de la función de densidad de la distribución gaussiana: $f(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$.

Puede además indicarse los valores σ_1 y σ_2. Si ambos son 0, entonces se calcularán a partir del tamaño del núcleo (k_1, k_2) con la siguiente fórmula:

$$\sigma_1 = 0.3 \cdot (\frac{k_1 - 1}{2} - 1) + 0.8$$

$$\sigma_2 = 0.3 \cdot (\frac{k_2 - 1}{2} - 1) + 0.8$$

Ejemplo 5

En este ejemplo utilizamos el algoritmo de Canny [5] de detección de bordes, usando para ello la función `cv2.Canny`. Este algoritmo funciona de la siguiente manera: en primer lugar aplica 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.

```python
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)
    ret = True
    while (ret):
        ret, frame = capture.read()
        if (ret):
            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): ret = False
```

ejemplos/ejemplo6.py

Como vemos el código para mostrar una imagen de la *webcam* es exactamente el mismo que el de mostrar un vídeo 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 6

Ahora simplemente vamos a combinar los dos ejemplos anteriores, reduciendo la imagen a la cuarta parte de su tamaño original. Esto lo hacemos aplicando dos veces `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

Capítulo 2

Otras librerías usadas

En este capítulo veremos una introducción a otras librerías usadas durante el trabajo. En primer lugar hablaremos sobre Android y el desarrollo de aplicaciones para este. Luego hablaremos de la librería Numpy.

2.1. 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 OpenGL.

  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 nuestra 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

La aplicaciones que nosotros vamos a realizar son muy sencillas y solo utilizarán el primero de todos estos componentes. 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 algunas características más del programa.

En el apéndice A.1 se incluye el código de Android realizado, donde se pueden ver el código de la actividad principal y su `layout`, así como el fichero "`manifest.xml`".

2.2. Numpy

Numpy (NUMerical PYthon) es una extensión para Python que agrega una gran cantidad 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.

Para una imagen en blanco y negro solo hace falta un dato por píxel: la intensidad. Sin embargo, cuando las imágenes son a color, hacen falta más valores. Estos valores así como el número usado variarán según el modelo de color usado (RGB, RGBA, HSV...). Los conjuntos de cada tipo de valores de los píxeles forman los llamados canales. Por ejemplo en el modelo RGB hay un canal para el rojo, otro para el verde y otro para el azul y cada uno de ellos contiene las intensidades de ese color en cada píxel.

Por tanto, nuestras imágenes de OpenCV serán arrays de Numpy cuyo atributo `shape` será de la forma `(h,w,n)` (o `(h,w)` si es en blanco y negro), donde `h` es la altura de la imagen, `w` es el ancho y `n` es el número de canales de la imagen. El atributo `ndim` será 2 si la imagen es en blanco y negro (y por tanto sólo tiene un canal) y 3 si tiene más de un canal.

Capítulo 3

Estabilización de imagen

En este capítulo trataremos algunos algoritmos de estabilización de imagen, incluyendo el intento de usar el acelerómetro. Cabe señalar que, debido a las motivaciones del trabajo comentadas anteriormente, durante todo el capítulo se supondrá que la cámara está estática, en el sentido de que el único movimiento producido por esta es leve y es el que queremos corregir.

3.1. 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 [11]. Existe una función en OpenCV que implemente este algoritmo: `goodFeaturesToTrack`. Después vemos a donde se han movido esos puntos en el siguiente fotograma mediante el algoritmo de flujo óptico de Lucas-Kanade[8]. 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.

3.1.1. 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"[6]. 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:

$$E(x,y) = \sum_{u,v} w(u,v)[I(u+x, v+y) - I(u,v)]^2 \tag{3.1}$$

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 el punto (x, y). 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:

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y \tag{3.2}$$

Donde I_x e I_y son las derivadas parciales de la función intensidad en las direcciones x e y respectivamente[1]. 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:

$$E(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} M \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.3}$$

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.

[1]Estas derivadas se aproximan como hemos explicado en el ejemplo 5 de 1.2

- Si λ_1 y λ_2 son positivos y suficientemente 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 [11] propusieron:

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

en lugar de la propuesta por Harris[6]

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 = \det(M) - k \cdot (\mathrm{tr}(M))^2$$

La idea de la función propuesta inicialmente por Harris era que no hacía falta calcular explícitamente los autovalores de la matriz y que se cumplía que:

- Si $|R|$ es pequeño, λ_1 y λ_2 también son pequeños.
- Si $R < 0$ entonces $\lambda_1 >> \lambda_2$ o bien $\lambda_2 >> \lambda_1$, en cuyo caso se trata de un borde.
- Si R es grande, entonces λ_1 y λ_2 también lo son y $\lambda_1 \sim \lambda_2$, por lo que se trata de una esquina.

Sin embargo Shi y Tomasi argumentaron que bastaba con que el mínimo de los dos autovalores fuese suficientemente grande, debido a que las diferencias de intensidad están acotadas por el valor máximo de intensidad de los píxeles, lo que impide que el mayor autovalor sea arbitrariamente grande. La función propuesta por Shi y Tomasi proporciona por tanto buenos resultados y, empíricamente se ha observado que en muchos casos mejores que los que proporciona la original de Harris.

De 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.

3.1.2. Algoritmo de Lucas-Kanade

Para estimar el movimiento de los puntos característicos de un fotograma a otro usaremos el algortimo de Lucas-Kanade [8]. Dicho algoritmo 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 (es decir, que la intensidad de un píxel se mantiene).

