

UNIVERSIDAD COMPLUTENSE
MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

**Técnicas de detección de
galaxias en imágenes tomadas
por telescopios por medio de
Computer Vision**

Miguel Ardura Zorita
Director: Carlos Gregorio Rodríguez

Abstract

This project addresses the detection of galaxies within a given raw telescope image. Since the input image comes with a high noise level, preprocessing steps are needed to successfully separate the interest regions, namely, the huge homogeneous pixel areas corresponding to galaxies and the small, isolated and highly illuminated pixel areas corresponding to stars. The detection of these two different objects, galaxies and stars, is carried out by means of two different processing paths, one for each object, and the outcome of them are images with the interest regions highlighted.

Resumen

Este proyecto aborda la detección de galaxias en una imagen sin procesar tomada por un telescopio. Puesto que las imágenes vienen con gran cantidad de ruido, se necesita preprocesarlas para poder diferenciar las zonas de interés, a saber, las grandes zonas homogéneas que corresponden a galaxias y las pequeñas y muy iluminadas zonas que corresponden a estrellas. La detección de estos dos tipos de objetos, galaxias y estrellas, es llevada a cabo por medio de dos caminos distintos, uno para cada objeto, y tienen como resultado imágenes con las zonas de interés destacadas.

Índice general

1 Motivación y objetivos	3
2 Librerías usadas	3
2.1 Ficheros FITS	4
2.1.1 Astropy	5
2.2 OpenCV	5
2.3 NumPy	5
3 Desarrollo del proyecto	6
3.1 Procesado de los ficheros FITS	6
3.2 Detección de los candidatos a galaxia	6
3.2.1 Procesado inicial: eliminación de ruido y estrellas menos luminosas	7
3.2.2 Destacando formas de galaxias con segmentación	8
3.2.3 Detección de las galaxias a través de sus contornos	9
3.3 Detección de candidatos a estrellas	12
3.3.1 Blob detection	14
4 Problemas encontrados	16
5 Conclusión y trabajo futuro	17
A Apéndices	17
A.1 Detalle de las opciones de thresholding y morphology	17
A.1.1 Thresholding	17
A.1.2 Morphology	18
A.2 Muestras del procesado de otras imágenes	18
A.3 Trabajo descartado	26
A.3.1 Sobel	26
A.3.2 Flow	26
A.3.3 Canny	26
A.3.4 3d	26
Bibliografía y referencias	26

1 Motivación y objetivos

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 FITS¹ 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 discriminación de objetos no candidatos a galaxias.

Así pues, mi objetivo era ser capaz de, partiendo de imágenes obtenidas de un telescopio cualquiera, usar OpenCV para crear las herramientas de procesado de imágenes necesarias para conseguir realizar un estudio completo de los objetos que contuviera, centrándome en galaxias y estrellas muy luminosas. Para ello había de ser capaz de encontrar una fuente de ficheros con imágenes astronómicas, a poder ser de acceso libre, y aprender a preparar las imágenes para el procesado. También necesitaba conocer qué métodos de procesado ofrece OpenCV y en qué consiste cada uno, al igual que comprobar cuál era el resultado de aplicarlos a mis imágenes de partida para decidir qué me convenía más usar. Después, necesité ser capaz de extraer la información que me interesaba de las imágenes, al igual que saber qué me podía esperar encontrar en las imágenes según qué técnica aplicara y qué técnicas se han usado hasta la fecha para tales fines. En este punto decidí centrar mis objetivos documentándome con trabajos publicados[1, 2, 3, 4, 5], una vez conocidas las características de las imágenes con las que iba a trabajar[6, 7].

2 Librerías 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).

¹Ver la sección 2.1

2.1 Ficheros FITS

FITS[8] 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 de metadatos. Surgió de la necesidad de disponer de un formato estándar con el que transferir datos astronómicos entre instalaciones científicas.

En 1982 la Unión Astronómica Internacional (IAU por sus siglas en inglés) decidió respaldar formalmente este sistema de intercambio de datos, y desde su asamblea general de 1988 existe un comité derivado de su comisión de datos astronómicos que se encarga expresamente de mantener los estándares de FITS y revisar, aprobar y mantener su futuras extensiones, el IAU Working Group

Un fichero FITS se compone de segmentos llamados HDU (Header/Data Unit). El primero es llamado 'Primary HDU', y generalmente consiste en un array que contiene un espectro de una dimensión, una imagen en dos dimensiones o un conjunto de datos en tres dimensiones.

Este HDU primario puede ir seguido de cualquier número de HDUs. Éstos se denominan *extensiones* y actualmente pueden ser de tres tipos:

- Una imagen, consistente en array de píxeles de dimensión 0-999
- Una tabla ASCII, que almacena información numérica en formato ASCII
- Una tabla binaria, en cuyas celdas pueden almacenarse arrays 1-dimensionales siempre que las dimensiones en una misma columna de la tabla coincidan

Cada HDU consiste en un encabezado en formato ASCII (Header Unit) seguido de una sección opcional que almacena el array de píxeles de la imagen (Data Unit). Cada Header Unit contiene una secuencia de claves con sus valores correspondientes con el siguiente formato

CLAVE = valor / comentario en forma de string

En estas claves se almacena, entre otros, información sobre las características del telescopio que tomó la imagen, coordenadas de la región a la que se apuntaba en el momento de producirse la observación y dimensiones y formato de la imagen generada. Por otro lado, los píxeles de la imagen almacenada en el Primary HDU o en las eventuales extensiones de tipo imagen, serán de uno de los cinco siguientes tipos actualmente soportados: enteros sin signo 8-bit, enteros con signo 16-bit, enteros con signo 32-bit, 32-bit IEEE float y 64-bit IEEE float.

Las imágenes usadas en este trabajo vienen almacenadas en el HDU primario y son 2-dimensionales y float 32. En el código adjunto `imageInfo.py` se muestra el contenido de los HDU del fichero `frame-i-002830-6-0398.fits`, usado a lo largo de esta memoria.

2.1.1 Astropy

Para poder interactuar con los ficheros FITS a través de Python he usado el paquete `astropy.io.fits` de la librería Astropy[9], que es un proyecto libre escrito en Python y diseñado para trabajar en Astronomía.

Este paquete es de un manejo sencillo y su uso se muestra en la sección 3.1.

