INTRODUCCIÓN A LA VISIÓN ARTIFICIAL. PROCESOS Y APLICACIONES

INTRODUCTION TO COMPUTER VISION. PROCESSES AND APPLICATIONS

TRABAJO FIN DE GRADO
DEPARTAMENTO DE ANÁLISIS Y MATEMÁTICA APLICADA
CURSO 2021-2022

AUTOR
BEATRIZ BORRELLA PETISCO

CODIRECTORES
D. ANTONIO LÓPEZ MONTES (UCM)
D. NICOLÁS ANTEQUERA RODRÍGUEZ (UCM)
D. ANTONIO MARTÍNEZ RAYA (UNED/UPM)

CONVOCATORIA: SEPTIEMBRE 2022

GRADO EN INGENIERÍA MATEMÁTICA
FACULTAD DE CIENCIAS MATEMÁTICAS
UNIVERSIDAD COMPLUTENSE DE MADRID

Dedicatoria

A mi abuelo Emilio.

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres la oportunidad y el esfuerzo realizado para que yo fuera capaz de finalizar la carrera, así como el apoyo y la ayuda constante. Siguiendo con mi familia, quería agradecer también a mis tíos y abuelos por los ánimos recibidos a lo largo de estos cuatros años y sobre todo en el momento final. Por último, a Álvaro, por ser un pilar fundamental y por estar a mi lado en los peores momentos.

Por otro lado, quiero agradecer a mis tutores el trabajo realizado. A Antonio López, por aceptar trabajar conmigo en este TFG desde el primer momento y conseguir darme soluciones a todos los problemas que le iba planteando. A Nicolas, por todo el esfuerzo y trabajo, por darme la idea inicial y por el aprendizaje recibido. Y a Antonio Martínez, por su atenta lectura del trabajo y todas las mejoras y correcciones recibidas.

Finalmente, mi agradecimiento a todos los amigos que me han acompañado estos cuatro años, sin ellos esto hubiera sido mucho más complicado y aburrido.

Resumen

La visión artificial es una disciplina cuyo principal objetivo es procesar y analizar imágenes del mundo real, con la finalidad de que estas puedan ser entendidas y tratadas por un ordenador. El concepto de visión artificial surgió en la década de los sesenta del siglo pasado. Desde entonces ha ido ganando importancia y funcionalidades hasta convertirse hoy en día en uno de los mejores métodos de identificación de imágenes. La principal aplicación de la visión artificial es el reconocimiento de patrones y es utilizada en campos como la vigilancia y la seguridad, el reconocimiento facial y el funcionamiento de sistemas robóticos.

El elemento básico y principal de la visión artificial es el píxel, es decir, cada una de las partes que forman una imagen digital. En rasgos generales, la visión artificial consiste en una serie de operaciones matemáticas sobre los píxeles de la imagen, que varían dependiendo de lo que se quiera conseguir en cada momento. Estas operaciones reciben el nombre de filtros y se realizan a través de máscaras.

En este trabajo comenzaremos con una introducción teórica a la visión artificial, estudiando detalladamente los componentes y las etapas del proceso que se lleva a cabo. Posteriormente, pondremos este proceso en práctica con tres casos diferentes. El objetivo final será llevar a cabo un proceso de visión artificial desde el momento en el que se toma una fotografía hasta que el ordenador es capaz de procesar los elementos que forman esa imagen.

Palabras clave: visión artificial, imagen, píxel, filtro, máscara

Abstract

Computer vision is a discipline whose main objective is to process and analyze images of the real world so that they can be understood and processed by a computer. The concept of machine vision emerged in the 1960s. Since then, it has been gaining importance and functionality and has become one of the best methods of image identification. The main application of computer vision is pattern recognition and it is used in fields such as surveillance and security, facial recognition and the operation of robotic systems.

The basic and main element of computer vision is the pixel, that is, each of the parts that make up a digital image. In general terms, computer vision consists of a series of mathematical operations on the pixels of the image, which vary depending on what is to be achieved at any given time. These operations are called filters and are performed through masks.

In this work we will begin with a theoretical introduction to computer vision, studying in detail the components and stages of the process that is carried out. Subsequently, we will put this process into practice with three different case studies. The final objective will be to be able to create an artificial vision process from the moment a photograph is taken until the computer is able to understand the elements that form that image.

Keywords: computer vision, image, pixel, filter, mask

Índice de Contenidos

1. Introducción a la Visión Artificial

1.1 Definición y Evolución histórica

La visión artificial es una disciplina científica que incluye métodos para adquirir, procesar y analizar imágenes del mundo real con el fin de producir información que pueda ser tratada por una máquina. Es una aplicación de la teórica clásica de matrices a dimensiones considerablemente grandes. Su objetivo principal es la deducción automática de la estructura y propiedades de un mundo tridimensional a partir de una o varias imágenes bidimensionales [1].

En comparación con la visión humana, ofrece una mejor evaluación de las magnitudes físicas y un mayor desempeño en las tareas rutinarias. Algunas de sus múltiples aplicaciones están relacionadas con la vigilancia y la seguridad, el reconocimiento facial, la programación de vehículos autónomos o el funcionamiento de sistemas robóticos mano-ojo [1].

El concepto de visión artificial surgió en la década de los sesenta y su principal objetivo era poder conectar una cámara de vídeo a un computador. Concretamente, la experimentación comenzó en 1959 cuando un grupo de neurofisiólogos le mostraron a un gato una serie de imágenes con la finalidad de comprender como las entendía y procesaba el animal. Descubrieron que sus primeras respuestas iban dirigidas a los bordes o líneas sólidas, con lo que concluyeron que el procesamiento de las imágenes empezaba con formas simples, como los bordes [2].

Al mismo tiempo, comenzó a desarrollarse la primera tecnología de escaneo artificial de imágenes, que permitió a los ordenadores digitalizar y adquirir las imágenes. En 1963 las computadoras fueron capaces de transformar imágenes bidimensionales en formas tridimensionales [2]. Uno de los trabajos que marcó el inicio de la visión artificial fue el realizado por Larry Roberts, creador de ARPAnet, en 1961. Creó un programa llamado “Mundo de Microbloques”, en el que un robot podía “ver” una estructura de bloques sobre una mesa, analizar el contenido y reproducirla desde otra perspectiva. Consiguió que la imagen mandada de la cámara al ordenador fuera procesada adecuadamente por este [3]. Además, en la década de 1960 apareció la Inteligencia Artificial como un campo de estudio académico y esta comenzó a usarse para resolver el problema de la visión artificial. En el año 2000, el estudio se focalizaba en el reconocimiento de objetos, y para el año 2001 se empezaron a crear aplicaciones de reconocimiento facial [2].

Actualmente la visión artificial comprende tanto la obtención como la caracterización e interpretación de los objetos contenidos en una imagen y es uno de los elementos sensoriales más importante para incrementar la autonomía en robótica. Adicionalmente, los métodos utilizados de visión artificial son bastante baratos en comparación con otros métodos que ofrecen resultados similares [4]

1.2 Conceptos preliminares sobre la visión artificial

Antes de explicar el proceso que se lleva a cabo en un sistema de visión artificial tenemos que hablar de los componentes que lo forman.

Para empezar, tenemos la iluminación. Su principal propósito es controlar la forma en que la cámara va a ver el objeto a capturar. Una buena iluminación puede llegar a simplificar considerablemente el posterior procesamiento de la imagen. Además, a través de la iluminación, podemos mejorar el contraste y normalizar cualquier variación de la iluminación ambiente [4].

Por otro lado, tenemos la cámara. Este dispositivo utiliza un juego de lentes y un sensor que recibe la luz reflejada por la escena y permite generar imágenes. La lente se utiliza para transmitir luz al sensor de una forma controlada y así poder tener una imagen enfocada de uno o varios objetos. Los sensores en una cámara son componentes sensibles a la luz que modifican su señal eléctrica en función de la intensidad luminosa que perciben [4].

Siguiendo con los componentes que forman el sistema de visión artificial tenemos el sistema de procesamiento, que recibe las imágenes e implementa funciones dependiendo del tipo de análisis que haya que realizar. El sistema de procesamiento está formado a su vez por tres componentes. En primer lugar, la tarjeta de adquisición, que permite transferir la imagen de la cámara a la memoria de la computadora. Por otro lado, los algoritmos de procesado, conocidos como la parte inteligente del sistema y que aplican las trasformaciones necesarias y las extracciones de información de las imágenes capturadas. Y finalmente, la interfaz, que, una vez realizado el procesamiento de la imagen, es la encargada de mostrar los resultados obtenidos [4].

Por último, tenemos que mencionar los actuadores externos. Se trata del conjunto de elementos que muestran los resultados obtenidos del procesamiento de la imagen y

pueden ser robots, monitores, dispositivos neumáticos e hidráulicos, autómatas programables, etc [4].

Conocidos los componentes de un sistema de visión artificial, podemos ahora pasar a explicar el esquema básico y ordenado que sigue el proceso [5]:

1. Adquisición y digitalización: es el proceso de capturar una imagen y pasarla a algún formato digital. Se utiliza una cámara para capturar la escena y después se envía a una unidad donde pueda ser procesada.
2. Pre-procesamiento: es el proceso que se encarga de eliminar información que no es de interés para el problema.
3. Segmentación: en este punto se reconocen y se extraen cada uno de los objetos presentes en la imagen. Se utilizan técnicas como la detección de bordes o la umbralización de regiones.
4. Extracción de características: se estudian las *"características*" apropiadas para la identificación de los objetos deseados.
5. Identificación de objetos: el último paso consiste en la utilización de un modelo de toma de decisión para decidir a qué categoría pertenece cada objeto.

En los próximos apartados desarrollaremos las partes más importantes de este proceso

1.3 ¿Qué es una imagen digital?

Para empezar, debemos definir el concepto de imagen. Comenzamos con las imágenes más básicas que son las que comúnmente conocemos como imágenes en blanco y negro. Una imagen de dos dimensiones es una función $f(x,y)$ cuyo valor es la intensidad lumínica (nivel de gris) en el punto (x,y) [6]. Es importante diferenciar una imagen real de una digital, la primera es una imagen continua con un número infinito de puntos, sin embargo, la segunda es una imagen discreta con un número finito de puntos.

Cada uno de los puntos de una imagen digital recibe el nombre de píxel. Cada píxel se identifica por la fila y la columna a la que pertenece. Gráficamente tienen la siguiente estructura:

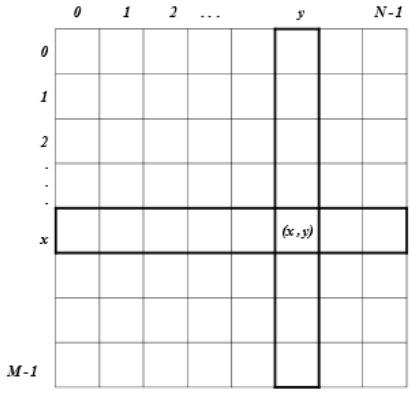


Figura 1.1: Estructura de una imagen digital [7]

Un píxel cuantifica la intensidad de la imagen en ese punto y, por tanto, posee un valor numérico. Al color negro se le asocia el valor 0 y al color blanco el valor $2n - 1$, siendo n el número de cortes diferentes [7]. Por ejemplo, si $n = 4$, entonces $negro = 0$ y $blanco = 2 * 4 - 1 = 15$, lo que significa que consideramos 15 intensidades diferentes de gris. Cuanto mayor sea n mayor será la homogeneidad de la imagen.

Los píxeles también influyen en otras características de la imagen. Por un lado, varía la resolución espacial o nitidez de la imagen, que mejora cuanto más elevado es el número de píxeles. Por otro lado, la resolución en amplitud o tonalidades de gris diferentes, que será mayor cuantos más niveles de gris haya [7].

Figura 1.2: Consecuencias de la variabilidad en el número de píxeles de una imagen [8]

Figura 1.3: Consecuencias del número de niveles de gris en una imagen [8]

Las imágenes a las que hemos hecho referencia son monocromáticas, es decir, solo tienen una banda de color, que es el gris. Sin embargo, generalmente trabajamos con imágenes multiespectrales, estas imágenes son una función con tantas componentes como bandas tenga la imagen [7]. Se definen como:

$$f(x,y) = f_1(x,y), f_2(x,y), ., f_n(x,y)).$$

Las imágenes en color que vemos en nuestro día a día tienen tres bandas que dependen del espacio de color, el más conocido es el *RGB (Red*, *Green*, *Blue)* [7]:

$$f(x,y) \ = \ (red(x,y), green(x,y), blue(x,y))$$

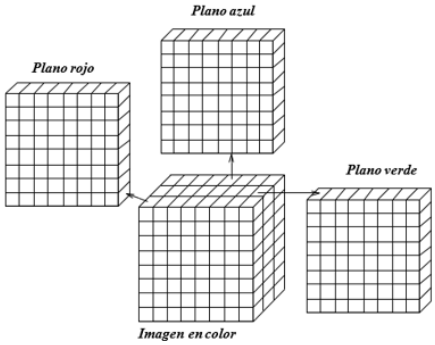


Figura 1.4: Esquema de una imagen en color descompuesta en cada uno de sus planos [7].

Otro concepto asociado a las imágenes y que debemos tener en cuenta ya que nos va a ser útil a la hora del pre-procesamiento son las máscaras digitales o filtros.

Comenzamos por el concepto de vecindad. Cada punto p de la imagen establece una relación de vecindad o adyacencia con los puntos que le rodean. Podemos tener dos tipos de vecindades: La más básica es la de orden 4 y considera a los dos vecinos horizontales y a los dos vecinos verticales. Un poco más compleja es la de orden 8 que se utiliza en la mayoría de los casos y considera los ocho vecinos que la rodean, dos verticales, dos horizontales y los cuatro diagonales [7].

	(x−1, y)	
(x, y−1)	(x, y)	(x, y+1)
	(x+1, y)	

Vecindad de Orden 4

(x−1, y−1)	(x−1, y)	(x−1, y+1)
(x, y−1)	(x, y)	(x, y+1)
(x+1, y−1)	(x+1, y)	(x+1, y+1)

Vecindad de Orden 8

Figura 1.5: Concepto de vecindad en una imagen [7]

Una vez establecido el concepto de vecindad podemos introducir las máscaras digitales. Las máscaras digitales, también conocidas como ventanas, filtros o plantillas, son matrices de tamaño reducido donde cada coeficiente (x, y) corresponde a un píxel. Son generalmente cuadradas y todas las que utilizaremos en este trabajo serán de tamaño 3x3. En cada una de las entradas de la matriz encontramos un coeficiente, w_i, cuyo valor dependerá de la máscara que estemos utilizando [7].

w1	*w2*	*w3*
w4	*w5*	*w6*
w7	*w8*	*w9*

Figura 1.6: Ejemplo máscara de dimensión 3x3 [7]

El proceso de aplicar estas máscaras recibe el nombre de convolución digital. La máscara se superpone sobre cada vecindario y siguiendo con el ejemplo de la máscara 3x3 se realiza la siguiente operación aritmética [7]:

$$C = \sum_{i=1}^{9} (z_i * w_i) = z_1 * w_1 + \cdots + z_9 * w_9$$

El resultado de la operación se le asigna al punto central del vecindario

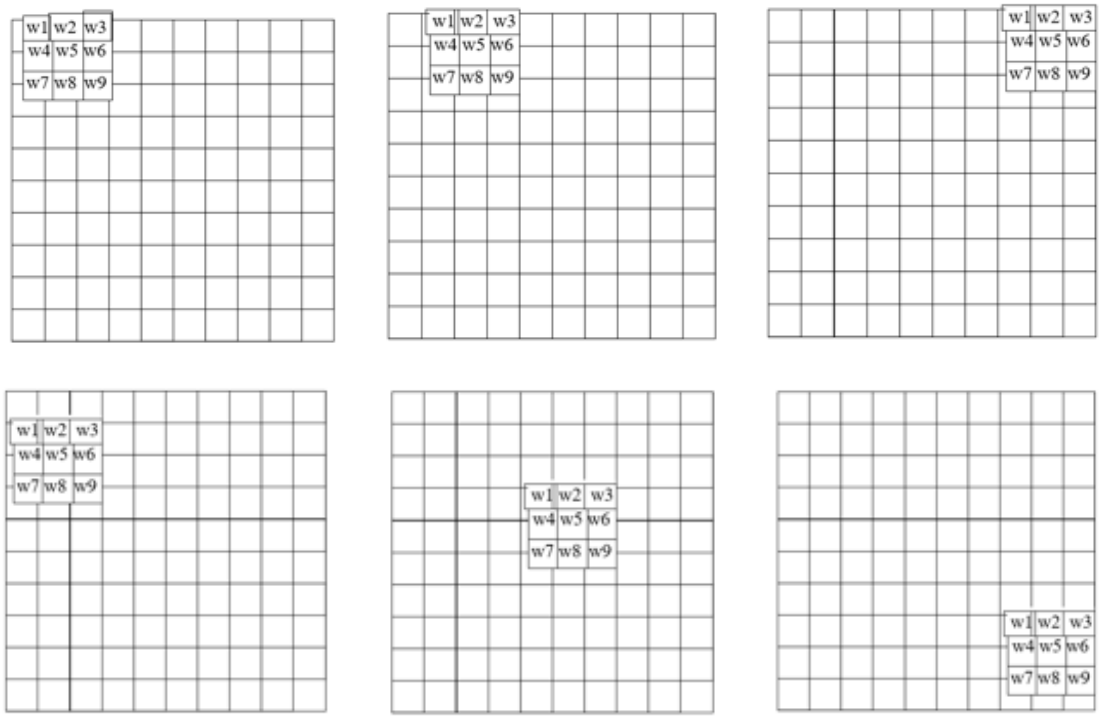


Figura 1.7: proceso de convolución de una máscara [7]

1.4 Fase de pre-procesamiento del sistema de visión artificial

1.4.1 Filtros espaciales

En esta fase tienen especial importancia los filtros que le aplicamos a la imagen. Los filtros son operaciones que se hacen sobre la imagen para provocar un cambio en esta [6]. Estos cambios lo que generalmente buscan es suavizar las imágenes, es decir, reducir el ruido y los elementos que no son de interés. Es importante que tengamos la clara la diferencia entre filtro y máscara. Vamos a utilizar el concepto de filtro de forma general para referirnos a cada uno de los procesos que le aplicamos a la imagen, mientras que las máscaras serán las matrices 3x3 formadas por coeficientes. En resumen, cada filtro tiene una matriz 3x3 asociada con sus respectivos coeficientes a la que daremos el nombre de máscara.

