UNIVERSIDAD COMPLUTENSE
MADRID

FACULTAD DE MATEMÁTICAS

Técnicas de preprocesado y procesado de imágenes astronómicas mediante OpenCV

Miguel Ardura Zorita
Director: Carlos Gregorio Rodríguez

Abstract

Here goes the English version of the abstract.

Índice general

1 Motivación **3**

2 Herramientas usadas **3**
2.1 Ficheros FITS 3
2.1.1 Astropy 4
2.2 OpenCV 4
2.2.1 Kernel en imágenes 4
2.3 Numpy 4
2.4 Matplotlib 4

3 Desarrollo del proyecto **4**
3.1 Procesado de los ficheros FITS 4
3.2 Uso de un filtro customizado 5
3.3 Segmentación 6
3.4 Detección de contornos 8
3.5 Estudio en paralelo: detección de candidatos a estrellas 10
3.6 Blob detection 12

4 Conclusión y trabajo futuro **15**

5 Problemas encontrados **15**

A Apéndices **16**
A.1 Muestras del procesado de otras imágenes 16
A.2 Trabajo descartado 16
A.2.1 Morphology 16
A.2.2 Canny 16
A.2.3 Sobel 16
A.2.4 Flow 16
A.2.5 3d 16

B Bibliografía **16**

1 Motivación

Este trabajo tenía la tarea radical de servirme de una primera aproximación a la disciplina de la Visión Artificial. Si bien aquí se presenta una aplicación concreta al campo de la Astronomía, mi objetivo era conocer las técnicas utilizadas en la creación, manipulación y procesado de imágenes mediante las herramientas de que se dispone ahora, tanto de hardware como de software. La elección concreta de la Astronomía como campo de estudio surgió al poco de reunirme con mi tutor, Carlos Gregorio, y responde a mi afición por tal ciencia y deseo de poder experimentar con las imágenes generadas por los telescopios a un nivel superior al de un mero aficionado.
Aunque en el camino se ha ido variando el rumbo del proyecto en multitud de ocasiones, finalmente decidí centrarlo en la detección semi-automática de galaxias tomando como input un fichero FITS[1] para una posterior clasificación en un trabajo futuro. El resultado del procesado es la detección de las estructuras candidatas a galaxias a través de sus contornos y la identificación de los objetos más brillantes de la imagen, con el fin de simplificar un trabajo posterior de discriminización de objetos no candidatos a galaxias.

2 Herramientas usadas

En esta sección se informa brevemente de las herramientas que he utilizado a lo largo del proyecto, tanto los tipos de ficheros (FITS), como las librerías usadas (Astropy, OpenCV, Numpy, Matplotlib).

2.1 Ficheros FITS

FITS son las siglas de Flexible Image Transport System, un formato de archivo que sirve para recoger, transportar y procesar datos consistentes en arrays multidimensionales y tablas que almacenan información con la estructura clave-valor, siendo éstos principalmente imágenes científicas y metadatos.

[1] Ver la sección 2.1

2.1.1 Astropy

2.2 OpenCV

2.2.1 Kernel en imágenes

2.3 Numpy

2.4 Matplotlib

3 Desarrollo del proyecto

3.1 Procesado de los ficheros FITS

El proceso de tratado de las imágenes empieza con la carga y lectura de los ficheros FITS. Esto se hace por medio de la librería Astropy. Como se explica en la sección 2.1, la matriz que representa la imagen que se usará aquí se corresponde con el primer HDU. Así, el siguiente código carga en 'img' el ndarray de numpy contenido en el archivo FITS de muestra:

```
#Cargar una imagen desde un fichero FITS
from astropy.io import fits
fitsFile="../TFG/frame-i-002830-6-0398.fits"
hdulist = fits.open(fitsFile)
img = hdulist[0].data
```

basicLoading.py

El siguiente código hace uso de OpenCV para mostrar la imagen antes cargada. Se realiza un mapeo lineal de los valores leídos del FITS a [0,1] para normalizar los valores de la imagen y que la función **imshow** no aparezca en blanco debido a la conversión al rango [0,255] que **imshow**[2] realiza a las imágenes en float 32:

```
#Mostrar una imagen con OpenCV
import cv2
import numpy as np

Min=abs(np.amin(img)) #Se toma el valor absoluto porque por
    errores en el CCD pueden existir mediciones ligeramente
    erradas. En concreto, negativas muy cercanas a cero
Max=np.amax(img)
img = 255*(img+Min)/Max
```

[2] http://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html#imshow

```
cv2.namedWindow("Image", cv2.WINDOW_NORMAL) #Se crea una ventana "Image"
cv2.imshow("Image",img) #Se dibuja img en la ventana Image
cv2.waitKey() #Esta sentencia muestra la ventana hasta que se presiona una tecla
```

basicLoading.py

La figura 1 muestra la imagen cargada con el script anterior.