2. Los objetos no se mueven muy rápido.

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

Las dos primeras suposiciones nos llevan a la siguiente igualdad:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{3.4}$$

Podemos aproximar por desarrollo de Taylor la parte derecha de la igualdad anterior, de donde obtenemos:

$$I(x+dx, y+dy, t+dt) = I(x, y, t)+I_x(x, y, t)\cdot dx+I_y(x, y, t)\cdot dy+I_t(x, y, t)\cdot dt \tag{3.5}$$

Sustituyendo en 3.4, despejando y dividiendo por dt obtenemos:

$$I_x u + I_y v + I_t = 0 \tag{3.6}$$

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 3.6 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 $Aw = -b$:

$$\begin{pmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_n) \end{pmatrix} \tag{3.7}$$

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

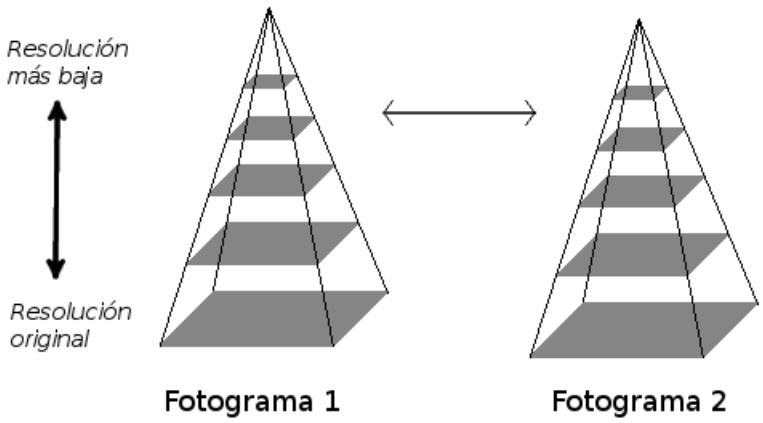
$$A^t A w = A^t b$$

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, como podemos observar en la imagen. 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` para usar una estimación inicial, 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 algoritmo 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.

3.1.3. 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:

$$\begin{pmatrix} x'_i \\ y'_i \\ 1 \end{pmatrix} = sH \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \tag{3.8}$$

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 determinado modelo matemático (en este caso una homografía) a partir de un conjunto de datos que contiene valores atípicos (valores numéricamente distantes a los del resto del conjunto). 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.

3.1.4. Código

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

import numpy as np
import cv2
import cv

feature_params = dict( maxCorners = 10000,
                       qualityLevel = 0.001,
                       minDistance = 5,
                       blockSize = 3 )

term = ( cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_COUNT, 30,
   0.001 )
lk_params = dict( winSize  = (50,50),
                  maxLevel = 4,
                  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__':
    import sys
    try:
        name = sys.argv[1]
        nout = sys.argv[2]
    except:
        print("Error, introduce los nombre de los ficheros de
   entrada y salida")
        sys.exit()
    frames = []
    cap = cv2.VideoCapture(name)
    fps = cap.get(cv2.cv.CV_CAP_PROP_FPS)
```

```
36      size = (int(cap.get( cv.CV_CAP_PROP_FRAME_WIDTH)),int(cap.
    get(cv.CV_CAP_PROP_FRAME_HEIGHT)))
37      nframes = int(cap.get(cv.CV_CAP_PROP_FRAME_COUNT))
38      ret, old_frame = cap.read()
39      old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
40      p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **
    feature_params)
41      cv2.cornerSubPix(old_gray, p0, (5, 5), (-1, -1), term)
42      frames.append(old_frame)
43      count = 1
44      ret = True
45      while(ret):
46          porcentaje = 100*count/nframes
47          print porcentaje, "% completed....................."
48          ret,frame = cap.read()
49
50          if ret:
51              frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY
    )
52              p2, trace_status = checkedTrace(old_gray,
    frame_gray, p0)
53
54              p1 = p2[trace_status].copy()
55              p0 = p0[trace_status].copy()
56
57              h, w = old_frame.shape[:2]
58
59              try:
60                  H, status = cv2.findHomography(p0, p1, cv2.
    RANSAC)
61              except:
62                  pass
63
64              old_frame = cv2.warpPerspective(frame, H, (w, h),
    flags=cv2.INTER_LINEAR + cv2.WARP_INVERSE_MAP)
65              frames.append(old_frame)
66              old_gray = cv2.cvtColor(old_frame, cv2.
    COLOR_BGR2GRAY)
67              p0 = cv2.goodFeaturesToTrack(old_gray, mask = None,
     **feature_params)
```