Para comenzar centraremos el estudio en los filtros espaciales. En este amplio grupo podemos diferenciar dos tipos de filtros, los lineales que explicaremos a continuación y los no lineales que veremos más adelante. Los filtros lineales reciben este nombre porque los valores de intensidad de los píxeles dentro de la región de procesamiento se combinan de manera lineal para generar el píxel resultado [6]. Dentro de los filtros lineales cabe

destacar los filtros de suavizado. Su principal objetivo es suavizar la imagen, es decir, reducir las variaciones de intensidad entre píxeles vecinos y eliminar el ruido. Esto consiste en modificar los píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos [6]. Dentro de los filtros de suavizado vamos a estudiar los tres tipos de filtros más conocidos y utilizados que son el filtro *Box*, el filtro de las medias y el filtro Gaussiano.

Comenzamos con el filtro *Box*. Es el más simple y antiguo de todos. En la siguiente imagen podemos ver la forma de este filtro:

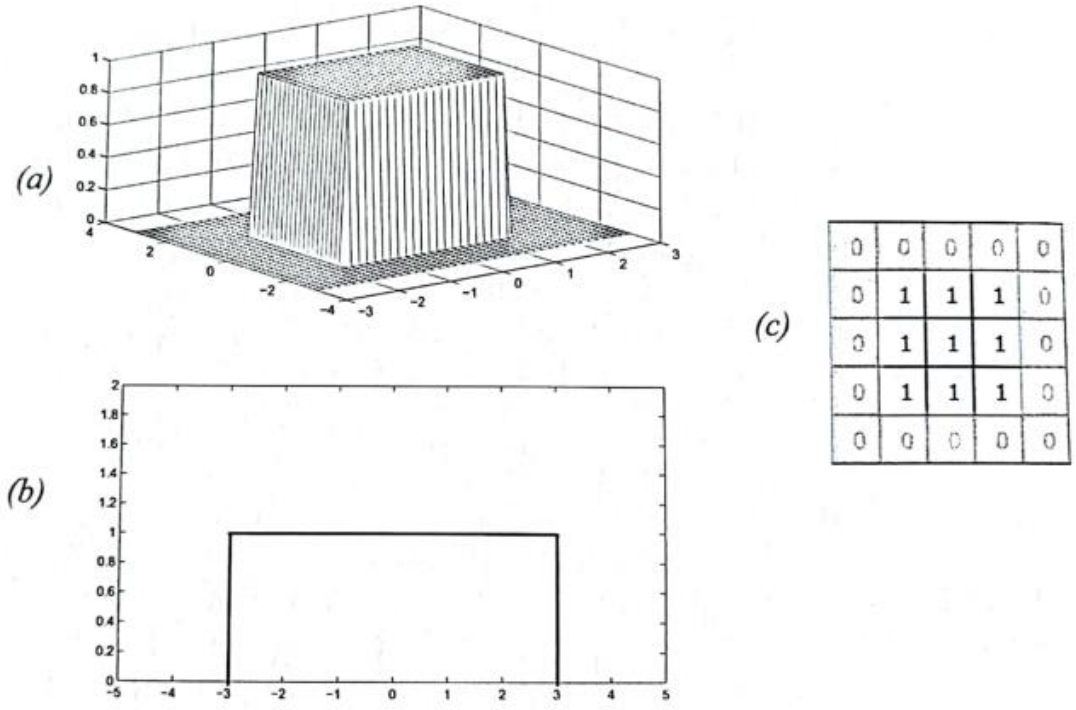


Figura 1.8: El filtro para suavizado Box: a) Función tridimensional, b) Función bidimensional y c) mascara que implementa el filtro [6]

Como podemos ver en la imagen este filtro parece una caja, de ahí su nombre, y afecta de manera uniforme a cada píxel de la imagen ya que todos los coeficientes tienen el mismo valor. Sin embargo, este filtro presenta unos bordes muy agudos, lo que hace que a pesar de su sencillez sea un filtro poco recomendable [6].

Siguiendo la idea del filtro *Box* aparece el filtro de la media. Este filtro consiste en reemplazar el valor de cada píxel por la media de los valores de los pixeles vecinos [9]. Un ejemplo de mascara 3x3 para este filtro seria la siguiente:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Figura 1.9: Ejemplo de mascara 3x3 para el filtro de la media

Este filtro presenta el mismo problema que el anterior y es que reduce muy bien el ruido, pero a la vez difumina mucho los bordes de los objetos [9].

Intuitivamente parece más razonable darle mayor peso al píxel central ya que es del que más información queremos conservar y menor peso a los pixeles vecinos. Atendiendo a este razonamiento aparece el filtro Gaussiano. En esencia, el filtro Gaussiano corresponde a la función de Gauss bidimensional y discreta dada por la fórmula [6]:

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Donde la desviación estándar σ representa el radio de cobertura de la función de Gauss tal y como vemos en la siguiente imagen:

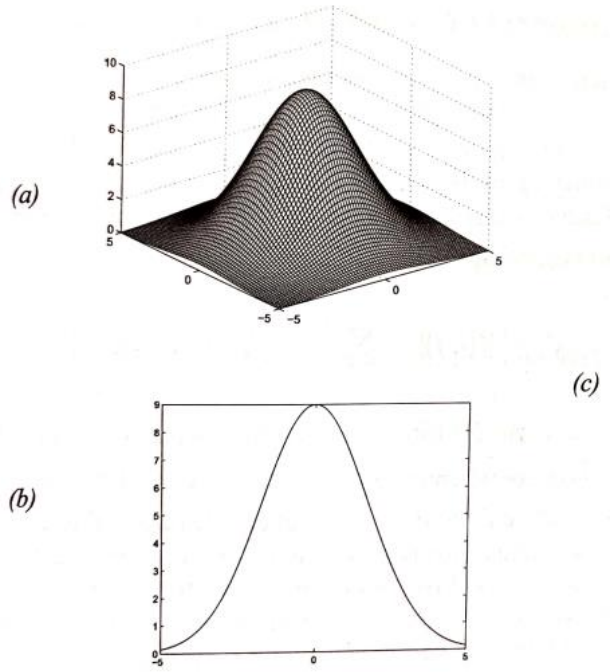


Figura 1.10: Filtrado Gaussiano. (a) Función tridimensional, (b) Función bidimensional y (c) mascara Gaussiana que implementa el filtro [6]

El elemento central del filtro representa el peso máximo que participa en la combinación lineal de la operación, mientras que los demás coeficientes tienen menor influencia según estos se alejan del centro del filtro. Una de las propiedades más relevantes de este filtro es que es invariante a las rotaciones, y en comparación con el filtro *Box* y el filtro de la media es mejor opción cuando el objetivo principal es mantener los bordes de los objetos de la imagen. Esto se debe a que este filtro permite suavizar las regiones en donde los valores de intensidad son homogéneos sin diluir de forma tan notable los bordes.

Aunque el filtro de Gauss consiga suavizar la imagen sin degradar por completo estructuras como puntos, líneas o bordes, es imposible aplicar filtros de suavizado lineales y mantener estas estructuras intactas. Es aquí donde aparecen los filtros no lineales. Su principal característica es que permiten resolver este problema de una mejor forma que el filtro de Gauss utilizando para ello operaciones no lineales [6]. Los filtros no lineales también calculan el resultado de un determinado píxel en la posición (x, y) utilizando una determinada región relativa a la imagen original. Dentro de estos filtros no lineales los más sencillos son los filtros máximos y mínimos [9] y se basan en las siguientes ecuaciones:

$$I'(x,y) = \min\{I(x+i, y+j) | (i,j) \in R\} = \min\left(R(x,y)\right)$$

$$I'(x,y) = \max\{I(x+i, y+j) | (i,j) \in R\} = \max\left(R(x,y)\right)$$

Donde $R(x, y)$ representa la región relativa a la posición (x, y) definida por el filtro que generalmente es un rectángulo 3x3 [6]. Veamos en la siguiente imagen el funcionamiento de estos filtros:

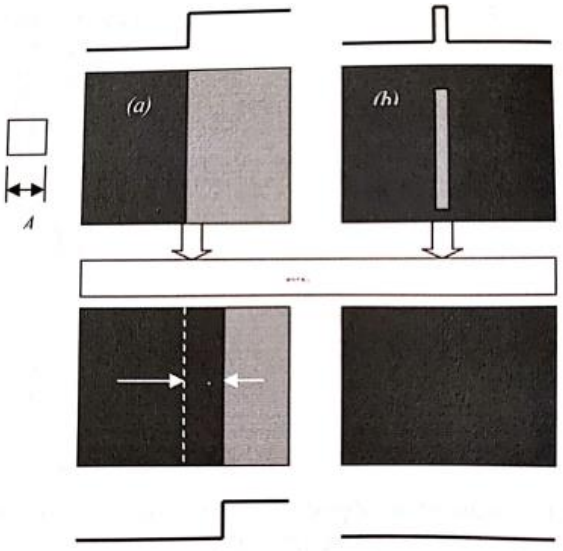


Figura 1.11: Efecto del filtro mínimo sobre diferentes formas locales en una imagen [6]

En esta figura vemos la imagen original arriba y el resultado obtenido por la operación filtro abajo. El valor A describe el ancho del filtro utilizado que también define al mismo tiempo la región R abarcada por él. En la imagen a) vemos representado un escalón que debido a la operación del filtro es desplazado hacia la derecha A unidades. Por otro lado, en la imagen b) la anchura de la línea contenida en la imagen es menor a la de A y por tanto, esta desaparece por efecto del filtro [6].

El funcionamiento de estos filtros se basa en mejorar fotos contaminadas con el tipo de ruido que se conoce como sal y pimienta. Este ruido simula la aparición aleatoria de píxeles blancos (sal) y píxeles negros (pimienta). El filtro de máximo selecciona el mayor valor dentro de una ventana ordenada de valores de nivel de gris, es decir, elimina los píxeles negros (ruido pimienta) [9]. Sin embargo, tiene algunos inconvenientes como que solo funciona cuando el ruido es exclusivamente de tipo pimienta y tiende a aclarar la imagen ya que agranda los píxeles con un valor permisible por la imagen (blanco o sal) [9]. Por su parte, el filtro de mínimo selecciona el menor valor dentro de una ventana ordenada de valores de nivel de gris, por lo que elimina los píxeles blancos (ruido sal), pero sus inconvenientes son que solo funciona cuando el ruido es exclusivamente de tipo sal y por la misma razón que el filtro anterior, tiende a oscurecer la imagen.

El último filtro no lineal que vamos a ver es el filtro de la mediana. Este filtro basa su cálculo en una medida de rango en un conjunto ordenado de datos, de manera que un dato con un valor muy grande o muy pequeño en comparación al valor del resto de datos de la región de interés $R(x, y)$ no influirá significativamente en el resultado final [6]. La manera de actuar de este filtro es visitar cada píxel de la imagen y reemplazar su valor por la mediana de los píxeles vecinos [9]. Considerando que $x_1, x_2, \dots, x_n$ son los datos de una muestra ordenada crecientemente, en el caso que n sea par la mediana de esa muestra se define como:

$$M_e = x_{\frac{n+1}{2}}$$

Si n es impar sigue la fórmula:

$$M_e = \frac{x_{\frac{n}{2}} + x_{\frac{n+1}{2}}}{2}$$

Entonces la expresión que sigue el filtro de la mediana es la siguiente:

$$I'(x, y) = M_e\big(R(x, y)\big)$$

A continuación, vemos un ejemplo de cómo se aplica este método considerando un filtro 3x3:

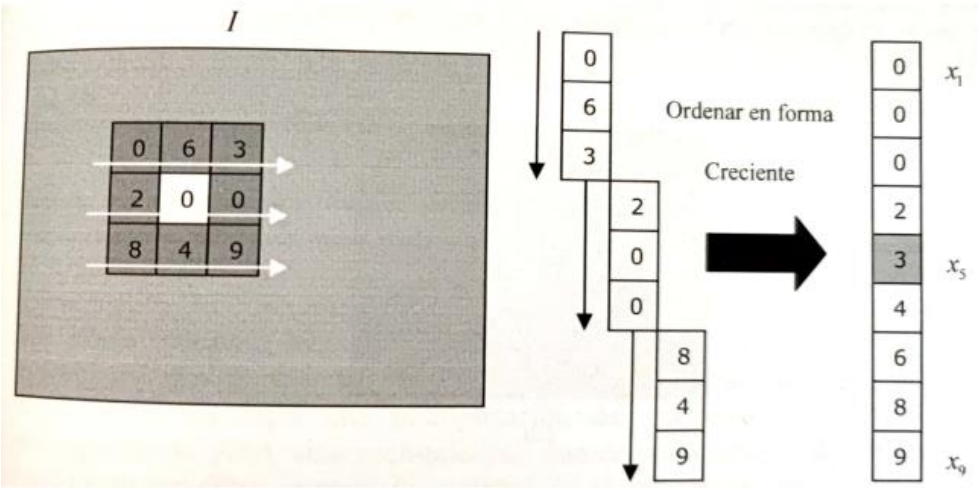


Figura 1.12: Ejemplo de cálculo de la mediana considerando un filtro 3x3 [6]

1.4.2 Filtrado espectral

Dejando a un lado los filtros espaciales, centramos ahora el estudio en el filtrado espectral. Su objetivo, al igual que el de los filtros anteriores es eliminar elementos que no son de interés para el estudio. Este tipo de filtros se basan en la idea de que una imagen se puede representar como la suma de dos componentes con diferentes escalas espaciales, por un lado, la versión de la imagen con las bajas frecuencias y por otro lado la de la imagen con las frecuencias altas [10]. En términos generales una señal unidimensional se puede descomponer en un conjunto infinito de señales de tipo seno con diferentes frecuencias, amplitudes y fases. El primer componente representa la amplitud media de la señal y tiene frecuencia igual a 0, el siguiente componente (componente fundamental) presenta la misma frecuencia que la señal original. Los demás componentes van adquiriendo frecuencias sucesivamente superiores y se denominan armónicos. A medida que se añaden componentes, la suma se aproxima cada vez más a la señal original, aunque necesitamos infinitos componentes para que la suma sea exacta. Esta suma total de componentes recibe el nombre de Serie de Fourier de la función original [10].

Podemos entender una imagen digital como una señal discreta bidimensional, de forma que su serie de Fourier es una suma finita de senos y cosenos que formará la imagen original cuando sumemos todos los términos. La suma parcial de las primeras

componentes es la formada por las frecuencias más bajas y producirá una versión de la imagen parecida al aplicar un filtro de paso-bajo a la imagen original. De la misma forma la suma de los últimos términos de la serie con frecuencias superiores generará una versión de la imagen con altas frecuencias o parecida a aplicar un filtro de paso-alto [10].

La transformada de Fourier de una función continua viene dada por la siguiente expresión:

$$F(u) = \int_{-\infty}^{\infty} f(x) * e^{-2\pi iux}\, dx$$

donde u es la variable en el espacio de la frecuencia. Análogamente, dada $F(u)$, podemos obtener la función $f(x)$ calculando la transformada inversa de Fourier como:

$$f(x) = \int_{-\infty}^{\infty} F(u) * e^{2\pi iux}\, du$$

Ambas funciones existen siempre que $f(x)$ sea continua e integrable y que $F(u)$ sea integrable [10].

Una vez introducida la Transformada de Fourier veamos su aplicación en este tipo de filtros a partir del Teorema de Convolución. Como hemos explicado antes, la operación de convolución de una imagen digital en el espacio bidimensional (x, y) consiste en la aplicación de un filtro o máscara de un determinado tamaño, píxel a píxel, operando con los valores de la imagen en el vecindario de cada píxel y con los pesos asociados al filtro que estamos aplicando. Consideramos la convolución discreta de una imagen digital $f(x, y)$ mediante el filtro $h(x, y)$, según el vecindario conocido por W como:

$$f(x, y) * h(x, y) = g(x, y) = \sum_{i,j \in W} \sum f_{ij}(x, y) \cdot h_{ij}(x, y)$$

Si $F(u, v)$ es la transformada de Fourier de $f(x, y)$ y $H(u, v)$ es la transformada de Fourier de $h(x, y)$, entonces $f(x, y) * h(x, y)$ tiene como transformada $F(u, v) \cdot H(u, v)$. Es decir,

$$f(x, y) * h(x, y) \Leftrightarrow F(u, v) \cdot H(u, v)$$

De la misma forma, la convolución en el dominio de la frecuencia se reduce a la multiplicación en el dominio espacial:

$$f(x, y) \cdot h(x, y) \Leftrightarrow F(u, v) * H(u, v)$$

[10].