Figura 1: Visualización del archivo frame-i-002830-6-0398.fits por medio de *basicLoading.py*.

3.2 Uso de un filtro customizado

Ante la problemática surgida en lo que respecta a la eliminación del ruido de las imágenes haciendo uso de las herramientas predefinidas en la librería OpenCV (como se muestra en la sección 5), creé un filtro con el que obtengo resultados más cercanos a lo que quería. En concreto consiste en el uso del método **filter2D**[3] de OpenCV. Dicho método sirve para crear un filtro a partir de un kernel (véase la sección 2.2.1) dado. En concreto, creé el kernel $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ y lo usé en la primera etapa del procesado de las imágenes. Tomando la imagen resultado de aplicar este filtro a la imagen leída de los

[3] `http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#filter2d`

archivos FITS, y aplicando técnicas de segmentación, obtuve resultados con los que pude seguir avanzando.

3.3 Segmentación

Me encontré con que, junto con la aplicación del filtro que creé, las técnicas de segmentación de las que provee OpenCV eran suficiente para eliminar gran cantidad del ruido existente en las imágenes. Esto me es de gran utilidad en el proceso de la detección de galaxias porque reduce enormemente la cantidad de información no útil para este propósito, como las regiones vacías o en las que los objetos luminosos no aparecen agrupados de forma ordenada.

En concreto uso el método de *thresholding*, que es una técnica de segmentación que consiste en una identificación de los valores de la imagen a través de una relación de equivalencia dada. En términos de Teoría de Conjuntos esto equivale a la construcción del espacio cociente del conjunto de píxeles por medio de la relación de equivalencia que dicte el método, y en el que a los píxeles relacionados se les asignará el mismo valor en el espacio cociente resultante.

Tales relaciones son, en el caso de segmentación por *thresholding*, de comparación con un umbral dado. OpenCV ofrece cinco tipos de segmentación usando este método[4]. A saber:

- **Binario:** Fijado un umbral, si el valor de un píxel excede ese umbral, se le asigna un valor fijado como máximo; si no, se le asignará 0.
- **Binario invertido:** Análogo al anterior, pero con las condiciones invertidas.
- **Truncado:** Si el valor de un píxel excede el umbral, se le asignará el umbral como nuevo valor. En caso contrario no se ve alterado.
- **Truncado a cero:** Si el valor de un píxel es menor que el del umbral, se le asigna el valor cero. En caso contrario no se ve alterado.
- **Truncado a cero invertido:** Análogo al anterior, pero con las condiciones invertidas.

El que me daba mejores resultados y decidí usar es el binario invertido.

En lo que respecta a los parámetros de la función, encontré que el umbral obtenido automáticamente por el algoritmo de Otsu y el valor máximo igual

[4] `http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=threshold#threshold`

a 1 daban los resultados más aproximados a lo que buscaba. El siguiente código recoge la aplicación de esto anterior:

```
from astropy.io import fits
import cv2
import numpy as np

fitsFile="../TFG/frame-i-002830-6-0398.fits"
hdulist = fits.open(fitsFile)
img = hdulist[0].data


Min=abs(np.amin(img))
Max=np.amax(img)
img = 255*(img+Min)/Max
imgFiltered=cv2.filter2D(img,-1,np.array
    ([[0,1,0],[1,0,1],[0,1,0]]))

_,binary=cv2.threshold(imgFiltered, cv2.THRESH_OTSU, 1.0, cv2.
    THRESH_BINARY_INV)
cv2.namedWindow("Image", cv2.WINDOW_NORMAL)
cv2.imshow("Image",binary)
cv2.waitKey()
```

customFilter+thresholding.py

y la figura 2 muestra la imagen que es generada por el código anterior a partir de la imagen antes vista.