```
68              cv2.cornerSubPix(old_gray, p0, (5, 5), (-1, -1),
    term)
69              count += 1
70      writer = cv2.VideoWriter(nout, cv.CV_FOURCC('M','J','P','G'
    ), fps, size)
71      for frm in frames:
72          writer.write(frm)
73      cap.release()
```

estabilizacion.py

En la línea 7 y en la 13 vemos los parámetros que usaremos para el algoritmo de Shi-Tomashi y para el de Lucas-Kanade respectivamente.

En la línea 18 podemos ver la función que calcula el flujo óptico, usando `cv2.calcOpticalFlowPyrLK()` primero para calcular la estimación de a donde se han movido los puntos en el segundo fotograma y una segunda vez para ver a donde van a parar estos puntos obtenidos del segundo fotograma al primero, para así comprobar la coincidencia de los puntos obtenidos con los originales y no tener en cuenta los que no tengan coincidencia.

De la línea 33 a la 44 inicializamos las variables que vamos a usar en el algoritmo, abrimos el vídeo, tomamos el primer fotograma y buscamos esquinas. Utilizamos la función `cv2.cornerSubPix()` para refinar la precisión de las esquinas.

Finalmente, dentro del bucle, vamos tomando cada fotograma, calculamos el flujo óptico con respecto al fotograma anterior en la línea 52, intentamos buscar la homografía en la línea 60 y la aplicamos en la línea 64. Por último en las líneas 67 y 68 buscamos las esquinas del fotograma que acabamos de transformar.

3.1.5. 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 suponemos que la cámara no sufre rotaciones y sólo hay traslación, en lugar de estimar la homografía podemos usar una traslación de razón la media o la mediana de lo que se 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 hay objetos moviéndose se producirá traslación aún cuando la cámara este completamente quieta. Sin embargo la mediana funciona bastante bien y es más rápida que el algoritmo del apartado anterior. Sin embargo si no requerimos rapidez no supone una gran mejora.

3.2. Algoritmo basado en la correlación de fase

Otro algoritmo bastante utilizado y más rápido que el anterior está basado en la correlación de fase [7]. Supone que la cámara no sufre rotaciones y que el único movimiento no deseado es de traslación y calcula esta traslación mediante la correlación de fase.

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:

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

Mediante la siguiente fórmula calculamos la llamada densidad espectral cruzada (normalizada con su producto):

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

Donde $\circ$ es el producto elemento a elemento (o 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 de r, y no quedamos con las cordenadas de este, que será el desplazamiento que se ha producido entre un fotograma y otro. OpenCV dispone de la función `cv2.phaseCorrelate(src1, src2[, window])` que se encarga de todo esto. Los dos primeros argumentos son las dos imágenes. El último argumento es opcional y es un array con los coeficientes para la función ventana.

Podemos encontrar el código de este algoritmo en el apéndice C.1.

3.3. Uso del acelerómetro para estabilizar

Se trata de intentar utilizar la información extra de los que disponemos, en concreto la 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 un PC procesa los archivos para intentar estabilizar la imagen: recoge los datos de aceleración en cada instante de medida 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.

3.3.1. 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 botón 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 eliminarla. Hay varias maneras de hacer esto, pero por fortuna, la mayoría de móviles Android 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 memoria en el apéndice A.1.

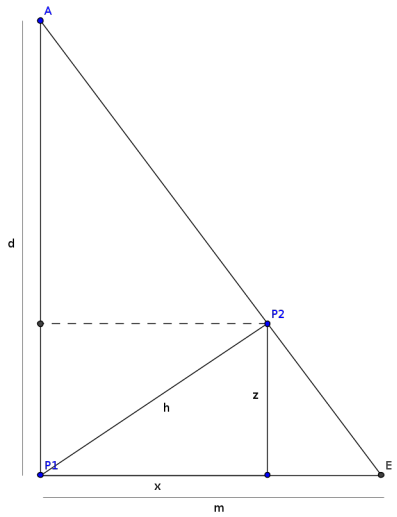
3.3.2. Programa de procesado de los datos

Este programa recibe el vídeo 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:

$$\begin{cases} v_{xi} = v_{x,i-1} + a_{xi}t_i \\ \\ x_i = x_{i-1} + \dfrac{v_{x,i-1} + v_{xi}}{2}t_i \end{cases} \tag{3.9}$$

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()` que calcula la interpolación lineal del conjunto de datos que tenemos en el instante deseado. Ya sabemos el desplazamiento de la 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.

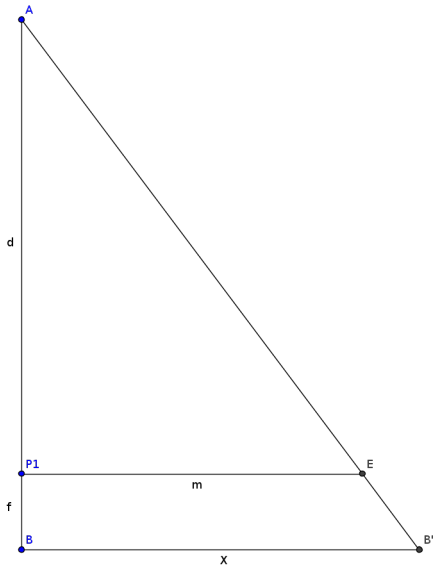


Tenemos, usando el teorema de Pitágoras:

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

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.

Como el desplazamiento lo tenemos en metros pero tenemos que tratar con píxeles. Para realizar este cambio utilizaremos la densidad de píxeles de la cámara: $\frac{\text{número de píxeles}}{\text{tamaño del sensor}}$.

El código se incluye al final en el apéndice A.2.

3.3.3. Resultados

Lamentablemente este método no produce resultados satisfactorios. En algunas ocasiones, sin llegar a producir buenos resultados, estos parecerían acercarse un poco a lo que desearíamos, otras veces no tanto. El mayo problema de esto es debido al error producido al calcular lo que se ha desplazado el teléfono móvil a partir de los datos del acelerómetro. Este error está provocado (como apuntan en la charla de Google *Sensor Fusion on Android Devices: A Revolution in Motion Processing* [10]) principalmente por dos motivos:

- El primero es por el error producido al aproximar la doble integral que tenemos que calcular para obtener la posición a partir de la aceleración. Según la exposición, manteniendo el dispositivo quieto durante un segundo, se produce un error de unos 20 centímetros.

- El segundo motivo es el realmente problemático. Se trata del posible error producido al calcular la aceleración lineal. Recordemos que esto se hacía restando la fuerza de la gravedad de los datos obtenidos por el acelerómetro, y para ello hay que calcular la fuerza de la gravedad. Un pequeño error calculando el ángulo de la gravedad supone un error enorme tras las dos integrales: una diferencia de un grado supondría un error de 8.5 metros.

Capítulo 4

Estabilización de la luz

4.1. Introducción

Por último vamos a ver por encima algunos de los recursos de que disponemos para trabajar con la luz y un algoritmo para intentar estabilizar la iluminación.

En primer lugar, hablemos de los histogramas. Un histograma es una gráfica que representa frecuencias, en este caso los niveles de intensidad de los píxeles. Es decir, relaciona cada nivel de intensidad de la imagen con el número de píxeles que tienen dicho nivel de intensidad. Por tanto cada canal de la imagen tiene su propio histograma; no obstante en este capítulo, por simplicidad, trabajaremos con imágenes en escala de grises, que solo tienen un único canal.