A continuación, vemos gráficamente el funcionamiento de estos filtros:

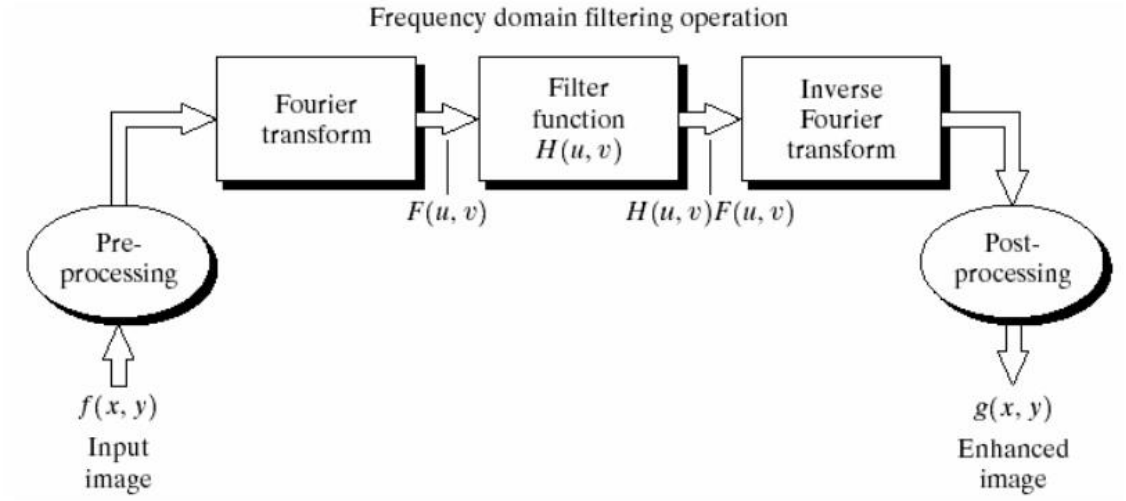


Figura 1.14: Diseño y aplicación de filtros en el dominio de la frecuencia, mediante la utilización de la Transformada de Fourier (TF) y de su inversa $(TF)^{-1}$ [9]

Por último, veamos unas imágenes de ejemplo para comprender la información que nos da este tipo de filtros.

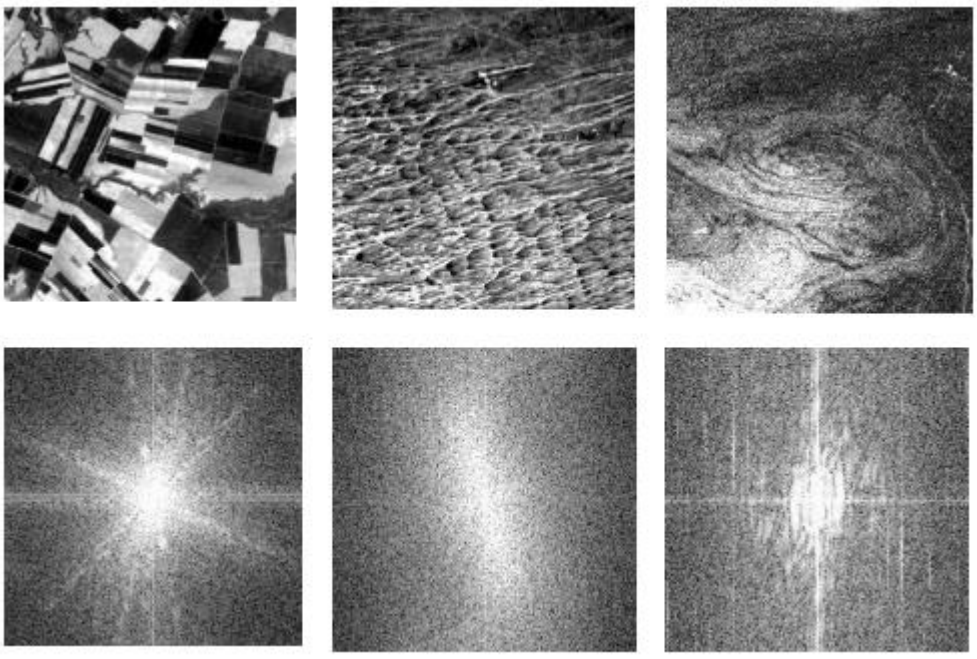


Figura 1.13: Ejemplo de tres imágenes con sus respectivas transformadas [10]

Aquí podemos ver tres distribuciones espaciales distintas y el efecto que provocan en la representación en forma de imagen de su transformada, teniendo en cuenta también el componente direccional de las frecuencias [10]. En la primera imagen vemos varias

parcelas agrícolas de distintos cultivos, lo que provoca una gran cantidad de altas frecuencias en varias direcciones. Por otro lado, la segunda imagen quiere representar un paisaje dunar, en la que observamos una orientación dominante NE (Norte, Este) – SO (Sur, Oeste) debido a la acción de los vientos. Por su parte, la transformada aparece en forma de nube, debido a que existe poca definición en la orientación dominante, no podemos decidir si es NE o SO. En la última imagen vemos una zona oceánica con bastantes corrientes en forma helicoidal. Esto hace que su transformada presente, por un lado, una serie de manchas próximas al origen, correspondientes a las altas frecuencias, y por otro, unas manchas más ligeras y difuminadas en los bordes, que corresponden a las bajas frecuencias.

1.5 Fase de Segmentación del Sistema de Visión Artificial

Una vez completada la etapa de pre-procesamiento, la imagen ya estará preparada para continuar con la siguiente fase, la de segmentación. Generalmente decimos que esta es la etapa más importante ya que es aquí donde vamos a obtener la información que queremos de la imagen.

La segmentación consiste en dividir una imagen digital en varias regiones o grupos de píxeles denominadas segmentos, es decir, se trata de un proceso de clasificación por píxel que asigna una categoría a cada píxel de la imagen analizada [7], en nuestro caso utilizaremos técnicas de segmentación para detectar los bordes de la imagen y como consecuencia poder identificar los objetos y el número de veces que aparecen.

Uno de los casos más simples para detectar objetos, sin entrar todavía en la detección de bordes, es la binarización de la imagen. Es decir, definimos un umbral llamado T y damos el valor 0 (negro) o 255 (blanco) a cada píxel dependiendo si el valor de dicho píxel es mayor o menor que nuestro umbral T. Siendo $f(x, y)$ cada uno de los píxeles de la imagen un ejemplo sería:

$$\begin{cases} Si\ f(x,y) < T\ entonces\ f(x,y) = 0 \\ Si\ f(x,y) \geq T\ entonces\ f(x,y) = 255 \end{cases}$$

Este procedimiento es muy eficaz cuando queremos identificar el número de objetos que hay sobre un fondo blanco, por ejemplo, funciona muy bien a la hora de detectar el número y palo de una baraja de cartas o la posición de las fichas en un tablero del tres en raya o incluso en un tablero de ajedrez. Para el resto imágenes con más colores o formas

en el fondo antes de poder identificar directamente los objetos tenemos que aplicar técnicas de detección de bordes.

El caso más básico es la detección de bordes binaria. Para empezar, tenemos que binarizar la imagen con la técnica explicada anteriormente y después, se procesan primero las filas de arriba a abajo y de izquierda a derecha haciendo para cada píxel [8]:

$$\begin{cases} f(x,y) = 1 \; si \; f(x,y-1) \; y \; f(x,y) \; están \; el \; niveles \; de \; intensidad \; opuestos \\ f(x,y) = 0 \; si \; f(x,y-1) \; y \; f(x,y) \; están \; el \; niveles \; de \; intensidad \; iguales \end{cases}$$

Se procede de igual forma para las columnas y se realiza la misma asignación. Una vez tengamos las dos imágenes estas se operan mediante la operación "OR" y se obtiene la imagen binaria de borde.

El problema aparece cuando añadimos más niveles de gris. En este caso se usan las derivadas direccionales (gradiente espacial) u operadores de primera derivada, que consiste en detectar las regiones de transición brusca de intensidad mediante diferenciación espacial [7]. El gradiente de una imagen en cualquier punto se define como un vector bidimensional dado por la siguiente ecuación [7]:

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix} = \begin{bmatrix} \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \\ \frac{f(y+\Delta y) - f(y-\Delta y)}{2\Delta y} \end{bmatrix}$$

Donde el vector G apunta en la dirección de variación máxima de f en el punto (x,y) por unidad de distancia siendo [7]:

$$\text{Magnitud: } |G| = \sqrt{{G_x}^2 + {G_y}^2} \qquad\qquad \text{Dirección: } \varphi(x,y) = \tan^{-1}\frac{G_y}{G_x}$$

Al obtener el gradiente de una imagen podemos obtener los bordes de esta con el método binario que hemos explicado antes de la forma:

$$\begin{cases} g(x,y) = 1 \; si \; |G[f(x,y)]| > T \\ g(x,y) = 0 \; si \; |G[f(x,y)]| \leq T \end{cases}$$

Siendo T un valor de umbral no negativo donde solo se considerarán significativos los píxeles que excedan el valor de T [7].

Un ejemplo gráfico de cómo funcionan las derivadas direccionales se ve con la siguiente serie de imágenes:

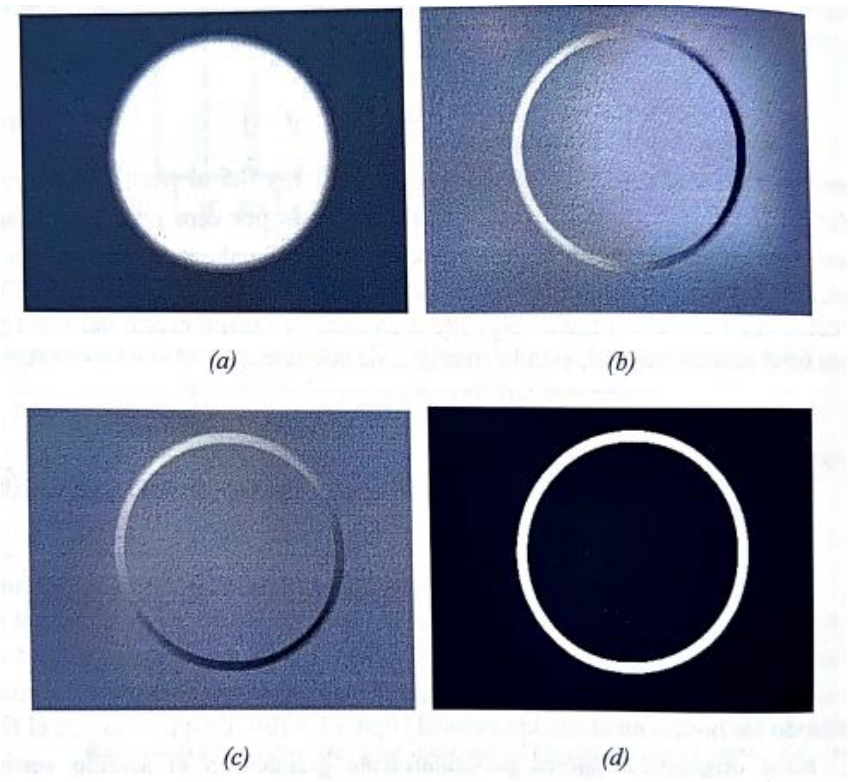


Figura 1.15: Funcionamiento de la primera derivada direccional. a) imagen original, b) primera derivada sobre el sentido horizontal, c) primera derivada sobre el sentido vertical, d) valor del gradiente [6]

Una vez que hemos obtenido el gradiente, utilizamos los operadores de primera derivada para obtener el valor del píxel que se evalúa con una relación de un umbral determinado [7]. Su tarea, al igual que la del gradiente, es eliminar el ruido de la foto y así hacer desaparecer bordes falsos. Algunos de estos operadores son el de Sobel, el de Prewitt y el de Roberts. Comenzaremos estudiando los operadores de Sobel y de Prewitt ya que son bastante similares. Como hemos explicado anteriormente, estos filtros calculan el gradiente de la imagen en cada píxel, lo que permite encontrar el mayor aumento de los píxeles claros a los oscuros y la tasa de cambio. El borde puede definirse como un conjunto de posiciones de píxeles contiguos en los que se produce un cambio repentino de los valores de intensidad en el píxel [11]. En ambos casos el algoritmo de estos operadores consta de dos máscaras 3x3, una para el gradiente horizontal Gx y otra para el gradiente vertical Gy y son las siguientes:

- Operador de Sobel

$$Gx = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

$$Gy = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

El proceso de convolución se realiza para cada píxel de la imagen multiplicando cada uno de los valores de la máscara de Sobel con los valores del píxel correspondiente de la imagen, sumando después los valores y sustituyendo el píxel de origen por el resultado, como podemos ver en la siguiente figura de ejemplo [11].

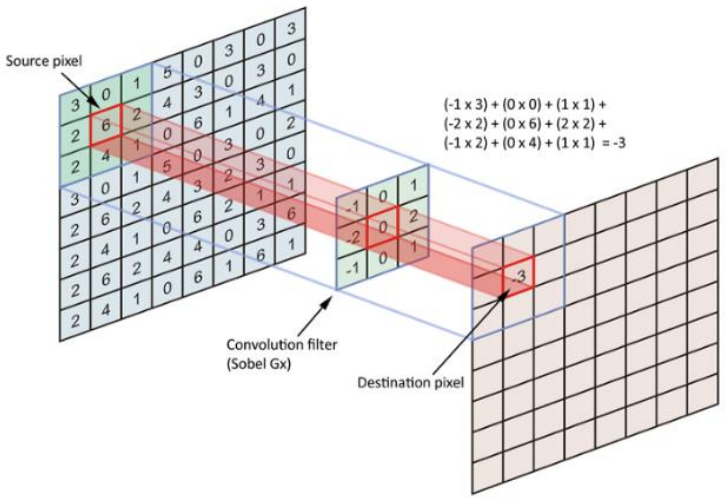


Figura 1.16: Ejemplo de convolución con el filtro de Sobel [11]

- Operador de Prewitt:

$$Gx = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$

$$Gy = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

Su funcionamiento es idéntico al operador de Sobel

Ambos operadores son prácticamente iguales y ofrecen unos resultados muy similares. La única diferencia que podemos encontrar en la práctica es que el operador de Sobel es ligeramente más sensible a los bordes diagonales que el de Prewitt [6].

Buscando un filtro diferente encontramos el operador de Roberts, el funcionamiento es el mismo. Consta de dos máscaras 3x3, una para el gradiente horizontal Gx y otra para el gradiente vertical Gy que tienen la siguiente estructura [12]:

$$Gx = z_6 - z_8$$

$$Gy = -z_1 + z_5$$

Este operador es bueno ante bordes diagonales, además, ofrece buenas prestaciones en cuanto a localización. Sin embargo, su gran inconveniente es que es extremadamente sensible al ruido y, por tanto, en algunos casos, la detección de bordes es muy pobre [12]

Veamos a continuación una tabla resumen con las máscaras de cada uno de los filtros explicados.

Operador	Filtros
Sobel	$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $Gy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Prewitt	$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ $Gy = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Roberts	$Gx = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$ $Gy = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Tabla 1.1: Resumen de los principales filtros de detección de bordes. Fuente: elaboración propia.

Obtenida la magnitud del gradiente podemos decir si un píxel es o no un borde y obtenemos así una imagen binaria de salida.

Los filtros más utilizados en la práctica y los que utilizaremos posteriormente en este trabajo son los explicados anteriormente. Sin embargo, es importante conocer los filtros de segunda derivada. La técnica que utilizan se basa en encontrar los pasos por cero, es decir, la transición de positivo a negativo o viceversa. Esto lo podemos estimar a partir de la segunda derivada [6]. Partimos de la definición de derivada para una función unidimensional:

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

Y a partir de esta ecuación obtenemos la de la segunda derivada:

$$\frac{\partial f^2}{\partial x^2} = f(x+1) - 2f(x) + f(x-1)$$

Como hemos comentado anteriormente, el objetivo es encontrar el paso por cero del valor del gradiente en cualquier dirección. Para ello vamos a utilizar el operador Laplaciano que se define como:

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$

Si consideramos ahora la ecuación de la segunda derivada en el Laplaciano obtenemos:

$$\frac{\partial^2 I(x,y)}{\partial x^2} = I(x+1,y) - 2I(x,y) + I(x-1,y)$$

$$\frac{\partial^2 I(x,y)}{\partial y^2} = I(x,y+1) - 2I(x,y) + I(x,y-1)$$

Por tanto, sustituyendo estas dos ecuaciones en la expresión del Laplaciano quedaría:

$$\nabla^2 I(x,y) = I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y)$$

Expresando esta ecuación en forma de filtro nos queda la siguiente mascara:

0	1	0
1	-4	1
0	1	0

Por último, veamos otro mecanismo que nos puede aportar bastante información acerca de la imagen e incluso de sus bordes u objetos que la forman. Hablamos en este punto de los histogramas de las imágenes. Estos son una representación gráfica de los valores de todos los píxeles de la imagen. En el eje x se comienza con el valor 0 correspondiente al negro y a la derecha se representa el valor máximo correspondiente al blanco. Si la imagen es en color se representan los histogramas de los tres valores de rojo, verde y azul. El histograma nos permite analizar la imagen según la distribución de grises ya que si hay valores muy próximos a 0 (resp. último valor) será una imagen oscura (resp. clara). Además, si el rango de valores del histograma es grande, la imagen tendrá un contraste alto, mientras que si este es pequeño entonces el contraste será bajo. También nos permite diferenciar el número de objetos y el tamaño de estos [7]. Algunos histogramas de ejemplo son:

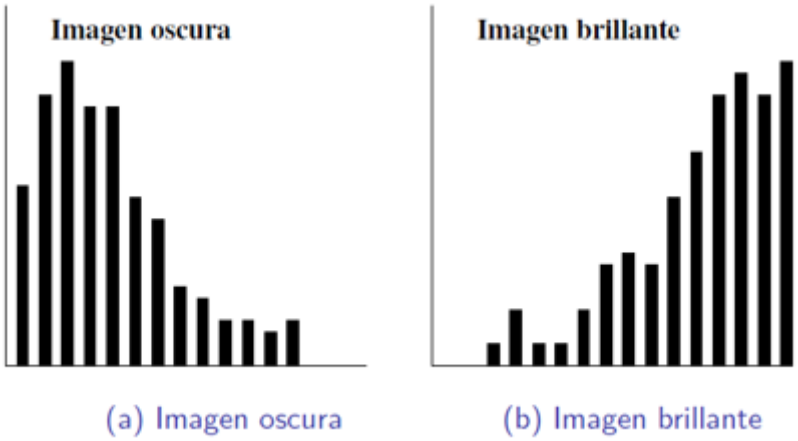


Figura 1.17: Ejemplo de histogramas según la iluminacion [7]

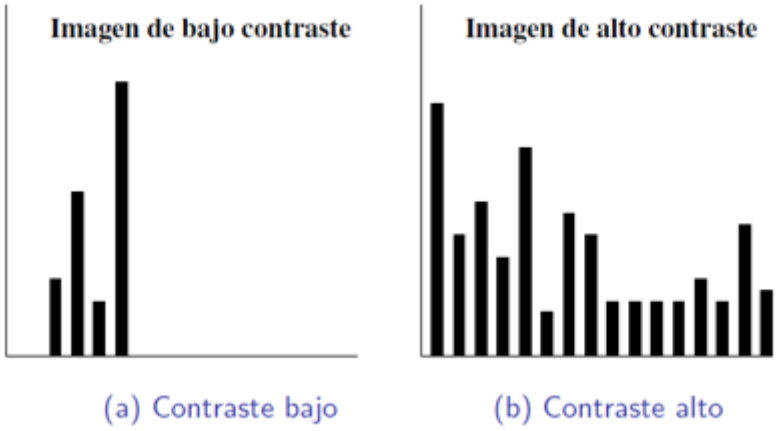


Figura 1.18: Ejemplo de histrogramas según el contraste [7]

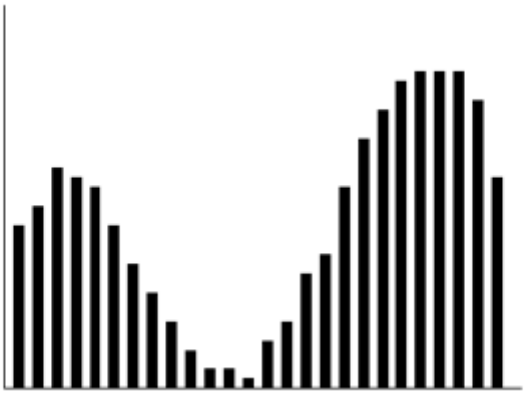


Figura 1.19: Ejemplo de histograma con dos objetos bien diferenciados [7]

1.6 Fase de interpretación del sistema de visión artificial

Una vez que hemos procesado la imagen pasamos al último paso que es el reconocimiento e interpretación de esta. El análisis de imágenes es un proceso que consiste en descubrir, identificar y comprender los patrones relevantes en el rendimiento de un trabajo basado en imágenes. Además, tiene como objetivo dotar a la máquina de una capacidad de aproximación similar a la de los humanos [13].

