ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

TRABAJO FIN DE GRADO

SISTEMA DE INVENTARIADO PARA DRONES USANDO VISIÓN ARTIFICIAL

Autor: Irene España Novillo

Directores:
Jaime Boal Martín-Larrauri
Miguel Manuel Martín Lopo

MADRID
Junio 2018

Copyright © 2018 Irene España Novillo

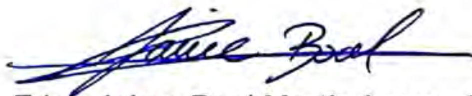
Este trabajo fue escrito con LaTeX y compilado en TeXmaker usando la distribución TeXLive 2017. Las familias de fuentes usadas son Bitstream Charter, Utopia, Bookman, y Computer Modern. A menos que se indique lo contrario, todas las figuras fueron creadas por el autor usando Microsoft Visio®, Microsoft Excel® y Microsoft Word®.

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
"Sistema de inventariado para drones usando visión artificial"
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2017/2018 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Irene España Novillo Fecha: 11 / 06 / 2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Jaime Boal Martín-Larrauri Fecha: 11 / 06 / 2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Miguel Manuel Martín Lopo Fecha: 11 / 06 / 2018

Scanned by CamScanner

Scanned by CamScanner

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor Dña. Irene España Novillo DECLARA ser el titular de los derechos de propiedad intelectual de la obra: "Sistema de inventariado para drones usando visión artificial", que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

Scanned by CamScanner

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a ..11.. de ..JUNIO.. de ..2018..

ACEPTA

Fdo. IRENE ESPAÑA NOVILLO

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Scanned by CamScanner

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

TRABAJO FIN DE GRADO

SISTEMA DE INVENTARIADO PARA DRONES USANDO VISIÓN ARTIFICIAL

Autor: Irene España Novillo

Directores:
Jaime Boal Martín-Larrauri
Miguel Manuel Martín Lopo

MADRID
Junio 2018

Copyright © 2018 Irene España Novillo

Este trabajo fue escrito con LaTeX y compilado en TeXmaker usando la distribución TeXLive 2017. Las familias de fuentes usadas son Bitstream Charter, Utopia, Bookman, y Computer Modern. A menos que se indique lo contrario, todas las figuras fueron creadas por el autor usando Microsoft Visio®, Microsoft Excel® y Microsoft Word®.

SISTEMA DE INVENTARIADO PARA DRONES USANDO VISION ARTIFICIAL

Autor: España Novillo, Irene.

Directores: Boal Martín-Larrauri, Jaime y Martín Lopo, Miguel.

Entidad colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

La visión artificial se presenta como una solución para realizar el proceso de inventario mediante drones. El presente proyecto describe un algoritmo de visión artificial que realiza la localización y lectura de códigos de barras y QR a partir de imágenes tomadas con diferentes orientaciones y condiciones de iluminación. El algoritmo consigue leer más de un 80% de los códigos empleando un tiempo medio de ejecución de 787,00 milisegundos, lo que no compromete el funcionamiento del dron.

Palabras clave: Visión artificial, Códigos de barras, Códigos QR, Drones, Inventario.

1. Introducción

Una actividad fundamental en la gestión de cualquier almacén es la realización del inventario. Sin embargo, realizar dicha tarea de forma manual requiere un incremento de recursos económicos y temporales, lo que la hace ineficiente. Los drones, con su vuelo estable, permiten automatizar el proceso, registrando la información de los paquetes que se encuentran en zonas difícilmente accesibles. Para extraer la información de los paquetes podría instalarse un lector de códigos en el dron pero el sensor adicional aumenta el consumo de batería y el peso del conjunto, disminuyendo la autonomía y dificultando el vuelo. El desarrollo de un algoritmo de visión artificial pretende solventar este problema.

2. Definición del proyecto

Se ha realizado la descripción e implementación de un algoritmo que permite localizar y leer códigos de barras y QR a partir de imágenes tomadas con diferentes orientaciones y condiciones de iluminación. Dicho algoritmo es capaz de ejecutarse en 787,00 milisegundos de media, es decir, en un tiempo razonable que no compromete el funcionamiento del dron.

3. Descripción del sistema

El algoritmo diseñado, sigue el proceso mostrado en la Figura 1. El procedimiento consta de tres fases ejecutadas de manera secuencial a la imagen que contiene el código: preprocesamiento, localización y lectura, dando como resultado la información decodificada. Además, después de la fase de localización y previamente a la fase de lectura pueden aplicarse una serie de correcciones a la imagen de forma opcional, solo en caso de que sean necesarias.

En primer lugar, durante la fase de preprocesamiento se pasa la imagen a escala de grises y se binariza mediante el método de Otsu como en [1] y [2]. De esta forma se elimina la información relativa al color. Para acabar con esta fase se realiza un cierre

y una apertura de la imagen con el fin de eliminar información no relevante para la lectura del código, como puede ser texto que rodea.

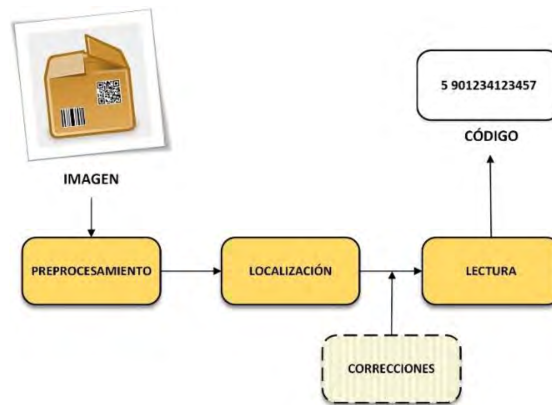


Figura 1: Diagrama de bloques general del sistema.

La fase de localización se basa en ubicar la posición del código dentro de la propia imagen. Se aplica, en primer lugar, el algoritmo de detección de bordes Canny [3]. Tras esto, se buscan contornos cerrados rectangulares y se seleccionan aquellos que posean un área mayor, los cuales serán candidatos a ser códigos de barras. Una vez que se ha obtenido la posición de los distintos rectángulos, se hace una corrección de la rotación de la imagen con el fin de dejar las barras del código de manera perpendicular a la horizontal. Para finalizar con esta fase se recorta la imagen, eliminando las partes externas al área del código.

La última fase es la lectura, en la cual se procede a extraer la información del código. Una vez que se poseen las imágenes recortadas y rotadas se realiza un escaneo de las mismas en busca de los símbolos que forman el código. El resultado es la secuencia de caracteres decodificados. Para llevar a cabo esta fase se emplean las funciones proporcionadas por la librería de lectura de códigos ZBar mientras que para aplicar las técnicas de las fases anteriores se hace uso de la libería de tratamiento de imágenes OpenCV.

4. Resultados

- En cuanto a los resultados de eficiencia, el algoritmo diseñado consigue detectar y leer más del 80% del total de los códigos de barras para el conjunto de imágenes probadas mientras que otro algoritmo que únicamente emplea ZBar, consigue leer más del 76,9%, tal y como se muestra en la Figura 2.
- Para los resultados de tiempo, el algoritmo diseñado emplea una media de 787,00 milisegundos mientras que el otro algoritmo emplea 654,83 en ejecutarse en la Raspberry Pi Zero W. El detalle de los tiempos aparece en la Figura 3.
- Los resultados obtenidos para los códigos QR son un 83,33% de códigos leídos tanto por el algoritmo diseñado como por el de ZBar.

5. Conclusiones

Se ha desarrollado un algoritmo de detección de códigos de barras y QR que evita

tener que leer la imagen completa para encontrar el código y que al mismo tiempo mejora ligeramente la eficacia en la decodificación. Además, los resultados sugieren que se podría utilizar el algoritmo desarrollado para leer códigos de barras con un tiempo de muestreo de 787 milisegundos, lo cual es compatible con la velocidad a la que se desplaza un dron.

Figura 2: Histograma del número de códigos de barras leídos por ambos algoritmos para imágenes con distintas condiciones de iluminación y orientaciones del código.

Figura 3: Gráfico del tiempo ejecución empleado por ambos algoritmos.

6. Referencias

[1] D. Chai and F. Hock, "Locating and Decoding EAN-13 Barcodes from Images Captured by Digital Cameras," *2005 5th Int. Conf. Inf. Commun. Signal Process.*, no. 1010, pp. 1595–1599, 2005.

[2] Y. Gu and W. Zhang, "QR code recognition based on image processing," *Int. Conf. Inf. Sci. Technol.*, pp. 733–736, 2011.

[3] F. Liu, J. Yin, K. Li, and Q. Liu, "An improved recognition method of PDF417 barcode," *2010 Chinese Conf. Pattern Recognition, CCPR 2010 - Proc.*, pp. 450–454, 2010.

INVENTORY SYSTEM FOR DRONES USING COMPUTER VISION

Author: España Novillo, Irene.
Supervisors: Boal Martín-Larrauri, Jaime and Martín Lopo, Miguel.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

Computer vision is presented as a solution to carry out the inventory process through drones. The present project describes an artificial vision algorithm that performs the localization and reading of bar codes and QR codes from images taken with different orientations and lighting conditions. The algorithm manages to read more than 80% of the codes, using an average execution time of 787,000 milliseconds, which does not compromise the operation of the drone.

Keywords: Computer vision, Bar codes, QR codes, Drones, Inventory.

1. Introduction

A fundamental activity in the management of any warehouse is the realization of the inventory. However, performing this task manually requires an increase in economic and temporal resources, which makes it inefficient. The drones, with their stable flight, allow to automate the process, registering the information of the packages that are in areas that are difficult to access. To extract the information from the packages a code reader could be installed in the drone but the additional sensor increases the battery consumption and the weight of the set, decreasing the autonomy and making the flight more difficult. The development of an artificial vision algorithm aims to solve this problem.

2. Project definition

The description and implementation of an algorithm that enables locating and reading bar and QR codes from pictures taken with different orientations and lighting conditions has been made. This algorithm is capable of executing in 787,00 milliseconds on average, that is, in a reasonable time that does not compromise the operation of the drone.

3. Algorithm description

The designed algorithm follows the process shown in Figure 1. The procedure consists of three phases executed sequentially to the image containing the code: preprocessing, localization and reading, resulting in decoded information. In addition, after the localization phase and before the reading phase, a series of corrections to the image can be applied optionally, only in case they are necessary.

First, during the preprocessing phase, the image is grayscale and binarized using the Otsu method as in [1] and [2]. In this way the information regarding the color is eliminated. To finish this phase, the image is closed and opened in order to eliminate non-relevant information to reading the code, such as surrounding text.

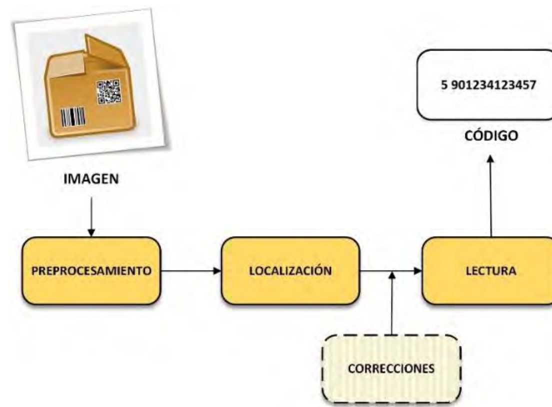


Figure 1: General block diagram of the system.

The location phase is based on locating the position of the code within the image itself. The Canny edge detection algorithm [3] is applied first. After this, rectangular closed contours are searched and those that have a larger area are selected, which will be candidates to be bar codes. Once the position of the different rectangles has been obtained, a correction of the rotation of the image is made in order to leave the bars of the code perpendicular to the horizontal. To finish with this phase, the image is cut out, eliminating the external parts to the code area.

The last phase is reading, in which we proceed to extract the information from the code. Once the cut and rotated images are possessed, they are scanned in search of the symbols that make up the code. The result is the sequence of decoded characters. In order to carry out this phase, the functions provided by the ZBar code reading library are used, while the OpenCV image processing software is used to apply the techniques of the previous phases.

4. Results

- Regarding the efficiency results, the designed algorithm manages to detect and read more than 80% of the total bar codes for the set of images tested while another algorithm that only uses ZBar, manages to read more than 76.9% , as shown in Figure 2.

- For the time results, the designed algorithm uses an average of 787.00 milliseconds while the other algorithm uses 654.83 in executing in the Raspberry Pi Zero W. The detail of the times appears in Figure 3.

- The results obtained for QR codes are 83.33% of codes read by both the designed algorithm and ZBar.

5. Conclusions

A bar code and QR detection algorithm has been developed which avoids having to read the complete image to find the code and at the same time slightly improves the decoding efficiency. In addition, the results suggest that the developed algorithm could be used to read bar codes with a sampling time of 787 millisecons, which is compatible with the speed at which a drone moves.

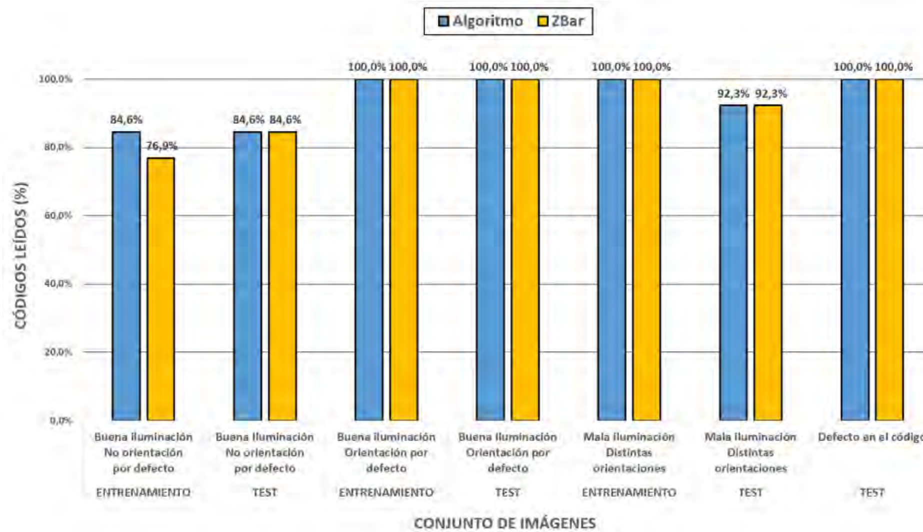


Figure 2: Histogram of the number of bar codes read by both algorithms for images with different lighting conditions and code orientations.

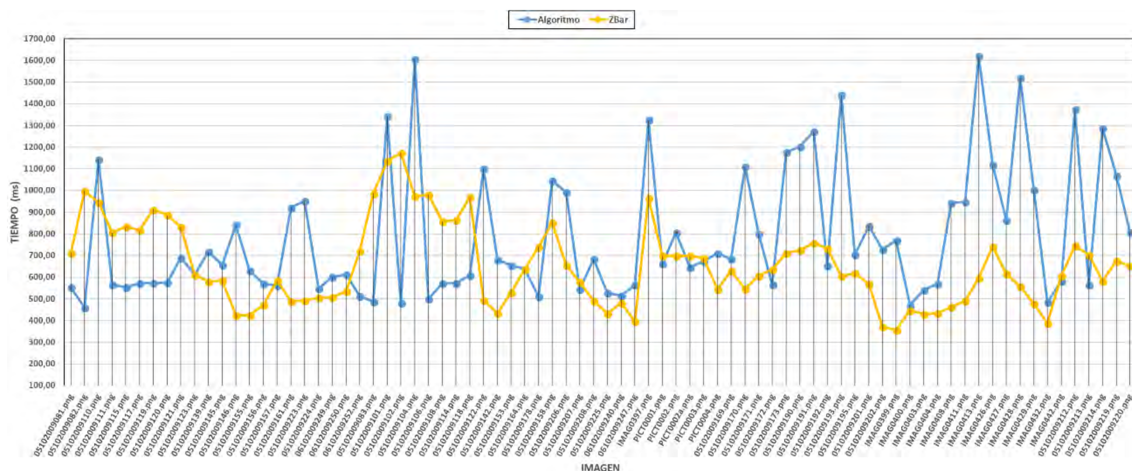


Figure 3: Graph of the execution time used by both algorithms.

6. References

[1] D. Chai and F. Hock, "Locating and Decoding EAN-13 Barcodes from Images Captured by Digital Cameras," 2005 5th Int. Conf. Inf. Commun. Signal Process., no. 1010, pp. 1595–1599, 2005.

[2] Y. Gu and W. Zhang, "QR code recognition based on image processing," Int. Conf. Inf. Sci. Technol., pp. 733–736, 2011.

[3] F. Liu, J. Yin, K. Li, and Q. Liu, "An improved recognition method of PDF417 barcode," 2010 Chinese Conf. Pattern Recognition, CCPR 2010 - Proc., pp. 450–454, 2010.

A mi abuelo Emeterio, sé que me cuidas desde el cielo.

A mi padre, a mi madre y a mi hermano.
Por el apoyo, la comprensión y el cariño.

A todas las mujeres ingenieras: pasadas, presentes y futuras.
Por aportar su granito de arena.

No es grande el que siempre triunfa,
sino el que jamás se desalienta.
Martín Descalzo
(1930–1991)

Agradecimientos

Mención especial merecen mis directores de proyecto, Jaime y Miguel, por estar siempre dispuestos a brindarme su ayuda, ya sea por correo electrónico o en el despacho, días laborables o fines de semana. Gracias por haber confiado en mí la realización de este trabajo, espero haber estado a la altura. Igualmente, gracias al resto de profesores que me han acompañado a lo largo de mis estudios universitarios de grado. Este proyecto es el resultado de cuatro años de aprendizaje en cada una de las clases que habéis impartido, no solo a nivel técnico y profesional sino también personal. Cuatro años haciendo del trabajo y la constancia los valores imprescindibles para conseguir cualquier meta.

Por otro lado, estoy inmensamente agradecida a mis padres y mi hermano, por darme los tres la oportunidad de estudiar en ICAI con su incansable esfuerzo. Al resto de mi familia, abuelos, tíos y primos, por haberse interesado siempre en mi trabajo y confiado en mi capacidad para salir adelante. En especial, mis abuelos y abuelas, por haber rezado más de una oración en tiempos de exámenes finales.