Las principales funciones a la hora de trabajar con histogramas son: `cv2.calcHist(images, channels, mask, histSize, ranges)` y `cv2.equalizeHist(src)`. La primera se encarga de calcular el histograma y la segunda lo ecualiza. La ecualización de un histograma es un método para mejorar el contraste de la imagen modificando el histograma. La idea intuitiva es encontrar el intervalo del histograma en el que se encuentran la mayor parte de los píxeles y de alguna forma "alargar" este intervalo aplanando el histograma. Lo que queremos es transformar una distribución (el histograma original) en otra; para esta transformación usaremos la función de distribución acumulada. Para un histograma $H(i)$ ($H(i)$ representa el número de píxeles con intensidad i en la imagen, con i entre 0 y 255 [1]) , su

[1]Supondremos durante todo el capítulo que la intensidad varía entre 0 y 255 porque trabajaremos con imágenes de 8 bits.

función de distribución acumulada $H'(i)$ es:

$$H^{'}(i) = \sum_{0 \leq j < i} H(j)$$

Por último tenemos que normalizar para que el valor máximo sea 255:

$$H^{'}(i) = \frac{H^{'}(i) - \text{mín}\{H^{'}(j)\}}{\text{máx}\{H'(j)\} - \text{mín}\{H'(j)\}} \cdot 255$$

Para finalmente poder aplicar la transformación:

$$equalized(x, y) = H^{'}(src(x, y))$$

Otra transformación parecida a la ecualización de un histograma es la normalización. En este caso se ajusta el valor mínimo del histograma al 0 y el máximo al 255, manteniendo la forma original del histograma. Esta operación se puede realizar con la función `cv2.normalize()` con los parámetros `alpha =0`,`beta=255` y `norm_type=cv2.NORM_MINMAX`.

4.2. Un algoritmo de normalización de luz

La ecualización es muy importante y es bastante usada para corregir el contraste de la imagen, pero por sí sola es completamente insuficiente a la hora de normalizar la iluminación. En el texto *Enhanced Local Texture Feature Sets for Face Recognition Under Difficult Lighting Conditions* [12] se propone un algoritmo para normalizar la iluminación para la posterior detección de caras.
Dicho algoritmo consta de tres pasos:

1. **Corrección gamma:** se trata de una transformación no lineal que dada una imagen I en escala de grises la reemplaza por $(\frac{I}{255})^{\gamma} \cdot 255$, con $\gamma \in [0, 1]$. Utilizaremos como parámetro $\gamma = 0.2$ [2]. Para hacer la potencia utilizamos la función `cv2.pow(I, γ )`.

2. **Diferencia gaussiana:** consiste en, dada una imagen, obtener dos imágenes desenfocadas de la original (una más desenfocada que la otra) utilizando filtro gaussiano y luego obtener la diferencia de ambas. Utilizaremos $\sigma_0 = 1$ y $\sigma_1 = 2$ para los filtros y para obtener la diferencia utilizaremos la función `cv2.subtract()`

[2] Usaremos estos valores porque son los propuestos en el papel original.

3. **Ecualización de contraste**: hay que reescalar la intensidad de la imagen para estandarizar el contraste. Un opción podría ser ecualizar o normalizar el histograma, pero en el papel citado proponen un método en dos fases:

$$I(x,y) \leftarrow \frac{I(x,y)}{(\text{media}(|I(x',y')|^a))^{1/a}} \tag{4.1}$$

$$I(x,y) \leftarrow \frac{I(x,y)}{(\text{media}(\text{mín}(\tau, |I(x',y')|)^a))^{1/a}} \tag{4.2}$$

Donde a es un componente de compresión y τ es el umbral de truncamiento. Se usará $a = 0.1$ y $\tau = 10$ [2].

El código lo podemos encontar el el apéndice C.2.

4.3. Corrección de la iluminación usando el sensor

Nuevamente intentamos usar para nuestros propósitos los sensores de que disponen los telefónos móviles. En este caso intentaremos utilizar los datos que nos proporciona el sensor de luz. Este sensor mide el nivel de luz ambiental en las unidades del Sistema Internacional para la iluminancia o nivel de iluminación, el lux.

La manera de acceder al sensor de luz es análoga a la de acceder al acelerómetro, por lo que los códigos son muy similares y no hay nada diferente de lo ya realizado. En cualquier caso se adjunta el código de la actividad principal en el apéndice B.1.

Una vez que tenemos los lux medidos en cada instante tenemos que ver que podemos hacer con ellos.

En primer lugar estableceremos una escala, para ello supondremos que nos encontramos en una habitación.En este caso podemos suponer que a partir de 400 lx la habitación está muy iluminada. Por tanto fijamos 400 como el máximo y reescalamos a un $k \in [0,1]$ dividiendo por 400. Queremos que estos datos representen lo iluminada que está la imagen. Como la relación entre lo iluminada que está una imagen y la iluminación ambiental no es lineal, elevamos los datos a un $\alpha \in [0,1]$, en concreto hemos decidido usar $\alpha = 1/4$ ya que tras varías pruebas es la que mejores resultados parece proporcionar.

Antes de ver cómo usamos estos datos, veamos primero la transformación que proponemos. Se trata de algo que en muchos programas de edición de imagen se conoce como ajuste de niveles. La idea es la siguiente, tenemos tres parámetros. Los dos primeros son un límite inferior y un límite superior.

Todos los píxeles de la imagen que tengan una intensidad igual o inferior al límite inferior pasan a ser 0. Análogamente todos los píxeles de la imagen que tengan una intensidad igual o superior pasan a ser 255. El límite inferior y superior pueden tomar valores de 0 a 255 siempre que el inferior sea menor o igual que el superior. El resto de valores comprendidos entre los límites inferior y superior se transforman al rango [0,255] normalizando y multiplicando por 255. El último valor(que llamaremos γ y supondremos entre 0.1 y 10) lo que hace es desplazar el "punto medio" de la imagen, que inicialmente se encuentra en 128 ($\gamma = 1$). Esto se realiza con una corrección gamma, elevando las intensidades de cada píxel a $1/\gamma$.

Los datos de que nosotros disponemos parten de la iluminación media y, por tanto podemos relacionarlos con los tonos medios de la imagen. Por tanto lo que queremos es, si la imagen es muy clara oscurecerla disminuyendo γ y si es muy oscura, aclararla aumentando γ. Esto lo hacemos mediante la transformación 10^{-2k+1}. Como límites mínimo y máximo utilizamos los percentiles 1 y 99 respectivamente.

De las pruebas realizadas podemos observar que el algoritmo funciona bastante bien pero tiene algunas limitaciones: en primer lugar, los cambios bruscos de iluminación,como por otra parte era imaginable, siguen dando problemas, ya que se sigue notando el cambio brusco y, algunas zonas muy oscuras de las imágenes aclaradas siguen quedando totalmente negras. Como ya hemos dicho, esto era esperable. Sin embargo existe otro problema y es que los cambios más leves el sensor algunas veces no los refleja. No obstante para cambios no muy bruscos de la iluminación, se puede usar este código, aunque probablemente haya que aplicar posteriormente otro (como el propuesto en el apartado anterior). Se adjunta el código en el apéndice B.2.

Apéndice A

Código del acelerómetro

A.1. Código de Android

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

A.1.1. MainActivity

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

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