2.2 OpenCV

OpenCV[10, 11] es una librería multiplataforma de código abierto escrita en C y C++ que permite trabajar con imágenes y visión artificial. Surgió originalmente en 1999 como un proyecto de Intel para mejorar el rendimiento de la CPU y ejecuciones en tiempo real y actualmente cuenta con más de 2500 algoritmos específicos de *computer visión* optimizados, abarcando tareas como el reconocimiento de objetos en imágenes y vídeos, el seguimiento de objetos en vídeo, comparación de imágenes con una base de datos dada, procesado de imágenes para mejorar su calidad, o incluso extraer modelos 3d de objetos. Tiene una comunidad de más de 47.000 usuarios, más de 7 millones de descargas y es actualmente usada por empresas privadas, grupos de investigación e incluso gobiernos. Su desarrollo actual está dirigido por Itseez.

En este trabajo se usa la versión 2.4 de la API para Python.

2.3 NumPy

NumPy[12] es una librería de código abierto resultado de la mejora de en 2005 de la reescritura de Numeric por parte de Travis Oliphant. Es una librería que permite trabajar con arrays multidimensionales e incorpora gran cantidad de funciones de alto nivel sobre estos arrays.

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:

```
1 #Cargar una imagen desde un fichero FITS
2 from astropy.io import fits
3 fitsFile = '../examples/Filters/frame-i-002830-6-0398.fits'
4 hdulist = fits.open(fitsFile)
5 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**[13] realiza a las imágenes en float 32:

```
1 #Mostrar una imagen con OpenCV
2 import cv2
3 import numpy as np
4
5 Min=abs(np.amin(img)) #Se toma el valor absoluto porque por
6     #errores en el CCD pueden existir mediciones ligeramente
7     #erradas. En concreto, negativas muy cercanas a cero
8 Max=np.amax(img)
9 img = 255*(img+Min)/Max
10
11 cv2.namedWindow("Image", cv2.WINDOWNORMAL) #Se crea una ventana
12     "Image"
13 cv2.imshow("Image",img) #Se dibuja img en la ventana Image
14 cv2.waitKey() #Esta sentencia muestra la ventana hasta que se
15     presiona una tecla
```

basicLoading.py

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

3.2 Detección de los candidatos a galaxia

A lo largo de esta sección se muestran los dos caminos seguidos para extraer la información sobre el contenido de la imagen con el fin de detectar galaxias

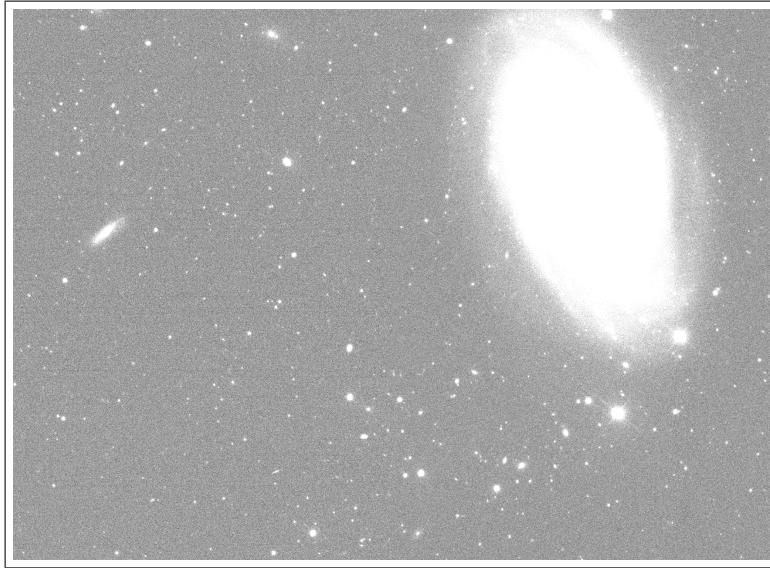


Figura 1: Visualización del archivo frame-i-002830-6-0398.fits por medio de *basicLoading.py*.

y estrellas que contenga.

3.2.1 Procesado inicial: eliminación de ruido y estrellas menos luminosas

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 4), creé un filtro con el que obtengo resultados más cercanos a lo que quería. En concreto consiste en el uso del método **filter2D**[14] de OpenCV. Dicho método sirve para crear un filtro a partir de una matriz dada.

La aplicación de un filtro a una imagen consiste principalmente en la combinación de una matriz, llamada kernel en este contexto, con la imagen original por medio de una convolución. Una convolución es una operación que consiste en combinar los valores del kernel con secciones de la imagen de igual dimensión para obtener unos nuevos valores para los píxeles de la imagen. Partiendo de una imagen y un kernel con uno de sus elementos marcado como *pivot*, el proceso es el siguiente:

1. Se superpone el elemento pivot del kernel con un píxel de la imagen original, con los elementos del resto del kernel superpuestos a los correspondientes píxeles de la imagen.

2. Se multiplican los coeficientes del kernel por el valor de los píxeles correspondientes de la imagen y se suma el resultado.
3. El resultado anterior se fija como nuevo valor del píxel de la imagen sobre el que se situó el pivote.
4. Se repiten los puntos 1, 2 y 3 para el resto de píxeles de la imagen original.

El resultado es una nueva imagen en la que los valores de los píxeles dependen tanto de los coeficientes del kernel como de la imagen anterior. Es por esto por lo que los filtros vienen dados principalmente por el kernel que los representa.

En concreto, creé el kernel $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ después de observar que el in-

tenso granulado innato de las imágenes que tomaba de los FITS hacía que al aplicar los filtros predefinidos de OpenCV el resultado fuera una imagen con gran cantidad de ruido añadido. Intentando destacar regiones luminosas y no tanto conservar las pequeñas regiones de pocos píxeles, decidí ponderar solo los píxeles de alrededor (arriba, abajo, izquierda y derecha) en lugar de considerar más píxeles vecinos, que era lo que añadía el ruido en el caso de los otros filtros.

Tomando la imagen resultado de aplicar este filtro a la imagen leída de los archivos FITS, y aplicando técnicas de segmentación, obtuve resultados con los que pude seguir avanzando.

3.2.2 Destacando formas de galaxias con 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[15], de comparación con un umbral dado. OpenCV ofrece cinco tipos de segmentación a través de su función **threshold** (ver A.1.1).

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[16] y el valor máximo igual a 1 daban los resultados más aproximados a lo que buscaba. El siguiente código recoge la aplicación de esto anterior:

```

1 from astropy.io import fits
2 import cv2
3 import numpy as np
4
5 fitsFile="..//examples//Filters//frame-i-002830-6-0398.fits"
6 hdulist = fits.open(fitsFile)
7 img = hdulist[0].data
8
9 Min=abs(np.amin(img))
10 Max=np.amax(img)
11 img = 255*(img+Min)/Max
12 imgFiltered=cv2.filter2D(img,-1,np.array
13     ([[0,1,0],[1,0,1],[0,1,0]]))
14
15 _, binary=cv2.threshold(imgFiltered, cv2.THRESHOTSU, 1.0, cv2.
16     THRESH_BINARY_INV)
17 cv2.namedWindow("Image", cv2.WINDOWNORMAL)
18 cv2.imshow("Image",binary)
19 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.