Dentro de esta fase vamos a identificar 4 cuatro subprocesos:

1. Detección, reconocimiento e identificación: con la detección pretendemos simplemente descubrir que hay algo en la imagen. Posteriormente, con el reconocimiento, el ordenador es capaz de identificar los objetos de la foto como algo familiar en base a su tamaño, forma y aplicaciones. Por último, con la identificación, el ordenador consigue darle un nombre o término al objeto que ha reconocido anteriormente [13].
2. Análisis: este paso consiste en dividir la imagen en unidades dibujando contornos y líneas de acuerdo con unos criterios establecidos de manera que se analice toda la imagen [13].
3. Clasificación: es la etapa en la que comparamos cada una de las unidades obtenidas por el análisis con características definidas anteriormente. Asignaremos un nombre de clase a las unidades diferenciadas que presenten las mismas características [13].
4. Deducción: esta última fase consiste en la combinación de las observaciones realizadas en la imagen con el conocimiento adquirido previamente a través de otras fuentes. Es aquí cuando el ordenador o interprete llega a conclusiones sobre la imagen [13].

2. Casos prácticos

2.1 Procesos de tratamiento de imágenes

Comenzamos ahora a aplicar los procesos de los que hemos hablado con anterioridad a imágenes reales para ver cuál es su verdadero efecto en estas y su funcionamiento.

Este ejercicio lo llevaremos a cabo en MATLAB (MATrix LABoratory**)** porque posee muchas funciones integradas que nos facilitan notablemente el trabajo. Al igual que hemos hecho en el capítulo teórico dividiremos el estudio en pre-procesamiento y segmentación.

Todas las figuras e imágenes que veremos a continuación son de elaboración propia y el código utilizado en los programas está basado en la referencia: [6]

2.1.1 Pre-procesamiento de las imágenes

Lo primero que tenemos que hacer es cargar la imagen en MATLAB, esto se hace con el comando *imread*. El primer paso que vamos a llevar a cabo es convertir la imagen a color original en una imagen escala de grises usando la función *rgb2gray*.

Figura 2.1: Imagen de muestra para la elaboración de este caso práctico. Fuente: obtenido del enlace https://www.agromatica.es/el-cultivo-de-girasoles/

Figura 2.2: Imagen a escala de grises obtenida por el software MATLAB a partir de la figura 2.1.

A partir de la foto en escala de grises comenzamos con el pre-procesamiento de la imagen. Vamos a empezar aplicando algunos filtros de suavizado. El objetivo de estos filtros es eliminar el ruido de la imagen. Aplicamos un filtro estándar que toma la misma proporción de cada píxel vecino, este es el conocido como filtro de la media explicado anteriormente y usaremos la máscara:

$$\frac{1}{9} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Por otro lado, vamos a aplicar el filtro de Gauss con una máscara Gaussiana 3x3 que tiene la estructura

$$\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Ejecutamos los programas con Matlab y obtenemos

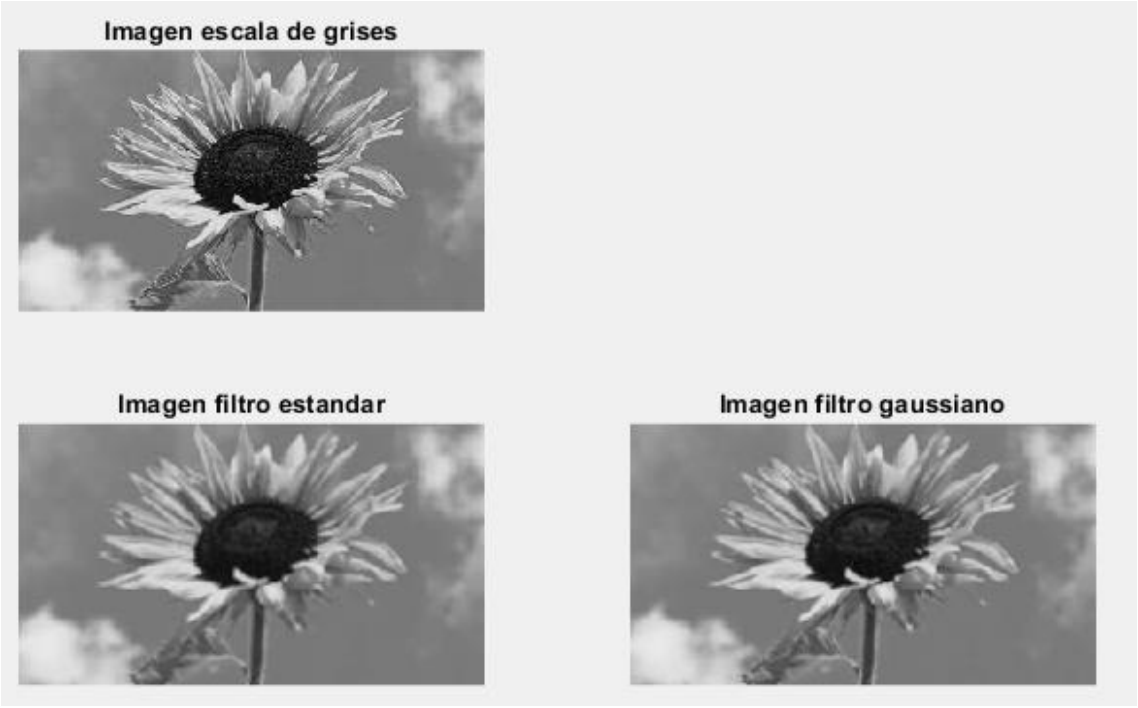


Figura 2.3: Comparativa entre el filtro estándar y el filtro Gaussiano, obtenida por del software MATLAB a partir de la figura 2.2

Vemos que ambos filtros suavizan los bordes de la imagen, siendo más intenso el filtro que toma el promedio de los pixeles externos ya que el filtro gaussiano le da más peso al píxel central y por tanto es más suave.

Sin embargo, la verdadera utilidad de estos filtros aparece cuando tenemos una fotografía con ruido. Vamos a aplicar ruido a la imagen para poder aplicarle después los filtros de suavizado y comprobar que estos de verdad funcionan. El filtro más fácil de aplicar manualmente es el de tipo sal y pimienta. Para ello convertimos aleatoriamente 500 píxeles de la imagen en un píxel de color blanco y otros 500 píxeles en color negro

Una vez tenemos la imagen con ruido le aplicamos los filtros anteriores a esta imagen y obtenemos lo siguiente:

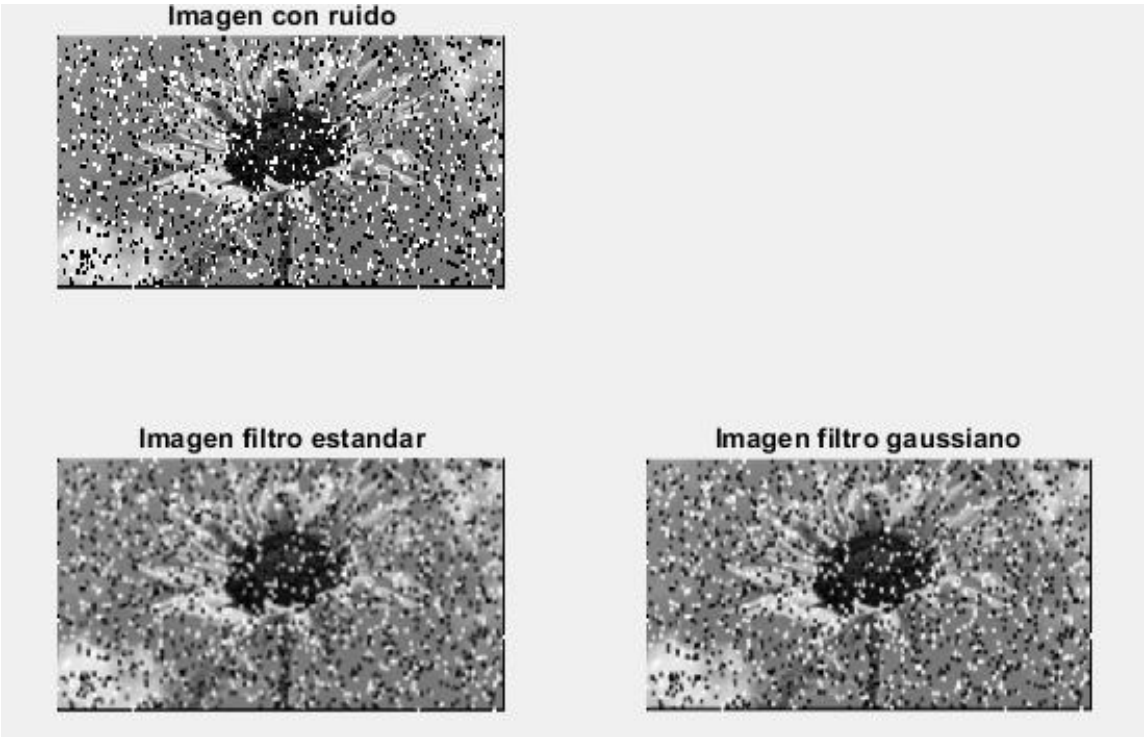


Figura 2.5: Comparativa entre el filtro estándar y el filtro Gaussiano en una imagen con ruido artificial. Obtenida por el software MATLAB a partir de la figura 2.2

Vemos que con estos filtros no podemos mejorar mucho la imagen, el problema es que la cantidad de ruido añadida es excesiva. Probamos entonces con un ruido artificial menos brusco. Además, como se ve que el filtro estándar es mas efectivo vamos a probar con máscaras de distinto tamaño de este tipo de filtro, en primer lugar, utilizamos una máscara 3x3 y en segundo lugar una máscara 9x9, los resultados son los siguientes:

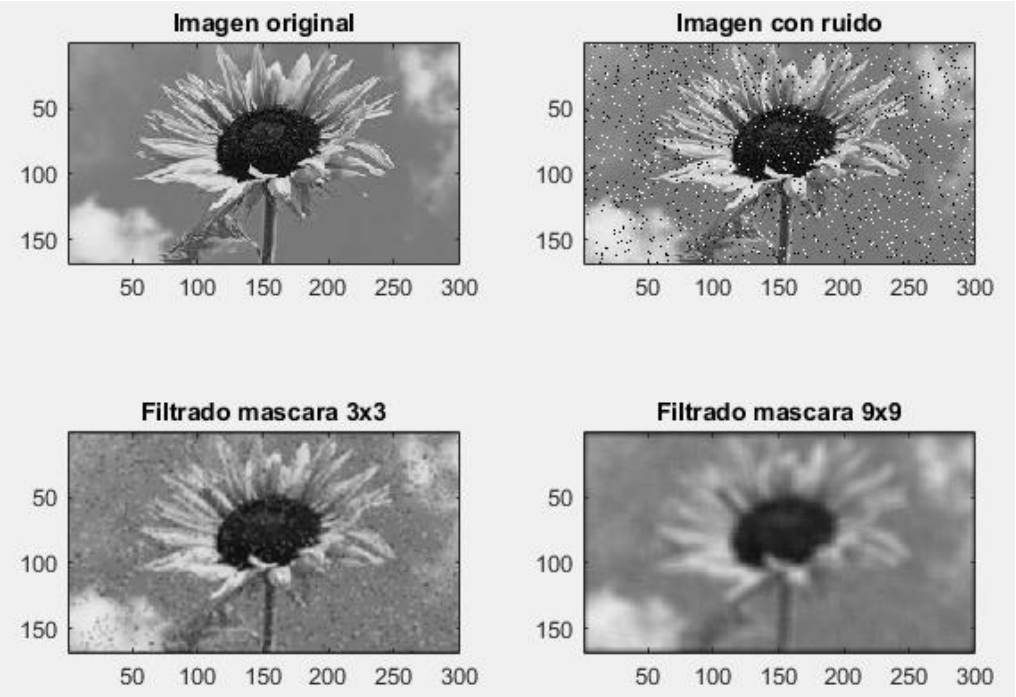


Figura 2.6: Imagen con filtrado artificial menos brusco y el efecto sobre ella de los filtros estándar con distinto tamaño de máscaras. Obtenido por el software MATLAB a partir de la figura 2.1

Vemos que en este caso el filtro estándar con mascara 3x3 es capaz de suavizar el ruido considerablemente bien. Por otro lado, el filtro estándar con mascara 9x9 hace desaparecer por completo el ruido, sin embargo, difumina demasiado los bordes.

Hasta ahora el ruido ha sido introducido por nosotros de manera artificial, sin embargo, ahora vamos a trabajar con una imagen con un ruido más natural.

Figura 2.7: Imagen con ruido. Obtenida del enlace: https://www.dzoom.org.es/wp-content/uploads/2009/11/4416660360_04d75258bb_b-734x371.jpg

Al aplicar los filtros de suavizado obtenemos:

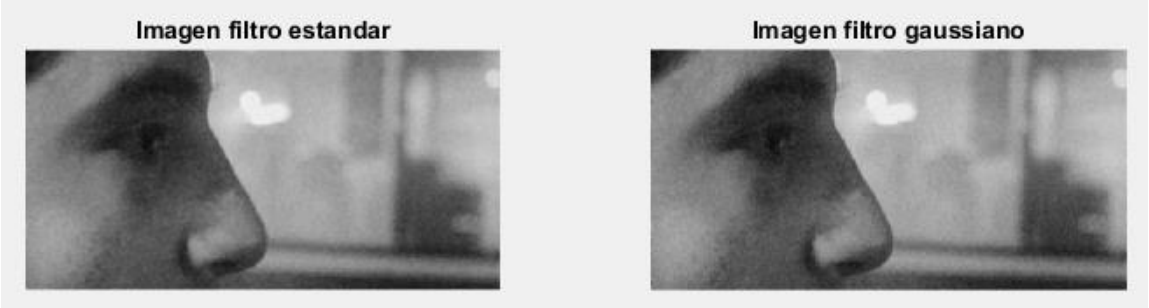


Figura 2.8: Comparativa entre el filtro estándar y el filtro Gaussiano en una imagen con ruido natural. Obtenida por el software MATLAB a partir de la figura 2.7

Aquí ya podemos ver que partimos de una imagen con bastante ruido natural y al aplicar los filtros es muy notable el suavizado que se produce en la imagen, sobre todo con el filtro estándar.