No puedo olvidarme de todas las grandes personas que he conocido durante estos años de universidad. Elena, mi fiel compañera de camino, gracias por la paciencia en los laboratorios y tus "luego te lo explico" durante las clases. Teresa, ejemplo de constancia y superación, por apoyarme siempre. Gracias a mi compi-charlas Conchita, a Marta por esas prácticas de redes acabadas en informes a las ocho de la tarde y Carlos, por siempre intentar sacarme una sonrisa con gifs de pingüinos. Gracias al resto de compañeros de clase por haber crecido desde septiembre de 2014 junto a mí y al resto de amigos de ICAI porque tenemos en común algo grande.

Gracias también a mis compañeros de residencia. Bea, más una hermana que una veterana, por ayudarme a salir de ese agujero negro que es primero de ingeniería y Juan, por las visitas bien recibidas a la 507, siempre has sabido levantarme el ánimo. A Dani, por hacer que este cuarto año de residencia haya pasado mucho más rápido de lo que esperaba, siempre tienes una buena historia que contarme. Gracias, en general, tanto al personal de la residencia como a los propios residentes, sois mi familia de Madrid.

A todas mis amigas manchegas, por conseguir sacarme de casa algún fin de semana que otro bajo el argumento de "tienes que despejarte", finalmente traducido en partidas de futbolín. Por desearme siempre buena suerte y haber estado tanto en los momentos buenos como en los que no lo han sido tanto. Ellas saben que las quiero. Gracias también a mi incansable corrector a distancia, por haber leído el Anexo B más veces que yo misma, por los consejos y el apoyo. Sin ti este proyecto no hubiese salido a flote.

Al resto de personas que no he mencionado (porque la lista sería infinita) pero que sé que siempre han estado ahí... GRACIAS.

Índice general

1. Introducción **1**
1.1. Motivación . . . 2
1.2. Objetivo . . . 2
1.3. Herramientas . . . 3
1.4. Organización del documento . . . 3

2. Clasificación y estructura de los códigos **5**

3. Estado del arte **11**
3.1. Preprocesamiento . . . 12
3.1.1. Escalado de grises . . . 12
3.1.2. Binarización . . . 13
3.1.3. Perfilado del contorno . . . 13
3.1.4. Filtrado de ruido . . . 13
3.1.5. Recortar . . . 14
3.1.6. Redimensionar . . . 16
3.1.7. Normalizar . . . 16
3.2. Localización . . . 16
3.2.1. Localización del código de barras . . . 16
3.2.1.1. Separar componentes conectados . . . 16
3.2.1.2. Líneas de escaneo en diferentes direcciones . . . 17
3.2.1.3. Definir una matriz de similitud . . . 17
3.2.1.4. Transformada Jerárquica de Hough (HHT) . . . 18
3.2.1.5. Algoritmo de selección de candidato . . . 18
3.2.1.6. Búsqueda en espiral . . . 18
3.2.1.7. Esqueletización . . . 20
3.2.1.8. Detección de bordes . . . 20
3.2.1.9. Búsqueda de patrón . . . 21
3.2.1.10. Extracción de la imagen . . . 22
3.2.2. Localización del código QR . . . 22
3.2.2.1. Detección de patrones de localización/tres esquinas . . . 22
3.2.2.2. Detección de patrones de alineamiento . . . 23
3.2.2.3. Estimación de la cuarta esquina . . . 23
3.2.2.4. Estimación del centro . . . 24
3.3. Correcciones . . . 24
3.3.1. Corrección de la rotación . . . 25
3.3.2. Corrección de la inclinación . . . 25

3.3.3. Adecuar líneas 26
3.4. Lectura 27
3.4.1. Lectura del código de barras 27
3.4.1.1. Clasificación jerárquica por características 27
3.4.1.2. Calcular línea perpendicular al código 27
3.4.1.3. Extracción del vector de bits 28
3.4.1.4. Detección de la anchura 29
3.4.1.5. Lectura de patrón izquierdo, centro y derecho 29
3.4.1.6. Aprendizaje por retropropagación de redes neuronales 29
3.4.1.7. Corrección de errores 29
3.4.2. Lectura del código QR 30
3.4.2.1. Generación de cuadrícula 30
3.4.2.2. Reverso del estándar internacional de codificación 30
3.4.2.3. Transformada jerárquica de Hough 30
3.4.2.4. Corrección de error Reed-Solomon 31
3.5. Tablas de técnicas 31

4. Preprocesamiento y localización **35**
4.1. Preprocesamiento 36
4.1.1. Escalado de grises 36
4.1.2. Binarización 37
4.1.3. Cierre 38
4.1.4. Apertura 39
4.1.5. Otras técnicas probadas 39
4.1.5.1. Ecualización adaptativa limitada del histograma de contraste . . 40
4.1.6. Resultados parciales de la versión previa del algoritmo 41
4.2. Localización 42
4.2.1. Canny 42
4.2.2. Encontrar contornos y calcular áreas 43
4.2.3. Recortar y rotar 45
4.2.4. Otras técnicas probadas 48
4.2.4.1. Sobel 48
4.2.4.2. Transformada Jerárquica de Hough (HHT) 49
4.2.5. Resultados de la versión previa del algoritmo 49

5. Lectura **51**
5.1. Lectura del código 52
5.2. Resultados de la versión previa del algoritmo 54

6. Resultados **57**
6.1. Resultados de eficacia 58
6.2. Resultados de tiempo 61
6.2.1. Resultados en ordenador 61
6.2.2. Resultados en la Raspberry Pi Zero W 67
6.3. Pruebas con códigos QR 69

7. Conclusiones **71**

8. Futuros desarrollos **73**

A. Instalación de herramientas **75**
A.1. Instalación de librerías en Raspberry 76
A.1.1. Instalación de OpenCV 76
A.1.2. Instalación de ZBar 76
A.2. Compilación del código para Raspberry 77
A.2.1. Compilación cruzada sobre Linux 77
A.2.2. Compilación directa sobre Raspberry 80

Bibliografía **83**

Índice de figuras

Figura 2.1. Estructura del código de barras EAN-13 6
Figura 2.2. Estructura del código de barras Code 128 6
Figura 2.3. Ejemplo de un código de barras tipo Code 39 7
Figura 2.4. Ejemplo de un código de barras tipo Code 93 7
Figura 2.5. Estructura del código PDF 417 8
Figura 2.6. Ejemplo de código QR 8
Figura 2.7. Estructura del código QR 8
Figura 3.1. Diagrama de bloques general del sistema 12
Figura 3.2. Proceso de dilatación de una imagen 14
Figura 3.3. Ejemplos de aplicación de la técnica de apertura 15
Figura 3.4. Ejemplos de aplicación de la técnica de cierre 15
Figura 3.5. Estructura del código de barras EAN-13 16
Figura 3.6. Técnica de líneas de escaneo en diferentes direcciones 17
Figura 3.7. Procedimiento de cálculo de HHT 19
Figura 3.8. Detección de líneas rectas mediante HHT 19
Figura 3.9. Método de búsqueda en espiral 20
Figura 3.10. Método de esqueletización 20
Figura 3.11. Ejemplo de detección de bordes mediante Canny 21
Figura 3.12. Ejemplo de detección de bordes mediante Sobel 21
Figura 3.13. Imagen tras realizar las cuatro divisiones 22
Figura 3.14. Estructura del código QR 22
Figura 3.15. Patrón de localización del código QR 23
Figura 3.16. Método de estimación de la cuarta esquina 24
Figura 3.17. Método de estimación del centro 24
Figura 3.18. Transformación de Perspectiva Inversa 25
Figura 3.19. Transformación de Control del Punto 26
Figura 3.20. Técnica para adecuar líneas 26
Figura 3.21. Línea perpendicular al código 27
Figura 3.22. Ejemplo de codificación Run-Length de un código de barras 28
Figura 4.1. Ejemplo de imagen en escala de grises 36
Figura 4.2. Ejemplo de imagen en color 36
Figura 4.3. Conversión de una imagen en color a escala de grises 37
Figura 4.4. Ejemplo de distintos tipos de binarización 38
Figura 4.5. Ejemplo de imagen tras haber realizado el cierre de la misma 38
Figura 4.6. Ejemplo de imagen tras haber realizado la apertura de la misma 39

Figura 4.7. Proceso realizado por CLAHE para limitar el histograma de contraste 40
Figura 4.8. Ejemplo de ecualización adaptativa limitada del histograma de contraste . . 40
Figura 4.9. Ejemplo de imagen tras haber aplicado la detección de bordes de Canny sobre la misma 42
Figura 4.10. Técnica de encontrar contornos y calcular áreas 43
Figura 4.11. Ejemplo de imagen tras haber encontrado y seleccionado los contornos cerrados con la mayor área 45
Figura 4.12. Ejemplo de cálculo de un ángulo de rotación negativo 46
Figura 4.13. Ejemplo de cálculo de un ángulo de rotación positivo 46
Figura 4.14. Ejemplo de imagen tras haber corregido la rotación del código 46
Figura 4.15. Ejemplo de imagen tras haber recortado el código 47
Figura 4.16. Ejemplo de las dos rotaciones 47
Figura 4.17. Ejemplo de imagen tras haber aplicado el operador de Sobel 48
Figura 4.18. Ejemplo de imagen tras haber aplicado la transformada jerárquica de Hough 49
Figura 5.1. Diagrama de bloques del funcionamiento de ZBar 52
Figura 5.2. Salida del programa tras la lectura del código de barras 53
Figura 5.3. Ejemplo de imagen original con código QR 54
Figura 5.4. Salida del programa tras la lectura del código QR 54
Figura 6.1. Histograma comparativo del porcentaje de códigos leídos 58
Figura 6.2. Ejemplos de imágenes con distintas condiciones de iluminación y orientación de los códigos 59
Figura 6.3. Gráfico comparativo del tiempo total de ejecución de las imágenes con buenas condiciones de iluminación y orientaciones del código distintas a la por defecto 63
Figura 6.4. Gráfico comparativo del tiempo total de ejecución de las imágenes con buenas condiciones de iluminación y orientaciones del código 64
Figura 6.5. Gráfico comparativo del tiempo total de ejecución de las imágenes con malas condiciones de iluminación y distintas orientaciones del código 66
Figura 6.6. Gráfico comparativo del tiempo total de ejecución de las imágenes con defectos en el código 67
Figura 6.7. Gráfico comparativo del tiempo total de ejecución de cada imagen en la Raspberry Pi Zero W 68
Figura 6.8. Gráfico comparativo del tiempo total de ejecución de las imágenes con buena/mala iluminación y distintas orientaciones del código QR 70
Figura A.1. Selección del compilador cruzado al crear el proyecto 78
Figura A.2. Configuración del prefijo y la ruta del compilador cruzado 79
Figura A.3. Incluir ruta al directorio donde se encuentran las librerías del toolchain para Raspberry 79
Figura A.4. Añadir al proyecto las librerías del toolchain para Raspberry 80

Índice de tablas

Tabla 3.1. Técnicas empleadas en la fase de preprocesamiento 32
Tabla 3.2. Técnicas empleadas en la fase de localización 32
Tabla 3.3. Técnicas empleadas para realizar las correcciones 33
Tabla 3.4. Técnicas empleadas en la fase de lectura 33
Tabla 4.1. Tiempo del algoritmo previo al definitivo en ejecutar las diferentes etapas de la fase de preprocesamiento 41
Tabla 4.2. Tiempo del algoritmo previo al definitivo en ejecutar las diferentes etapas de la fase de localización 50
Tabla 5.1. Tiempo del algoritmo previo al definitivo en ejecutar la fase de lectura 55
Tabla 6.1. Resultados de tiempo del algoritmo definitivo y de la aplicación directa de ZBar para imágenes con buenas condiciones de iluminación y orientaciones distintas a la de por defecto 63
Tabla 6.2. Resultados de tiempo del algoritmo definitivo y de la aplicación directa de ZBar para imágenes con buenas condiciones de iluminación y orientaciones iguales a la de por defecto 64
Tabla 6.3. Resultados de tiempo del entrenamiento del algoritmo definitivo y de la aplicación directa de ZBar para imágenes con malas condiciones de iluminación y distintas orientaciones 66
Tabla 6.4. Resultados de tiempo del algoritmo definitivo y de la aplicación directa de ZBar para imágenes que poseen un defecto en el código de barras 67
Tabla 6.5. Resultados de tiempo del algoritmo definitivo y de la aplicación directa de ZBar para imágenes con códigos QR y buena iluminación y distintas orientaciones del código y para imágenes con mala iluminación e igualmente diferentes orientaciones 69

1
Introducción

La única cosa peor a empezar algo
y fracasar... es no empezar algo.
Seth Godin
(1960–)

El primer capítulo presenta las razones fundamentales por las que se ha decidido llevar a cabo este proyecto, así como su principal objetivo y herramientas empleadas en su ejecución. Además, aporta al lector una visión general de la organización del documento con el fin de facilitar la lectura del mismo.

La realización del inventario es una actividad básica en el proceso de producción de cualquier empresa. Poseer un registro ordenado y detallado tanto de los artículos disponibles como de los que han sido vendidos es una vía para eliminar problemas como el agotamiento de recursos o la sobreproducción, los cuales conllevarían costes adicionales.

El problema que se presenta aparece a la hora de realizar el inventario de un almacén. Dicha tarea puede resultar bastante tediosa si se realiza de forma manual, incluso con las técnicas actuales en las que se emplean lectores para registrar los códigos de los distintos paquetes. Es por ello que se ha propuesto automatizar el proceso, para lo cual, una de las posibles soluciones es utilizar drones que realicen dicho inventariado, aumentando así la eficiencia y eliminando el error humano. Los drones, al ser capaces de mantener un vuelo estable y controlado de manera autónoma, representan una de las mejores soluciones para conseguir registrar paquetes que se encuentren en zonas de elevada altura y difícil acceso.

Sin embargo, un nuevo problema se plantea a la hora de determinar el mecanismo de extracción y almacenamiento de la información de cada paquete. La solución propuesta con mayor frecuencia actualmente consiste en un lector de código de barras y QR instalado en el dron. Dicha solución, como se comprobará más adelante, no es la más óptima, por lo que en este proyecto se plantea el uso de visión artificial como vía para automatizar el proceso de inventariado mediante drones.

La visión artificial pretende conseguir que un procesador sea capaz de identificar imágenes del mundo que nos rodea y actuar en consecuencia, al igual que lo haría un ser humano.

Es ampliamente utilizada en la actualidad por muchas tecnologías debido a que permite implementar una gran variedad de funciones como reconstrucción de escenas, detección de eventos y reconocimiento de objetos.

Por lo tanto, dada una imagen, el sistema a desarrollar debe ser capaz de identificar la posición del código, decodificarlo y comprobar si este es erróneo o no. Este sistema, a su vez, implementado sobre el dron haría que la problemática de la realización del inventariado del almacén quedara resuelta.

1.1. Motivación

La solución de realizar el inventariado de los artículos mediante drones en lugar de personas resulta más eficiente por distintas razones. En primer lugar, el dispositivo puede alcanzar zonas de difícil acceso como puntos muy elevados de pilas de paquetes sin poner en riesgo la integridad física del trabajador. Además, se elimina el error humano a la hora de realizar la lectura de los códigos.

Por otro lado, para llevar a cabo el inventariado de un almacén empleando un dron, típicamente se suele instalar un lector de código de barras sobre el mismo. No obstante, esto plantea diversos inconvenientes. En primer lugar, dicho sensor genera un peso adicional, con lo que el vuelo del dron se ve entorpecido. Igualmente, el lector disminuye la autonomía del dron al aumentar el consumo de batería.

Cabe plantearse, por tanto, la existencia de una solución alternativa más óptima para realizar la localización y lectura de los códigos. Una de las opciones es emplear visión artificial. Esta tecnología aprovecha la cámara que suele estar ya instalada en el dron con el fin de ayudar al operador a controlarlo para tomar imágenes de los paquetes, y procesa los códigos de identificación de inventarios mediante un algoritmo implementado sobre *hardware* ligero y de bajo coste. Dicho *hardware* podría ser un microcontrolador tipo Raspberry Pi Zero W [Wwwaf], mediante el cual, además, sería posible enviar el resultado de la decodificación a una estación base en tierra de forma inalámbrica.

Además, es necesario determinar cuáles de las técnicas existentes de visión artificial aplicada a la lectura de códigos es más adecuada para esta práctica y mejorar otra serie de técnicas con el fin de adaptarlas a las necesidades planteadas.

De esta forma, se consiguen eliminar los problemas que plantea tanto la automatización del proceso de realización del inventariado como la instalación del lector de código de barras en el dron, supliendo las necesidades a la vez que se mejora el proceso.

1.2. Objetivo

El objetivo principal del proyecto es la descripción e implementación de un algoritmo que permita localizar y leer códigos de barras y QR a partir de imágenes tomadas con diferentes orientaciones y condiciones de iluminación. Dicho algoritmo debe ser capaz de ejecutarse en tiempo real, es decir, en un tiempo razonable que no comprometa el funcionamiento del dron.

Como ya se ha explicado con anterioridad en el documento, con este software se pretende hacer más eficiente el proceso de inventariado de almacenes que se realizan mediante drones, reduciendo, de esta manera, tiempo y coste.

1.3. Herramientas

Con el fin de llevar a cabo el proyecto propuesto se ha empleado como lenguaje de programación C++ y la librería para visión artificial OpenCV [Wwwu] la cual proporciona numerosas funciones para el procesamiento de imágenes. La elección de C++ se debe, por un lado, a que es un lenguaje compilado y, por tanto, es más rápido que los lenguajes interpretados ya que el código se traduce a un archivo que puede ejecutarse tantas veces como sea necesario sin tener que repetir el proceso de traducción. Así, el tiempo entre ejecución y ejecución es mínimo. Por el contrario, los lenguajes interpretados son más lentos al necesitar ser traducidos a lenguaje máquina con cada ejecución [Wwwd]. Por otro lado, OpenCV está programada en C++ y para hacerla compatible con otros lenguajes como Python posee una capa de software que traduce el código programado por el usuario en Python a C++ [Wwwau]. Empleando C++ de nuevo se consigue ahorrar tiempo al evitar esa traducción, haciendo el proceso más rápido y eficiente.