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, introduce el estudio de *contornos* en la imágenes obtenidas, lo que lleva a la siguiente etapa del procesado.

3.2.3 Detección de las galaxias a través de sus contornos

Con el fin de identificar y aislar las regiones oscuras que aparecen aplicando thresholding, incorporo el uso del método de OpenCV que detecta contornos,

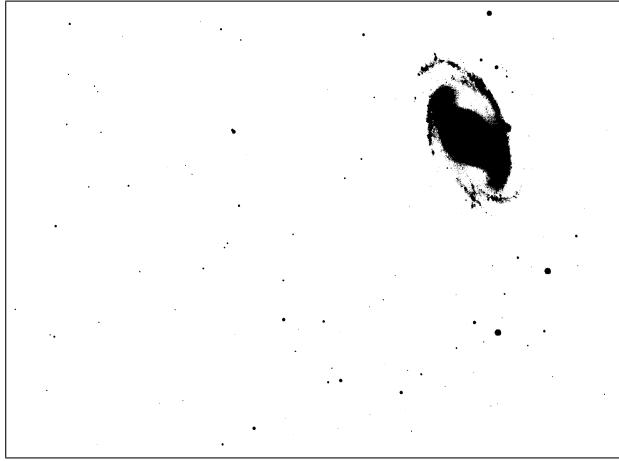


Figura 2: Imagen resultado de aplicar lo anterior a la imagen vista antes.

findContours[17]. Esta función toma como input una imagen de 8 bits, un modo y un método y devuelve la lista de contornos. Como modo utilice 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**[18], 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 -*-
2 from astropy.io import fits
3 import cv2
4 import numpy as np
5
6 fitsFile = "/examples/Filters/frame-i-002830-6-0398.fits"
7 hdulist = fits.open(fitsFile)
8 img = hdulist[0].data
9
10 Min=abs(np.amin(img))
11 Max=np.amax(img)
12 img = 255*(img+Min)/Max
13 imgFiltered=cv2.filter2D(img,-1,np.array
14     ([[0,1,0],[1,0,1],[0,1,0]]))
```

```

16 _ , binary=cv2.threshold(imgFiltered, cv2.THRESH_OTSU, 1.0, cv2.
    THRESH_BINARY_INV)
17 cv2.namedWindow("Image", cv2.WINDOW_NORMAL)
18 cv2.imshow("Image",binary)
19 cv2.waitKey()
20 contours, _ = cv2.findContours(cv2.convertScaleAbs(binary),cv2.
    RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
21 cv2.drawContours(binary,contours,-1,(0,0,0),2) #Este método
    dibuja los contornos en la imagen binary
22
23 cv2.namedWindow("Contours with bounding rectangles", cv2.
    WINDOW_NORMAL)
24
25 max_area=1
26 X,Y,W,H=0,0,0,0
27 for cnt in contours:
28     x,y,w,h = cv2.boundingRect(cnt)
29     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
31 cv2.imshow("Contours with bounding rectangles",binary)
32 cv2.waitKey()
33 cropped=binary[Y:Y+H,X:X+W]
34 cv2.imshow('crop',255*cropped)
35 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
2 X,Y,W,H=0,0,0,0
3 for cnt in contours:
4     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 cálculo de contornos.

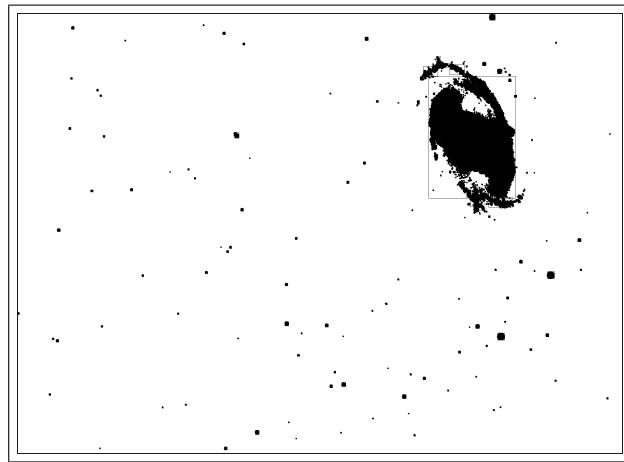


Figura 3: Rectángulos ajustando los contornos.

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

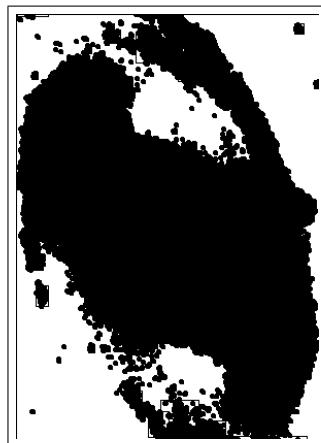


Figura 4: Recorte del contorno identificado.

3.3 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 **threshold** antes mencionados, y lo combino con la función **morphologyex**[19] 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 **morphologyex** 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 rodean. 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 provee de varias operaciones de segmentación (ver el apéndice A.1.2). 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 4).

El siguiente código muestra el uso de la operación cierre:

```

1 # -*- coding: utf-8 -*-
2 from astropy.io import fits
3 import cv2
4 import numpy as np
5
6 fitsFile = ".../examples/Filters/frame-i-002830-6-0398.fits"
7 hdulist = fits.open(fitsFile)
8 img = hdulist[0].data
9
10 Min=np.amin(img)
11 Max=np.amax(img)
12 img = 255*(img+abs(Min))/Max
13 imgFiltered=cv2.filter2D(img,-1,np.array(
14     ([[0,1,0],[1,0,1],[0,1,0]])))
15 _,imgBase=cv2.threshold(imgFiltered, cv2.THRESH_OTSU, 0, cv2.
16     THRESH_TOZERO_INV)
17
18 imgProcessed=cv2.morphologyEx(imgBase, cv2.MORPH_CLOSE, (3,3),
19     iterations=5)
```

blobDetection.py

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



Figura 5: 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.

3.3.1 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.