En este caso vamos a probar también con algún filtro de tipo no lineal para ver como funcionan. Los filtros no lineales que vamos a utilizar son el filtro de máximo y mínimo. El resultado es el siguiente:

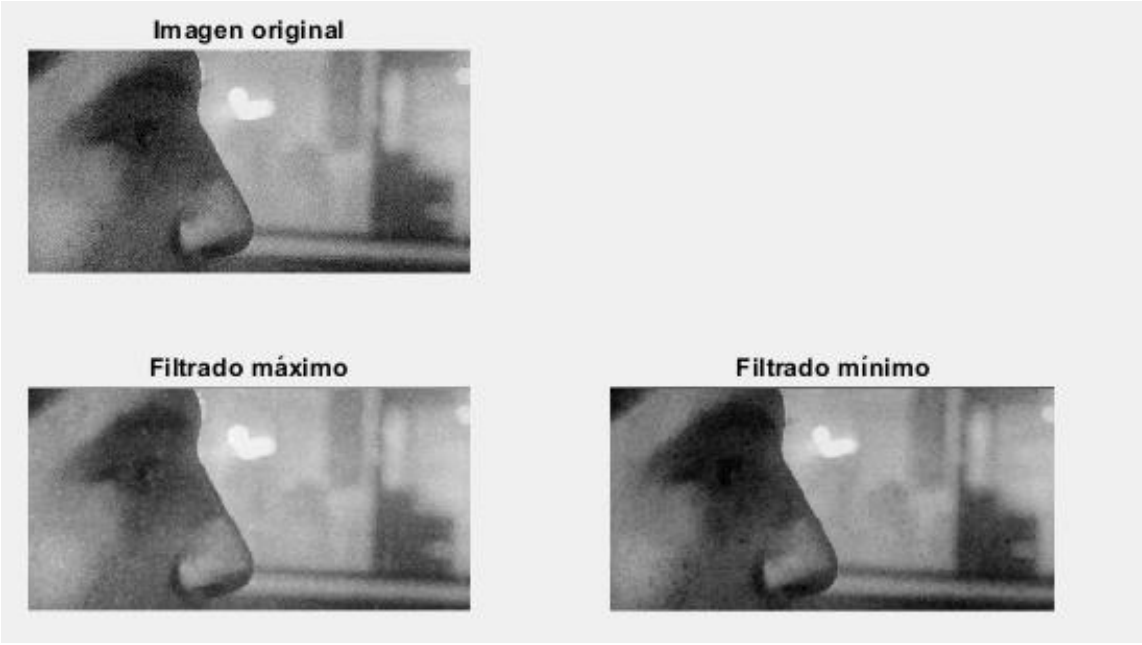


Figura 2.9: Comparativa entre en filtro máximo y mínimo en una imagen con ruido natural. Obtenida por el software MATLAB a partir de la figura 2.7

Podemos observar como se aclara y se oscurece la imagen dependiendo del tipo de filtro utilizado.

2.1.2 Segmentación

Una vez que ya hemos filtrado las imágenes podemos pasar a la segunda parte del proceso que es la que conocemos como segmentación. Aquí centraremos todos nuestros esfuerzos en la detección de bordes y la identificación de distintos objetos. Para identificar objetos con distinta intensidad de color el método más simple y en ocasiones, el más efectivo es la detección de bordes binaria. Para ello vamos a definir un nivel de umbral y le damos el color blanco a los píxeles que sobrepasen ese umbral y el color negro a los que no lo alcancen. Para la imagen de los girasoles tomamos un umbral de 140.

Figura 2.10: Ejemplo de binarización de una imagen. Obtenida por el software MATLAB a partir de la figura 2.1

Al ser una imagen con muchas tonalidades de gris es complicado conseguir detectar todos los pétalos de la flor. Sin embargo, en el momento en el que tomamos una imagen de ciertos objetos apoyados en una mesa, vemos que este filtro es muy práctico para poder diferenciarlos, un ejemplo sería el siguiente:

Figura 2.11: Ejemplo de binarización de una imagen. Obtenida por el software MATLAB. Fuente: elaboración propia.

Debemos tener cuidado con la sombra que nos ha quedado en la esquina inferior izquierda para no confundirla con otro objeto, pero se pueden ver claramente los cuatro objetos distintos que hay en la imagen.

Ahora pasaremos a aplicar los filtros de detección de bordes comenzando con el filtro de Sobel. El procedimiento que vamos a seguir es aplicar las dos mascaras correspondientes al filtro de Sobel, Gx en horizontal y Gy en vertical. Posteriormente calculamos el vector gradiente y normalizamos este valor. Con eso ya obtendremos la imagen final con los bordes detectados. Veamos imágenes del proceso para la foto del girasol.

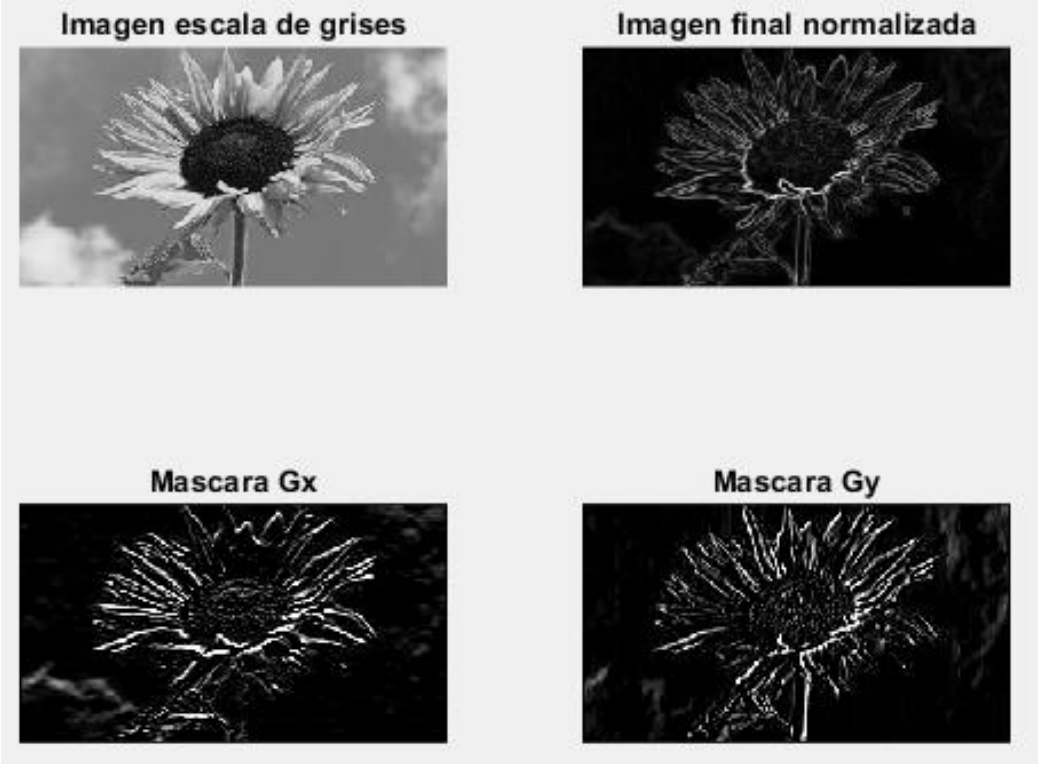


Figura 2.12: Ejemplo de proceso de aplicación del filtro de Sobel. Obtenida por el software MATLAB a partir de la figura 2.2

Ahora vamos a aplicar el método para una imagen con muchos bordes y vemos que funciona igual de bien.

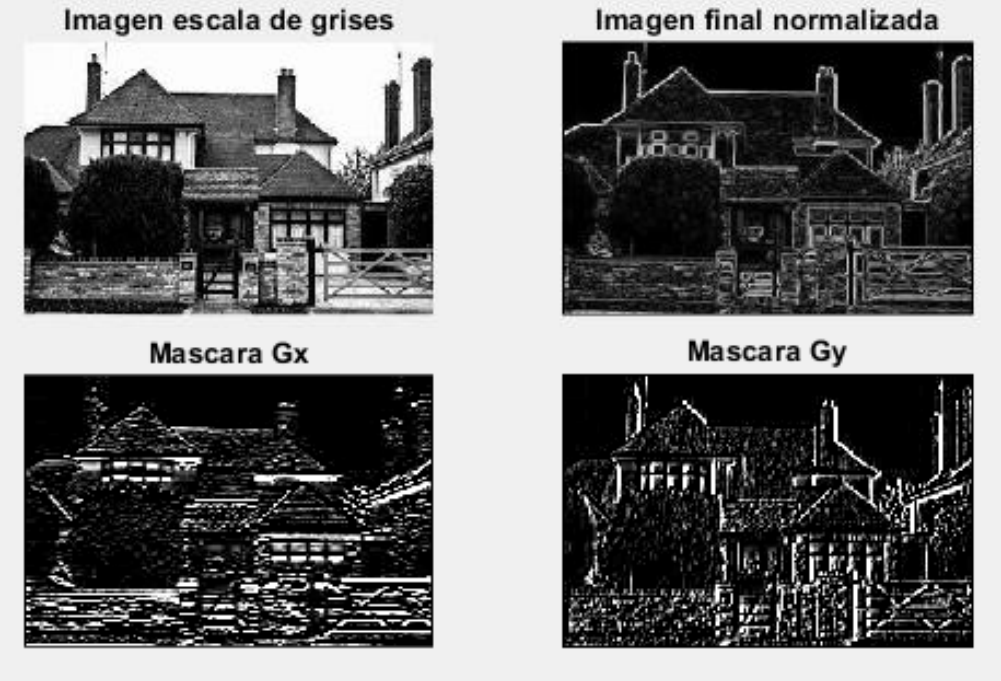


Figura 2.13: Ejemplo de aplicación del filtro de Sobel. Obtenida con MATLAB a partir de la imagen disponible en el enlace https://www.sandradavidson.com/wp-content/uploads/2021/03/selling-a-house-after-probate.jpg

En este caso hemos aplicado el filtro de Sobel paso a paso. Hay una función en MATLAB que al proporcionarle una imagen aplica el filtro de Sobel directamente, u otros como el filtro de Prewitt. Vamos a utilizar esta función de MATLAB para hacer una comparación entre estos dos filtros de detección de bordes.

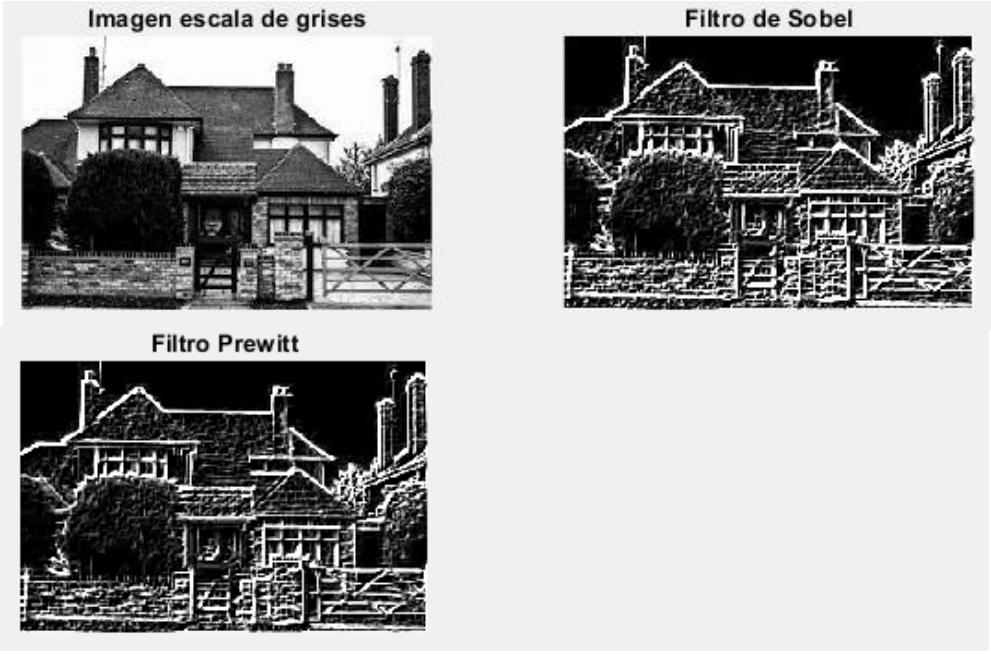


Figura 2.14: Comparativa entre el filtro de Sobel y el filtro de Prewitt. Obtenida por el software MATLAB a partir de la figura 2.13

Podemos ver que los bordes que nos devuelve esta función son más agresivos que los que hemos obtenido al principio. Esto se debe a que en el primer caso hemos normalizado la imagen final y, sin embargo, en el segundo caso no lo estamos haciendo. También vemos que la diferencia entre ambos filtros es prácticamente insignificante ya que los dos cumplen bien con su objetivo.

Por último, vamos a estudiar los histogramas de las imágenes. En muchos casos los histogramas nos permiten diferenciar el número de objetos distintos que hay. Veamos algún ejemplo con una imagen que tiene unos colores bien diferenciados

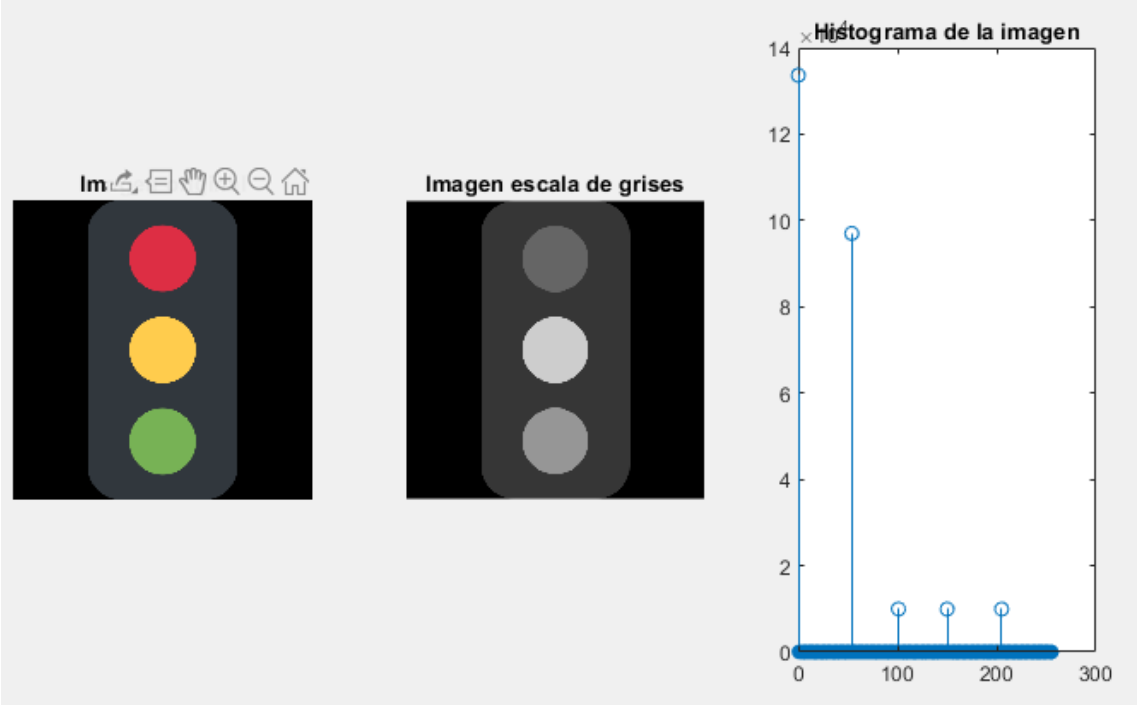


Figura 2.15: Ejemplo de histograma de una imagen. Obtenido con el software MATLAB a partir del enlace https://media.istockphoto.com/id/1186867446/es/vector/sem%C3%A1foro-colores-icono-vectorial-sobre-fondo-blanco.webp?s=612x612&w=is&k=20&c=e0BoTwKHfLzYh6-ZHtL9oIMd-hfiFoypdW3d0lzF6-A=

Como podemos ver la información que nos da el histograma es clara, aparecen cinco colores distintos, y nos informa también de la cantidad en la que estos aparecen. Es fácil identificar que la primera barra corresponde al color negro, el fondo de la imagen. La segunda barra corresponde a la estructura del semáforo y las otras barras restantes corresponden a los tres colores distintos del semáforo ya que vienen en la misma proporción.

Lo vemos también en una foto menos clara y obtenemos:

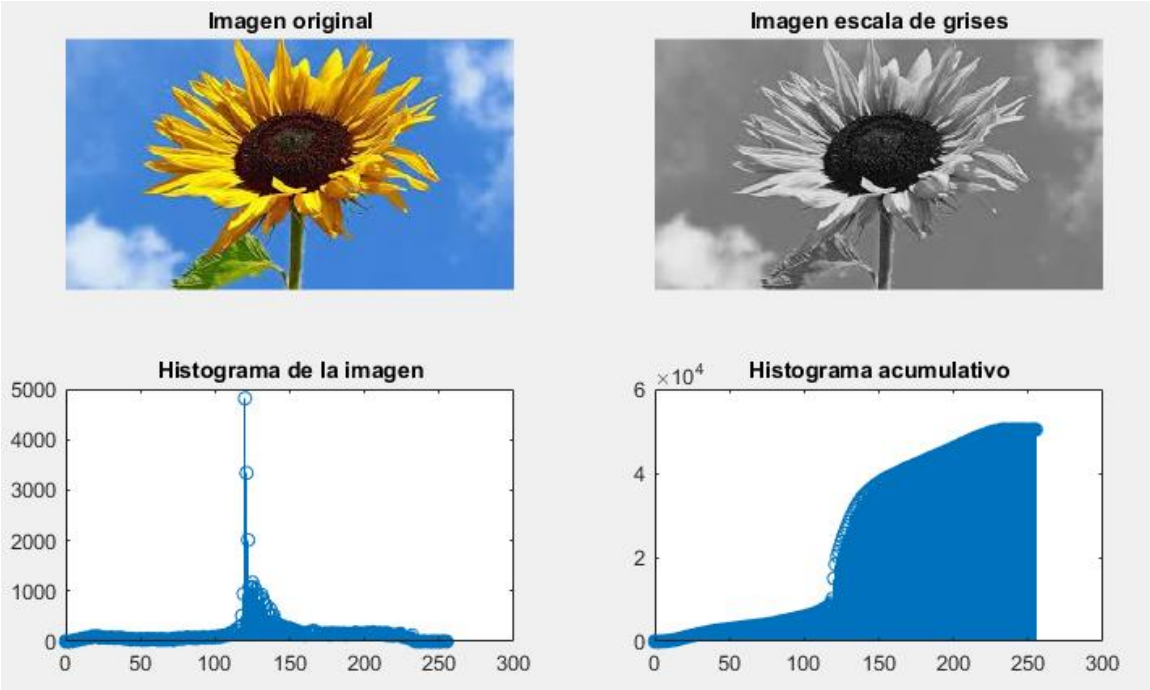


Figura 2.16: Ejemplo de histograma de una imagen. Obtenido por el software MATLAB a partir de la figura 2.2

En el histograma de la imagen podemos ver un color que predomina sobre el resto y que suponemos que será el tono de gris correspondiente al azul del fondo. Además, en esta imagen hemos añadido el histograma acumulativo que parece indicar que la imagen tiene dos claras tonalidades ya que da un salto muy pronunciado alrededor del valor 120.