Además, se ha hecho uso de la librería ZBar [Wwwbc] que permite la lectura y decodificación de diferentes tipos de códigos de barras y QR. El algoritmo ha sido implementado sobre el *hardware* Raspberry Pi Zero W.

1.4. Organización del documento

La memoria se articula en ocho capítulos siendo el presente el primero de ellos, el cual introduce el problema tratado en el proyecto. El siguiente recoge una clasificación de las diferentes tipologías de códigos de barras y QR existentes. El tercero contiene la revisión de la literatura existente sobre visión artificial aplicada a la lectura de dichos códigos. Los dos siguientes capítulos explican detalladamente en qué consiste el algoritmo desarrollado. Tras esto, en el capítulo seis se realiza un análisis de los resultados obtenidos tras las pruebas efectuadas sobre dicho algoritmo y en el siete se exponen las conclusiones extraídas del trabajo efectuado. Finalmente, se comentan algunos de los posibles desarrollos a elaborar en un futuro.

Además, se incluye un anexo que recoge los diferentes procedimientos llevados a cabo para la instalación de las herramientas empleadas y, por último, la bibliografía con las referencias citadas a lo largo del documento.

2

Clasificación y estructura de los códigos

La ciencia es la clasificación
sistemática de la experiencia.
George Henry Lewes
(1817–1878)

En el siguiente capítulo se describen los distintos tipos de códigos de barras y QR más usados, además del proceso que emplea cada uno de ellos para codificar la información. De esta forma, se pretende mejorar la comprensión de la estructura de los códigos con el fin de facilitar la interpretación de las posteriores cuestiones tratadas en el documento.

En primer lugar, cabe destacar que, generalmente, los códigos empleados para realizar el inventariado de los productos se clasifican en dos grandes categorías: lineales y bidimensionales. En el primer grupo se encajan aquellos códigos que mapean la información en un formato unidimensional, basándose en la anchura de las barras, de tal forma que quedaría codificada en una especie de vector. Por otro lado, los códigos bidimensionales emplean la estructura de una matriz para almacenar la información por lo que esta quedaría codificada en varias líneas de puntos [Wwwap].

Los códigos lineales empleados con mayor frecuencia en la actualidad son los siguientes:

- **EAN**: cuyas siglas se corresponden con European Article Number y es el empleado para etiquetar todos los productos comercializados principalmente en el mercado europeo, aunque también es usado en otros países del resto del mundo. El más común es el EAN-13, formado por 13 dígitos y con una estructura fija dividida en cuatro partes diferenciadas. Únicamente permite codificar caracteres numéricos [Wwwas], [Wwwq].

  En la Figura 2.1, se pueden apreciar las distintas partes del código como las zonas de silencio (1), las cuales sirven para delimitar el comienzo y fin del código de barras y son imprescindibles para los lectores electrónicos que se poseen en numerosos

Figura 2.1. Estructura del código de barras EAN-13. Modificado de [Wwwad].

establecimientos. El resto de grupos contienen información que permite clasificar el artículo por país de procedencia (2), empresa productora (3) e identificador individual del artículo (4). El dígito de control (5) permite realizar una comprobación de que la posterior decodificación se ha realizado de forma correcta ya que sigue un algoritmo específico para su cálculo, el cual será detallado en secciones posteriores [Wwwas].

- **Code 128**: es un código muy utilizado para logística y la paquetería, con el fin de etiquetar los productos. Surge ante la necesidad de poseer mayor cantidad de información en un espacio más reducido. Permite codificar tanto caracteres numéricos como alfanuméricos y todos los caracteres pertenecientes a la tabla ASCII, incluyendo los de control. Posee una longitud variable, de hasta 106 caracteres diferentes [Wwwam], [Wwws].

Figura 2.2. Estructura del código de barras Code 128 [Wwwt].

La estructura del código consta de seis partes mostradas en la Figura 2.2: a la izquierda del código una zona de silencio de longitud equivalente a dos caracteres y, a continuación, el carácter de inicio (determinando el tipo de caracteres que se representan). Seguidamente, un número variable de caracteres ASCII (representando la información del producto), el dígito o carácter de control, el carácter de parada (carácter auxiliar que indica el final del código) y, finalmente, a la derecha, otra zona de silencio equivalente a dos caracteres. Los caracteres especiales incluidos en la zona de los datos, sirven para definir, junto con el carácter de inicio, la simbología empleada y como separador de campos de longitud variable [Wwwam], [Wwwt].

- **Code 39**: posee una longitud variable permitiendo codificar hasta un total de 43 caracteres entre los que se incluyen números, letras mayúsculas y algunos caracteres especiales. Uno de sus inconvenientes es que necesita más espacio para codificar la información que otros tipos de código, como por ejemplo el Code 128 (explicado anteriormente). Con esto resulta más difícil el etiquetado de objetos de tamaño reducido. Cada carácter consta de cinco barras y cinco espacios [Wwwq], [Wwwan].

Figura 2.3. Ejemplo de un código de barras tipo Code 39. Modificado de [Wwwt].

Un ejemplo de código de barras Code 39 se muestra en la Figura 2.3 en la que se puede apreciar la estructura del mismo. Como se observa, posee un carácter de inicio y otro de final, situados a los extremos del código e identificados con la codificación correspondiente al asterisco (*). Entre estos caracteres se sitúa la información, que puede incluir caracteres especiales como el espacio en este caso. Tras la información se añade el código de auto-verificación (en este caso 28) con lo que no es necesario incluir un carácter de control. Dicho código se calcula sumando el valor de control de cada carácter del código (exceptuando los de inicio y fin) y dividiéndolo entre 43. El resto de la división será el código de auto-verificación [Wwwt].

- **Code 93**: fue desarrollado con el fin de aumentar la densidad de información del Code 39, permitiendo codificar hasta 47 caracteres alfanuméricos como máximo, con una longitud variable. Es usado por el servicio postal de Canadá, en sectores militares y automotrices, entre otros. Cada carácter consta de tres barras y tres espacios [Wwwq], [Wwwao], [Wwwg], [Wwwh].

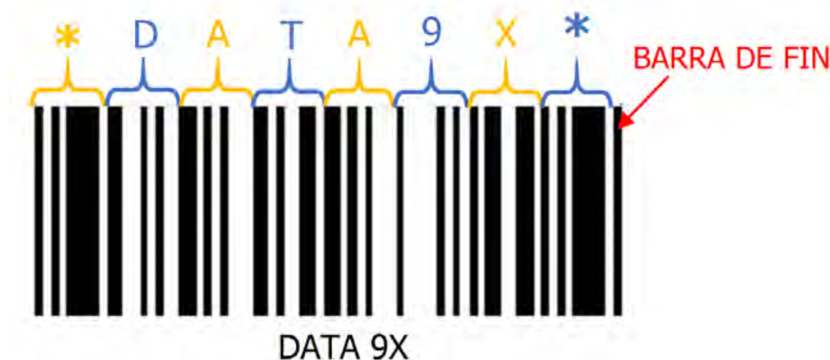


Figura 2.4. Ejemplo de un código de barras tipo Code 93. Modificado de [Wwwg].

La estructura del Code 93 puede apreciarse en la Figura 2.4 y está formada por: un carácter de comienzo (*), el mensaje codificado, un carácter de fin (*) y una única barra de fin. Además se pueden añadir caracteres de auto-verificación del código, al igual que se hacía en el Code 39, que irían situados justo antes del carácter de fin [Wwwg].

Por otro lado, los códigos bidimensionales más utilizados serían:

- **PDF 417**: es un formato de código de barras formado por varios códigos lineales apilados mediante un algoritmo con lo que puede contener mayor cantidad de información. Se emplea principalmente en tarjetas de transporte e identificación. Consta de 3 a 90 filas donde cada una de ellas es un pequeño código de barras lineal [Wwws], [Wwwav].

Figura 2.5. Estructura del código PDF 417. Modificado de [Wwwav].

En la Figura 2.5 se observa la estructura del PDF 417. De esta forma, cada una de las filas consta de una zona de silencio, un patrón de inicio con información que identifica el formato de la codificación, un indicador de fila izquierda que contiene información sobre la fila, las palabras del código que representan la información, un indicador de fila derecha con más información sobre la fila, un patrón de parada para simbolizar el final de la fila y otra zona de silencio [Wwwav].

- **Código QR**: está formado por una matriz de puntos diseñada para un escaneo rápido de la información. Puede contener distintos tipos de información como enlaces a páginas web, un mapa de localización, un correo electrónico o un perfil de una red social. Es muy empleado en Japón y son de la forma que se muestra en la Figura 2.6 [Wwwaq], [Wwwa].

Figura 2.6. Ejemplo de código QR [Wwwax].

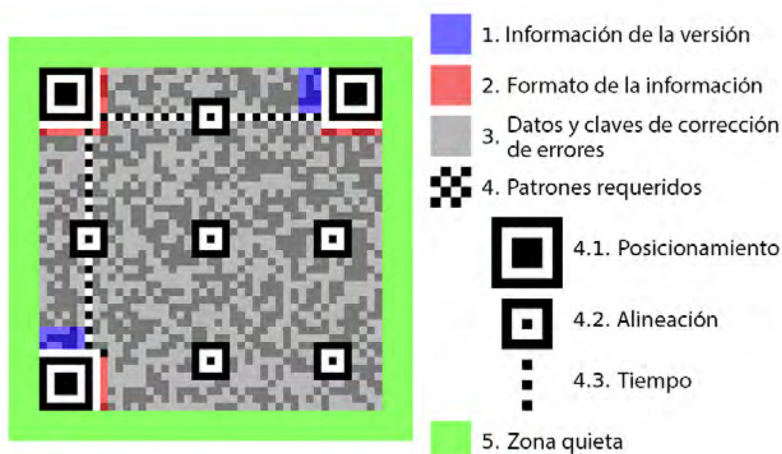


Figura 2.7. Estructura del código QR. Modificado de [Wwwa].

En la Figura 2.7 se observa la estructura del código QR, en la cual se pueden distinguir diferentes tipos de información como la versión (1), formato (2), claves de corrección

de errores (3) y patrones (4). Cabe destacar el patrón de posicionamiento (4.1), puesto que es clave para detectar la localización y orientación del código. Además, el patrón de alineación (4.2) facilita la lectura del código ayudando a determinar con mayor precisión la posición del código y el patrón de tiempo (4.3) permite detectar el ancho del código. Finalmente, el código QR al igual que el de barras, posee una zona de silencio (5) que lo delimita pero no posee ningún tipo de información [Wwwa].

3 Estado del arte

Solos podemos hacer muy poco;
unidos podemos hacer mucho.
Hellen Keller
(1880–1968)

El tercer capítulo expone el procedimiento que el algoritmo a desarrollar deberá seguir para detectar y leer los códigos. Además, se realiza una revisión de la literatura publicada por diversos investigadores sobre dicha cuestión, con el fin de analizar las distintas técnicas empleadas hasta la fecha.

El esquema presentado en la Figura 3.1 muestra el proceso general para la obtención del código de barras o QR decodificado dada una imagen capturada por una cámara. El algoritmo a desarrollar consta de tres fases ejecutadas de manera secuencial: preprocesamiento, localización y lectura. Además, después de la fase de localización y previamente a la fase de lectura pueden aplicarse una serie de correcciones a la imagen de forma opcional, solo en caso de que sean necesarias. A continuación, se desarrollan cada una de dichas fases por separado con el fin de analizar las técnicas empleadas en su ejecución.

Cabe destacar que, antes de aplicar cualquiera de los algoritmos de preprocesamiento, es necesario tener en cuenta la calibración de la cámara. Debido a que las lentes introducen distorsión, los bordes rectos de los objetos, pueden aparecer curvos y los colores con tonalidades diferentes a las originales. La calibración permite obtener una mayor fidelidad de forma que la imagen tomada reproduzca de manera precisa la realidad. En cualquier aplicación de visión artificial, el primer paso debe ser, siempre que sea posible, calibrar la cámara. Para ello, de forma general, podría emplearse una rejilla de calibración (consistente de un conjunto de puntos) con coordenadas conocidas del mundo real y de la imagen, a partir de las cuales se pueden obtener los coeficientes de distorsión y los parámetros intrínsecos (la distancia focal, el centro óptico y el coeficiente de inclinación) de la cámara [Wwwv], [Wwwe].

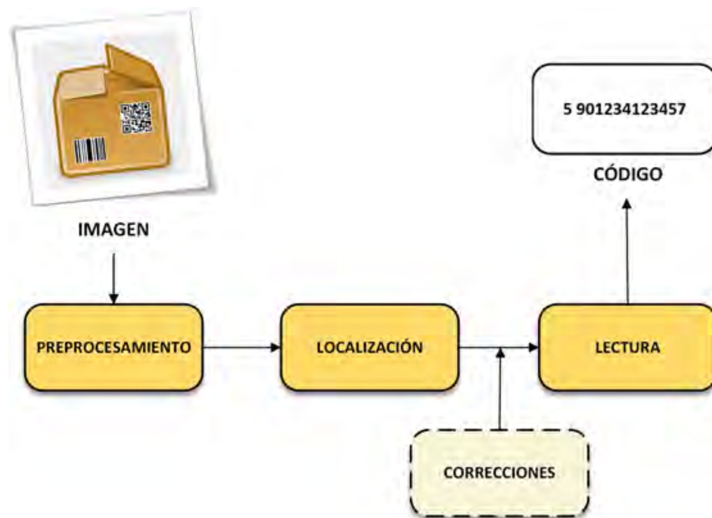


Figura 3.1. Diagrama de bloques general del sistema.

3.1. Preprocesamiento

La fase de preprocesamiento consiste en realizar un primer análisis de la imagen y aplicarle una serie de modificaciones de tal forma que se consiga filtrar el ruido y eliminar la información que no va a ser de utilidad para las fases posteriores. Además, pueden corregirse ciertos defectos que existan en la posición de la imagen, todo ello con el fin de mejorarla para las siguientes etapas.

Los métodos de preprocesamiento pueden aplicarse indistintamente a códigos de barras o QR ya que afectan a la imagen como tal en su conjunto. La Tabla 3.1 presenta diferentes técnicas empleadas en esta fase y las cuales se dividen en siete grupos. Los dos métodos de los apartados 3.1.1. y 3.1.2. suelen ser los primeros en aplicarse, mientras que el resto de métodos normalmente son empleados tras los otros dos.

3.1.1. Escalado de grises

Consiste en transformar la imagen capturada en escala RGB a escala de grises. De esta forma, queda representada únicamente por su luminancia (en un rango de valores que se extiende desde el blanco al negro).

Esta técnica es el primer paso por muchas referencias usadas [Ohb+04], [CH05], [GZ11], [Liu+08], [FK07], [LL06], [LZ10] y [Liu+10]. Destaca [FK07] que, tras aplicar primero el escalado de grises globalmente a toda la imagen, realiza este paso de forma local. Para ello, divide primero la imagen en cuatro regiones iguales (puesto que se obtienen mejores resultados que al emplear una única región o dividirla en dieciséis regiones), y, posteriormente, calcula el histograma de grises para cada una de ellas. De esta forma, se consiguen adaptar más eficientemente los niveles de gris a cada zona con el fin de que el histograma abarque todo el rango de luminosidad de cada una de ellas. Se considera útil realizar el escalado de grises en el caso que nos concierne ya que se elimina la información relativa al color, innecesaria para la lectura de los códigos, que en la práctica totalidad de los casos son negros sobre fondo blanco.

3.1.2. Binarización

La binarización reduce el rango de valores que toman los píxeles de una imagen a únicamente dos, 1 y 0, blanco y negro respectivamente. Para ello se suelen emplear métodos basados en la definición de un umbral. Así, dependiendo de si un valor se encuentra por encima o por debajo de dicho umbral, se le asigna 1 o 0.

Para códigos de barras, [Ohb+04] emplea el **Método del Umbral Global** en el cual se define un único umbral para toda la imagen. Además [FK07], igualmente [Ohb+04] y [GZ11], aplican el **Método del Umbral Adaptativo Local** para códigos QR, con lo que, en primer lugar, la imagen se divide en bloques de píxeles de igual tamaño y, tras esto, establece un umbral para cada bloque. En cuanto a la división, [Ohb+04] emplea una rejilla de nueve cuadrados de 60x60 píxeles que aplica al centro de la imagen (cuyo tamaño no se especifica), mientras que [FK07] divide la imagen en cuatro regiones. El umbral para cada bloque según [Ohb+04] se calcula tomando el valor de la mediana del histograma para cada área y, después, se define como umbral común para todas las áreas el valor mínimo de los nueve umbrales calculados. [GZ11] y [FK07] no detallan específicamente cómo realizan este proceso. Existe una variante llamada **Método del Umbral Adaptativo Multinivel** que engloba las dos técnicas anteriores y es empleada, por ejemplo, por [Liu+08]. Finalmente, una alternativa sería emplear el **Método de Otsu** basado en la varianza de los tonos de gris, como se hace en [CH05] y [GZ11].

Una buena solución sería, como en [GZ11], diferenciar entre dos casos: cuando existan condiciones normales de luz, emplear el Método del Umbral Global y el método del Umbral Adaptativo Local en condiciones de luz desigual.

3.1.3. Perfilado del contorno

Consiste en perfilar los bordes de las formas geométricas que configuran los códigos puesto que en algunas imágenes pueden verse borrosos. [YS07] simplemente trata de localizar todos los contornos cerrados de la imagen mientras que [LZ10] aplica el operador de Laplace para atenuar los componentes de alta frecuencia.