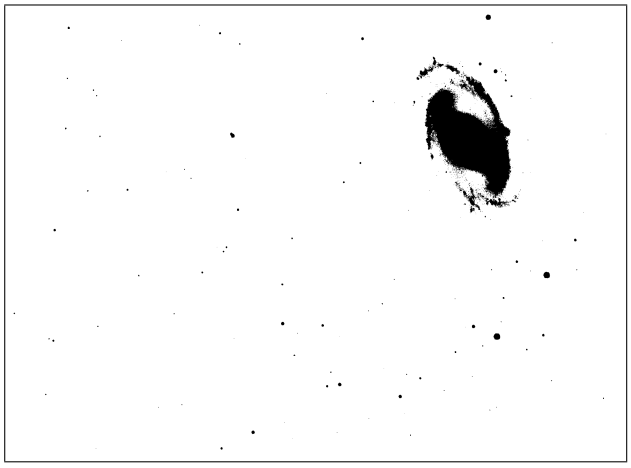

Figura 2: Imagen resultado de aplicar lo anterior a la imagen vista antes.

Como se puede observar, gran parte del granulado de la imagen original ha desaparecido y en su lugar se distinguen motas aisladas y una gran estructura.

Se aprecia claramente una zona central de la que salen brazos finos hacia el exterior.

A fin de conseguir que esta detección fuera automática, introduje el estudio de *contornos* en la imágenes obtenidas, lo que lleva a la siguiente etapa del procesado.

3.4 Detección de contornos

Con el fin de identificar y aislar las regiones oscuras que aparecen aplicando *thresholding*, incorporo el uso del método que detecta contornos **findContours**[5] de OpenCV. Este método toma como input una imagen de 8 bits, un modo y un método y devuelve la lista de contornos. Como modo utilizo el que devuelve los contornos en una estructura anidada (**RETR_TREE**), y como método de aproximación de los contornos, uso el que comprime los segmentos para almacenar solo sus extremos (**CHAIN_APPROX_SIMPLE**). El objetivo es usar el método de OpenCV que determina el rectángulo que se ajusta a cada contorno, y entre todos esos rectángulos quedarse con el de mayor área, que es el que contiene al contorno candidato a estructura de interés. Para esto hago uso del método **boundingRect**[6], que dado un contorno, devuelve cuatro parámetros que corresponden a las dos coordenadas del vértice superior izquierdo, la base y la altura del rectángulo horizontal que ajusta al contorno.

El siguiente script muestra su uso:

```
# -*- coding: utf-8 -*-
from astropy.io import fits
import cv2
import numpy as np

fitsFile="../TFG/frame-i-002830-6-0398.fits"
hdulist = fits.open(fitsFile)
img = hdulist[0].data


Min=abs(np.amin(img))
Max=np.amax(img)
img = 255*(img+Min)/Max
imgFiltered=cv2.filter2D(img,-1,np.array
    ([[0,1,0],[1,0,1],[0,1,0]]))
```

[5] http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#findcontours

[6] http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#boundingrect

```
16 _,binary=cv2.threshold(imgFiltered, cv2.THRESH_OTSU, 1.0, cv2.
       THRESH_BINARY_INV)
   cv2.namedWindow("Image", cv2.WINDOW_NORMAL)
18 cv2.imshow("Image",binary)
   cv2.waitKey()
20 contours, _ = cv2.findContours(cv2.convertScaleAbs(binary),cv2.
       RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
   cv2.drawContours(binary,contours,-1,(0,0,0),2) #Este método
       dibuja los contornos en la imagen binary
22
   cv2.namedWindow("Contours with bounding rectangles", cv2.
       WINDOW_NORMAL)
24
   max_area=1
26 X,Y,W,H=0,0,0,0
   for cnt in contours:
28   x,y,w,h = cv2.boundingRect(cnt)
     X,Y,W,H,max_area = (x,y,w,h,w*h) if w*h>max_area and w*h<2040*
       1480 else (X,Y,W,H,max_area)
30   cv2.rectangle(binary,(x,y),(x+w,y+h),(0,0,0),1) #Este método
       dibuja los rectángulos que ajustan los contornos en la imagen
        binary
   cv2.imshow("Contours with bounding rectangles",binary)
32 cv2.waitKey()
   cv2.imwrite("img3.png",255*binary)
34 cropped=binary[Y:Y+H,X:X+W]
   cv2.imshow('crop',255*cropped)
36 cv2.imwrite("img3Rect.png", 255*cropped)
   cv2.waitKey()
```

contours+BoundingRects.py

y la figura 3 muestra la imagen resultante.