Otras operaciones que podemos realizar sobre la imagen y que se van a ver reflejadas en los histogramas son un aumento de contraste o de iluminación. Comenzamos aplicando un aumento de contraste:

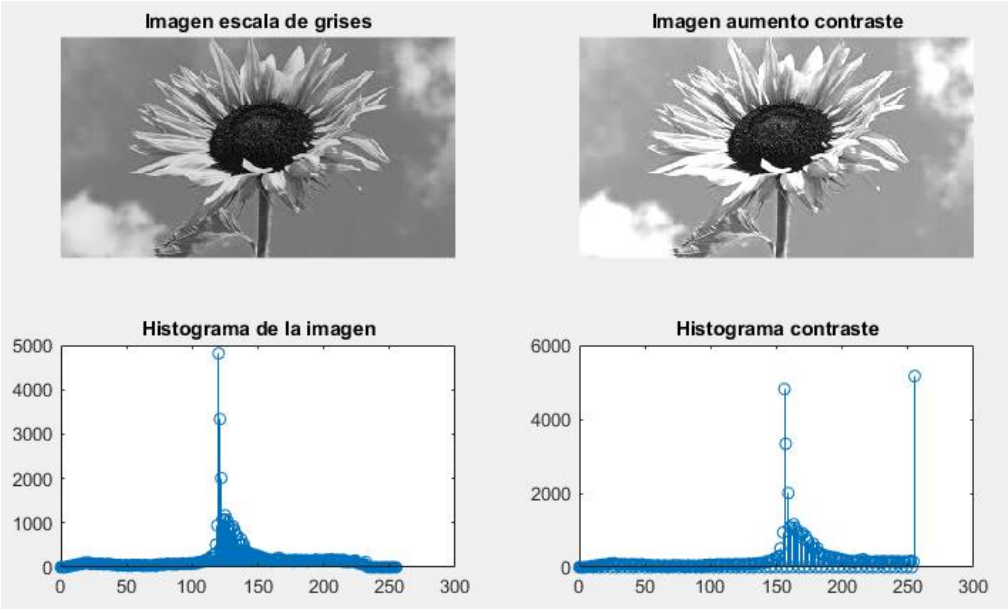


Figura 2.17: Ejemplo del efecto en el histograma de una imagen al aplicar un aumento de contraste. Obtenida por el software MATLAB a partir de la figura 2.2

Apreciamos que los tonos claros se hacen más claros y aparecen tonos blancos, es decir, el histograma queda desplazado a la derecha.

Si aplicamos un aumento de iluminación sumamos una cantidad x a cada píxel y lo que va a provocar este cambio es que el histograma se desplace todo x unidades a la derecha. Tomando x = 50, obtenemos:

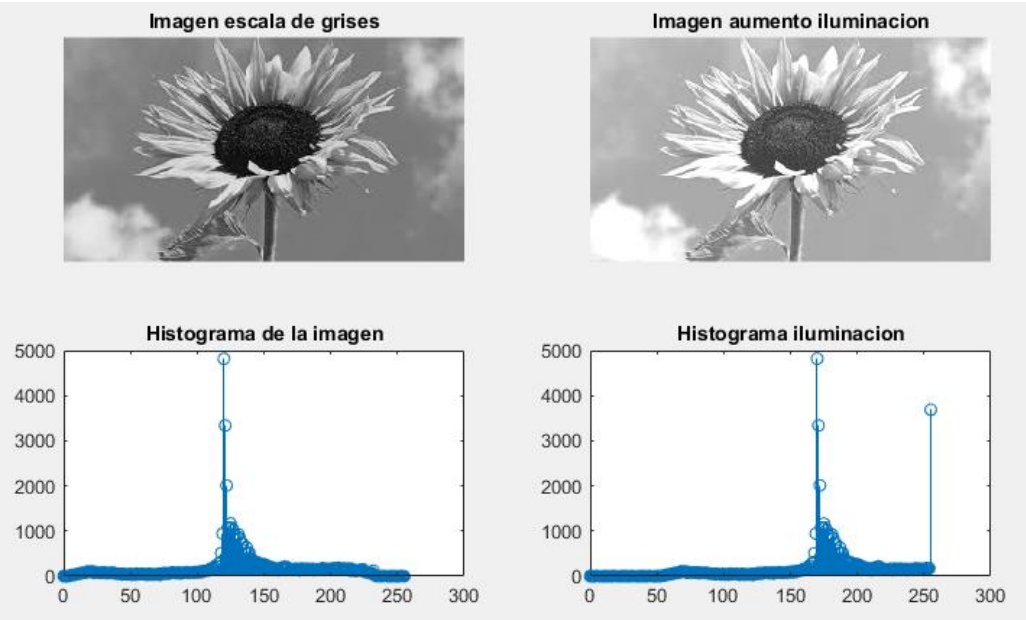


Figura 2.18: Ejemplo del efecto en el histograma de una imagen al aplicar un aumento de iluminación. Obtenida por el software MATLAB a partir de la figura 2.2

2.2 Detección de círculos en una imagen

2.2.1 Programa básico

En este capítulo vamos a construir un programa en MATLAB [14] que nos permita detectar los círculos que hay en una imagen, sin importar el color de estos.

Figura 2.19: Disposición y colores de los círculos a detectar [14]

El objetivo es ser capaces de detectar todos los círculos, es decir, conocer la posición en la que se encuentra cada uno de ellos y el número total sin importar que unos estén encima de otros o incluso estén cortados por los bordes de la imagen.

La clave de este programa se basa en la función nativa de MATLAB *imfindcircles* que es capaz de encontrar los círculos de una imagen en función de ciertos parámetros de entrada. Los parámetros de entrada son los siguientes [15]:

- Imagen de entrada: es la imagen sobre la que queremos encontrar los círculos.
- Intervalo de radio: indica los valores entre los que se encuentra el radio de los círculos de la imagen.
- *'ObjectPolarity'*: indica la polaridad del objeto y tiene dos valores: *'bright'* que es el predeterminado y significa que los círculos son más claros que el color de fondo, o *'dark'*, que por el contrario indica que los círculos son más oscuros que el fondo.
- Sensitivity: es el factor de sensibilidad que se encuentra en el intervalo [0,1], a medida que aumenta también lo hace la capacidad detección de círculos de la función.

Por otro lado, los parámetros de salida que devuelve son:

- Centro: da la coordenada del centro de cada círculo detectado en forma (x, y).
- Radio: estimación del radio de cada círculo detectado.

Antes de aplicar esta función tenemos que saber si el parámetro *'ObjectPolarity'* es *dark* o *bright*, para ello pasamos la imagen a escala de grises obteniendo:

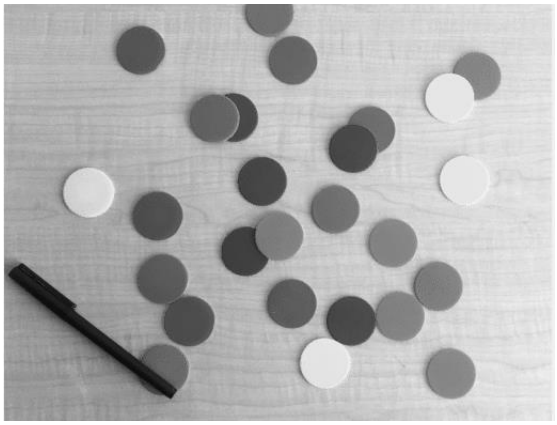


Figura 2.20: Imagen en escala de grises. Obtenida por el software MATLAB a partir de la figura 2.19

Vemos que por lo general los círculos son más oscuros, entonces aplicamos la función con el parámetro *'ObjectPolarity'* como *dark* y obtenemos la siguiente imagen:

Figura 2.21: Imagen resultante al aplicar función imfincircles con el *parámetro 'ObjectPolarity' como dark. Obtenida por el software MATLAB a partir de la figura 2.19*

Como vemos en esta imagen hemos conseguido detectar todos los círculos excepto los amarillos. Los parámetros de rango de radio y precisión los hemos ido variando hasta obtener la imagen anterior.

Sin embargo, todavía nos falta detectar los círculos amarillos, para ello cambiamos el parámetro *'ObjectPolarity'* a *bright*, de esta forma detectamos únicamente los círculos amarillos.

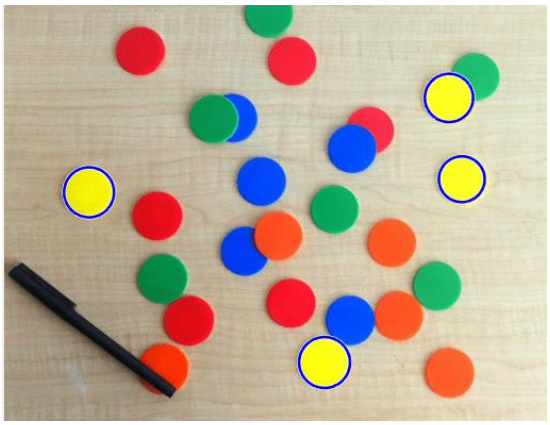
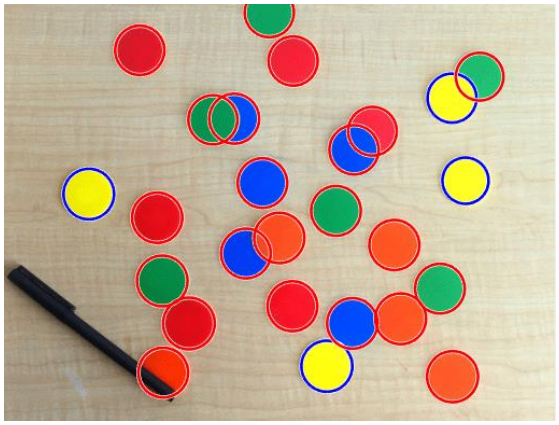


Figura 2.21: Imagen resultante al aplicar función imfincircles con el *parámetro 'ObjectPolarity' como bright. Obtenida por el software MATLAB a partir de la figura 2.19*

Por último, necesitamos combinar ambas detecciones para tener todos los círculos. Finalmente obtendríamos la imagen siguiente, el recuento total de círculos, los centros de cada uno de los círculos detectados y el radio de estos.



```
Número de círculos totales:
    26
```

```
centers =                radii =

  269.5139  280.9540     23.1311
  270.2059   53.4154     22.8021
  241.6942  166.6623     22.9987
  127.6821   43.5068     22.8620
  325.2254  296.9161     23.4278
  418.3759  346.3603     23.6626
  406.6388  264.6628     23.0307
  143.8577  198.8586     23.3131
  326.8204  135.8333     22.8387
  310.0395  192.4702     23.0760
  364.2372  224.2076     23.2211
  173.3600  297.0064     23.9707
  343.9257  118.5755     22.5573
  226.3205  232.0504     23.0467
  215.4539  106.9479     23.1273
  443.5499   67.9399     22.1388
  370.1770  292.6391     23.5315
  148.5849  257.5070     23.4646
  257.3500  217.6176     23.3227
  248.0655    8.4524     22.8361
  149.5031  342.5729     23.0755
  195.6333  107.9883     22.7678
```

Figura 2.22. Imagen que consigue detectar todos los círculos de la imagen. Numero de círculos totales y coordenadas del centro y radio de cada uno de ellos. Obtenida por el software MATLAB a partir de la figura 2.19

2.2.2 Aplicación al juego del tres en raya

Una de las aplicaciones de este tipo de detección de círculos aparece cuando queremos enseñar a una inteligencia artificial como jugar al juego del tres en raya. Para que esta pueda realizar los movimientos primero tiene que entender el tablero del que generalmente obtendrá una imagen para procesarla. Es en este punto donde entra esta aplicación de la visión artificial. Lo que queremos conseguir en este capítulo es que dada una imagen, podamos extraer la información que hay en ella de forma que sea comprensible por un ordenador. Vamos a partir de una imagen cualquiera en la que tenemos un tablero de nueve casillas con fichas en forma circular de dos colores diferentes, correspondiendo cada color a cada uno de los dos jugadores enfrentados.

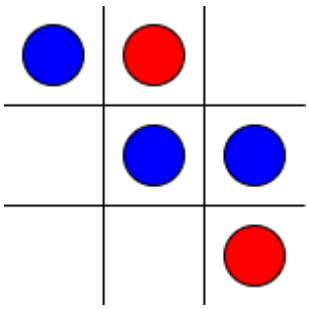


Figura 2.23: Disposición del tablero de tres en raya. Imagen obtenida a partir del software Draw.io

En este caso es bastante obvio que los círculos a detectar son más oscuros que el color de fondo, en cualquier caso, transformamos la imagen a escala de grises para generalizar el proceso por si acaso en algún momento cambia el color de fondo de la imagen. Obtenemos la siguiente imagen:

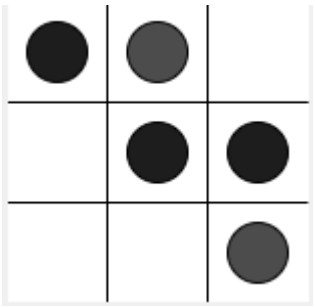


Figura 2.24: Imagen del tablero de tres en raya en escala de grises. Obtenida por el software MATLAB a partir de la figura 2.23

Vemos claramente que los colores de los círculos son más oscuros que el fondo de la imagen, por tanto, usamos la función *imfindcircles* con 'ObjectPolarity' como 'dark. De esta manera la función encuentra todos los círculos de la siguiente forma:

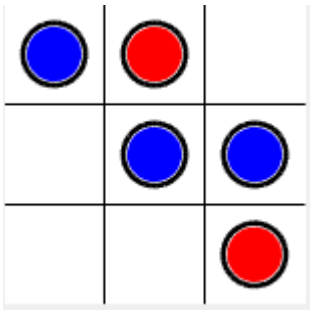


Figura 2.25: Detección de los círculos que forman el tablero del tres en raya. Obtenida por el software MATLAB a partir de la figura 2.23

A partir de esta imagen podemos ver que el programa es capaz de detectar todos los círculos, además nos devuelve el centro de cada uno de los círculos que ha detectado:

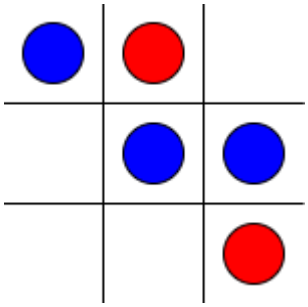
```
centers =

   25.0009   25.0009
  124.8543  124.8543
  125.0000   74.9999
   75.0000   25.0014
   74.9999   74.9999
```

Figura 2.26: Coordenadas del centro de cada uno de los círculos detectados en el tablero. Obtenida por el software MATLAB a partir de la figura 2.23

El siguiente paso es conseguir diferenciar en que casillas se encuentran las fichas y de qué color es cada una. Para ello vamos a dividir la imagen en nueve regiones iguales, es decir, calculamos las dimensiones de la imagen y lo dividimos entre 3. Posteriormente calculamos a qué región pertenece cada uno de los centros de las circunferencias detectadas. Para conocer el color de cada ficha accedemos al valor del píxel central, este paso lo podemos hacer tanto en la imagen en color o en la imagen en escala de grises, en ambos casos tenemos que utilizar la función *impixel* que nos devolverá tres valores. En la foto en formato *RGB* cada valor corresponderá al valor de cada color (Red, Green y Blue) y en la imagen en grises se repetirá el mismo número las tres veces. En este caso hemos decidido hacerlo con la imagen en color por lo que si el color del círculo es rojo solo obtendremos un valor en la tercera componente, si por el contrario es de color azul el valor lo obtendremos en la primera coordenada.

Por último, para obtener la información de manera que el ordenador pueda procesarla vamos a dar la salida en forma de una matriz 3x3. Cada entrada de la matriz corresponde con cada una de las casillas del tablero, además en los coeficientes de la matriz

encontramos un 1 si la ficha que hay es de color azul, si hay un 2 la ficha es de color rojo y si hay un 0 todavía no hay ninguna ficha en esa casilla. Para el tablero inicial con el que hemos empezado el ejemplo, la salida del programa sería la siguiente:

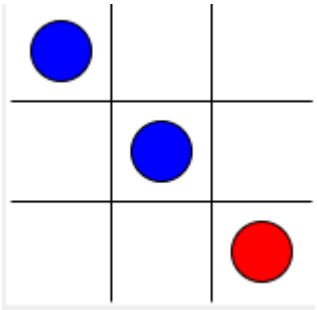


```
numero_fichas =

     5


tablero =

     1     2     0
     0     1     1
     0     0     2
```

Figura 2.27: Salida del programa que detecta todos los círculos del tablero. Obtenida por el software MATLAB a partir de la figura 2.23

A continuación, veremos otras configuraciones distintas del tablero para comprobar que el programa funciona para cualquier número de fichas y distribuciones de estas.