3.1.4. Filtrado de ruido

Permite eliminar el ruido de la imagen con el fin de suprimir ciertos componentes que impiden apreciar el código con claridad y que no son relevantes para su lectura. Existen diferentes técnicas de filtrado:

- **Dilatación**: es un operador morfológico cuya función es expandir las formas de los objetos que aparecen en las imágenes por medio de un elemento estructural que ha sido previamente elegido. Tal como se expresa en la Figura 3.2 , el proceso para realizar la dilatación de una imagen sería, en primer lugar, seleccionar el elemento estructural que va a emplearse (1). Tras esto, se iría pasando dicho elemento por toda la imagen y cada vez que alguno de los píxeles de dicho elemento toque con un píxel del borde del objeto, se añade un píxel a dicho borde (2). Pueden darse casos en los que el elemento estructural no toque al objeto, con lo que no se añadirá ningún pixel (3). De esta forma, la imagen original queda expandida (4).

  En [Ohb+04] esta técnica se emplea con el fin de rellenar píxeles (agujeros) con lo que se consigue espesar las diferentes formas del código.

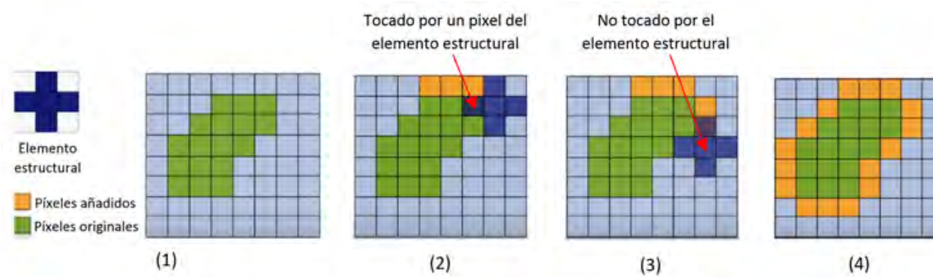


Figura 3.2. Proceso de dilatación de una imagen. (1) Imagen original y elemento estructural. (2) Elemento estructural toca la imagen. (3) Elemento estructural no toca la imagen. (4) Imagen resultante. Modificado de [Wwwaz].

- **Apertura y cierre**: al igual que la dilatación, son operadores morfológicos. Antes de explicar estas técnicas, cabe destacar que la **erosión** es el operador contrario a la dilatación, con lo que encoge el objeto, haciéndolo más delgado. Para ello se sigue el procedimiento idéntico al explicado en la Figura 3.2 solo que eliminando píxeles del objeto. La **apertura**, como se muestra en la Figura 3.3, consiste en una dilatación aplicada sobre una erosión, de tal forma que se consiguen eliminar los píxeles del fondo que no pertenecen a los objetos (píxeles de ruido). El **cierre**, por otro lado, se trata de realizar una erosión de una dilatación, con lo que, como se muestra en la Figura 3.4, se eliminan agujeros o huecos que existan en los objetos de la imagen. [Liu+08] y [FK07] aplican las técnicas estándar de apertura y cierre con el fin de eliminar el ruido.

- **Filtro de la mediana**: se selecciona el valor de la mediana para un área de píxeles vecinos y los píxeles que se encuentren fuera de este valor son eliminados puesto que se identifican como ruido, de acuerdo a [Sze11], libro que define una serie de algoritmos y técnicas de visión artificial. Aplicado en [LZ10].

- **Muestreo piramidal**: es una representación multi-escala en el que se aumenta y se disminuye la escala de la imagen con el fin de eliminar ruido. Es usado por [Liu+10].

- **Filtro gaussiano**: simula una distribución gaussiana, donde el valor máximo aparece en el píxel central y disminuye hacia los extremos más rápidamente cuanto mayor sea el parámetro de desviación típica. Se emplea con el fin de suavizar las imágenes y eliminar el ruido. Es aplicado por [Lu+06] en uno de los casos del algoritmo desarrollado para el cual el código posee perturbaciones alrededor de los bordes de las barras o ruido sal y pimienta (característico por cubrir toda la imagen con una serie de píxeles blancos y negros dispersos). Resulta interesante el empleo de un filtro gaussiano en este caso puesto que el ruido sal y pimienta se elimina mejor con un filtro de la mediana.

3.1.5. Recortar

Consiste en recortar la imagen para eliminar el fondo que sea innecesario reduciendo su tamaño, y, por tanto, el tiempo de proceso. Únicamente [FK07] recorta la imagen y, en términos generales, no sería muy recomendable realizar este paso a menos que se haya localizado la posición del código previamente puesto que se podría eliminar parte de la información relevante.

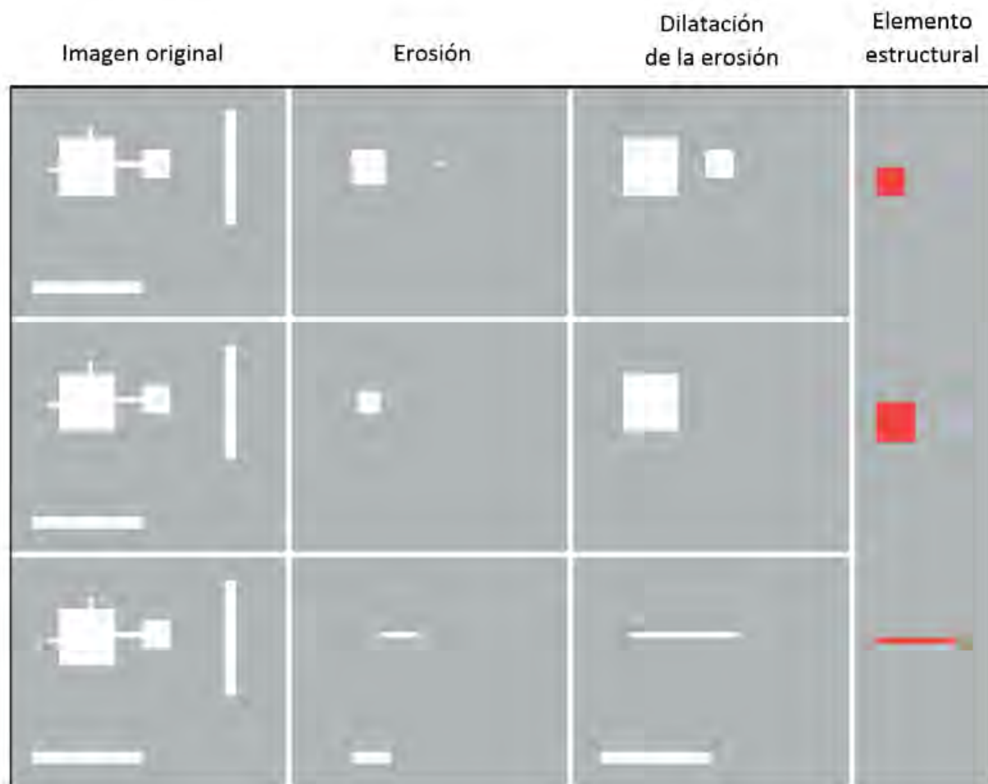


Figura 3.3. Ejemplos de aplicación de la técnica de apertura. Modificado de [Cor11].

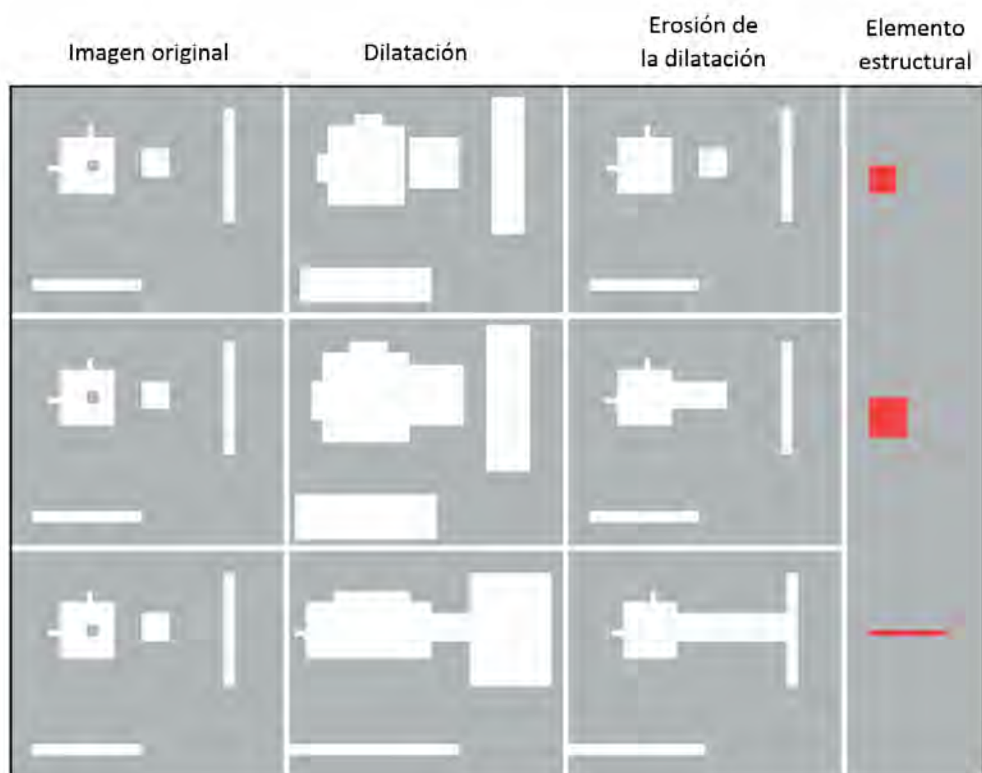


Figura 3.4. Ejemplos de aplicación de la técnica de cierre. Modificado de [Cor11].

3.1.6. Redimensionar

Redimensionar es una técnica que forma parte de las transformaciones geométricas aplicadas a las imágenes [Sze11] y que consiste en modificar las dimensiones de la misma. En [Ohb+04] la imagen se reescala reduciendo a la mitad las coordenadas del eje X y del eje Y con el fin de reducir los costes de cálculo en la siguiente fase.

3.1.7. Normalizar

También conocido como el proceso de expansión de contraste, aporta mayor definición a los distintos elementos de la imagen. Modifica el histograma de la imagen de tal forma que todos los niveles de la escala de grises sean empleados.

Como última técnica de preprocesamiento a aplicar, [GZ11] sugiere realizar la normalización de la imagen. Para ello, en primer lugar, se comprueba el número de versión del código QR de acuerdo al estándar. A continuación, lo divide en celdas cuadradas remuestreando el centro de cada una de ellas y finalmente, las normaliza.

3.2. Localización

La fase de localización se basa en ubicar la posición del código dentro de la propia imagen. Cabe destacar que esta fase, a diferencia de la etapa anterior, se ejecutará de forma distinta dependiendo de si se trata de un código de barras o un código QR.

La Tabla 3.2 presenta diferentes técnicas empleadas en esta fase. Para localizar códigos de barras se poseen ocho técnicas mientras que para los códigos QR se poseen cuatro.

3.2.1. Localización del código de barras

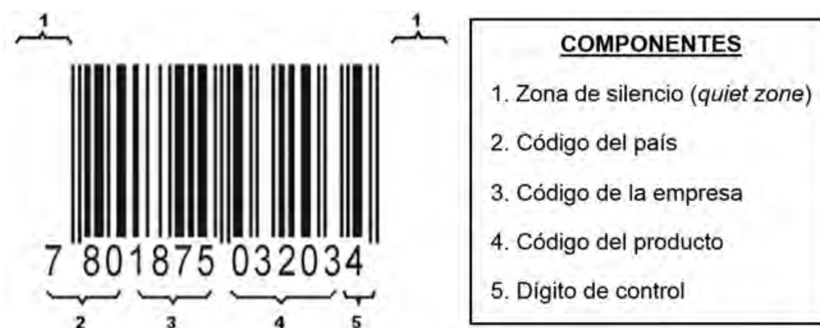


Figura 3.5. Estructura del código de barras EAN-13. Modificado de [Wwwad].

3.2.1.1. Separar componentes conectados

Los componentes conectados son regiones definidas de píxeles adyacentes con el mismo valor [Sze11]. Resultan de gran utilidad para detectar letras o, en este caso, formas determinadas como rectángulos. Tanto [YS07] como [CH05] realizan, en primer lugar, una detección de componentes conectados para, posteriormente, agruparlos y clasificarlos de tal forma que se poseen individualmente cada una de las barras que forman el código. De esta forma se consigue extraer el código del resto de la imagen. Si se emplease este método resultaría útil, además, dividir los componentes en grupos de forma similar a la mostrada en la Figura 3.5. Es decir, separar las barras en los conjuntos que representen el código del país (2), de la empresa (3),

del producto (4) y, por último, el dígito de control (5). Así, se facilitaría su posterior lectura y decodificación en diferentes partes.

3.2.1.2. Líneas de escaneo en diferentes direcciones

Consiste en trazar líneas rectas a través de la imagen en diferentes direcciones y, para cada línea, se obtiene el nivel de escala de grises de los píxeles que atraviesa. Este proceso se muestra en la Figura 3.6, donde, en primer lugar, se posee la imagen original (1) y después se trazan las líneas de escaneo en diferentes direcciones, horizontal y diagonal con 45° (2). De esta forma, si se detectan zonas con un patrón de píxeles negros y blancos alternos en alguna de las líneas, se sabrá dónde se encuentra el código de barras. Dichos patrones pueden apreciarse en los histogramas calculados de algunas líneas escaneadas en (3) y (4). Esta técnica es aplicada por [Lu+06].

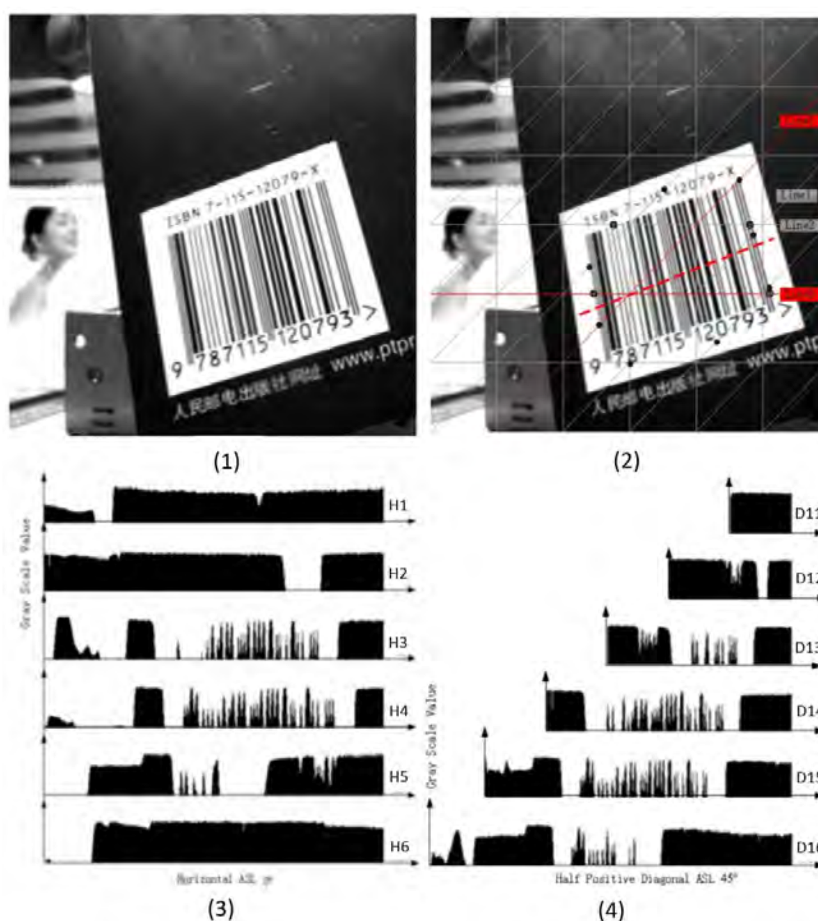


Figura 3.6. Técnica de líneas de escaneo en diferentes direcciones. (1) Imagen original. (2) Imagen con líneas de escaneo. (3) Histogramas de escala de grises de las líneas horizontales de (2). (4) Histograma de escala de grises de las líneas diagonales (45°) de (2). Modificado de [Lu+06].

3.2.1.3. Definir una matriz de similitud

Es una estructura que contiene información sobre los pares de líneas que han sido empleadas en el escaneo de la imagen. De acuerdo a [Lu+06] las variables contenidas en la matriz son: las dos líneas con la mayor longitud de patrón de código de barras encontrado, dicha longitud, posición y valor del vector de la escala de grises. Estos parámetros ayudarán a conocer, además, la calidad de la imagen con el fin de determinar las próximas técnicas a aplicar. De acuerdo a la

Figura 3.6, en la imagen (3) las dos líneas que se incluirían en la matriz de similitud serían H3 y H4 puesto que tienen el patrón más largo, y en la imagen (4) serían D14 y D15.

3.2.1.4. Transformada Jerárquica de Hough (HHT)

Es una técnica de extracción de características, muy usada para detectar líneas, aunque puede usarse para detectar otros tipos de formas como círculos y elipses. Dicha transformada trabaja de la siguiente forma: en primer lugar, define un punto y, para dicho punto traza todas las líneas que lo atraviesan. Esto se hace mediante coordenadas polares (donde $\boldsymbol{r}$ es la distancia del origen a la línea y θ es el ángulo), simplemente rotando una línea que atraviese el punto desde 0° a 180°: a medida que θ cambia, $\boldsymbol{r}$ cambiará también. Estos cambios de ángulo y distancia generan sinusoidales, de la forma que se muestra en la Figura 3.7. En (1) y (3) se puede apreciar el procedimiento explicado anteriormente en el que se van variando $\boldsymbol{r}$ y θ, de forma que se tracen todas las rectas que pasan por el punto de color azul. En (2) se aprecia el punto exacto en el que las tres sinusoidales se cortan, indicando que éstas están alineadas y dichos valores de $\boldsymbol{r}$ y θ definen la recta que une los tres puntos. Esto puede apreciarse mejor en la Figura 3.8, donde los puntos de corte de las sinusoidales entre ellas indican que existe alineación entre los puntos verde, rojo y amarillo y dan la recta que los une.