Para determinar el rectángulo de mayor área, se predefine un rectángulo inicial de alto y ancho iguales a cero para en cada iteración sobre los contornos comparar el rectángulo que ajusta al contorno de la iteración con el de máximo área actual. Se corresponde con la siguiente sección del código:

```
1 max_area=1
  X,Y,W,H=0,0,0,0
3 for cnt in contours:
    x,y,w,h = cv2.boundingRect(cnt)
5   X,Y,W,H,max_area = (x,y,w,h,w*h) if w*h>max_area and w*h<2040*
      1480 else (X,Y,W,H,max_area)
```

contours+BoundingRects.py

La condición $w*h < 2040*1480$ descarta el contorno que corresponde al borde exterior de la imagen, pues eventualmente puede aparecer como resultado del

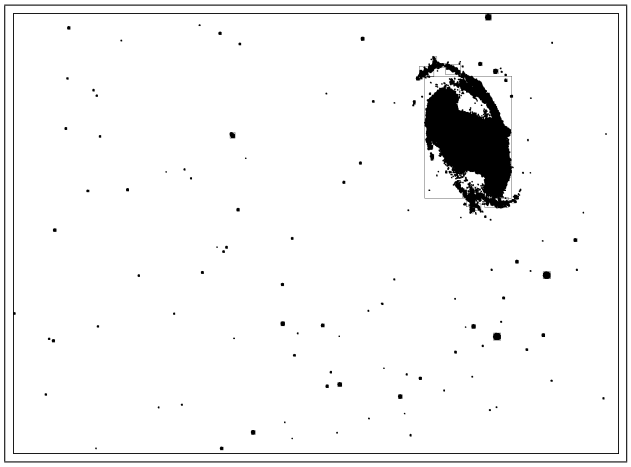

Figura 3: Rectángulos ajustando los contornos.

cálculo de contornos.

La figura 4 muestra el resultado de recortar la imagen 3 por el rectángulo de mayor área detectado.

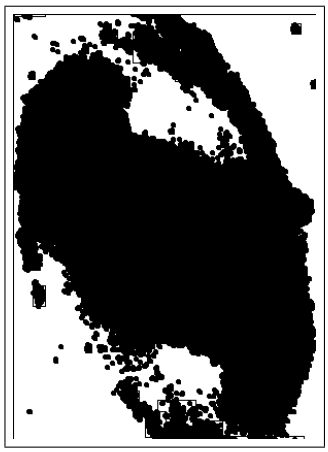

Figura 4: Recorte del contorno identificado.

3.5 Estudio en paralelo: detección de candidatos a estrellas

Con el fin de identificar todas las posibles galaxias presentes en la imagen, surge la necesidad de determinar cuándo una región luminosa se corresponde con una estrella y cuándo con una galaxia, pues ambos objetos pueden presentar tamaños y formas similares.

A diferencia de la discretización anterior, me centro ahora en la identificación de los objetos que por su luminosidad se destacan sobre su alrededor. Para ello hago un uso combinado del filtro customizado y función de *thresholding* antes mencionados, y lo combino con la función **morphology**[7] de OpenCV, que proporciona herramientas para el uso de transformaciones morfológicas.

Una transformación morfológica es básicamente una operación que provee de información sobre la estructura de una imagen por medio de la aplicación de un elemento estructurador. Generalmente consistirá en la convolución de la imagen de partida con un kernel. La naturaleza de este kernel será lo que determine la imagen resultado.

Las transformaciones implementadas disponibles en **morphology** se basan en dos operaciones básicas: la *dilatación* y la *erosión*. En ambos casos se convoluciona con un kernel que generalmente tiene forma rectangular o circular y a cada píxel se le asigna un nuevo valor dependiendo de los valores de los píxeles que lo rodeen . En el caso de la dilatación se toma el máximo valor de los píxeles del entorno definido por el kernel, y en el caso de la erosión el mínimo. Cuando la imagen es binaria estos valores se corresponden con 1 y 0 respectivamente.

Combinando estas dos transformaciones, OpenCV dispone de las siguientes operaciones:

- **Apertura (open):** es el resultado de aplicar una erosión seguida de una dilatación.

$$apertura(imagen, kernel) = \\ dilatación(erosión(imagen, kernel), kernel)$$

- **Cierre (close):** es el resultado de aplicar una dilatación seguida de una erosión.

$$cierre(img, kernel) = erosión(dilatación(imagen, kernel), kernel)$$

- **Gradiente:** es la diferencia de la erosión y la dilatación de una imagen.

$$gradiente(imagen, kernel) = \\ dilatación(imagen, kernel) - erosión(imagen, kernel)$$

- **Top Hat:** es la diferencia de la imagen inicial y su apertura.