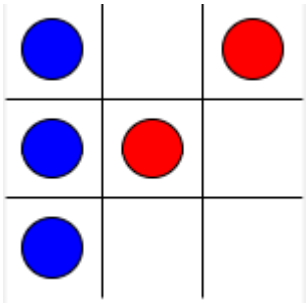
- Ejemplo tablero 1



```
numero_fichas =

     3


tablero =

     1     0     0
     0     1     0
     0     0     2
```

Figura 2.28: Ejemplo de disposición de tablero y salida del programa que detecta los círculos. Obtenido por el software MATLAB a partir de la imagen creada por el software Draw.io

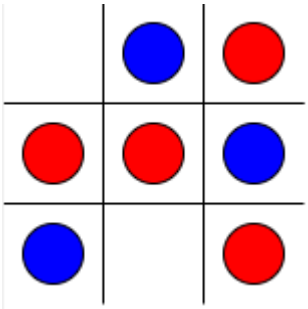
- Ejemplo tablero 2



```
numero_fichas =

     5


tablero =

     1     0     2
     1     2     0
     1     0     0
```

Figura 2.29: Ejemplo de disposición de tablero y salida del programa que detecta los círculos. Obtenido por el software MATLAB a partir de la imagen creada por el software Draw.io

- Ejemplo tablero 3



```
numero_fichas =

     7


tablero =

     0     1     2
     2     2     1
     1     0     2
```

Figura 2.30: Ejemplo de disposición de tablero y salida del programa que detecta los círculos. Obtenido por el software MATLAB a partir de la imagen creada por el software Draw.io

2.3 Detección de objetos en una imagen

2.3.1 Objetivo y funcionamiento

Con este programa pretendemos unir todos los conceptos aprendidos a lo largo del trabajo y ponerlos en práctica para realizar un verdadero proceso de visión artificial. El objetivo es que dada una imagen con diferentes objetos podamos ser capaces de procesarla para que un ordenador pueda entender los tipos de objetos que hay en ella. Para ello partiremos de una imagen con varios objetos, le aplicaremos algunos filtros de suavizado para mejorar sus características y posteriormente utilizaremos técnicas de detección de objetos para asignarles un valor [16]. Este valor lo podemos asociar a cada tipo de objeto para que el ordenador sepa diferenciarlo.

2.3.2 Procedimiento

Vamos a partir de una foto tomada por mí misma con la cámara del móvil, en esta foto vemos diferentes objetos sobre un fondo oscuro. Lo hemos elegido así para que hubiera mejor contraste con el color de los objetos, pero podría tratarse de un fondo blanco o de cualquier otro color. La imagen con la que vamos a trabajar es la siguiente:

Figura 2.31: Disposición de los distintos objetos. Fuente: elaboración propia.

En ella podemos ver cinco objetos distintos, empezando por la parte superior y de izquierda a derecha tenemos un ratón, una llave, una calculadora, una goma de borrar y, por último, un bolígrafo.

Una vez elegida la imagen comienza el pre-procesamiento de esta. Básicamente lo que tenemos que hacer es convertirla a escala de grises y posteriormente eliminar el ruido ya que cualquier ruido en el fondo podría confundirse después con otro objeto. Para lo primero utilizamos la función *rgb2gray*, y para lo segundo utilizamos un nuevo filtro conocido como filtro de Wiener.

El filtro de Wiener entra dentro del grupo de los filtros lineales óptimos y de forma general, consiste en una señal de entrada $x(n)$, una respuesta deseada $d(n)$, y un filtro lineal de respuesta impulsional $h(n)$. Este filtro es alimentado por $x(n)$ y produce una salida $y(n)$. La diferencia entre la señal de salida del filtro $y(n)$ y la señal deseada $d(n)$ es el error de estimación $e(n)$. El objetivo de este filtro es conseguir que el error $e(n)$ sea, en un sentido estadístico, lo más pequeño posible. Para ello se utiliza la minimización del valor cuadrático medio del error [17]:

$$\xi = \{|e(n)|^2\}$$

Las imágenes que obtenemos tras el pre-procesamiento son las siguientes:

Figura 2.32: Pre-procesamiento de la imagen. Obtenida por el software MATLAB a partir de la figura 2.31

Siendo la primera la imagen original en escala de grises y la segunda la imagen obtenida tras aplicar el filtro de Wiener donde podemos apreciar un fondo suave sin ruido.

El siguiente paso es binarizar la imagen, tomando un umbral, en este caso 0.45, le damos el color blanco a los pixeles con un valor inferior al umbral seleccionado y el color negro a los pixeles con un valor superior. Esto lo realizamos con la función *im2bw* y la imagen que obtenemos es la siguiente:

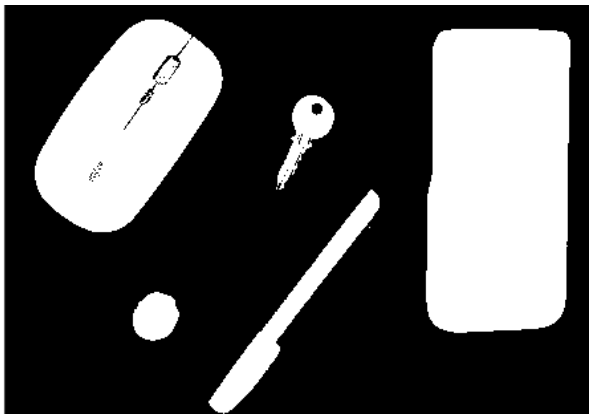


Figura 2.33: Binarizacion de la imagen. Obtenida por el programa MATLAB a partir de la figura 2.32

Sin embargo, con esta imagen todavía no podemos detectar los diferentes objetos ya que tenemos que seguir eliminando ruido. El proceso que queremos llevar a cabo es homogeneizar las regiones en las que haya un objeto, contabilizarlas y calcular algunas de sus medidas, pero si hacemos zoom sobre alguno de los objetos, como por ejemplo, la

llave, vemos que hay regiones más pequeñas que no pertenecen a la región principal de la propia llave.

Figura 2.34: Regiones a eliminar o unificar. Obtenida a partir de MATLAB a partir de la figura 2.33

Estas regiones nos van a dar problemas, por lo que tenemos que eliminarlas o unificarlas a la imagen central. Vamos a hacer ambas cosas. En primer lugar, eliminamos las regiones de píxeles que tengan menos de 130 píxeles con la función *bwareaopen* y posteriormente rellenamos los espacios en negro que hay entre las regiones principales de los objetos y las regiones pequeñas que lo rodean con la función *imclose*. Obtenemos el siguiente resultado:

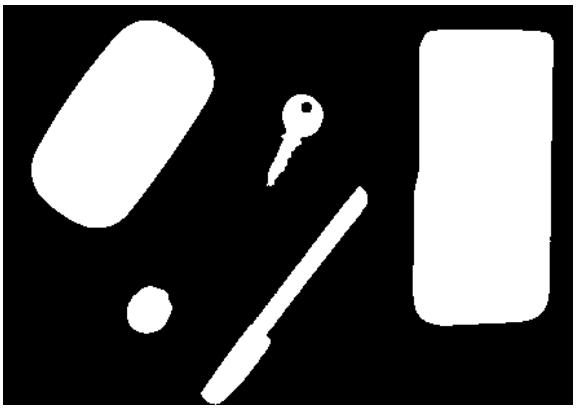


Figura 2.35: Eliminacion de las regiones que generan problemas. Obtenida por el software MATLAB a partir de la figura 2.33

Aquí ya no tenemos regiones perdidas, por tanto, podemos identificar las que pertenecen a objetos distintos. Para ello lo que hacemos es detectar los bordes de las regiones en blanco usando la función *bwboundaries* y le asignamos un color diferente a cada una de las distintas regiones encontradas. Obtenemos la imagen:

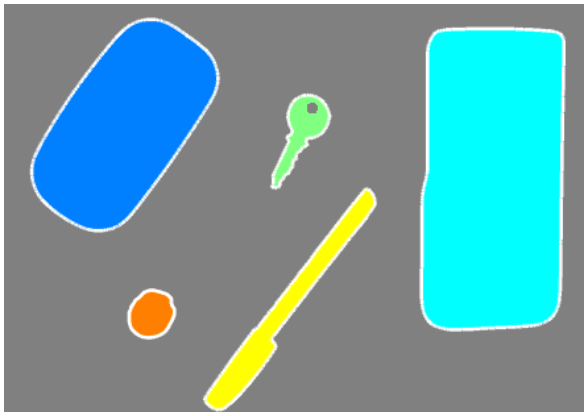


Figura 2.36: Identificación de las regiones de los diferentes objetos. Obtenida por el software MATLAB a partir de la figura 2.35

El último paso es calcular los parámetros que nos van a permitir identificar qué objetos son. Para ello utilizamos la función *regionprops* que calcula el área y el centro de cada región identificada. Por otro lado, gracias a la función *boundary* que calculaba los bordes podemos obtener el perímetro. Con todas estas medidas vamos a calcular una métrica dada por la fórmula:

$$metrica = \frac{4 * \pi * area}{perimetro^2}$$

Finalmente, añadimos este valor a la imagen cerca de la región que está midiendo y obtenemos:

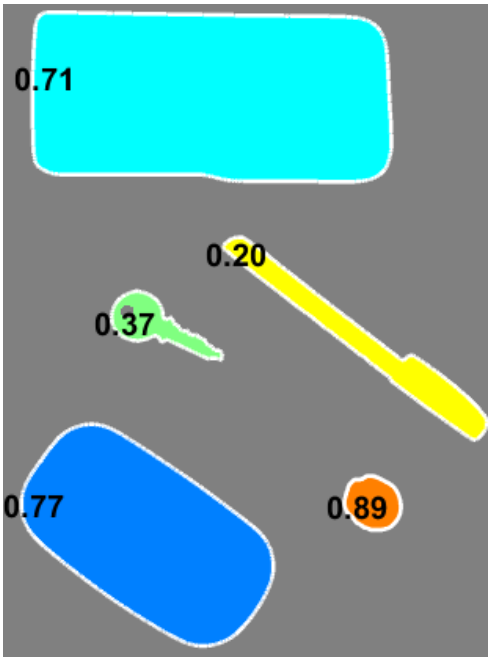


Figura 2.37: Métricas asociadas a las regiones de cada objeto. Obtenida por el software MATLAB a partir de la figura 2.36

Estos valores también los podemos obtener en un vector que será el que le pasaremos al ordenador. Como anteriormente le habremos asignado a cada tipo de objeto su métrica correspondiente, el ordenador ya será capaz de identificar el número y el tipo de objetos de la imagen.

3. Conclusiones: debilidades y fortalezas de la visión artificial

La primera comparativa que deberíamos tener en cuenta una vez estudiada la visión artificial, es la diferencia entre lo que nos proporciona la visión humana y la visión artificial. Mientras que la primera es la mejor para la interpretación de las escenas complejas no estructuradas, la segunda destaca por la medición cuantitativa en escenas estructuradas, siendo más rápida y precisa que la visión humana. Esta ventaja tiene numerosas aplicaciones que comentaremos a continuación.

Tras una breve introducción a la visión artificial, el concepto mejor interiorizado es que la visión artificial pretende que, dada una imagen, un ordenador sea capaz de procesarla y entender los elementos que hay en ella. Uno de los campos en los que los sistemas de visión artificial han supuesto grandes mejoras es la automatización de la industria, en concreto, es claro el ejemplo de las cadenas de producción. Un sistema de visión artificial es capaz de inspeccionar miles de piezas por minuto, además, si este sistema se ha construido con una resolución de cámara óptima, puede inspeccionar detalles de objetos que son demasiado pequeños para la vista humana. Otra ventaja es que permite eliminar el contacto físico entre el sistema de prueba y la pieza a evaluar, lo que evita posibles daños y costes de mantenimiento de los componentes mecánicos [18].

Por otro lado, una aplicación que se utiliza en comercios, aeropuertos e incluso, nuestras casas, son los sistemas de videovigilancia. La visión artificial, gracias al reconocimiento facial, permite detectar personas no reconocidas, sospechosas e incluso delincuentes en busca y captura. Pero los sistemas de vigilancia no solo se limitan al reconocimiento de personas. Esta vigilancia puede estar orientada en sistemas de asistencia en quirófanos y hospitales. En estos casos vigila los procedimientos y puede alertar si hay pacientes en riesgo o alguna desviación en el procedimiento quirúrgico [19].

Finalmente, dentro de las aplicaciones que he encontrado más llamativas se encuentra la capacidad de comprensión que tienen los sistemas de visión artificial. Las máquinas tienen una mayor capacidad para gestionar y comprender conceptos complejos. Algunos ejemplos concretos son el proyecto "InnerEye" de *Microsoft Research Cambridg*e que busca facilitar la comprensión de la forma que tiene un tumor cerebral, o el proyecto "Triton", un software de *Gauss Surgical*, con el que se trata de estimar la pérdida de sangre durante una cesárea [19].

Son claras las numerosas ventajas que nos proporciona la visión artificial. Sin embargo, debemos tener en cuenta algunas de sus limitaciones y debilidades. Para empezar, como hemos comentado al principio, la visión artificial encuentra serios problemas a la hora de identificar conceptos complejos no estructurados. A lo largo de los últimos años esta tecnología ha levantado muchas polémicas debido a reconocimientos erróneos. Algunos ejemplos son etiquetar a personas de raza negra como orangutanes o basarse en la vestimenta y el origen étnico para catalogar a una persona como un riesgo [19]. Es aquí donde nos damos cuenta de que es necesario seguir desarrollando estas tecnologías, mejorando los métodos de entramiento y ampliando los ejemplos con los que se construyen estos sistemas.

Finalmente, comentaremos algunas líneas de investigación y trabajos futuros que podrías surgir a partir de este.

En primer lugar, ligado a las debilidades de la visión artificial, aunque no es un tema relacionado con la línea de investigación, no se puede obviar la afección ética que provocan los fallos de esta tecnología. Algunos ejemplos son los fallos racistas comentados anteriormente o el uso de la visión artificial con fines delictivos. Por ello, se puede sugerir abrir una línea que investigue el uso de esta tecnología con fines poco éticos.

Por otro lado, un trabajo futuro muy interesante sería partir de uno de los casos prácticos realizados en este trabajo, en concreto, el de la identificación del tablero de tres en raya. El objetivo sería construir una inteligencia artificial que aprenda a jugar a este juego. Podría tratarse, por ejemplo, de un brazo robótico, que sería el encargado de la parte mecánica del juego, es decir, de realizar el movimiento de las fichas. El aprendizaje se llevaría a cabo mediante una red neuronal. A esta red neuronal podríamos entrenarla mediante métodos de aprendizaje supervisado. Para empezar, podríamos pasarle algunas

jugadas de ejemplo para que sepa el movimiento que tiene que hacer dependiendo de la estructura de tablero que encuentre. Posteriormente, la dejaríamos jugando con otra máquina ya entrenada o con cualquier persona, recibiendo una “recompensa” cuando sea capaz de ganar la partida y un “castigo” cuando pierda. El objetivo final será crear una máquina que haya aprendido a jugar al tres en raya desde cero y sea capaz de ganar siempre, o al menos, no perder.

Referencias

[1] Contaval, "¿Qué es la visión artificial y para qué sirve?," *Deteccion Blog*, 2016. https://www.contaval.es/que-es-la-vision-artificial-y-para-que-sirve/ (accessed Sep. 05, 2022).

[2] IBM, "¿Qué es la Visión Artificial?," *IBM*, 2022. https://www.ibm.com/es-es/topics/computer-vision (accessed Aug. 26, 2022).

[3] R. A. Aquino Castro, "Reconocimiento E Interpretación Del Alfabeto Dactilológico De La Lengua De Señas Mediante Tecnología Móvil Y Redes Neuronales Artificiales," Universidad Mayor de San Andrés, 2018.

[4] "Sistemas de Visión Artificial, Historia, Componentes y Procesamiento de Imágenes." https://1library.co/document/y9r89vry-sistemas-de-vision-artificial-historia-componentes-y-procesamiento-de-imagenes.html (accessed Aug. 24, 2022).

[5] J. Molleda Meré, "Técnicas de visión por computador para la reconstrucción en tiempo real de la forma 3D de productos laminados," Universidad de Oviedo, 2008.

[6] E. Cuevas, D. Zaldívar, and M. Pérez, *Procesamiento digital de imágenes con MATLAB y Simulink*, 1st ed. Mexico: RA-MA, 2010.

[7] N. L. Fernández García, "Introduccion a la Visión Artificial. Visión Artificial Avanzada." Escuela Politecnica Superior Universidad de Cordoba, pp. 1–434. [Online]. Available: www.freelibros.me

[8] J. F. Vélez Serrano, A. B. Moreno Díaz, A. Sánchez Calle, and J. L. E. Sánchez-Marín, *Vision por Computador*. RA-MA, 2003.

[9] R. C. González and R. E. Woods, *Digital Image Processing*, 3ª. Pearson Prentice Hall, 2008.

[10] R. A. Schowengerdt, *Remote Sensing: Models and Methods for Image Processing*. Academic Press, 1997.

[11] S. Hadri, "Image Processing Best Practices — C++ Part 2," *Medium*, 2020. https://soubhihadri.medium.com/image-processing-best-practices-c-part-2-c0988b2d3e0c

[12] L. Sáez Acosta, "Detección de bordes en una imagen." Universidad de Jaén, 2015. [Online]. Available: https://docplayer.es/4877825-Deteccion-de-bordes-en-una-imagen.html

[13] E. García-Meléndez, “Módulo VII : Sistemas de Información Geográfica y Teledetección, análisis visual de imágenes.” Escuela de Negocios (EOI), 2007.

[14] MathWorks, “Detect and Measure Circular Objects in an Image,” 2022. https://es.mathworks.com/help/images/detect-and-measure-circular-objects-in-an-image.html (accessed Sep. 04, 2022).

[15] MathWorks, “‘imfindcircles,’” 2022. https://es.mathworks.com/help/images/ref/imfindcircles.html#:~:text=imfindcircles convierte las imágenes en,función im2single antes de procesarlas (accessed Aug. 25, 2022).

[16] A. e Ingenia, “DETECCION Y RECONOCIMEINTO DE OBJETOS en Matlab | DETECCION DE LINEAS en Matlab | Visión Artificial.” 2021. [Online]. Available: https://www.youtube.com/watch?v=RG5F5oM5btE

[17] PhysioNet, “Filtrado Lineal Óptimo: Filtrado de Wiener.” http://physionet.cps.unizar.es/~eduardo/docencia/tds/librohtml/wiener1.htm (accessed Sep. 04, 2022).