De esta forma, se consigue localizar un grupo de líneas rectas paralelas que forman el código de barras. En [Muñ+99] los valores recogidos por la transformada se almacenan en una matriz de acumulación tras lo cual se fija el ángulo en el que considera que se posee la mejor dirección para analizar el código de barras. Se extrae el vector generado en dicha dirección y se representa, lo que permitirá decodificarlo en la fase posterior.

3.2.1.5. Algoritmo de selección de candidato

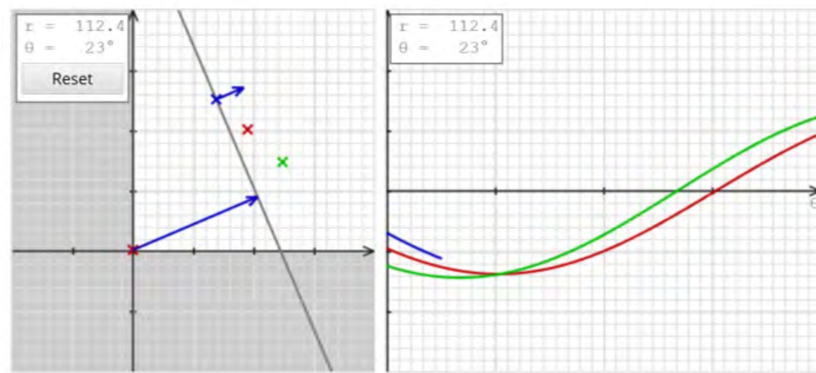
Consiste en, una vez que se han separado los componentes conectados, aplicar una serie de condiciones para seleccionar las regiones del código de barras que mejor las satisfagan. Dichas regiones serán las que pasarán a ser decodificadas. En [YS07] se proponen las siguientes desigualdades:

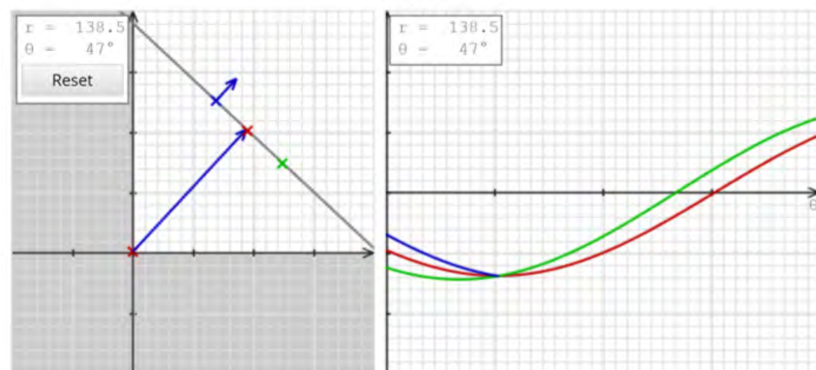
- *Area >BC_MIN_AREA*
- *BC_MIN_RATIO <= height/width <= BC_MAX_RATIO*
- *Area >= max (area of the candidates)*

En las cuales, los valores de *BC_MIN_AREA*, *BC_MIN_RATIO* y *BC_MAX_RATIO* serán determinados después de haber ensayado con múltiples imágenes. Con este método pueden no obtenerse los resultados óptimos ya que las condiciones son demasiado generales y, para casos específicos de fotos con baja calidad, podría no ser efectivo. Además, calcular los valores especificados anteriormente conllevaría un mayor nivel de complejidad y de tiempo de procesamiento al tratarse de un método iterativo que necesita un elevado número de repeticiones para llevarse a cabo.

3.2.1.6. Búsqueda en espiral

Este método se basa en escanear trazando una espiral desde el centro de la imagen hacia fuera como se muestra en la Figura 3.9 (2). En [Ohb+04], esto permite encontrar una línea recta perteneciente al código de barras y perpendicular a la cual traza otras líneas que atraviesan todo el código de barras. De esta forma puede detectar tanto el resto de barras que componen





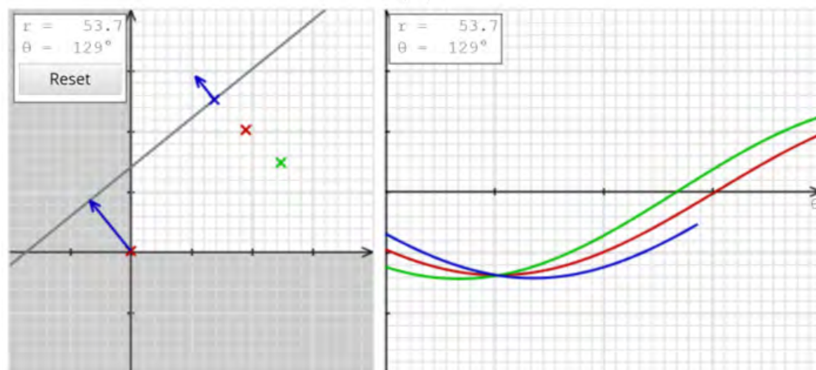


Figura 3.7. Procedimiento de cálculo de HHT. (1) Recta que pasa por el punto azul. (2) Recta que une los tres puntos. (3) Recta que pasa por el punto azul. Modificado de [Wwwr].

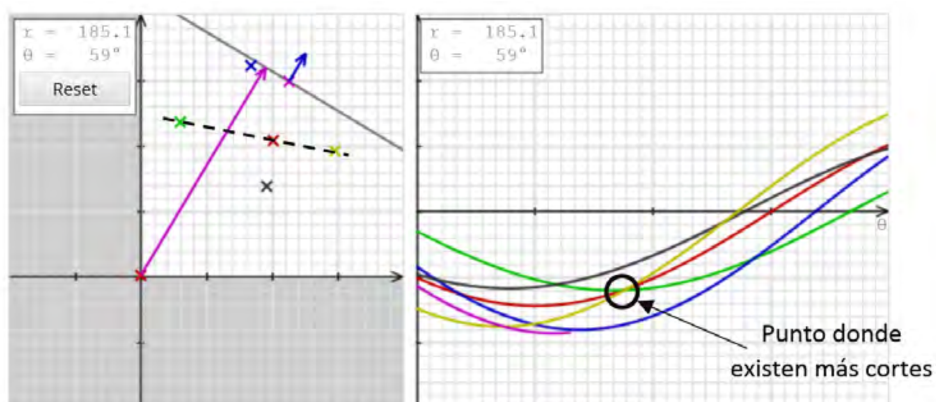


Figura 3.8. Detección de líneas rectas mediante HHT. Modificado de [Wwwr].

Figura 3.9. Método de búsqueda en espiral. (1) Imagen original binarizada. (2) Trazado de la espiral. (3) Detección de la longitud del código mediante intersecciones entre líneas del algoritmo y reales del código [Ohb+04].

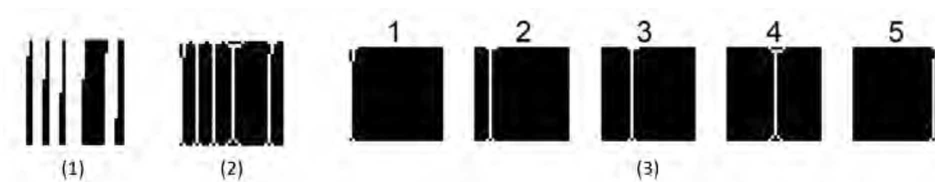


Figura 3.10. Método de esqueletización. (1) Imagen original binarizada. (2) Imagen esqueletizada. (3) Separación de los cinco componentes que forman la imagen. Modificado de [CH05].

el código como su longitud (anchura del código), observando las intersecciones entre las líneas trazadas por el algoritmo y las reales de la imagen (3).

3.2.1.7. Esqueletización

La esqueletización de imágenes binarias reduce los objetos a representaciones de un pixel de anchura. Esto puede resultar útil para la extracción de características y para representar la topología de dichos objetos. [CH05] detecta mediante este método cada uno de los componentes, es decir, las barras, que aparecen en los bloques en los que ha dividido la imagen del código original. El procedimiento se muestra en la Figura 3.10. Una vez binarizada la imagen original (1), se aplica el método de esqueletización, reduciendo la anchura de las barras a un píxel (2), de esta forma es fácil identificar el patrón de las mismas y separar los componentes (3) del código, como ya se indicó previamente en el punto 3.2.1.1.

3.2.1.8. Detección de bordes

Consiste en encontrar los límites de los objetos, los cuales suelen ocurrir, de forma cualitativa, entre regiones de diferentes intensidades, colores o texturas. Los artículos estudiados emplean dos métodos independientes:

- **Basado en vertical (*Vertical-based edge detection operator*)**: es un operador que emplea diez píxeles vecinos para detectar la posición de un píxel. De esta forma, para que un píxel sea considerado como un borde, debe de haber al menos cuatro píxeles blancos y cuatro negros a su alrededor [LZ10].

- **Canny**: algoritmo que aplica varias etapas para la detección de los bordes. En primer lugar, elimina el ruido (aplicando, por ejemplo, un filtro gaussiano) y, tras esto, obtiene el gradiente de cada píxel. A continuación, reduce el ancho de los bordes y, basándose en los valores del gradiente determina los bordes potenciales. Finalmente, se asegura de que no existan errores revisando los píxeles de los bordes. [Liu+10] usa esta técnica que resultaría de gran utilidad en el caso de los códigos de barras puesto que el patrón que forman los bordes de las barras destacaría en la imagen, con lo que sería más sencillo

reconocerlo (una serie de rectángulos alargados y próximos). Un ejemplo del resultado tras haber aplicado el filtro se muestra en la Figura 3.11.

Figura 3.11. Ejemplo de detección de bordes mediante Canny. (1) Imagen original. (2) Imagen tras haber aplicado la técnica de detección de bordes Canny. Modificado de [Wwww].

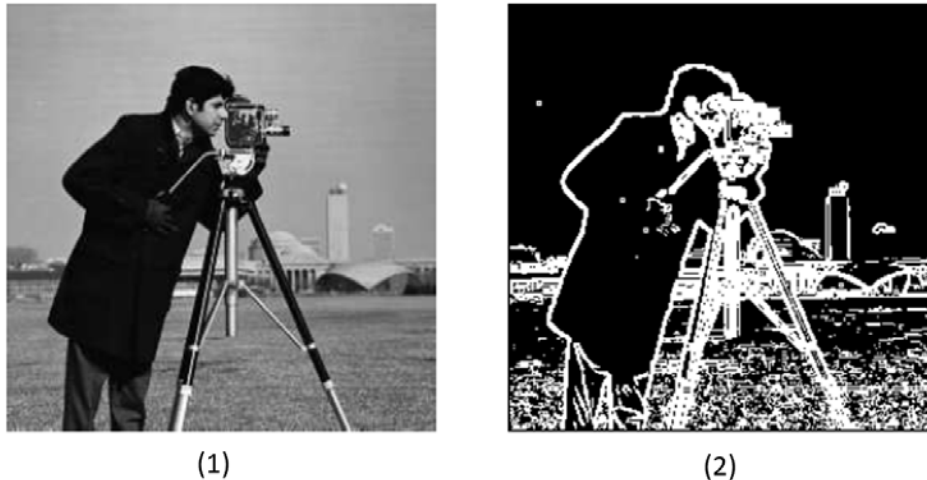


Figura 3.12. Ejemplo de detección de bordes mediante Sobel. (1) Imagen original. (2) Imagen tras haber aplicado la técnica de detección de bordes del operador de Sobel [Wwwo].

Otra opción si se desean reducir los tiempos de procesamiento, es emplear el operador de **Sobel**. Dicho algoritmo es nombrado en ciertos estudios como [GZ11] y [LZ10] pero no se aplica como tal. Hace una aproximación del gradiente de la función de intensidad de forma que el resultado muestra cómo de abruptamente o suavemente cambia una imagen en cada punto y, por lo tanto, la probabilidad de que represente un borde. En la Figura 3.12 se muestra un ejemplo de una imagen que aplica esta técnica.

3.2.1.9. Búsqueda de patrón

Se basa en buscar en los estándares publicados para códigos de barras patrones de inicio y fin tras lo cual aplica un algoritmo específico de búsqueda de dichas combinaciones de elementos. La búsqueda se realiza escaneando la imagen mediante dos líneas paralelas a las barras que forman el código [Liu+10].

3.2.1.10. Extracción de la imagen

Como se muestra en la Figura 3.13, se puede realizar una división superior (y1) e inferior (y2) y otra derecha (x2) e izquierda (x1) que permitan separar el código del resto de la imagen.

Cabe destacar que ésta es una técnica adicional que aplica [LZ10] como paso siguiente a la localización, pero previo a la lectura del código. No sería estrictamente necesario realizar este procedimiento, aunque ayuda a reducir la información a la única imprescindible para la etapa posterior.

Figura 3.13. Imagen tras realizar las cuatro divisiones [LZ10].

3.2.2. Localización del código QR

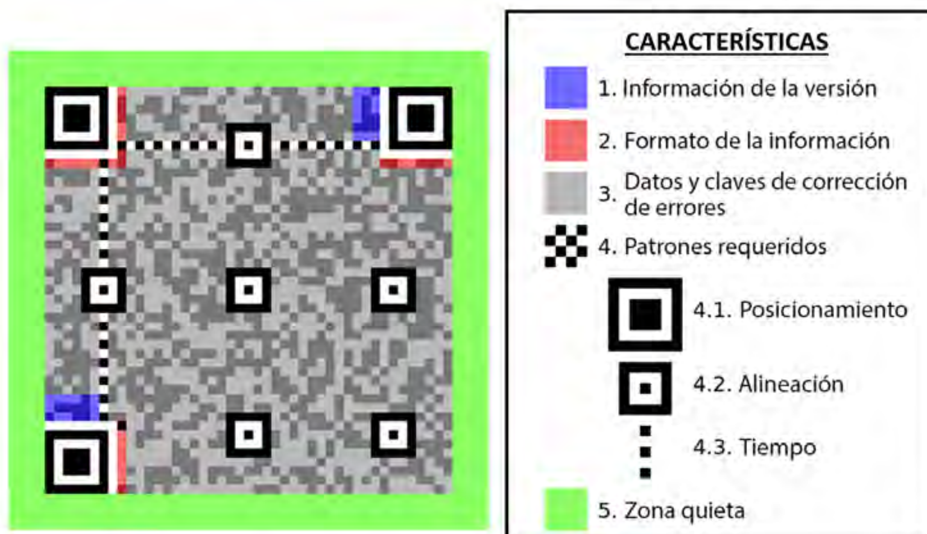


Figura 3.14. Estructura del código QR. Modificado de [Wwwa].

3.2.2.1. Detección de patrones de localización/tres esquinas

El código QR posee en tres de sus esquinas patrones de localización compuestos siempre por un cuadrado negro de 7x7 píxeles, relleno por un cuadrado blanco de 5x5 píxeles y éste, relleno a su vez, de un cuadrado negro de dimensiones 3x3. Dichos patrones aparecen indicados por la característica 4.1 de la Figura 3.14 y sirven para determinar tanto la posición como la orientación del código.

[Ohb+04], [GZ11], [Liu+08], [FK07] y [LL06] aplican esta técnica, para lo que buscan una ratio de píxeles oscuros-claros de 1:1:3:1:1 (Figura 3.15) en direcciones paralelas a cualquiera de los lados del código QR. Sin embargo, como cabe la posibilidad de que este patrón se repita en la zona de los datos, deben encontrarse de una forma rápida las tres esquinas que además deben estar alineadas entre sí. [Liu+08] para resolver este problema se apoya también en los patrones de tiempo (característica 4.3 de la Figura 3.14).

Esta técnica es, sin duda, una de las más eficientes para localizar los QR y la más empleada en la actualidad. Cabe destacar que estos patrones fueron diseñados específicamente con la función de encontrar el código, luego lo más acertado sería aplicar este método.

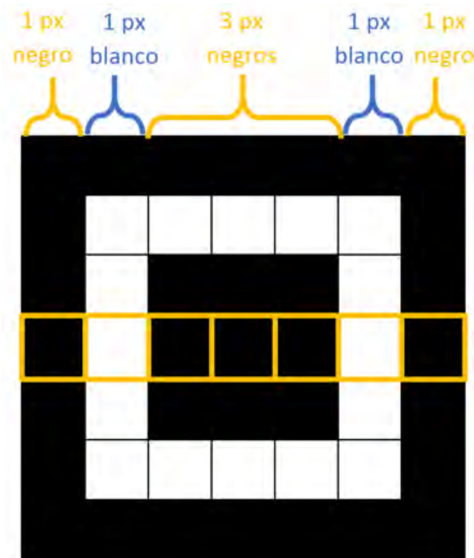


Figura 3.15. Patrón de localización del código QR. Ratio 1:1:3:1:1 (1 píxel negro:1 píxel blanco:3 píxeles negros:1 píxel blanco:1 píxel negro).

3.2.2.2. Detección de patrones de alineamiento

Los patrones de alineamiento o alineación aparecen como característica 4.2 de la Figura 3.14. Como puede observarse están formados por un módulo negro de un pixel, rodeado por un módulo blanco de 3x3 píxeles y, éste a su vez, rodeado por un cuadrado negro de 5x5 píxeles. Normalmente se poseen seis patrones de alineación, pero este número puede variar dependiendo de la versión del código.

[Liu+08] y [LL06] se apoyan en estos patrones no solo para determinar con mayor precisión la posición del código sino también para corregirla si ésta se encuentra torcida.

3.2.2.3. Estimación de la cuarta esquina

En la Figura 3.14 se aprecia la existencia de patrones de localización en tres de las cuatro esquinas del código QR. La cuarta esquina no posee ningún patrón, pero su posición puede determinarse una vez conocido dónde se encuentran las tres restantes.

Para ello, [Ohb+04] propone un algoritmo en el que traza dos líneas rectas: una desde la esquina superior derecha y otra desde la esquina inferior izquierda, acercando el punto de cruce de ambas rectas hasta que toque el área del código. Este proceso se muestra en la Figura 3.16, aunque hay que tener en cuenta que existen ciertos casos en los que el código QR no posee una celda o píxel negro justo en el punto de la cuarta esquina. [GZ11] también emplea un método

de escaneo en línea recta desde las esquinas próximas hasta conseguir una intersección que determine la posición de la cuarta.