[7] `http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=morphologyex#morphologyex`

$$tophat(imagen, kernel) = imagen - apertura(imagen, kernel)$$

- **Black Hat:** es la diferencia del cierre y la imagen inicial.

$$blackhat(imagen, kernel) = cierre(imagen, kernel) - imagen$$

Se puede ver un ejemplo de la aplicación de estas operaciones en la sección A.2.1.

A través de la experimentación encontré que la operación cierre conseguía descartar la mayor cantidad de ruido de la mayoría de las imágenes, salvo en algunos casos, en los que por la naturaleza de las imágenes los procedimientos aquí usados dan resultados erróneos (véase sección 5.

El siguiente código muestra el uso de la operación cierre:

```
# -*- coding: utf-8 -*-
from astropy.io import fits
import cv2
import numpy as np

fitsFile="../TFG/frame-i-002830-6-0398.fits"
hdulist = fits.open(fitsFile)
img = hdulist[0].data

Min=np.amin(img)
Max=np.amax(img)
img = 255*(img+abs(Min))/Max
imgFiltered=cv2.filter2D(img,-1,np.array
    ([[0,1,0],[1,0,1],[0,1,0]]))
_,imgBase=cv2.threshold(imgFiltered, cv2.THRESH_OTSU, 0, cv2.
    THRESH_TOZERO_INV)

imgProcessed=cv2.morphologyEx(imgBase, cv2.MORPH_CLOSE, (3,3),
    iterations=5)
```

blobDetection.py

y la figura 5 muestra la imagen resultante de aplicar en cinco iteraciones la operación cierre.

Una vez que se tiene una imagen binaria y con solo los elementos más brillantes de la imagen original, creé un detector de *blobs*, como se explica en la siguiente sección.

3.6 Blob detection

Un *blob* es un grupo de píxeles de una imagen que comparten la propiedad de tener valores similares. En la figura 5 cada región oscura aislada es un *blob*, y el objetivo de esta sección es detectarlos.

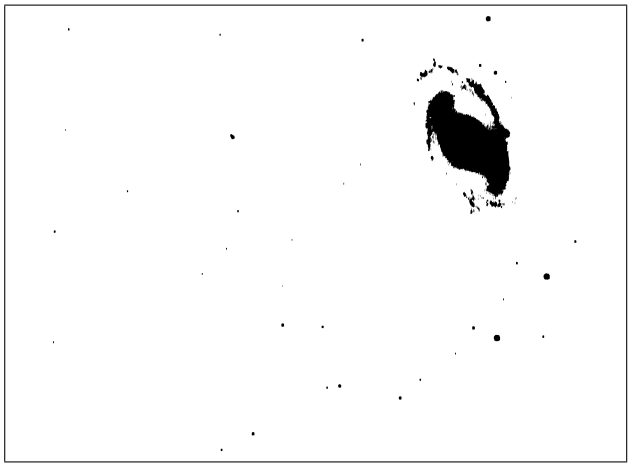

Figura 5: Operación cierre.

En OpenCV se dispone de la función **SimpleBlobDetector** para lograr esto. Basta con fijar una serie de parámetros que se refieren a unas características básicas (a saber: color, tamaño y forma).

El siguiente código muestra el detector de *blobs* que creé y los parámetros con los que obtuve mejores resultados, que son una convexidad mínima de 0.5 y un valor de la elongación mínimo de 0.1:

```
#Código basado en el manual https://www.learnopencv.com/blob-
    detection-using-opencv-python-c/

im = cv2.imread("img4.png") #Carga de la imagen generada
    aplicando la operación cierre

#Creación del contenedor de los parámetros
params = cv2.SimpleBlobDetector_Params()

#Filtrado por convexidad
params.filterByConvexity = True
params.minConvexity = 0.5
#Filtrado por elongación
params.filterByInertia = True
params.minInertiaRatio = 0.1

#Creación del detector con los parámetros fijados
detector = cv2.SimpleBlobDetector(params)

#Detección de los blobs
keypoints = detector.detect(im)

#La siguiente función marca con una circunferencia los blolbs
```

```
    detectados
im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array
    ([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.namedWindow("Keypoints",cv2.WINDOW_NORMAL)
cv2.imshow("Keypoints", im_with_keypoints)
cv2.imwrite('BLOBTEST.png', im_with_keypoints)
cv2.waitKey(0)
```

blobDetection.py

y la figura 6 muestra el resultado de aplicar el detector de *blobs*.