```
55          File file = new File(Environment.
    getExternalStorageDirectory() + File.separator +
    Accfilename);
56
57          //write the bytes in file
58          if(!file.exists())
59          {
60               try {
61                 file.createNewFile();
62             } catch (IOException e) {
63                 e.printStackTrace();
64             }
65          }
66          try {
67             outputAcc = new FileOutputStream(file);
68         } catch (FileNotFoundException e) {
69             e.printStackTrace();
70         }
71
72      }
73
74
75      @Override
76      public boolean onKeyDown(int keyCode, KeyEvent event)
77      {
78          if (keyCode == KeyEvent.KEYCODE_VOLUME_UP)
79          {
80            if (recording) {
81                  camcorderView.stopRecording();
82                  mSensorManager.unregisterListener(this);
83                  try {
84                      outputAcc.close();
85                  } catch (IOException e) {
86                      e.printStackTrace();
87                  }
88                 finish();
89             } else {
90                 recording = true;
91                  mSensorManager.registerListener(this,
    mAccelerometer , SensorManager.SENSOR_DELAY_FASTEST);
92                  camcorderView.startRecording();
```

```
93
94              SimpleDateFormat fecha = new SimpleDateFormat("HH:
    mm:ss.SSS");
95              String fecha_str = fecha.format(new Date());
96              String output = fecha_str+"\n";
97              try {
98                  outputAcc.write(output.getBytes());
99              } catch (IOException e) {
100                 e.printStackTrace();
101             }
102
103            }
104             return true;
105         }
106         return super.onKeyDown(keyCode, event);
107     }
108
109     @Override
110     public void onAccuracyChanged(Sensor sensor, int accuracy)
    {
111
112     }
113
114     @Override
115     public void onSensorChanged(SensorEvent event) {
116
117         SimpleDateFormat fecha = new SimpleDateFormat("HH:mm:ss
    .SSS");
118         String fecha_str = fecha.format(new Date());
119
120         //esto es asi porque la pantalla esta girada
121         float x = event.values[1];
122         float y = event.values[0];
123         float z = event.values[2];
124         String output = Float.toString(x)+" "+Float.toString(y)
    +" "+" "+fecha_str+"\n";
125         try {
126             outputAcc.write(output.getBytes());
127         } catch (IOException e) {
128             e.printStackTrace();
129         }
```