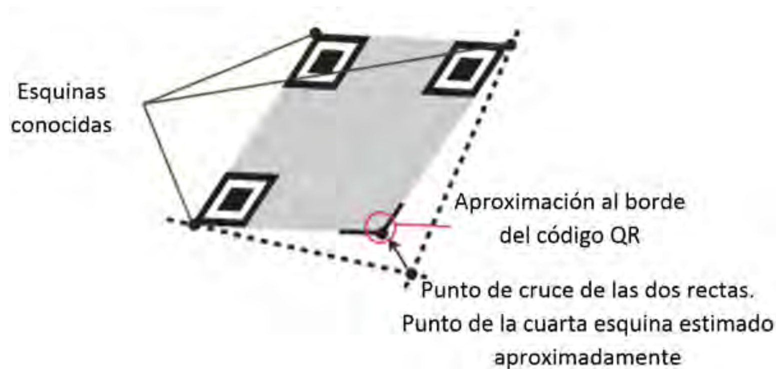


Figura 3.16. Método de estimación de la cuarta esquina. Modificado de [Ohb+04].

3.2.2.4. Estimación del centro

La posición del píxel central del código QR también puede ser determinada como presentan [GZ11], [Liu+08] y [LL06]. [GZ11] emplea un algoritmo que une los tres vértices conocidos formando un triángulo isósceles y traza, desde la esquina superior izquierda del código una línea recta hasta que corta con el centro del lado opuesto, siendo este punto de corte también el centro del código (Figura 3.17). [LL06] también emplea este método y además, resalta que el centro coincide con uno de los patrones de alineamiento. Por otro lado, [Liu+08] prefiere realizar un escaneo desde ocho direcciones diferentes para hallar el centro.

El método empleado por [LL06] y [GZ11] puede resultar de gran interés para localizar la posición exacta del código y para facilitar la posterior lectura y decodificación.

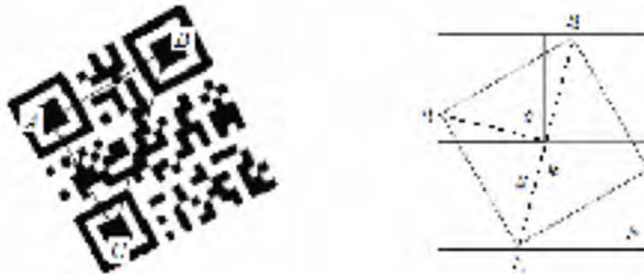


Figura 3.17. Método de estimación del centro [GZ11].

Como conclusión general de la fase de localización, cabe considerar la existencia de más de un código de barras o QR en las imágenes tomadas por el dron (caso en el que la foto ha sido tomada de varios paquetes). Por ello, el algoritmo escogido debe ser ligero, con el fin de localizar en el menor tiempo posible todos los códigos existentes que, además, pueden poseer distintas orientaciones.

3.3. Correcciones

Una vez que se ha localizado el código, se le pueden aplicar un conjunto de correcciones con el fin de rectificar algunas de las características que presenta. La realización de esta fase es opcional ya que pueden darse casos en los que no sea necesaria. Conviene aplicar las

correcciones únicamente al código, por lo que se debe llevar a cabo tras la fase de localización. Por otro lado, puesto que se pretende mejorar el área del código, es recomendable realizar las correcciones previamente a la fase de lectura, para que, de este modo, resulte más sencillo su posterior decodificación. Las técnicas empleadas en esta fase se muestran en la Tabla 3.3.

3.3.1. Corrección de la rotación

A veces, la imagen tomada contiene el código ligeramente girado hacia uno de los lados. En [GZ11] se propone un algoritmo que, tras haber localizado el código y detectado el ángulo de rotación, mediante interpolación bilineal corrige su posición. Cabe destacar que este paso puede ser aplicado antes de la localización siempre que se corrija la rotación de la imagen completa, no solamente del código.

3.3.2. Corrección de la inclinación

Se aplica cuando el código se encuentra en una posición que no es perpendicular a la del horizonte y existen distintas técnicas:

- **Transformada Jerárquica de Hough (HHT)**: como ya se explicó anteriormente en la sección 3.2.1.4, este método traza una serie de líneas a través de la imagen mediante las cuales puede determinar grupos de símbolos. [YS07] emplea dicha transformada con el fin de calcular el ángulo de inclinación que poseen los elementos del código y poder corregirlo.
- **Transformación de Perspectiva Inversa**: como puede apreciarse en el la Figura 3.18, si la imagen ha sido tomada con cierta inclinación, como muestra la gráfica de la derecha (de coordenadas u y v), [Ohb+04] usa las ecuaciones 3.1 y 3.2 para normalizar la forma del código, como aparece en la gráfica de la izquierda (de coordenadas x e y). Los coeficientes c_0 a c_7 de las ecuaciones anteriores pueden obtenerse mediante las relaciones entre los puntos de ambas gráficas, mostradas en la Figura 3.18. [Liu+10] también emplea esta técnica.

$$u = \frac{c_0 x + c_1 x + c_2}{c_6 x + c_7 y + 1} \tag{3.1}$$

$$v = \frac{c_3 x + c_4 x + c_5}{c_6 x + c_7 y + 1} \tag{3.2}$$

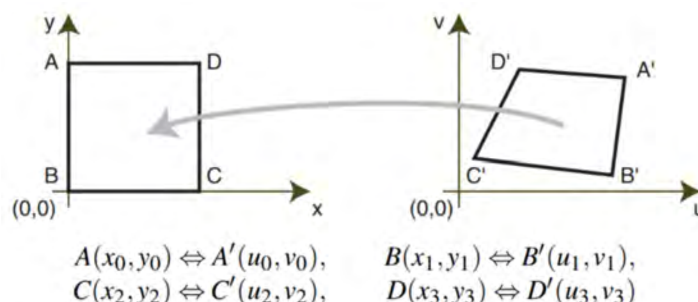


Figura 3.18. Transformación de Perspectiva Inversa. Modificado de [Ohb+04].

- **Transformación de Control del Punto**: [GZ11] detecta los cuatro vértices del código QR como puede apreciarse en la Figura 3.19 y, aplicando las ecuaciones que relacionan cada punto en la imagen inclinada con su homólogo (P_1' con P_1, etc.) en la imagen corregida, se consigue rectificar la inclinación del código.

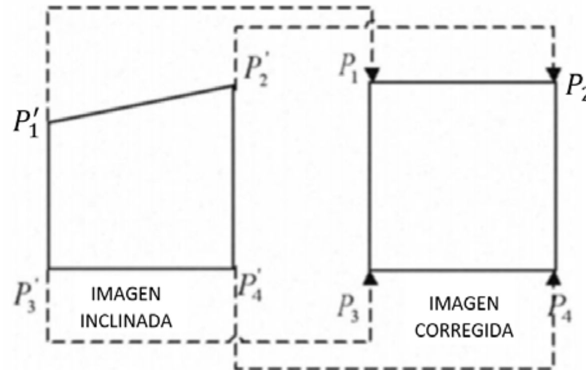


Figura 3.19. Transformación de Control del Punto [GZ11].

3.3.3. Adecuar líneas

Este es un procedimiento que consiste en aplicar una serie de técnicas al código de barras con el fin de mejorar la región antes de la decodificación, como se muestra en la Figura 3.20. Así, [YS07] ajusta los valores de intensidad de la imagen y tras esto, y ayudado por la matriz de similitud y el algoritmo de selección de candidato (técnicas explicadas en los puntos 3.2.1.3 y 3.2.1.5), consigue aportar una mayor definición al código en el caso de que existiesen zonas en las que no se distinguen unas barras de otras. [Liu+10] también aplica una serie de filtros para reducir el ruido y técnicas para corregir la inclinación como se ha visto anteriormente.

Figura 3.20. Técnica para adecuar líneas. (a) Imagen original, (b) Imagen tras la corrección. Modificado de [YS07].

3.4. Lectura

La última fase es la lectura, en la cual se procede a extraer la información del código. Una vez que se posee dicha información, debe decodificarse e interpretarse para comprobar si el código es correcto.

Al igual que en la fase anterior, la lectura se realiza de forma diferente para los códigos de barras y los QR. Para los primeros se han encontrado siete técnicas diferentes y cuatro para los segundos. Todas ellas se muestran en la Tabla 3.4.

3.4.1. Lectura del código de barras

3.4.1.1. Clasificación jerárquica por características

Consiste en, una vez que se posee la matriz de similitud de la imagen, clasificar sus características de forma jerárquica, de tal forma que se pueda determinar una estrategia para decodificar las barras directamente o no. Esto dependerá de si se posee una buena calidad en el código o si, por el contrario, es preferible aplicar otro preprocesamiento antes de su lectura.

[Muñ+99] define tres grupos en los que clasificar los distintos códigos atendiendo a tres características. Así, cada uno de estos grupos se define con un rango de valores para la longitud del código y para los niveles de gris. El primer grupo contiene los rangos en los que el código tendría la máxima calidad, el segundo se identifica con rangos en los que calidad sería intermedia y en el tercer grupo se encajarían los códigos que no se incluyen en ninguno de los anteriores y que poseen baja calidad. Con pertenencia a uno de los dos primeros grupos, la decodificación se realizaría directamente, mientras que si el código forma parte del tercer grupo se aplicarían una serie de técnicas previamente.

3.4.1.2. Calcular línea perpendicular al código

Figura 3.21. Línea perpendicular al código. Modificado de [Wwwap].

La lectura del código se realiza de forma perpendicular a las barras, es por esto que se traza una línea imaginaria que sea ortogonal a las mismas, tal y como se muestra en la Figura 3.21 destacada en color rojo. [Ohb+04], [CH05],[LZ10] y [YS07] usan este método, destacando [Ohb+04] que para calcular dicha línea emplea una de las barras del código detectadas en la fase anterior de localización. [YS07] además la emplea para definir las barras rectas si existen puntos en los que estén unidas y se desconozca su anchura, ya que va realizando la lectura a través de la línea y comprobando que dicha anchura no sea superior a un valor máximo.

Hallar esta línea resulta de gran utilidad sobre todo para realizar la lectura ya que se pueden ir almacenando las anchuras de las diferentes barras, como se explicará en más detalle en los apartados 3.4.1.3 y 3.4.1.4.

3.4.1.3. Extracción del vector de bits

A partir de la línea perpendicular al código obtenida mediante las técnicas explicadas en el apartado anterior 3.4.1.2, se realiza una lectura de izquierda a derecha en la que se van anotando los valores de los píxeles activos con el fin de conocer dónde empieza y acaba cada barra del código. En el caso de que el código de barras esté invertido, una opción sería realizar su lectura también en la dirección opuesta (de derecha a izquierda) y comprobar ambas lecturas con el código de control puesto que solo una de ellas coincidirá con éste. Esto puede llevarse a cabo de las siguientes formas:

- **Establecer umbral**: consiste en establecer un valor límite y, de forma similar al proceso de binarización, si el valor del píxel en escala de grises se encuentra por debajo de dicho umbral, se le asigna un cero y se identificaría con parte de una barra blanca. Así se aplica este método en [CH05] y [Muñ+99]. En [YS07] el umbral se establece como un máximo para la anchura de las barras.

- **Pares de bits de inicio y fin**: teniendo en cuenta únicamente cuándo empieza y acaba una barra se puede extraer el vector de píxeles activos. Así, [YS07] empleando el umbral establecido anteriormente, si el número de píxeles activos en una columna es mayor que el umbral, se le asigna el valor 1 a dicha columna, en caso contrario se le asigna 0.

- **Codificación Run-Length:** se aplica este conocido método de codificación en el cual cada una de las barras se codifica por una pareja de valores (*w,v*), o lo que es lo mismo (*anchura, valor*). El primer valor indica el número de veces seguidas que aparece repetido el segundo valor. En la Figura 3.22 se muestra un ejemplo de esta codificación, donde cada cuadrado gris representa un píxel. De esta forma, [CH05] y [LZ10] van contando el número de píxeles seguidos que hay en una columna, determinando así su anchura (w) que codifican junto con su color, blanco (1) o negro (0).

  El método Run-Length es un método de codificación que emplean algunos formatos sin pérdida de información para almacenar el contenido multimedia. Resulta más fácil de implementar que los otros dos métodos y proporciona una lectura muy rápida de la estructura del código para la posterior decodificación.

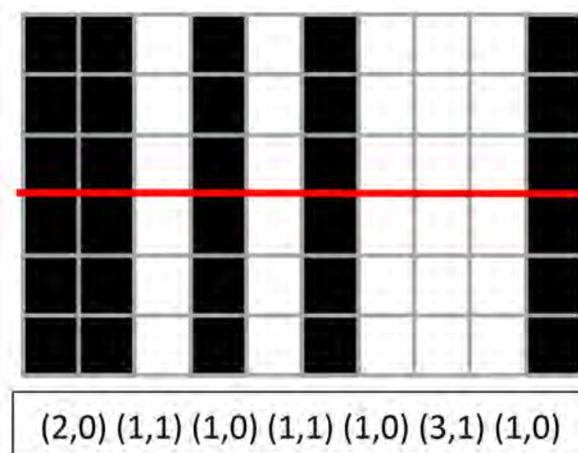


Figura 3.22. Ejemplo de codificación Run-Length de un código de barras.

3.4.1.4. Detección de la anchura

Este es un procedimiento que consiste en obtener la anchura de cada una de las barras de las que se compone el código. Se obtiene de forma directa tras haber aplicado el anteriormente expuesto en el punto 3.4.1.2 ya que, al poseer el vector de bits correspondiente al código de barras, para detectar la anchura se puede, o bien contar el número de píxeles con valor 1 (anchura de barras blancas) y con valor 0 (anchura de barras negras) o bien, la codificación Run-Length que la proporciona directamente.

La primera opción la emplean [YS07] y [Muñ+99] mientras que la segunda opción la aplica [CH05]. Por otro lado, [Ohb+04] no obtiene como resultado tras el trazado de la línea perpendicular al código una lectura bi-nivel, sino en escala de grises con ocho niveles. Esto se debe a que las columnas del código de barras pueden poseer cuatro tipos diferentes de anchura y el decodificador debe elegir qué umbral es mejor para cada tipo de columna. Podrían emplearse, por lo tanto, cuatro niveles para codificar la anchura de las barras blancas y otros cuatro niveles para codificar la anchura de las barras negras.

3.4.1.5. Lectura de patrón izquierdo, centro y derecho

Consiste en dividir el código en partes (de izquierda a derecha siempre) con el fin de que el decodificador extraiga la información presente en cada uno de estos grupos.

Para ello, [LZ10] parte del vector de bits para separar el carácter de inicio (primer dígito del identificador del país), los datos de la parte izquierda (códigos de país y empresa), los datos de la parte derecha (código de producto) y el carácter final que se corresponde con el dígito de control, como puede observarse en la Figura 3.5. [CH05] divide el código en tantas partes como dígitos representa, es decir, para el caso más común que es el del EAN-13, realiza trece subdivisiones.

Cabe destacar que esta técnica también resulta muy útil en la fase de lectura ya que se extrae toda la información y se le proporciona al decodificador de forma que pueda realizar su función de forma más eficiente. Además, se debe tener en cuenta que en todos los casos la decodificación se llevaría a cabo empleando el inverso del estándar internacional para codificación de códigos de barras EAN-13.

3.4.1.6. Aprendizaje por retropropagación de redes neuronales

Las redes neuronales consisten en un conjunto de nodos de procesamiento conectados a otros numerosos nodos de forma que imitan el comportamiento de las neuronas y los axones biológicos. El algoritmo de retropropagación o propagación hacia atrás es uno de los métodos de aprendizaje más conocidos y en el cual, se produce un estímulo desde adelante hacia atrás y va atravesando numerosas capas de nodos de procesamiento.

Esta técnica, precursora de la inteligencia artificial es empleada por [YS07] para, en primer lugar, realizar el aprendizaje del algoritmo de lectura y decodificación y, posteriormente, que la fase de reconocimiento del código se realice de manera automática. Éste es un método muy interesante pero que, sin embargo, requiere una elevada cantidad de ejemplos para entrenar el sistema.

3.4.1.7. Corrección de errores

Una vez que se ha realizado la lectura y decodificación del código de barras se puede realizar una corrección de errores con el fin de comprobar que el resultado obtenido tras estos procesos

es el correcto. La técnica de ***checksum*** consiste en realizar la suma de verificación con los dígitos extraídos del código de barras y el resultado se compara con el dígito de control (característica 5 de la Figura 3.5), si son iguales, la decodificación es correcta, si no coinciden, ha habido algún error. [CH05] emplea este método y, para hallar el valor del *checksum*, emplea el mismo método que se lleva a cabo para calcular el dígito de control del código. Esto es, ordenando los dígitos **de derecha a izquierda**, se suman los dígitos de posiciones impares, el resultado se multiplica por tres y se le suman los dígitos de las posiciones pares obteniendo la suma total. A continuación, se busca la decena inmediatamente superior y se le resta dicha suma total. El resultado final es el dígito de control pero si es múltiplo de 10 el dígito de control es 0 [Wwwas].

Por ejemplo, para el **código 123456789041** el dígito de control se calcularía de la siguiente forma:

1. Se ordenan los dígitos de derecha a izquierda: 1 4 0 9 8 7 6 5 4 3 2 1.
2. Se suman los dígitos de las **posiciones impares**: 1 4 0 9 8 7 6 5 4 3 2 1 $\rightarrow 1+0+8+6+4+2=$ **21**.
3. El resultado se multiplica por 3: $21 \cdot 3 =$ **63**.
4. Por otro lado, se suman los dígitos de las **posiciones pares**: 1 4 0 9 8 7 6 5 4 3 2 1 $\rightarrow 4+9+7+5+3+1=$ **29**.
5. Se suman los resultados de los pasos 3 y 4: $63+29=$ **92**.
6. La decena inmediatamente superior al resultado anterior es **100**.
7. Por lo que el **dígito de control** será la resta del resultado del paso 6 menos el 5: $100-92=$ **8**.

Quedando el código de la forma: 123456789041**8** [Wwwas].