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/
2 im = cv2.imread("img4.png") #Carga de la imagen generada
    aplicando la operación cierre
4 #Creación del contenedor de los parámetros
```

```

6| params = cv2.SimpleBlobDetector_Params()
8| #Filtrado por convexidad
9| params.filterByConvexity = True
10| params.minConvexity = 0.5
11| #Filtrado por elongación
12| params.filterByInertia = True
13| params.minInertiaRatio = 0.1
14|
15| #Creación del detector con los parámetros fijados
16| detector = cv2.SimpleBlobDetector(params)
17|
18| #Detección de los blobs
19| keypoints = detector.detect(im)
20|
21| #La siguiente función marca con una circunferencia los blobs
22| #detectados
23| im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array
24| ([], (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
25|
26| cv2.namedWindow("Keypoints",cv2.WINDOW_NORMAL)
27| cv2.imshow("Keypoints", im_with_keypoints)
28| cv2.waitKey(0)

```

blobDetection.py

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

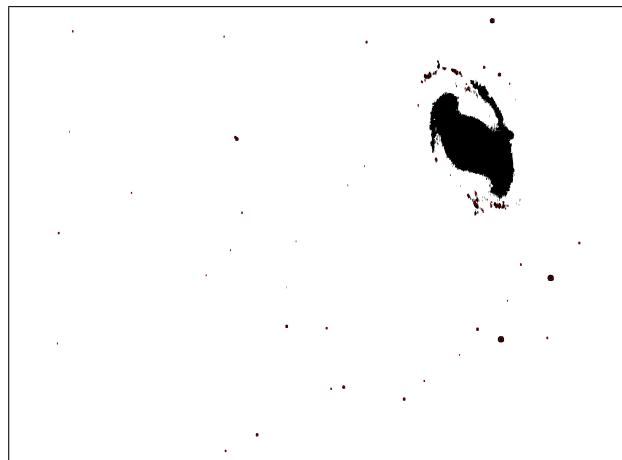


Figura 6: Detección de *blobs*.

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

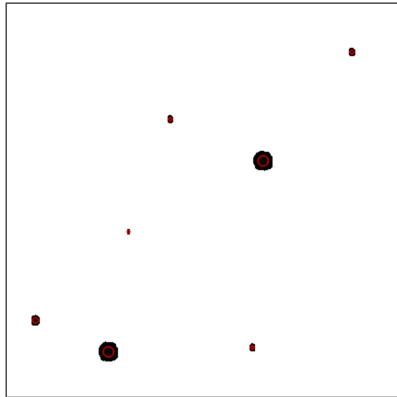


Figura 7: Detección de *blobs* en detalle.

4 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 comenzar a trabajar.

riamente hacer uso de las funciones de OpenCV.

5 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. Esto serviría para salvar el inconveniente de la gran disparidad de las imágenes, que lleva a la necesidad de ajustar para cada imagen los valores de los parámetros usados para realizar una clasificación satisfactoria.

A Apéndices

A.1 Detalle de las opciones de thresholding y morphology

A.1.1 Thresholding

Los cinco tipo de segmentación aplicables en el método **threshold** son:

- **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.

A.1.2 Morphology

El método **morphology** dispone de las siguientes operaciones morfológicas:

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

$$\begin{aligned} \text{apertura}(\text{imagen}, \text{kernel}) = \\ \text{dilatación}(\text{erosión}(\text{imagen}, \text{kernel}), \text{kernel}) \end{aligned}$$

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

$$\text{cierre}(\text{img}, \text{kernel}) = \text{erosión}(\text{dilatación}(\text{imagen}, \text{kernel}), \text{kernel})$$

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

$$\begin{aligned} \text{gradiente}(\text{imagen}, \text{kernel}) = \\ \text{dilatación}(\text{imagen}, \text{kernel}) - \text{erosión}(\text{imagen}, \text{kernel}) \end{aligned}$$

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

$$\text{tophat}(\text{imagen}, \text{kernel}) = \text{imagen} - \text{apertura}(\text{imagen}, \text{kernel})$$

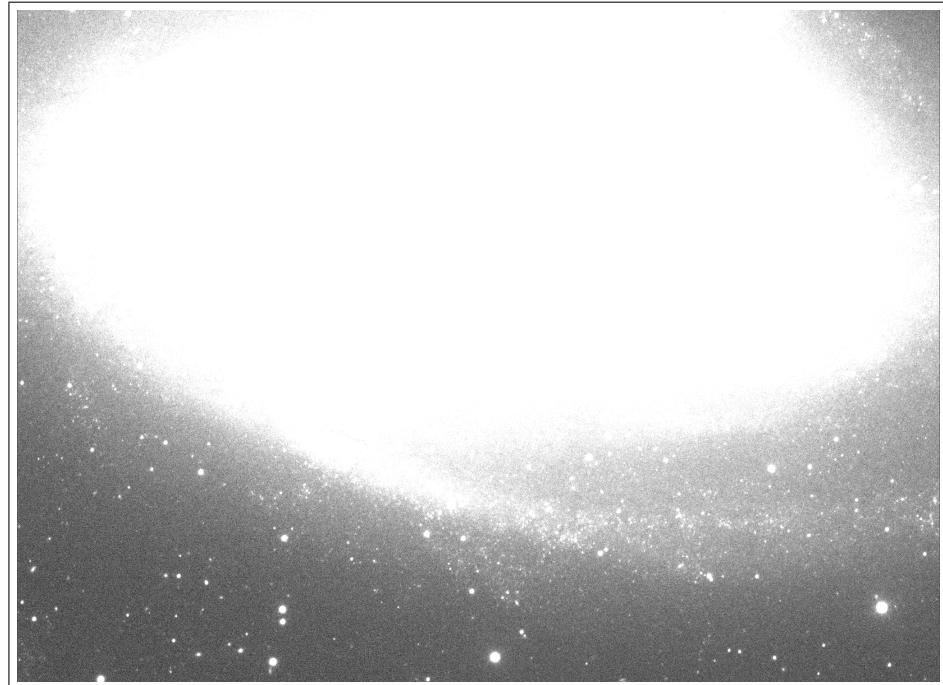
- **Black Hat:** es la diferencia del cierre y la imagen inicial.

$$\text{blackhat}(\text{imagen}, \text{kernel}) = \text{cierre}(\text{imagen}, \text{kernel}) - \text{imagen}$$

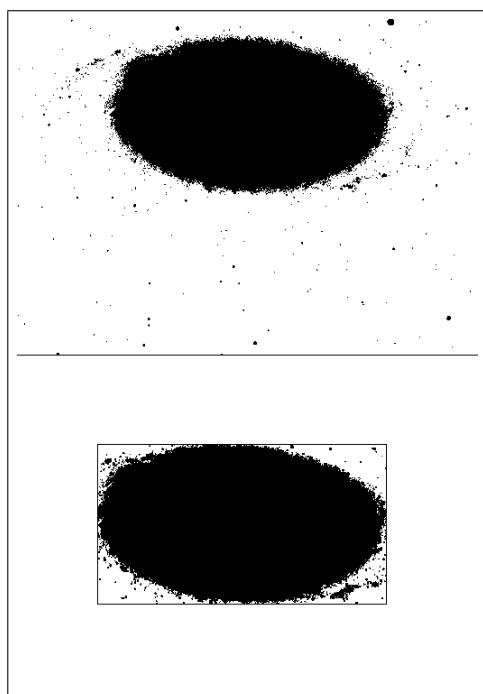
En el script `morphExample.py` se puede ver el efecto de estas operaciones morfológicas.

A.2 Muestras del procesado de otras imágenes

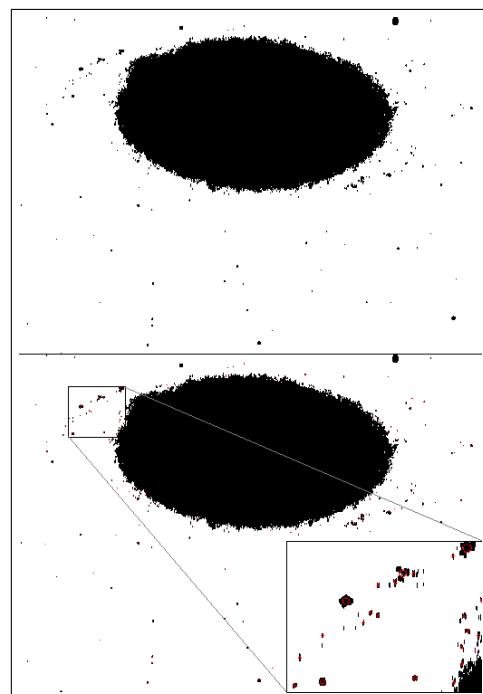
Las figuras 8, 9, 10, 11 12, 13 y 14 muestran el resultado de aplicar a otras imágenes obtenidas de ficheros FITS los mismo procedimientos que se han mostrado a lo largo del punto 3. Se observa cómo de algunas imágenes se obtienen resultados espurios fruto de la necesidad de ajustar los parámetros de algunas funciones casi de manera independiente por la naturaleza de esas imágenes. Se pueden ver ejemplos de este ajuste dinámico, material del trabajo que seguiría al descrito aquí, en los scripts adjuntos.