```
130     }
131
132     @Override
133     public boolean onCreateOptionsMenu(Menu menu) {
134         getMenuInflater().inflate(R.menu.main, menu);
135         return true;
136     }
137
138 }
```

Android/Acelerometro/MainActivity.java

A.1.2. 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/Acelerometro/vcorderView.java

A.1.3. 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).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
     package="com.mobvacc.videorecorder"
     android:versionCode="1"
     android:versionName="1.0" >
     <uses-permission android:name="android.permission.CAMERA"><
    /uses-permission>
     <uses-permission android:name="android.permission.
    RECORD_AUDIO"></uses-permission>
     <uses-permission android:name="android.permission.
    WRITE_EXTERNAL_STORAGE"></uses-permission>
     <uses-feature android:name="android.hardware.camera" />
     <uses-feature android:name="android.hardware.camera.
    autofocus" />

     <uses-sdk
         android:minSdkVersion="9"
         android:targetSdkVersion="18" />

     <application
         android:allowBackup="true"
         android:icon="@drawable/ic_launcher"
         android:label="@string/app_name"
         android:theme="@style/AppTheme" >
         <activity
             android:name="com.mobvacc.videorecorder.
    MainActivity"
             android:label="@string/app_name" >
             <intent-filter>
                 <action android:name="android.intent.action.
    MAIN" />

                 <category android:name="android.intent.category
    .LAUNCHER" />
             </intent-filter>
```

```
29          </activity>
30      </application>
31  </manifest>
```

Android/Acelerometro/AndroidManifest.xml

A.1.4. 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/Acelerometro/res/layout/activity_main.xml

A.2. Código de procesado de los datos

A continuación se incluye el programa que procesa los datos recogidos por el programa de Android.

```
#!/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]

            if date != old_date:
                array.append( fline )
                old_date = date
    acc_file.close()
    return array

def calculate_movement(filename):
    array = get_array(filename)
```

```
34      t0 = 0.0
35      x0 = 0.0
36      vx0 = 0.0
37      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      s = np.sign(x)
66      h = np.sqrt(x*x+z*z)
67      m = h*d/(d-z)
68      return s*(m/d)*(d+f)
69
70  if __name__ == '__main__':
71
72      import sys
73      try:
74          name = sys.argv[1]
```

```
75          accname = sys.argv[2]
76      except:
77          print("Error, introduce nombre del archivo de entrada")
78          sys.exit()
79      frames = []
80      nout = "out-" + name
81      cap = cv2.VideoCapture(name)
82      x, y, z = calculate_movement(accname)
83
84
85      fps = cap.get(cv2.cv.CV_CAP_PROP_FPS)
86      fps=16
87      size = (int(cap.get( cv.CV_CAP_PROP_FRAME_WIDTH)),int(cap.
    get(cv.CV_CAP_PROP_FRAME_HEIGHT)))
88      t = 0.0
89      f = 0.004
90      dist = 1.4
91
92      ret, old_frame = cap.read()
93      old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
94
95      frames.append(old_frame)
96
97      while(ret):
98          ret,frame = cap.read()
99          if ret:
100             cv2.imshow('original',frame)
101             frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY
    )
102             t = cap.get(cv2.cv.CV_CAP_PROP_POS_MSEC)/1000
103             h, w = old_frame.shape[:2]
104
105             #calculamos la traslacion
106             dx = np.interp(t,[d[0] for d in x], [d[1] for d in
    x])
107             dy = np.interp(t,[d[0] for d in y], [d[1] for d in
    y])
108             dz = np.interp(t,[d[0] for d in z], [d[1] for d in
    z])
109
110             dx = recalculate_pos(dx, dz, dist, f)
```

```
111             dy = recalculate_pos(dy, dz, dist, f)
112
113             dx = dx*149321.0
114             dy = dy*149321.0
115             M = np.array([[1, 0, dx],[0, 1, dy]])
116
117             old_frame = cv2.warpAffine(frame, M, (w,h), flags=
    cv2.INTER_LINEAR + cv2.WARP_INVERSE_MAP)
118             old_gray = cv2.cvtColor(old_frame, cv2.
    COLOR_BGR2GRAY)
119
120             frames.append(old_frame)
121
122     writer = cv2.VideoWriter(nout, cv.CV_FOURCC('M','J','P','G'
    ), fps, size)
123     for frm in frames:
124         writer.write(frm)
125     cv2.destroyAllWindows()
126     cap.release()
```