3.4.2. Lectura del código QR

3.4.2.1. Generación de cuadrícula

Consiste en trazar una cuadrícula que permita dividir el código QR en secciones de datos para que sea más fácil realizar la decodificación. [Liu+08] aplica esta división una vez encontrados los patrones de localización y de alineación ya que únicamente consiste en unir los centros de los patrones de alineación hasta que intersequen con los patrones de localización.

3.4.2.2. Reverso del estándar internacional de codificación

Se incluye este procedimiento en un apartado diferenciado, al contrario de cómo se ha hecho con el código de barras ya que la mayoría de los artículos, una vez encontrada la localización del código, realizan directamente su decodificación aplicando el reverso del estándar internacional de codificación de códigos QR. Así lo hacen [GZ11], [Liu+08] y [LL06].

3.4.2.3. Transformada jerárquica de Hough

Se puede realizar la lectura del código QR aplicando nuevamente esta transformada. Según [FK07], la idea consiste en buscar líneas rectas en un cierto espacio para lo que resulta útil aplicar la Transformada Jerárquica de Hough ya que transforma la imagen en coordenadas de ángulo-magnitud.

3.4.2.4. Corrección de error Reed-Solomon

Este es un código cíclico no binario que se emplea como estándar en la corrección de errores de los datos transmitidos en comunicaciones. Se basa en el cálculo de un polinomio y su posterior interpolación, a partir de la cual se pueden recuperar los datos originales. [Liu+08] emplea este método para corregir errores cuando ciertas partes importantes del código se encuentran distorsionadas.

3.5. Tablas de técnicas

Las tablas que se muestran en este apartado (Tabla 3.1-3.4) resumen las técnicas empleadas por una serie de estudios realizados hasta el momento sobre el uso de visión artificial para la localización y lectura de códigos de barras y QR. Dichas tablas permiten extraer conclusiones generales sobre la información de la revisión de la literatura.

En la fase de preprocesamiento (Tabla 3.1), destaca la enorme frecuencia con la que se aplican las técnicas de escalado de grises y binarización. Esto se debe a que, tanto los códigos de barras como QR suelen ser en blanco y negro, por lo que la información del color de la imagen no es necesaria y puede ser eliminada. El filtrado de ruido, por otro lado, no es esencial, a no ser que la calidad de la imagen sea baja y existen numerosas técnicas distintas que puede aplicarse.

Por otro lado, en la fase de localización (Tabla 3.2), destacan las técnicas relativas a los códigos de barras puesto que no existe una tendencia clara en cuanto a cuál es el mejor método a aplicar, por lo que será necesario probar sobre imágenes reales numerosos algoritmos de localización de códigos de barras antes de decantarse por la mejor opción. Sin embargo, para los códigos QR, la técnica de detección de los patrones de localización parece ser la más empleada. Además, dicha técnica podría apoyarse con alguna de las tres restantes con el fin de asegurar la correcta localización del código.

La fase de las correcciones (Tabla 3.3) resultaría útil aplicarla estrictamente en los casos que sea necesario, puesto que no existe un elevado número de estudios que la emplean. Además, no existe ninguna técnica que destaque sobre las demás.

Finalmente, para realizar la lectura de los códigos de barras (Tabla 3.4), parece que la tendencia es calcular la línea perpendicular al código, para posteriormente, extraer el vector de bits y detectar la anchura de cada barra. Aunque no destaquen los métodos de corrección de errores, sí que es necesario realizar una comprobación de que la decodificación se ha llevado a cabo de manera correcta mediante el dígito de control. Para los códigos QR, la mayoría de los estudios optan por aplicar directamente el inverso al estándar internacional de codificación una vez que el código ha sido localizado, aunque igualmente sería conveniente realizar una corrección de errores posterior a la decodificación.

REFERENCIAS	TÉCNICAS DE PREPROCESAMIENTO										
				Filtrado del ruido							
	Escalado	Binarización	Perf. contorno	Dilatación	Aper./Cier.	Filt. med.	Muest. piram.	Filt. gauss.	Recortar	Redimensionar	Normalizar
Eisaku et al, 2004 **[Ohb+04]**	✓	✓		✓						✓	
Chai y Hock, 2005 **[CH05]**	✓	✓									
Xiangju et al, 2006 **[Lu+06]**	✓							✓			
Liu Y. y Liu M., 2006 **[LL06]**	✓	✓									
Youssef y Salem, 2007 **[YS07]**		✓	✓								
Falas y Kashani, 2007 **[FK07]**	✓	✓			✓				✓		
Liu et al, 2008 **[Liu+08]**	✓	✓			✓						
Li y Zeng, 2010 **[LZ10]**	✓	✓	✓			✓					
Liu et al, 2010 **[Liu+10]**	✓	✓					✓				
Gu y Zhang, 2011 **[GZ11]**	✓	✓									✓
TOTAL	**9**	**9**	**2**	**1**	**2**	**1**	**1**	**1**	**1**	**1**	**1**

Tabla 3.1. Técnicas empleadas en la fase de preprocesamiento. *Perf. contorno* se refiere a la técnica de perfilado del contorno, *Aper./Cier.* se refiere a Apertura/Cierre, *Filt. med.* a Filtro de la mediana, *Muest. piram.* a Muestreo piramidal y *Filt. gauss.* a Filtro gaussiano.

REFERENCIAS	TÉCNICAS DE LOCALIZACIÓN														
	CÓDIGOS DE BARRAS											CÓDIGOS QR			
								Det. bor.							
	Sep. comp. con.	Lín. esc. dif. dir.	Def. mat. sim.	HHT	Alg. selec. cand.	Bus. espiral	Esqueletización	Bas. en vertical	Canny	Bus. patrón	Extrac. imagen	Det. pat. loc./esq.	Det. pat. alin.	Est. cuarta esq.	Est. centro
Muñiz et al, 1999 **[Muñ+99]**				✓											
Eisaku et al, 2004 **[Ohb+04]**						✓						✓		✓	
Chai y Hock, 2005 **[CH05]**	✓						✓								
Liu Y. y Liu M., 2006 **[LL06]**												✓	✓		✓
Xiangju et al, 2006 **[Lu+06]**		✓	✓												
Youssef y Salem, 2007 **[YS07]**	✓				✓										
Falas y Kashani, 2007 [6] **[FK07]**												✓			
Liu et al, 2008 **[Liu+08]**												✓	✓		✓
Li y Zeng, 2010 **[LZ10]**								✓			✓				
Liu et al, 2010 **[Liu+10]**									✓	✓					
Gu y Zhang, 2011 **[GZ11]**												✓		✓	✓
TOTAL	**2**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**5**	**2**	**2**	**3**

Tabla 3.2. Técnicas empleadas en la fase de localización. *Sep. comp con.* se refiere a la técnica de Separar componentes conectados, *Lín. esc. dif. dir.* se refiere a Líneas de escaneo en diferentes direcciones, *Def. mat. sim.* a Definir matriz de similitud, *HHT* a Transformada Jerárquica de Hough, *Alg. selec. cand.* a Algoritmo de selección de candidato, *Bus. espiral* a Búsqueda en espiral, *Det. bor.* a Detección de bordes, *Bas. en vertical* a Basado en vertical, *Bus. patrón* a Búsqueda de patrón, *Extrac. imagen* a Extracción de la imagen, *Det. pat. loc./esq.* a Detección de los patrones de localización/tres esquinas, *Det. pat. alin.* a Detección de patrones de alineación, *Est. cuarta esq.* a Estimación de la cuarta esquina y *Est. centro* a Estimación del centro.

REFERENCIAS	TÉCNICAS DE CORRECCIÓN					
		Corr. inclinación				
	Corr. rotación	HHT	Transf. persp. inv.	Transf. cont. punt.		Adec. líneas
Eisaku et al, 2004 **[Ohb+04]**			✓			
Youssef y Salem, 2007 **[YS07]**		✓				✓
Falas y Kashani, 2007 **[FK07]**	✓				✓	
Liu et al, 2010 **[Liu+10]**		✓				✓
Gu y Zhang, 2011 **[GZ11]**	✓			✓		
TOTAL	**2**	**1**	**2**	**1**	**1**	**2**

Tabla 3.3. Técnicas empleadas para realizar las correcciones. *Corr. rotación* se refiere a la técnica de Corrección de la rotación, *Corr. inclinación* se refiere a Corrección de la inclinación, *Transf. persp. inv* a Transformación de la perspectiva inversa, *Transf. cont. punt.* a Transformación de control del punto y *Adec. líneas* a Adecuar líneas.

REFERENCIAS	TÉCNICAS DE LECTURA												
	CÓDIGOS DE BARRAS									CÓDIGOS QR			
			Ext. vec. bits										
	Clas. jer. car.	Lín. perp. cod.	Estimar umbral	Pares bits ini/fin	Cod. Run-Lenght	Det. anch.	Lect. pat. iz/ce/de	BPNN	Cor. err.	Gen. cuad.	Rev. est. int.	HTT	Reed-Solomon
Muñiz et al, 1999 **[Muñ+99]**			✓			✓							
Eisaku et al, 2004 **[Ohb+04]**		✓				✓							
Chai y Hock, 2005 **[CH05]**		✓	✓		✓	✓	✓		✓				
Liu Y. y Liu M., 2006 **[LL06]**											✓		
Xiangju et al, 2006 **[Lu+06]**	✓												
Youssef y Salem, 2007 **[YS07]**		✓	✓	✓		✓		✓					
Falas y Kashani, 2007 [6] **[FK07]**												✓	
Liu et al, 2008 **[Liu+08]**										✓	✓		✓
Li y Zeng, 2010 **[LZ10]**		✓			✓		✓						
Gu y Zhang, 2011 **[GZ11]**											✓		
TOTAL	**1**	**4**	**3**	**1**	**2**	**3**	**2**	**1**	**1**	**1**	**3**	**1**	**1**

Tabla 3.4. Técnicas empleadas en la fase de lectura. *Clas. jer. car.* se refiere a la técnica de Clasificación jerárquica por características, *Lín. perp. cod.* se refiere a Línea perpendicular al código, *Ext. vec. bits* a Extracción del vector de bits, *Pares bits ini/fin* a Pares de bits de inicio y fin, *Det. anch.* a Detección de la anchura, *Lect. pat. iz/ce/de* a Lectura de patrón izquierdo, centro y derecho, *BPNN* a Aprendizaje por retropropagación de redes neuronales, *Cor. err.* a Corrección de errores, *Gen. cuad.* a Generación de cuadrícula, *Rev. est. int.* a Reverso del estándar internacional de codificación, *HHT* a Transformada Jerárquica de Hough y *Reed-Solomon* al método de corrección de errores Reed-Solomon.

4 Preprocesamiento y localización

Buscad, y hallaréis.

Mateo 7:7

El presente capítulo describe de forma detallada el trabajo llevado a cabo en relación a las fases de preprocesamiento y localización de los códigos, así como las técnicas concretas seleccionadas para cada una de estas fases del algoritmo desarrollado. Además, se muestran una serie de resultados de pruebas parciales ejecutadas sobre una versión del algoritmo previa a la definitiva.

Para llevar a cabo las fases de preprocesamiento y localización se hará uso de una librería de tratamiento de imágenes denominada OpenCV. Dicha librería de código abierto proporciona una infraestructura para aplicaciones de visión artificial [Wwwu]. En primer lugar, antes de aplicar cualquier modificación sobre la imagen, ésta debe ser leída por el programa. De esta forma, dicha imagen se toma desde el directorio en el que haya sido guardada y el algoritmo la carga en color, tal y como ha sido capturada originalmente. En el sistema definitivo, el algoritmo tomará la imagen directamente desde la cámara. Tras esto, se comprueba que la imagen ha sido cargada correctamente, en cuyo caso, se pasaría a la fase de preprocesamiento. En el caso contrario, el programa muestra un mensaje de error por pantalla y finaliza su ejecución.

La librería OpenCV posee una clase denominada `cv::Mat` la cual permite almacenar y manipular los píxeles de una imagen. `cv::Mat` constituye una parte fundamental de OpenCV al ser utilizada por la mayoría de las funciones que define. Éstas, generalmente toman un objeto `cv::Mat` de entrada, lo procesan y generan un nuevo objeto `cv::Mat` de salida [Wwwak].

Para representar una imagen en formato digital se emplea la estructura de matriz, almacenando en cada celda el valor de un píxel. Como es bien conocido, el píxel es la unidad básica de una imagen digitalizada y se representa con un valor numérico que indica su color [Wwwaw]. En las imágenes (1) de las Figuras 4.1 y 4.2 se pueden apreciar los píxeles que las

forman. Así, para una imagen en escala de grises únicamente se posee una matriz de valores que indican el nivel de la intensidad de color de gris (imagen (2) de la Figura 4.1), mientras que para una imagen a color se tratará de una matriz multidimensional en la que cada canal representa la intensidad de unos colores: azul, verde o rojo, si se encuentra en formato RGB (imagen (2) de la Figura 4.2). Un objeto `cv::Mat` posee la estructura de una matriz y, además, puede estar formado de distintos canales, dependiendo del formato de la imagen que contenga [Wwwak].

Figura 4.1. Ejemplo de imagen en escala de grises. (1) Píxeles de la imagen. (2) Matriz con valores de los píxeles. Modificado de [Wwwak].

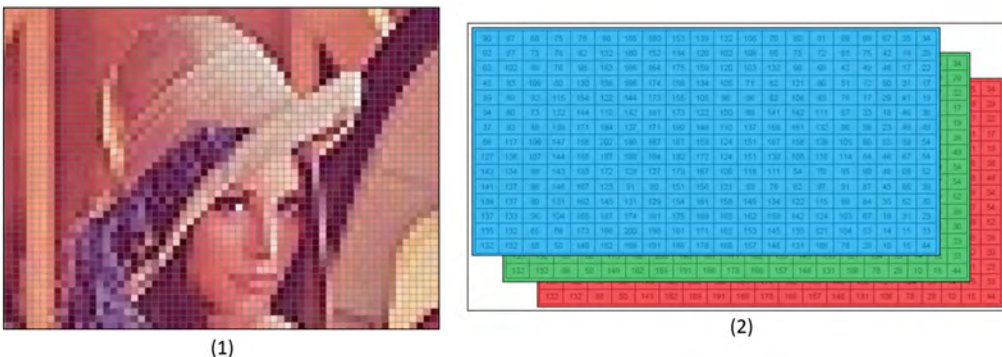


Figura 4.2. Ejemplo de imagen en color. (1) Píxeles de la imagen. (2) Matriz con valores de los píxeles. Modificado de [Wwwak].

4.1. Preprocesamiento

Una vez se ha cargado la imagen en el programa, se comienza la primera fase del algoritmo: el preprocesamiento. En esta fase se aplican una serie de técnicas de forma secuencial: escalado de grises, binarización, cierre y apertura.

4.1.1. Escalado de grises

Con el objetivo de eliminar la información relativa al color de la imagen, ésta se convierte a escala de grises. Los códigos de barras codifican la información en el número de barras, anchura y espaciado de las mismas [Wwwap], por lo que el color no es relevante para su lectura.

Para pasar la imagen de color a escala de grises se emplea la función `cv::cvtColor` la cual convierte una imagen de un espacio de color a otro. La declaración de la función es la siguiente [Wwwab]:

```
void cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)
```

Donde `src` es la imagen de entrada, `dst` es la imagen de salida, `code` es el código de conversión de los espacios de color y `dstCn` es el número de canales de la imagen destino. Si `dstCn` es 0, el número de canales se deriva automáticamente de los parámetros `src` y `code`. `dstCn` puede excluirse de la función. En la Figura 4.3 se muestra un ejemplo de una imagen a color (1) transformada a escala grises (2). Para realizar esta conversión, el valor que toma el parámetro `code` es `CV_BGR2GRAY`.

Figura 4.3. Conversión de una imagen en color a escala de grises. (1) Imagen original. Obtenida de la base de datos *no.2 plain* [Wwwp]. (2) Imagen en escala de grises.

4.1.2. Binarización

Como ha sido comentado anteriormente, el color no es relevante para la lectura del código. Por lo tanto, con la binarización se reduce más la cantidad de información relativa al color de la imagen. Con la función `cv::thresholding` se realizan binarizaciones globales, tomando la imagen en escala de grises y asignando a cada pixel un valor 0 o 1 dependiendo de si el valor de dicho píxel se encuentra por encima o por debajo de cierto umbral [Wwwab], [Wwwaa]. Cabe destacar la existencia de otra función denominada `cv::adaptiveThreshold` mediante la cual se pueden efectuar binarizaciones adaptativas, en las cuales se define un umbral diferente para cada bloque de píxeles dentro de la imagen.

En la Figura 4.4 pueden apreciarse los distintos tipos de binarización que se probaron con el fin de identificar el mejor para el caso de los códigos: el método del umbral medio adaptativo (1), el método del umbral adaptativo de Gauss (2), el del umbral global (3) y el de Otsu (4). La descripción de algunos de estos métodos se realizó en la subsección 3.1.2 del capítulo 3. El método del umbral adaptativo medio establece el valor del umbral como la media de los valores de los píxeles vecinos, y en el método del umbral adaptativo de Gauss, el valor umbral es la suma ponderada de los valores de píxeles vecinos, siendo los pesos una ventana gausiana [Wwwaa].

El método seleccionado para realizar la binarización en el algoritmo diseñado ha sido el Método de Otsu debido a que es uno de los que generan un mejor contraste de las barras que forman el código y el fondo de la imagen como se aprecia en la imagen (4) de la Figura 4.4. Con el método del umbral global se obtiene un resultado similar, sin embargo, es necesario ajustar el valor de dicho umbral. Por otro lado, los métodos adaptativos (imágenes (1) y (2) de la Figura 4.4) eliminan el relleno interior de las barras negras dejando únicamente el contorno de las mismas, por lo que pierden información del código.