[18] Cognex, “Introducción a la Visión Artificial. Una guía para la automatización de procesos y mejorasde calidad,” no. November. 2014. [Online]. Available: http://www.ikusmen.com/documentos/descargas/3cbb38_Introduction to Machine Vision.pdf

[19] M. Salas, “Visión artificial y personas: cuando el roce hace el cariño,” *BBVA Next Technologies*, 2021. https://www.bbvanexttechnologies.com/pills/vision-artificial-y-personas-cuando-el-roce-hace-el-carino/ (accessed Sep. 05, 2022).

Apéndice A – Código de los programas de procesamiento de imágenes

A.1 Procesos de pre-procesamiento y segmentación en imágenes

A.1.1 Pre-procesamiento

```
%Pre-procesamiento
%Comenzamos leyendo la imagen
A=imread('C:\Users\34684\Documents\TFG\MATLAB\practica2\objetos.jpeg')
;
%medimos su tamaño
[m,n] = size(A);

%para ver la imagen desde matlab
imshow(A);
Ag = rgb2gray(A);
subplot(2,2,1)
imshow(Ag);
title('Imagen escala de grises')

%Obtenemos el mapa de pixeles de la foto:
Ag;
%y si queremos el valor de un pixel concreto:
Ag(1,1) %nos devuelve el valor del primer pixel

%Segunda parte:
%Vamos a aplicar distintos filtros de suavizado

%Filtro 1:
%Filtro 3x3 que suaviza la imagen utilizando el promedio de los
pixeles
%vecinos

[m n] = size(Ag);
Ag = double(Ag);
Agr = Ag;
for r = 2:m-1
    for c = 2:n-1
        Agr(r,c) = 1/9*(Ag(r-1,c-1)+Ag(r-1,c)+Ag(r-1,c+1)+...
        Ag(r,c-1)+Ag(r,c)+Ag(r,c+1)+Ag(r+1,c-1)+...
        Ag(r+1,c)+Ag(r+1,c+1));
    end
end

Agr = uint8(Agr);
imshow(Agr)
title('Imagen filtro estandar')

%Filtro 2:
%Filtro de Gauss utilizando la mascara Gaussiana 3x3

Agg = Ag;
for r = 2:m-1
    for c = 2:n-1
        Agg(r,c) = 1/16*(Ag(r-1,c-1)+2*Ag(r-1,c)+Ag(r-1,c+1)+...
        2*Ag(r,c-1)+4*Ag(r,c)+2*Ag(r,c+1)+Ag(r+1,c-1)+...
```

```
        2*Ag(r+1,c)+Ag(r+1,c+1));
    end
end

Agg = uint8(Agg);
imshow(Agg)
title('Imagen filtro gaussiano')

%Otra forma para ver la efectividad de los filtros de suavizado es
añadir
%ruido a una imagen y posteriormente aplicar los filtros de suavizado
% Añadimos ruido de tipo pimienta

Agruido = Ag;
[m,n] = size(Agruido);
for v = 1:1000
    x = round(rand*m);
    y = round(rand*n);
    %Como MATLAB no indexa a partir de 0 protegemos el programa y
hacemos
    %que empiece en 1
    if x == 0
        x = 1;
    end
    if y == 0
        y = 1;
    end
    %Tambien se protege el programa
    if x == 600
        x = 598;
    end
    if y == 800
        y = 798;
    end
    Agruido(x,y) = 255;
    Agruido(x,y+1) = 255;
    Agruido(x+1,y) = 255;
    Agruido(x+1,y+1) = 255;
    Agruido(x+2,y+1) = 255;
    Agruido(x+2,y) = 255;
end
%se calculan 1000 puntos de ruido con valor 0 y se repite el proceso
%anterior
for v = 1:1000
    x = round(rand*m);
    y = round(rand*n);
    %Como MATLAB no indexa a partir de 0 protegemos el programa y
hacemos
    %que empiece en 1
    if x == 0
        x = 1;
    end
    if y == 0
        y = 1;
    end
    %Tambien se protege el programa
    if x == 600
        x = 598;
    end
    if y == 800
        y = 798;
```

```
    end
    Agruido(x,y) = 0;
    Agruido(x,y+1) = 0;
    Agruido(x+1,y) = 0;
    Agruido(x+1,y+1) = 0;
    Agruido(x+2,y+1) = 0;
    Agruido(x+2,y) = 0;
end

Agruido = uint8(Agruido);

subplot(2,2,4)

imshow(Agruido)
title('Imagen con ruido')


%Utilizamos una función nativa de MATLAB para agregar un ruido mas
suave

fn=imnoise(Ag,'salt & pepper',0.05);


%Ahora utilizamos el filtro estándar con una mascara de 3x3 y después
con una de 9x9

h1=fspecial('average');
h2=fspecial('average',[9,9]);
media1=imfilter(fn,h1);
media2=imfilter(fn,h2);

subplot(2,2,1),subimage(Ag),title('Imagen original');
subplot(2,2,2),subimage(fn),title('Imagen con ruido');
subplot(2,2,3),subimage(media1),title('Filtrado mascara 3x3');
subplot(2,2,4),subimage(media2),title('Filtrado mascara 9x9');


%Una vez tenemos la imagen con ruido volvemos a ejecutar los filtros
%de suavizado con distintas imágenes, no cambia nada mas del código.
```

A.1.2 Segmentación

```
%Una vez que tenemos la imagen filtrada podemos dar como concluido el
%pre-procesamiento y pasamos ahora a la segmentacion
%En esta parte nos centraremos en la deteccion de los bordes de la
imagen
%Comenzamos por la detección de bordes binaria

imshow(A);
Ag = rgb2gray(A);
subplot(1,2,1)
imshow(Ag);
title('Imagen escala de grises')


for i = 1:m
    for j = 1:1668
       if I_gray(i,j) > 145
           I_gray(i,j) = 255;
       else
           Ag(i,j) = 0;
```

```
       end
    end
end
subplot(1,2,1)
imshow(I_gray)

%Para la segunda imagen cambiamos el umbral
imshow(A);
Ag = rgb2gray(A);
subplot(1,2,1)
imshow(Ag);
title('Imagen escala de grises')

for i = 1:m
    for j = 1:1465
       if Ag(i,j) > 162
           Ag(i,j) = 255;
       else
           Ag(i,j) = 0;
       end
    end
end
subplot(1,2,2)
imshow(Ag)
title('Imagen binaria')



%Continuamos usando el filtro de sobel
Agsobel = Ag;
Agsobel = double(Agsobel);
[m,n] = size(Agsobel);
%Creamos matrices de ceros
Gx = zeros(size(Agsobel));
Gy = zeros(size(Agsobel));
%Primera parte: Aplicamos los filtros de Sobel a la imagen Gx en la
%direccion x y a Gy en la direccion y
for r = 2:m-1
    for c = 2:n-1
        Gx(r,c) = -1*Agsobel(r-1,c-1)-2*Agsobel(r-1,c)-Agsobel(r-
1,c+1)...
            +Agsobel(r+1,c-1)+2*Agsobel(r+1,c)+Agsobel(r+1,c+1)
        Gy(r,c) = -1*Agsobel(r-1,c-1)+Agsobel(r+1,c+1)-2*Im(r,c-1)...
            +2*Agsobel(r,c+1)-Agsobel(r+1,c-1)+Agsobel(r+1,c+1)
    end
end
%Segunda parte: calculamos el valor total del gradiente
Gt = sqrt(Gx.^2+Gy.^2);
%Valos maximo del gradiente
VmaxGt = max(max(Gt));
%Normalizamos el gradiente a 255
GtN = (Gt/VmaxGt)*255;

GtN = uint8(GtN)

%Ahora aplicamos el filtro de Sobel y el de Prewitt directamente con
la función de MATLAB y no normalizamos

sf=fspecial('sobel');
sc=sf';
b1=imfilter(im1,sf);
```

```
b2=imfilter(im1,sc);
b3=imadd(b1,b2);
subplot(2,2,1),imshow(im1);
title('Imagen escala de grises')
%imshow(b2);
%imshow(b1);
imshow(b3);
title('Filtro de Sobel')

sf1 = fspecial('prewitt');
sc1 = sf1';
b11=imfilter(im1,sf1);
b21=imfilter(im1,sc1);
b31=imadd(b11,b21);
imshow(b31);
title('Filtro Prewitt')


% Pasamos ahora a estudiar los histogramas
[cont,x] = imhist(A);
subplot(1,2,2)
stem(x,cont)

%hacemos un histograma acumulativo
va = 0;
for v = 1:256
    H(v) = va+cont(v);
    va = H(v);
end
%subplot(2,2,2)
stem(x,H)

%ahora vamos a aumentar el contraste de la imagen
B = I*1.3;
imshow(B)
[cont,x] = imhist(B);
stem(x,cont)

%ahora aumentamos la iluminación, para ello sumamos los niveles que
%queramos iluminar la imagen, lo que ocurre en el histograma es que se
%desplaza 20 unidades hacia la derecha
C = I+20;
imshow(C)
[cont,x] = imhist(C);
stem(x,cont)
```

A.2 Detección de círculos en una imagen

A.2.1 Programa básico

```
% Reconocimiento de circulos en una imagen.
clc
clear all
close all
%Comenzamos cargando la imagen
image = imread('circulos.png');
imshow(image)
```

```
% Antes de aplicar esta funcion es recomendable saber si los circulos
a
% detectar son mas o menos brillantes que el fondo, para ello pasamos
la
% imagen a escala de grises.

gray_image = rgb2gray(image);
imshow(gray_image)

% Vemos que la mayoria de los circulos son mas oscuros que el fondo,
sin
% embargo, la funcion por defecto busca circulos mas claros que el
fondo,
% por lo que tenemos que añadir los parametros 'ObjectPolarity' and
'dark'

[centers,radii] = imfindcircles(image,[15
35],'ObjectPolarity','dark',...
    'Sensitivity',0.9)

imshow(image)
h = viscircles(centers,radii);

%Hasta ahora hemos conseguido que reconozca todos los circulos excepto
los
%amarillos ya que estos eran mas claros que el fondo. Vamos a cambiar
los
%parametros de la funcion imfindcircles

[centersBright,radiiBright] = imfindcircles(image,[15 35], ...
    'ObjectPolarity','bright','Sensitivity',0.92);
imshow(image)

%Ahora queremos que aparezcan todos los circulos marcados

hBright = viscircles(centersBright, radiiBright,'Color','b');

[centersBright,radiiBright,metricBright] = imfindcircles(image,[15
35], ...
    'ObjectPolarity','bright','Sensitivity',0.92,'EdgeThreshold',0.1);

delete(hBright)
hBright = viscircles(centersBright, radiiBright,'Color','b');

h = viscircles(centers,radii);

numero = length(radii) + length(radiiBright);
disp('Número de círculos totales:')
disp(numero)
```

A.2.2 Aplicación al juego de tres en raya

```
%Deteccion de circulos en el tres en raya
clc
clear all
close all
%Comenzamos cargando la imagen del primer tablero
```

```
image = imread('tablero1.png');
imshow(image)

%La pasamos a tonos grises para ver si los circulos son mas claros o
%oscuros que el fondo

gray_image = rgb2gray(image);
imshow(gray_image)

%Pasamos la funcion con 'ObjectPolarity' como 'dark'

[centers,radii] = imfindcircles(image,[15
35],'ObjectPolarity','dark',...
    'Sensitivity',0.9)

imshow(image)
h = viscircles(centers,radii);

%Vemos que nos detecta todos los circulos, y nos da los centros de los
%circulos, lo que nos falta es saber en que cuadrante de la imagen
estan

%Comenzamos calculando las dimensiones de la imagen

m = size(image)


%Sabemos entonces que el ancho son 149 y el largo 150, lo que tiene
sentido
%porque pretende ser una imagen cuadrada.

%Ahora tenemos que dividir la imagen en 9 regiones iguales y hacer
%corresponder cada circulo a su region
centro = centers;

n = length(radii);
numero_fichas = n

pos = [0,0,0;0,0,0;0,0,0];
color = [0,0,0;0,0,0;0,0,0];

for i = 1:n
    if centro(i) < m(1)/3 & centro(i+n) < m(2)/3
        pos(1,1) = 1;
        if pos(1,1) == 1
            color_pos1 = impixel(image,centro(i),centro(i+n));
            if color_pos1(3) ~= 0
                color(1,1) = 1;
            else
                color(1,1) = 2;
            end
        end
    elseif centro(i) >= m(1)/3 & centro(i) < 2*m(2)/3 & centro (i+n) <
m(2)/3
        pos(1,2) = 1;
        if pos(1,2) == 1
            color_pos2 = impixel(image,centro(i),centro(i+n));
            if color_pos2(3) ~= 0
                color(1,2) = 1;
```

```
            else
                color(1,2) = 2;
            end
        end
    elseif centro(i) >= 2*m(1)/3 & centro(i+n) < m(2)/3
        pos(1,3) = 1;
        if pos(1,3) == 1
            color_pos3 = impixel(image,centro(i),centro(i+n));
            if color_pos3(3) ~= 0
                color(1,3) =1;
            else
                color(1,3) = 2;
            end
        end
    elseif centro(i) < m(1)/3 & centro(i+n) > m(2)/3 & centro(i+n) <
2*m(2)/3
        pos(2,1) = 1;
        if pos(2,1) == 1
            color_pos4 = impixel(image,centro(i),centro(i+n));
            if color_pos4(3) ~= 0
                color(2,1) = 1;
            else
                color(2,1) = 2;
            end
        end
    elseif centro(i) >= m(1)/3 & centro(i) < 2*m(2)/3 & centro(i+n) >
m(2)/3 & centro(i+n) < 2*m(2)/3
        pos(2,2) = 1;
        if pos(2,2) == 1
            color_pos5 = impixel(image,centro(i),centro(i+n));
            if color_pos5(3) ~= 0
                color(2,2) = 1;
            else
                color(2,2) = 2;
            end
        end
    elseif centro(i) >= 2*m(1)/3 & centro(i+n) > m(2)/3 & centro(i+n)
< 2*m(2)/3
        pos(2,3) = 1;
        if pos(2,3) == 1
            color_pos6 = impixel(image,centro(i),centro(i+n));
            if color_pos6(3) ~= 0
                color(2,3) = 1;
            else
                color(2,3) = 2;
            end
        end
    elseif centro(i) < m(1)/3 & centro(i+n) >= 2*m(2)/3
        pos(3,1) = 1;
        if pos(3,1) == 1
            color_pos7 = impixel(image,centro(i),centro(i+n));
            if color_pos7(3) ~= 0
                color(3,1) = 1;
            else
                color(3,1) = 2;
            end
        end
    elseif centro(i) >= m(1)/3 & centro(i) < 2*m(2)/3 & centro(i+n) >=
2*m(2)/3
        pos(3,2) = 1;
        if pos(3,2) == 1
```

```
            color_pos8 = impixel(image,centro(i),centro(i+n));
            if color_pos8(3) ~= 0
                color(3,2) = 1;
            else
                color(3,2) = 2;
            end
        end
    elseif centro(i) >= 2*m(1)/3 & centro(i+n) >= 2*m(2)/3
        pos(3,3) = 1;
        if pos(3,3) == 1
            color_pos9 = impixel(image,centro(i),centro(i+n));
            if color_pos9(3) ~= 0
                color(3,3) = 1;
            else
                color(3,3) = 2;
            end
        end
    end
end

%obtenemos finalmente una matriz 3x3 que representa el tablero e
indica con
%el numero 1 que en esa posicion hay una ficha azul, con el numero 2
que
%hay una ficha roja y con el numero 0 que no hay ninguna ficha en esa
%posicion.

tablero = color
```

A.3 Detección e identificación de objetos en una imagen

```
%Deteccion e identificacion de objetos

%Vamos a partir de una imagen con varios objetos y queremos obtener un
%valor numerico que el ordenador sea capaz de entender como
identificador
%de ese tipo de objeto concreto.

clc
clear all
close all

%Comenzamos cargando la imagen y pasandola a escala de grises

image = imread('objetos_negro2.jpeg');
imshow(image)
image = rgb2gray(image);
imshow(image)

%Suavizamos la imagen aplicando el filtro de weiner (luego tenemos que
%probar con el de Gauss por ejemplo)

image = wiener2(image,[50,50]);
figure
imshow(image)

%Ahora binarizamos la imagen

image = im2bw(image,0.45);
```

```
figure
imshow(image)

%Seguimos eliminando el ruido y eliminamos las regiones que tengan
menos de
%130 pixeles para que no confundamos ruido con otros objetos

image = bwareaopen(image,130);
figure
imshow(image)

%Ahora rellenamos los espacios de los objetos

se = strel('disk',5);
image = imclose(image,se);
figure
imshow(image)

%Con esto hemos conseguido tener los espacios bien delimitados, ahora
vamos
%a utilizar los bordes de los objetos para delimitar las regiones y
%caracterizar cada objeto

[B,L] = bwboundaries(image,'noholes');
imshow(label2rgb(L,@jet,[.5 .5 .5]))
hold on

for k = 1: length(B)
    boundary = B{k}; %bucle para dibujar los contornos de las regiones
    plot(boundary(:,2),boundary(:,1),'w','LineWidth',2)
end

%Ahora vamos a empezar a obtener los parametros que nos van a permitir
%identificar los objetos
%Extraemos la informacion del centro del objeto y del area del mismo

stats = regionprops(L,'Area','Centroid');
threshold = 0.94;
for k = 1:length(B)
    boundary = B{k};
    delta_sq = diff(boundary).^2;
    perimeter = sum(sqrt(sum(delta_sq,2))); %Calculamos el perimetro
    area = stats(k).Area; %Calculamos el area
    metric = 4*pi*area/perimeter^2; %Calculamos una metrica
    metric_string = sprintf('%2.2f',metric); %Asignamos cada metrica a
cada region
    if metric > threshold
        centroid = stats(k).Centroid;
        plot(centroid(1),centroid(2),'ko')
    end
    text(boundary(1,2)-
35,boundary(1,1)+13,metric_string,'Color','k','FontSize',15,'FontWeigh
t','bold')
end
```