estabilizacion_acelerometro.py

Apéndice B

Código del sensor de luz

B.1. Código de Android

Solo incluimos el código de la actividad principal, porque el resto de archivos son iguales que los del código del acelerómetro.

```
1  package com.illucorder.illurecorder;
2
3
4
5  import android.os.Bundle;
6  import android.app.Activity;
7  import android.view.Menu;
8  import java.io.File;
9  import java.io.FileNotFoundException;
10 import java.io.FileOutputStream;
11 import java.io.IOException;
12 import java.text.SimpleDateFormat;
13 import java.util.Date;
14 import java.util.Scanner;
15 import com.illucorder.illurecorder.R;
16 import com.illucorder.illurecorder.R.id;
17 import com.illucorder.illurecorder.R.layout;
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 mLightSensor;
33     private vcorderView camcorderView;
34     private boolean recording = false;
35     FileOutputStream outputLight;
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 Lightfilename="light"+dateString+".txt";
48         //camera init
49         //
50         camcorderView = (vcorderView) findViewById(R.id.
    vcorderView1);
51         camcorderView.setOutputFile(filename);
52
53         //light sensorr init
54         mSensorManager = (SensorManager) getSystemService(
    Context.SENSOR_SERVICE);
55         mLightSensor = mSensorManager.getDefaultSensor(Sensor.
    TYPE_LIGHT);
56
57         //open sensor file
```

```
58          File file = new File(Environment.
    getExternalStorageDirectory() + File.separator +
    Lightfilename);
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             outputLight = 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                      outputLight.close();
88                  } catch (IOException e) {
89                      e.printStackTrace();
90                  }
91                 finish();
92             } else {
93                 recording = true;
94                  mSensorManager.registerListener(this,
    mLightSensor , 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                 outputLight.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         //float x = event.values[0];
124         //
125         Scanner st;
126         int lux = -1;
127         try {
128             st = new Scanner(new File("/sys/devices/virtual/
    lightsensor/switch_cmd/lightsensor_file_state"));
129             lux = st.nextInt();
130             st.close();
131         } catch (FileNotFoundException e) {
132             e.printStackTrace();
```

```
133         }
134
135         String output = Float.toString(lux)+" "+fecha_str+"\n";
136         try {
137             outputLight.write(output.getBytes());
138         } catch (IOException e) {
139             e.printStackTrace();
140         }
141     }
142
143     @Override
144     public boolean onCreateOptionsMenu(Menu menu) {
145         getMenuInflater().inflate(R.menu.main, menu);
146         return true;
147     }
148
149 }
```

Android/Luz/MainActivity.java

B.2. Código 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[1]
            aux_date = date_str.split(":")
            date = ((int(aux_date[0])*60 + int(aux_date[1]))*60
    + float(aux_date[2])) - t0
            fline = [float(aux[0]), date]

            if date != old_date:
                array.append( fline )
                old_date = date
    acc_file.close()
    return array


def correct_gamma(img, correction):

    temp = img.copy()/255.0
    temp = np.array(temp, dtype=np.float32)
    temp = cv2.pow(temp, 1./correction)*255.0
```

```
37      return temp
38
39  def adjust_levels(im,bot,mid,top):
40      lut = np.zeros(256)
41      for i in xrange(256):
42          if i <= bot:
43              lut[i]=0
44          elif i >= top:
45              lut[i] = 255
46          else:
47              lut[i]=255*(i-bot)/(top-bot)
48      aux = cv2.LUT(im, lut)
49      aux = np.array(aux, dtype=np.uint8)
50      if mid < 0.1:
51          mid = 0.1
52      aux = correct_gamma(aux, mid)
53      aux = np.array(aux, dtype=np.uint8)
54      return aux
55
56  def autolevels(im, mid=1):
57      inf = np.percentile(im,1)
58      sup = np.percentile(im,99)
59      return adjust_levels(im, inf,mid,sup)
60
61  def draw_hist(im, name):
62      h = np.zeros((300,256,3))
63      if len(im.shape)!=2:
64          im = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
65      hist_item = cv2.calcHist([im],[0],None,[256],[0,256])
66      cv2.normalize(hist_item,hist_item,0,255,cv2.NORM_MINMAX)
67      hist=np.int32(np.around(hist_item))
68      for x,y in enumerate(hist):
69          cv2.line(h,(x,0),(x,y),(255,255,255))
70      y = np.flipud(h)
71      y = 255 -y
72      cv2.imshow(name, y)
73
74
75  if __name__ == '__main__':
76
77      import sys
```

```
78      try:
79          name = sys.argv[1]
80          illuname = sys.argv[2]
81      except:
82          print("Error, introduce nombre del archivo de entrada")
83          sys.exit()
84      frames = []
85      nout = "out-" + name
86      cap = cv2.VideoCapture(name)
87      array = get_array(illuname)
88      i = 0
89      fps = cap.get(cv2.cv.CV_CAP_PROP_FPS)
90      fps = 16
91      size = (int(cap.get( cv.CV_CAP_PROP_FRAME_WIDTH)),int(cap.
    get(cv.CV_CAP_PROP_FRAME_HEIGHT)))
92      ret = True
93      while(ret):
94          ret,frame = cap.read()
95          t = cap.get(cv2.cv.CV_CAP_PROP_POS_MSEC)/1000
96          t = t*1.197
97          if ret:
98              i = 0
99              while i < len(array)-1:
100                 if (array[i][1]<=t) and (t<array[i+1][1]):
101                     break
102                 i = i+1
103             lux = array[i][0]
104             if lux > 400:
105                 lux = 400
106             k = pow((lux/400),1./4)
107             x = -2*k+1
108             y = pow(10,x)
109
110
111             gr =  cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
112             out = autolevels(gr, y)
113             out = cv2.cvtColor(out, cv2.COLOR_GRAY2BGR)
114             frames.append(out)
115
116
117
```