(a) Imagen base.



(b) Detección de la galaxia por su contorno.

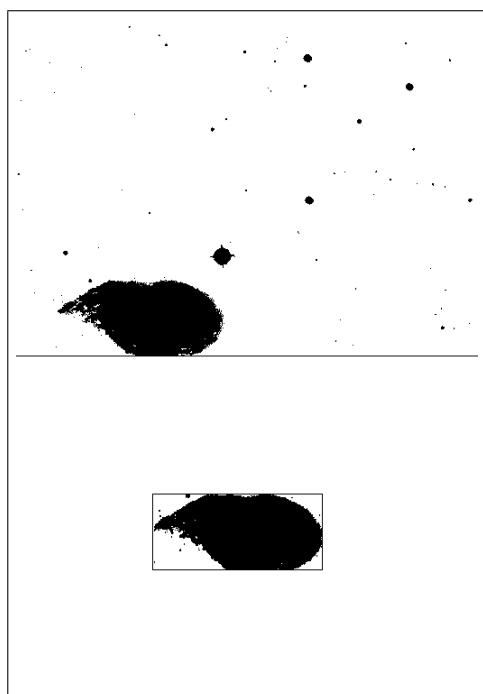


(c) Detección de las estrellas más luminosas.

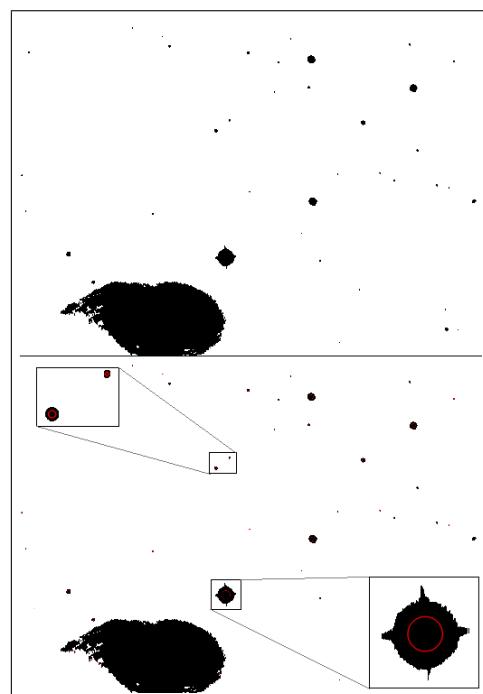
Figura 8: Galaxia m81.



(a) Imagen base.



(b) Detección de la galaxia por su contorno.

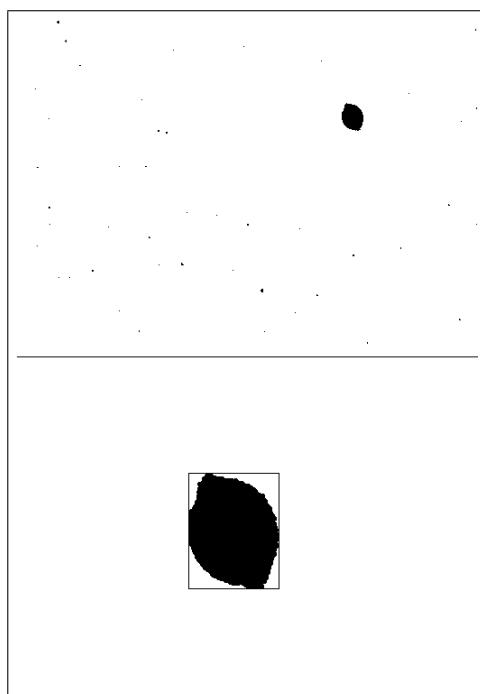


(c) Detección de las estrellas más luminosas.

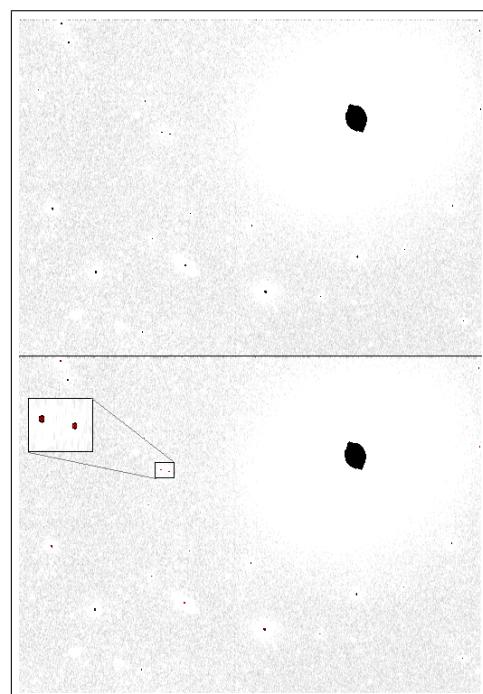
Figura 9: Galaxia m66.



(a) Imagen base.

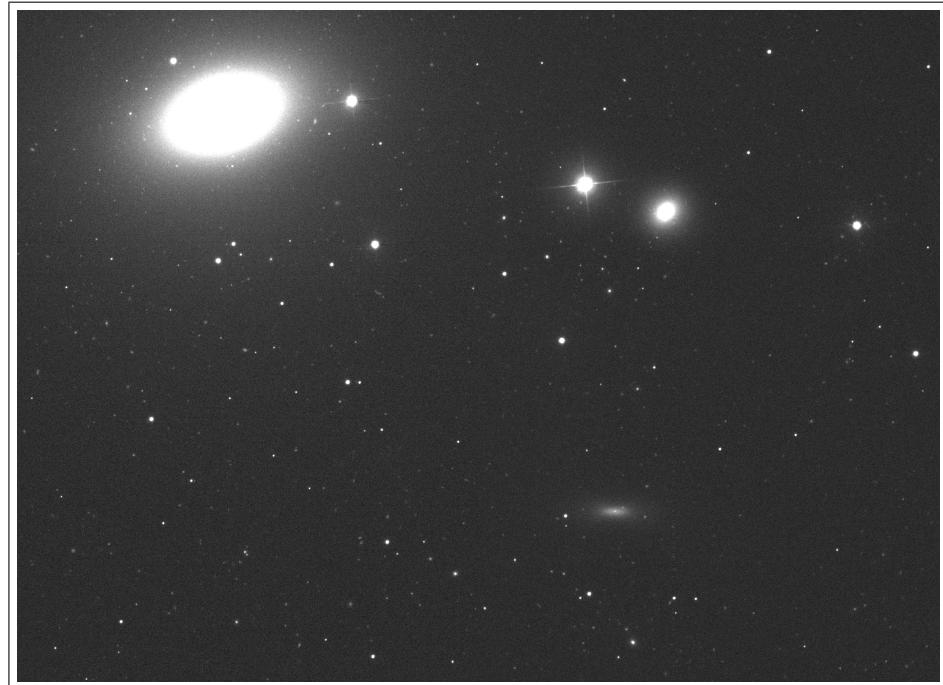


(b) Detección de la galaxia por su contorno.

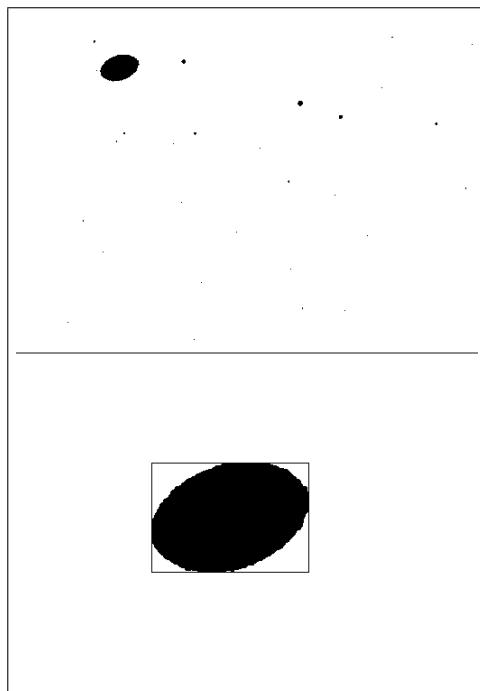


(c) Detección de las estrellas más luminosas.

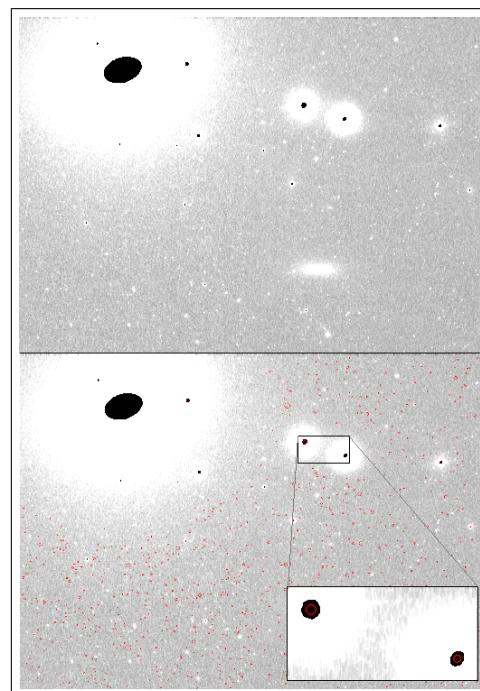
Figura 10: Galaxia m91.



(a) Imagen base.

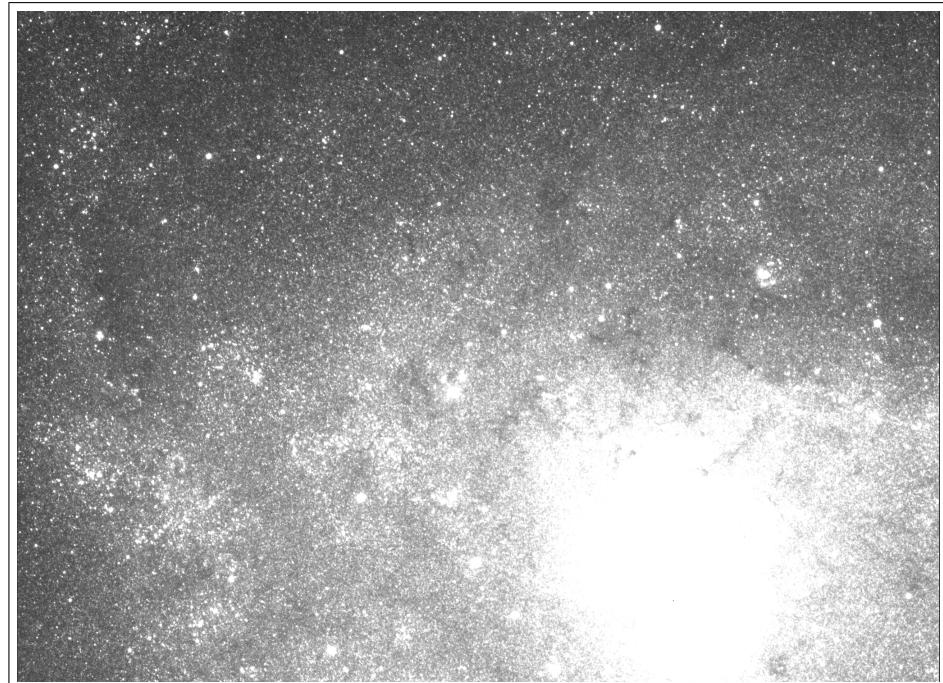


(b) Detección de la galaxia por su contorno.

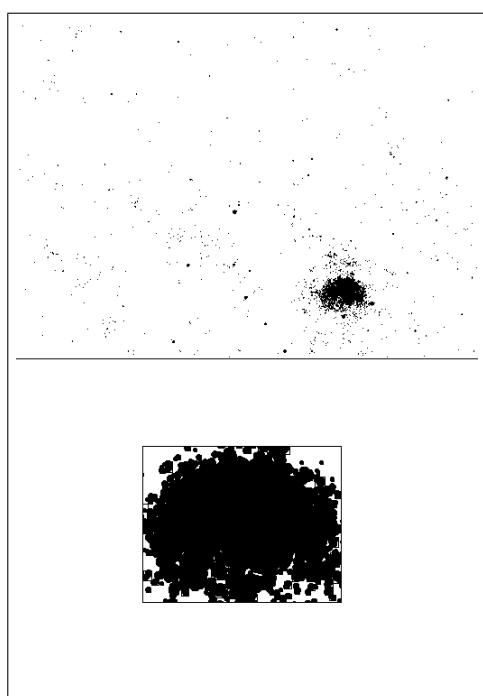


(c) Detección de las estrellas más luminosas.

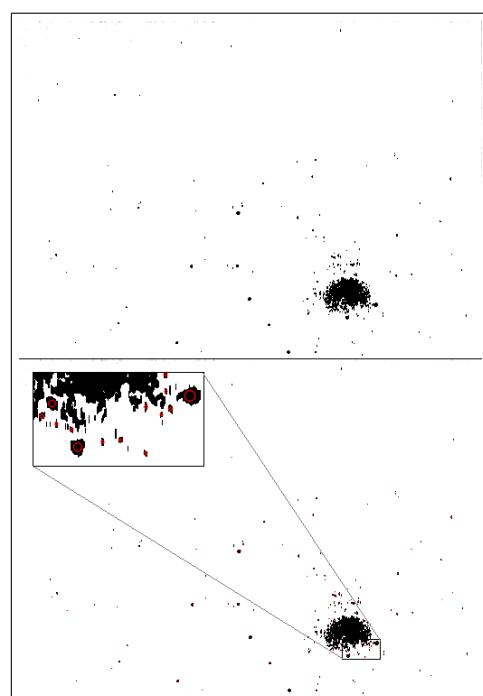
Figura 11: Galaxia m59.



(a) Imagen base.



(b) Detección de la galaxia por su contorno.

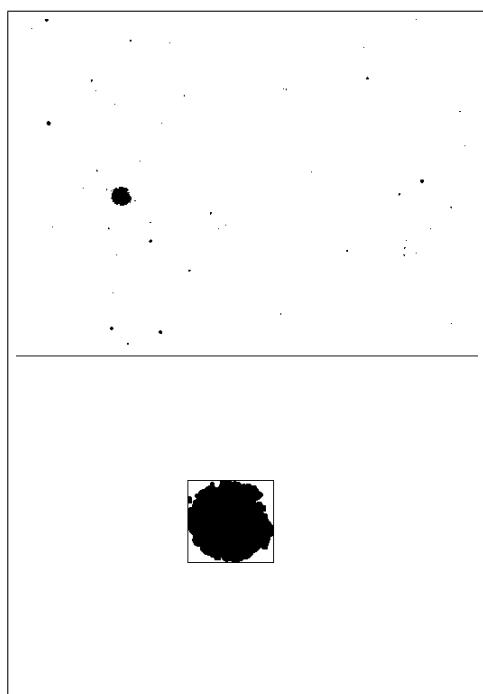


(c) Detección de las estrellas más luminosas.

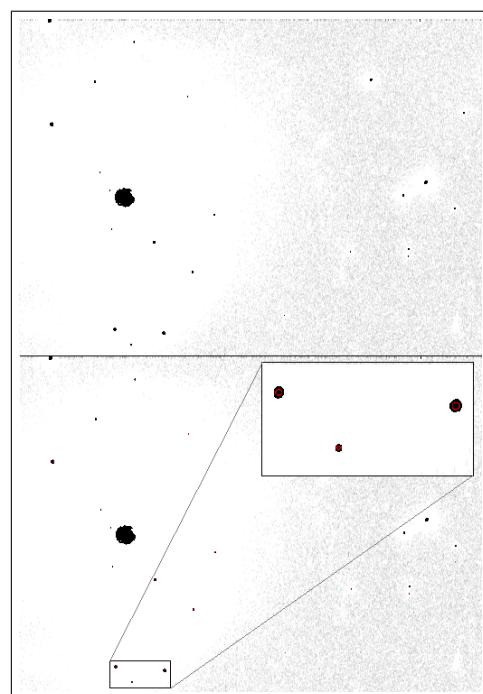
Figura 12: Galaxia m33.



(a) Imagen base.

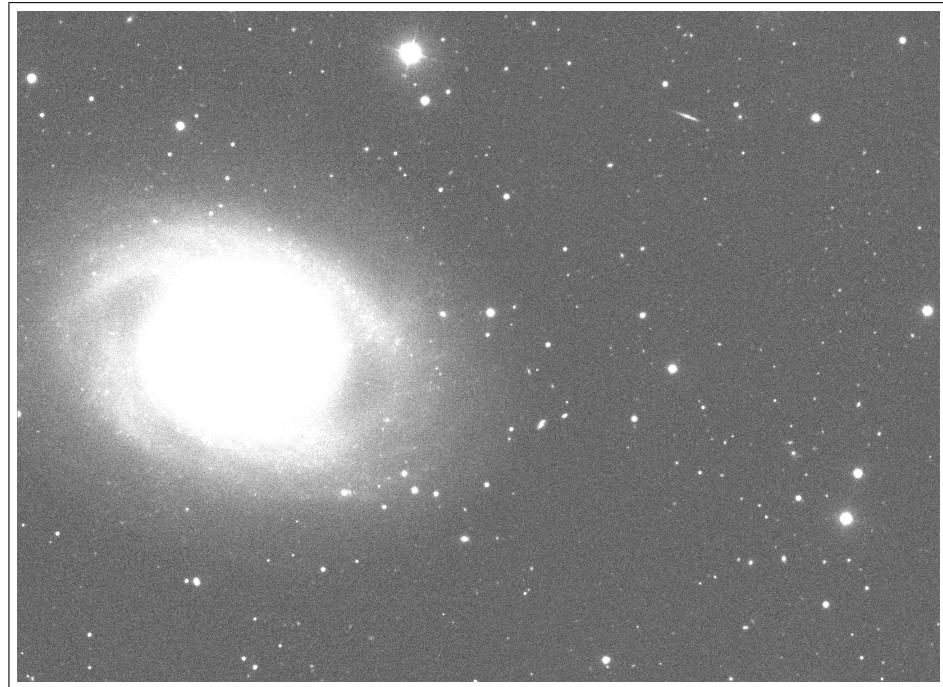


(b) Detección de la galaxia por su contorno.

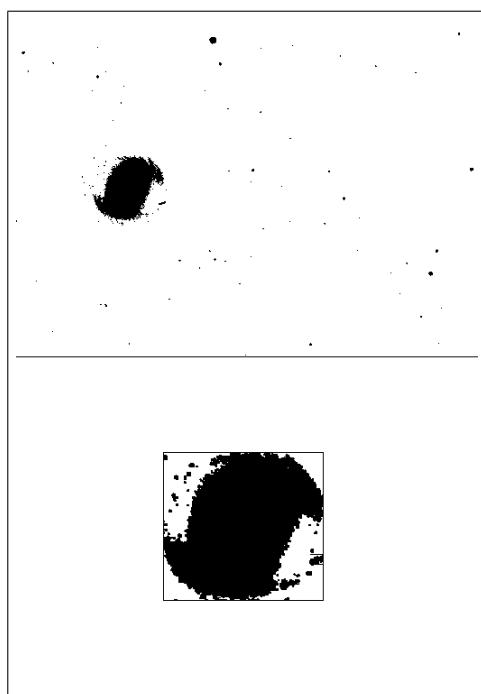


(c) Detección de las estrellas más luminosas.

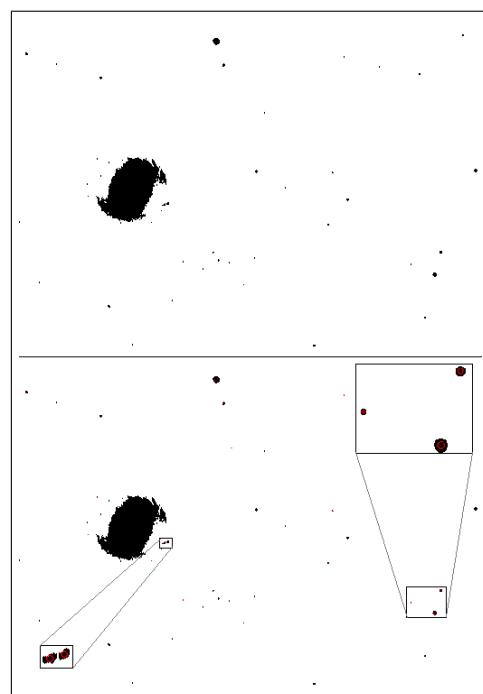
Figura 13: Galaxia m74.



(a) Imagen base.



(b) Detección de la galaxia por su contorno.



(c) Detección de las estrellas más luminosas.

Figura 14: Galaxia m95.

A.3 Trabajo descartado