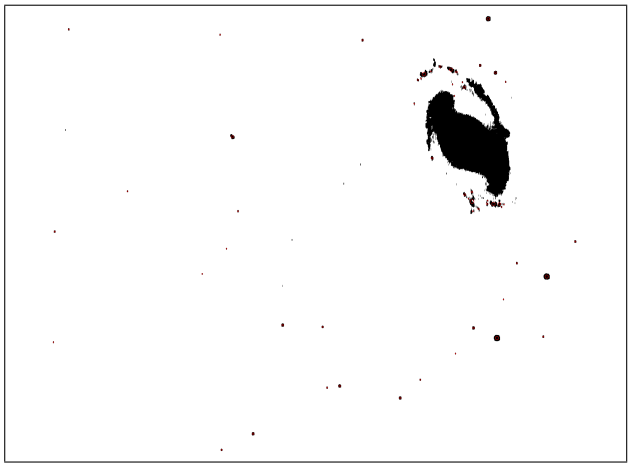

Figura 6: Detección de *blobs*.

En la figura 7 se ve en detalle una región de la imagen 6.

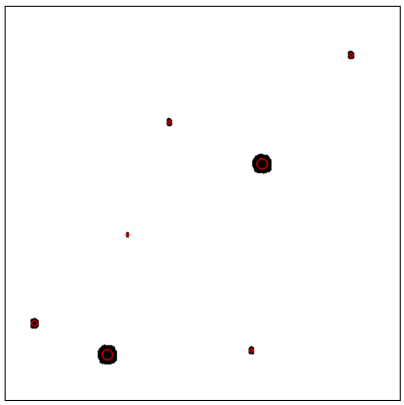

Figura 7: Detección de *blobs* en detalle.

4 Conclusión y trabajo futuro

En resumen, se han definido métodos para, partiendo de una imagen de un fichero FITS, identificar tanto las grandes regiones luminosas candidatas a galaxias como objetos más brillantes de la imagen. El siguiente paso sería continuar con la clasificación de las imágenes ya simplificadas y separadas en sus zonas de interés. Esto se podría hacer de varias maneras. Por ejemplo, si se dispusiera de una gran base de datos con ficheros FITS etiquetados por los objetos astronómicos que contienen, se podrían usar técnicas de Machine Learning para crear clasificadores automáticos con ajuste dinámico de los parámetros de todas las funciones usadas.

5 Problemas encontrados

A la hora de realizar este trabajo me he encontrado principalmente con dos problemas. El primero es la falta de imágenes con las que trabajar y sobre todo de calidad adecuada para trabajar con ellas a nivel amateur-académico iniciado. Si bien se pueden encontrar muchos sitios web con ficheros FITS de acceso libre tras buscar un poco en Internet, pocos eran los que disponían de una base de datos que alguien profano en la materia pudiera entender y manejar con soltura en un primer acercamiento al manejo de los mismos.

Finalmente di con el sitio web de un proyecto que libera ficheros FITS etiquetados por su contenido, lo que me resultó muy útil para acceder de manera directa a imágenes de galaxias, a fin de empezar a trabajar con ellas. Este sitio es "The Sloan Digital Sky Survey: Mapping the Universe" (www.sdss.org). Lamentablemente para mí, el buscador interno de la página dejó de funcionar y tuve problemas para conseguir nuevos ficheros con los que trabajar.

El segundo problema es la naturaleza de las imágenes captadas por los telescopios. Al ser el principal objetivo del trabajo aprender a usar OpenCV, intenté primero obtener resultados haciendo uso de sus funciones predefinidas, pero me encontré con que parecían estar hechas principalmente para su aplicación en imágenes tomadas en el espectro visible. Las imágenes astrónomicas obtenidas de los ficheros FITS distan mucho de ser tan nítidas como las obtenidas por cualquier cámara de fotos, y si se representan tal y como son leídas directamente del FITS probablemente no se distingan nada más que gránulos grises. Esto es de esperar, puesto que no están hechas para ser mostradas, al menos, en crudo, pero explica por qué necesité crear mi propio filtro para obtener algún resultado. Una vez superado esto pude satisfactoriamente hacer uso de las funciones de OpenCV.

A Apéndices

A.1 Muestras del procesado de otras imágenes

A.2 Trabajo descartado

A.2.1 Morphology

testMorefilters

A.2.2 Canny

testCanny

A.2.3 Sobel

testEdges

A.2.4 Flow

testFlow

A.2.5 3d

test3d

B Bibliografía