```
118     writer = cv2.VideoWriter(nout, cv.CV_FOURCC('M','J','P','G'
        ), fps, size)
119     for frm in frames:
120         writer.write(frm)
121     cv2.destroyAllWindows()
122     cap.release()
```

ilum_sensor.py

Apéndice C

Otros códigos

C.1. Código del algoritmo de correlación de fase

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

import numpy as np
import cv2
import cv

if __name__ == '__main__':

    import sys
    try:
        name = sys.argv[1]
        nout = sys.argv[2]
    except:
        print("Error, introduce nombre de los ficheros de
    entrada y salida")
        sys.exit()
    cap = cv2.VideoCapture(name)
    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)))
    ret, old_frame = cap.read()
    old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
    ret = True
    frames = []
```

```
23      while(ret):
24          ret,frame = cap.read()
25      if ret:
26              frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY
    )
27              h, w = old_frame.shape[:2]
28              old_gray = np.float32(old_gray)
29              frame_gray = np.float32(frame_gray)
30              dx, dy = cv2.phaseCorrelate(old_gray, frame_gray)
31              M = np.array([[1, 0, dx],[0, 1, dy]])
32              old_frame = cv2.warpAffine(frame, M, (w,h), flags=
    cv2.INTER_LINEAR + cv2.WARP_INVERSE_MAP)
33              frames.append(old_frame)
34              old_gray = cv2.cvtColor(old_frame, cv2.
    COLOR_BGR2GRAY)
35
36
37      writer = cv2.VideoWriter(nout, cv.CV_FOURCC('M','J','P','G'
    ), fps, size)
38      for frm in frames:
39          writer.write(frm)
40      cap.release()
```

estabilizacion_pc.py

C.2. Código del algoritmo de nomalización de luz

```
1  import cv2
2  import cv2.cv as cv
3  import numpy as np
4
5
6  def correct_gamma(img, dst, correction):
7
8      temp = img.copy()/255.0
9      temp = np.array(temp, dtype=np.float32)
10     temp = cv2.pow(temp, correction)*255.0
11     return temp
```

```
12
13 def doCeq(im,a,t):
14     aint = np.abs(im)
15     m = np.power(aint, a)
16     m = np.mean(m)
17     m = np.power(m,1.0/a)
18
19     im = im/m
20     aint = np.abs(im)
21
22     #m = np.array([[pow(min(t,x),a) for x in y] for y in aint])
23     m = np.minimum(aint, t)
24     m = np.power(m, a)
25
26     m = np.mean(m)
27     m = np.power(m,1.0/a)
28     im = im/m
29
30     im = t*np.tanh(im/t)
31     im = cv2.convertScaleAbs(im, alpha=127, beta=0)
32     return im
33
34 def process_image(img):
35
36     gr =  cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
37     gr = np.array(gr, dtype=np.uint8)
38     gf = correct_gamma(gr, gr, 0.2)
39     b1 = cv2.GaussianBlur(gf,(0,0), 1,1)
40     b2 = cv2.GaussianBlur(gf,(0,0), 2,2)
41     b2 = cv2.subtract(b1, b2)
42     gr = cv2.convertScaleAbs(b2, alpha=127, beta=127)
43     gr = doCeq(gr, 0.1, 10)
44     #gr = cv2.normalize(gr,alpha = 0,beta = 255,norm_type = cv2
       .NORM_MINMAX)
45     return gr
46
47 if __name__ == '__main__':
48     import sys
49     try:
50         name = sys.argv[1]
51         nout = sys.argv[2]
```

```
52      except:
53          print("Error, introduce los nombre de los ficheros de
    entrada y salida")
54          sys.exit()
55
56      frames = []
57      cap = cv2.VideoCapture(name)
58      fps = cap.get(cv2.cv.CV_CAP_PROP_FPS)
59      fps = 16
60      size = (int(cap.get( cv.CV_CAP_PROP_FRAME_WIDTH)),int(cap.
    get(cv.CV_CAP_PROP_FRAME_HEIGHT)))
61      nframes = int(cap.get(cv.CV_CAP_PROP_FRAME_COUNT))
62      count = 1
63      ret = True
64
65      while(ret):
66          ret,frame = cap.read()
67          if ret:
68              out = process_image(frame)
69              out = cv2.cvtColor(out, cv2.COLOR_GRAY2BGR)
70              frames.append(out)
71              porcentaje = 100*count/nframes
72              print porcentaje, "% completed
    ........................"
73              count += 1
74      writer = cv2.VideoWriter(nout, cv.CV_FOURCC('M','J','P','G'
    ), fps, size)
75      for frm in frames:
76          writer.write(frm)
77      cap.release()
```

luz.py

Bibliografía

[1] Android reference. http://developer.android.com/reference.

[2] The numpy reference manual. http://docs.scipy.org/doc/numpy/numpy-ref-1.8.0.pdf, 2013.

[3] The opencv reference manual. http://docs.opencv.org/opencv2refman.pdf, 2013.

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

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

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

[7] C. D. Kuglin and D. C. Hines. The phase correlation image alignment method. *IEEE Conference on Cybernetics and Society*, pages 163–165, 1975.

[8] 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.

[9] Bryan S. Morse. Edge detection and gaussian related mathematics. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MORSE/edges.pdf.

[10] David Sachs. Sensor fusion on android devices: A revolution in motion processing. http://www.youtube.com/watch?v=C7JQ7Rpwn2k. min 23:20.

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

[12] Xiaoyang Tan and Bill Triggs. Enhanced local texture feature sets for face recognition under difficult lighting conditions. In *IEEE Transactions on Image Processing 19*, pages 1635–1650, 2010.