Esta sección contiene el trabajo realizado pero no utilizado por no proporcionar los resultados que buscaba.

A.3.1 Sobel

En el script SobelExample.py se puede ver el resultado de aplicar el operador Sobel. No lo usé porque no conseguía eliminar el ruido y la forma de la galaxia resultaba demasiado alterada.

A.3.2 Flow

El script flowExample.py muestra el resultado de aplicar el método cornerEigenValsAndVecs, que sirve para determinar esquinas en las imágenes por medio de autovalores y autovectores. No lo usé porque no destacaba suficiente la forma de la galaxia sobre el resto de la imagen.

A.3.3 Canny

En el script CannyExample.py se muestra el resultado de aplicar el algoritmo de Canny para detectar aristas. No lo usé porque al necesitar preprocesar la imagen base para usarla como input del método Canny, se producía un ruido que producía resultados no deseados.

A.3.4 3d

En el script 3dExample.py se muestra un primer acercamiento a la clasificación de los objetos en la imagen por medio de su visualización 3d. Lo acabé descartando porque la enorme dimensión de las imágenes de los ficheros FITS superaba la capacidad de los equipos en los que he trabajado y no pude avanzar en ese sentido.

Bibliografía y referencias

- [1] A. Jandir, U. Mendoça, and E. Costa. A mathematical morphology approach to the star/galaxy characterization. *Journal of the Brazilian Computer Society*, 3(3), apr 1997.
- [2] Zhi-Yong Liu, Kai-Chun Chiu, and Lei Xu. Improved system for object detection and star/galaxy classification via local subspace analysis.

Neural Network Analysis of Complex Scientific Data: Astronomy and Geosciences, 16:437–451, apr 2003.

- [3] Luc Simard, Christopher N. A Willmer, et al. The deep groth strip survey. ii. hubble space telescope structural parameters of galaxies in the groth strip. *The Astrophysical Journal Supplement Series*, 142:1–33, sep 2002.
- [4] Devendra Singh Dhami. Morphological classification of galaxies into spirals and non -spirals. Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Master of Sciences. <http://homes.soic.indiana.edu/ddhami/Master\%20Thesis.pdf>, may 2015.
- [5] Christian Achgill, Devendra Singh, and Tasneem Alowaiseq. Extraction of morphological features of galaxies using computer vision methods. Technical report, Indiana University, 2015.
- [6] The photon’s path from stars to the telescope. <http://slittlefair.staff.shef.ac.uk/teaching/phy217/lectures/principles/101/index.html>.
- [7] Telescopes and spectrographs. <http://www.open.edu/openlearn/science-maths-technology/science/telescopes-and-spectrographs/content-section-1.5.5#fig001-011>, 2016.
- [8] Fits. <http://fits.gsfc.nasa.gov>.
- [9] Astropy. <http://www.astropy.org>.
- [10] Opencv. <http://opencv.org>.
- [11] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O’Reilly, 2008.
- [12] Numpy. <http://www.numpy.org>.
- [13] Método imshow. http://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html\#imshow.
- [14] Método filter2d para la creación de filtros. <http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html\#filter2d>.
- [15] Método threshold. http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=threshold\#threshold.

- [16] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Trans. Sys., Man., Cyber*, 9(1):62–66, jan 1979.
- [17] Método findcontours. [#findcontours](http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours).
- [18] Método boundingrect. [#boundingrect](http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours).
- [19] Método morphologyex. [#morphologyex](http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=morphologyex).