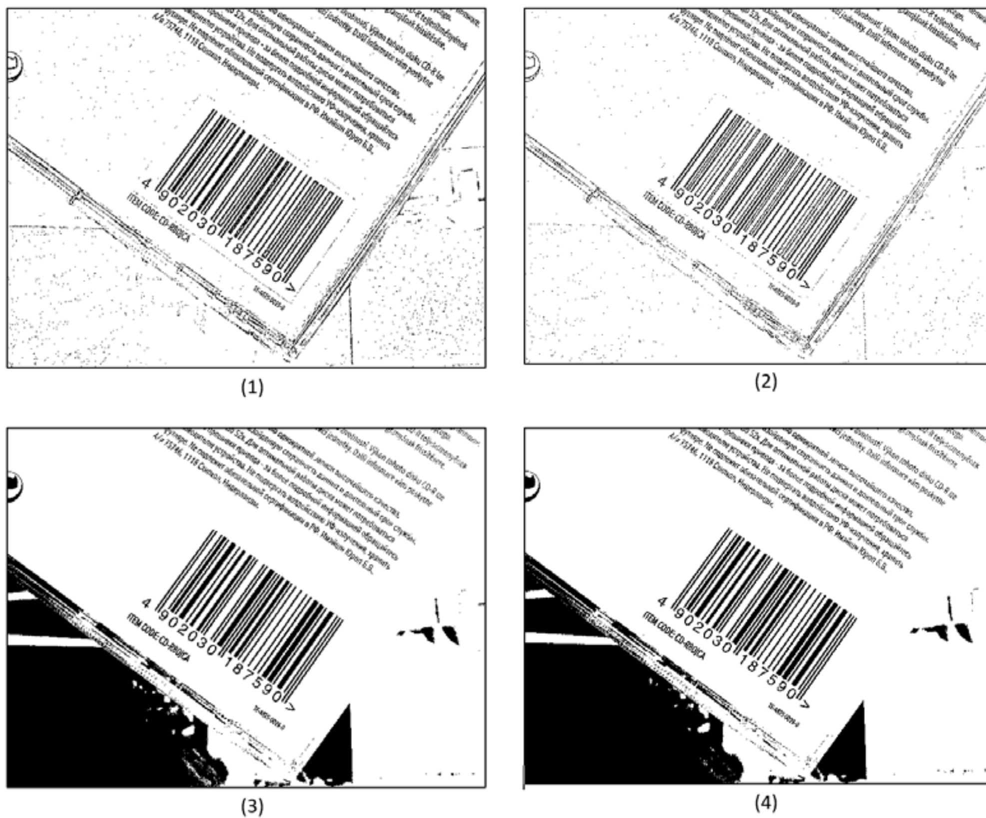


Figura 4.4. Ejemplo de distintos tipos de binarización. (1) Método del umbral adaptativo medio. (2) Método del umbral adaptativo de Gauss. (3) Método del umbral global. (4) Método de Otsu.

4.1.3. Cierre

Con el objetivo de filtrar el ruido de la imagen se hace el cierre de la misma. En el apartado 3.1.4, se especificó el funcionamiento de este operador morfológico que elimina huecos del código haciendo uso de un elemento estructural.

Figura 4.5. Ejemplo de imagen tras haber realizado el cierre de la misma.

Es, por tanto, el primer paso, obtener el elemento estructural mediante la función `cv::getStructuringElement`. El elemento seleccionado es un rectángulo de 3 píxeles de ancho y de 2 píxeles de largo, y éste se pasa como parámetro a la función `cv::morphologyEx` la cual realiza el cierre de la imagen. Dicha función recibe otra serie de parámetros de entrada

de entre los que destaca el código (número entero) que indica el tipo de operación morfológica que se desea realizar, en este caso `MORPH_CLOSE` [Wwwz]. Se seleccionó dicho elemento estructural debido a que, tras una serie de pruebas de entrenamiento del algoritmo, era con el que el se conseguía leer una mayor cantidad de códigos.

Un ejemplo del cierre de una imagen se aprecia en la Figura 4.5 en la cual puede observarse cómo la imagen es muy similar a la original binarizada, con la salvedad de que las líneas de texto existentes en la parte superior de la imagen han sido prácticamente eliminadas.

4.1.4. Apertura

Al igual que ocurría con el cierre en el apartado anterior (4.1.3), la apertura tiene como fin continuar filtrando el ruido de la imagen. En el apartado 3.1.4 se especificó el funcionamiento de este operador morfológico que elimina píxeles del fondo de la imagen que no pertenecen a los objetos, haciendo uso igualmente de un elemento estructural.

Para realizar la apertura se sigue un proceso idéntico al del cierre. En primer lugar, se obtiene el elemento estructural mediante la función `cv::getStructuringElement`, que en este caso se trata de un rectángulo de dimensiones 14 píxeles de ancho y 7 píxeles de largo. Dicho elemento se pasa como parámetro a la función `cv::morphologyEx` al igual que el código que inidica el tipo de operación morfológica a realizar: `MORPH_OPEN` [Wwwz]. De la misma forma que ocurrió con el elemento estructural del cierre, el de la apertura se seleccionó debido a que, tras una serie de pruebas, era con el que se leía una cantidad más elevada de códigos.

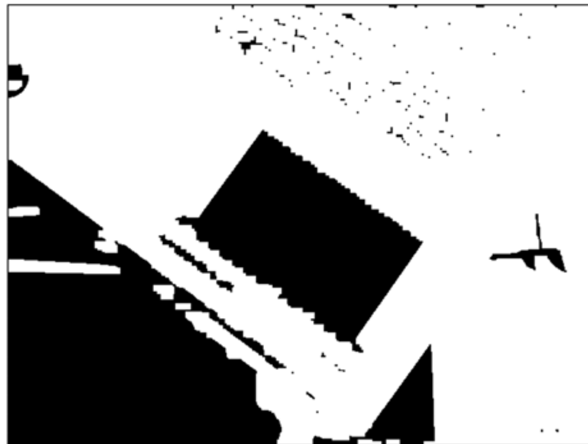


Figura 4.6. Ejemplo de imagen tras haber realizado la apertura de la misma.

Como ejemplo de apertura de una imagen se observa la Figura 4.6 en la cual puede apreciarse cómo esta operación conserva los objetos que poseen un tamaño igual al del elemento estructural o mayor, eliminando el resto.

4.1.5. Otras técnicas probadas

En este apartado se describe otra técnica que fue ensayada sobre las imágenes durante la fase de preprocesamiento pero que, finalmente, por distintos factores que se explicarán a continuación, no fue incluida en el algoritmo.

4.1.5.1. Ecualización adaptativa limitada del histograma de contraste

Ecualizar el histograma de una imagen consiste en obtener, a través de una transformación, un histograma con una distribución uniforme. De esta forma, en el caso de una imagen en escala de grises, existirá el mismo número de píxeles para cada nivel de gris [Wwwar].

La ecualización adaptativa limitada del histograma de contraste, también conocida como CLAHE por sus siglas en inglés (*Contrast Limited Adaptive Histogram Equalization*) es una variante de la ecualización adaptativa. Las ecualizaciones adaptativas se diferencian de las ecualizaciones normales en que calculan varios histogramas, cada uno correspondiente a una sección distinta de la imagen. Éstos se usan para redistribuir los valores de iluminación de dicha imagen. Sin embargo, en este tipo de ecualizaciones, se tiende a sobreamplificar el contraste en zonas de la imagen en las que el histograma está muy concentrado, causando, al mismo tiempo, la amplificación del ruido en dichas zonas. Es decir, se aumenta en exceso el contraste en las secciones de la imagen en las que el histograma posee valores constantes, aumentando también el ruido.

Por lo tanto, CLAHE, limita la amplificación de dicho contraste, para, de esta forma, reducir el problema de la amplificación del ruido [Wwwal]. En la Figura 4.7 se puede apreciar este proceso de limitar el histograma en las zonas en las que se encuentra concentrado, mientras que en la Figura 4.8 puede observarse la comparativa entre la imagen original (1) y la imagen tras el proceso de ecualización (2), en la cual se ha aumentado el contraste tomando más píxeles valores oscuros cercanos al negro, principalmente en las zonas de valores constantes. Además, se aprecia cómo en dichas zonas comienzan a aparecer píxeles de ruido. CLAHE con el límite en el histograma evita que dicho ruido aumente excesivamente.

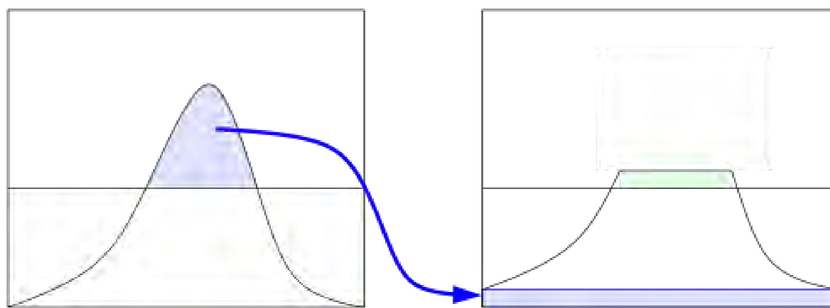


Figura 4.7. Proceso realizado por CLAHE para limitar el histograma de contraste [Wwwat].

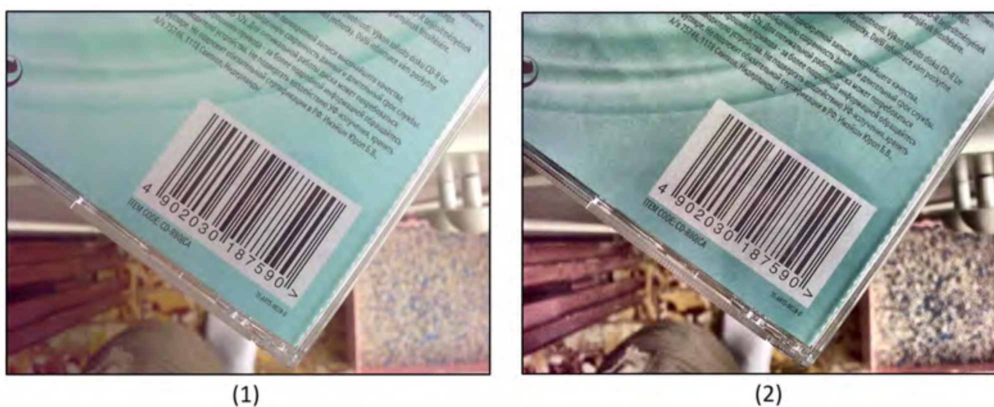


Figura 4.8. Ejemplo de ecualización adaptativa limitada del histograma de contraste. (1) Imagen original. (2) Imagen tras aplicar la ecualización.

En un principio, se pensó incluir esta ecualización tras la carga de la imagen, con el objetivo de mejorar la calidad de la imagen, sobre todo en los casos en los que las condiciones de iluminación eran malas. Sin embargo, la transformación elevaba considerablemente el tiempo de ejecución del algoritmo. Además, el ratio de códigos leídos con la ecualización frente a los leídos sin la ecualización no era demasiado elevado. Por lo tanto, finalmente la ecualización ha sido eliminada del algoritmo.

4.1.6. Resultados parciales de la versión previa del algoritmo

En esta sección se hace un análisis de los resultados obtenidos tras realizar una serie de pruebas de tiempo sobre una versión del algoritmo previa a la definitiva. Para ello, se ha tomado una muestra representativa de imágenes de tres tipos: con buena iluminación y distintas orientaciones,—ninguna siendo la orientación por defecto (barras perpendiculares a la horizontal y números situados en la parte inferior del código pudiendo ser leídos de izquierda a derecha)—, buena iluminación y la orientación por defecto, y con mala iluminación y distintas orientaciones. Las imágenes empleadas pertenecen a la base de datos *no.2 plain* obtenida de [Wwwp]. Las pruebas se han llevado a cabo en una máquina virtual con el sistema operativo Ubuntu 16.04. Sus recursos asignados son 4 GB de memoria RAM y 2 hilos de procesador. La máquina virtual se ejecuta sobre un ordenador con procesador Intel® Core™ i7-4500U CPU @ 1.80 GHz (Turbo Boost hasta 2.40 GHz), el sistema operativo Windows 8.1 de 64 bits y 8 GB de memoria RAM.

Imagen	Tiempo preprocesamiento (ms)				
	Carga	Escalado y binarización	Cierre	Apertura	Total
05102009082	10,35	3,72	0,58	0,68	**15,33**
05102009115	10,64	2,82	0,54	0,78	**14,78**
05102009139	11,33	2,67	0,58	0,77	**15,35**
05102009104	13,32	3,62	1,00	1,00	**18,34**
05102009108	11,48	4,27	0,81	1,16	**17,72**
05102009122	12,08	4,03	1,14	0,76	**18,01**
05102009170	10,98	2,92	0,62	0,76	**15,28**
05102009192	10,55	2,72	0,54	0,72	**14,53**
05102009201	11,13	3,63	0,60	0,69	**16,05**
Promedio	**11,31**	**3,38**	**0,71**	**0,81**	**16,15**

Tabla 4.1. Tiempo del algoritmo previo al definitivo en ejecutar las diferentes etapas de la fase de preprocesamiento. Todas las imágenes poseen formato *.png*. Los valores máximos de cada columna aparecen sombreados en gris.

La Tabla 4.1 recoge los resultados de tiempo de ejecución en milisegundos obtenidos para cada una de las técnicas empleadas en la fase de preprocesamiento. Como puede apreciarse, la carga de la imagen es el paso más costoso en cuanto a tiempo de ejecución, con un promedio muy superior al del resto de las etapas. Esto se debe a que el programa debe buscar la imagen en el directorio indicado, leerla por completo e ir almacenándola píxel a píxel en una variable de tipo `cv::Mat`. Cabe destacar, que dicha carga se realiza desde la memoria RAM del dispositivo, ya que el dron, al tomar los fotogramas del exterior los irá almacenando en esta memoria. También es necesario tener en cuenta que la resolución de las imágenes es de 640x480 píxeles,

con lo que el tiempo de carga aumentará o disminuirá notablemente en función de si las dimensiones son mayores o menores que éstas.

Por otro lado, los procesos de escala de grises y binarización conjuntamente suponen un aumento del tiempo medio de ejecución de en torno a 3,38 milisegundos, algo más elevado que la aplicación de los operadores morfológicos de cierre y apertura, cuyo tiempo medio de ejecución es mínimo, siendo 0,71 milisegundos para el cierre y 0,81 milisegundos para la apertura. En promedio, la fase de preprocesamiento supondría unos 16,15 milisegundos del tiempo total de ejecución.

4.2. Localización

Tras haber realizado la fase de preprocesamiento sobre la imagen, en la que se intenta eliminar el máximo posible de información no relevante para la lectura del código, el siguiente paso a realizar es la localización del mismo. Dicho preprocesamiento, al filtrar elementos de la imagen innecesarios, facilita la fase de localización, cuyo objetivo consiste en detectar la posición exacta del código. Las técnicas aplicadas para conseguir esto son: detección de bordes mediante Canny, encontrar contornos, calcular áreas, recortar el código y rotarlo.

4.2.1. Canny

El algorimo de Canny consta de múltiples etapas que le permiten detectar una amplia gama de bordes de una imagen. El funcionamiento de cada una de dichas etapas está especificado en la sección 3.2.1.8. Mediante la función `cv::Canny` que proporciona OpenCV se puede aplicar esta técnica de manera sencilla. La función toma como parámetro de entrada la imagen resultante del proceso de apertura (apartado 4.1.4). Además, se deben dar a la función otra serie de parámetros numéricos que indican valores umbrales [Wwwx]. En el caso del algorimo desarrollado, se tomaron los valores utilizados en [Wwwf] puesto que posteriormente también servirá dicha referencia de apoyo para la siguiente técnica a aplicar (apartado 4.2.2).

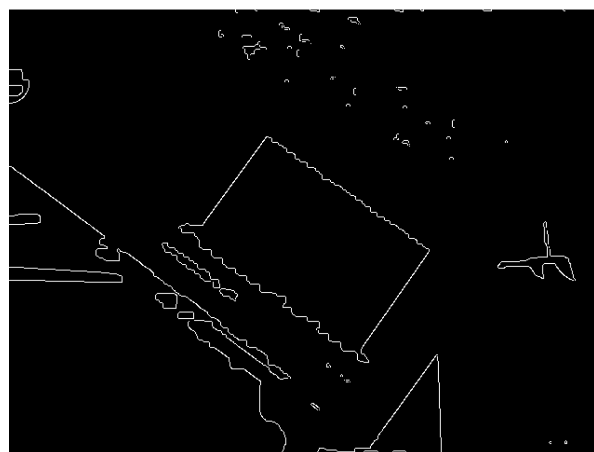


Figura 4.9. Ejemplo de imagen tras haber aplicado la detección de bordes de Canny sobre la misma.

El resultado de la imagen tras haber aplicado el algoritmo de Canny se muestra en la Figura 4.9. Destaca en ella el rectángulo inclinado respecto a la horizontal y que corresponde a la zona en la que se sitúa el código de barras.

4.2.2. Encontrar contornos y calcular áreas

Tras haber detectado los bordes de la imagen mediante Canny, el siguiente paso es encontrar dichos bordes o contornos y almacenarlos en una variable para poder tratarlos más adelante. Esto se lleva a cabo mediante la función `cv::findContours` cuya declaración se muestra a continuación [Wwwac]:

```
void findContours(InputOutputArray image, OutputArrayOfArrays contours, int mode,
    int method, Point offset=Point())
```

Donde `image` es la imagen de entrada, `contours` es la variable en la que se guardan los contornos encontrados, `mode` indica el método de recuperación del contorno, que en el caso que nos concierne será `CV_RETR_EXTERNAL` para que devuelva únicamente los contornos externos y `method` es el tipo de método de aproximación de los contornos, siendo en este caso `CV_CHAIN_APPROX_SIMPLE`, el cual comprime los segmentos horizontales, verticales y diagonales dejando únicamente sus puntos finales. Finalmente, el parámetro `offset` se corresponde con un desplazamiento que se le aplica a cada punto. Este último parámetro es opcional y en el algoritmo se omite. En las imágenes (1) y (2) de la Figura 4.10 puede observarse que partiendo de la detección de bordes de Canny, se han encontrado los contornos externos de los objetos.

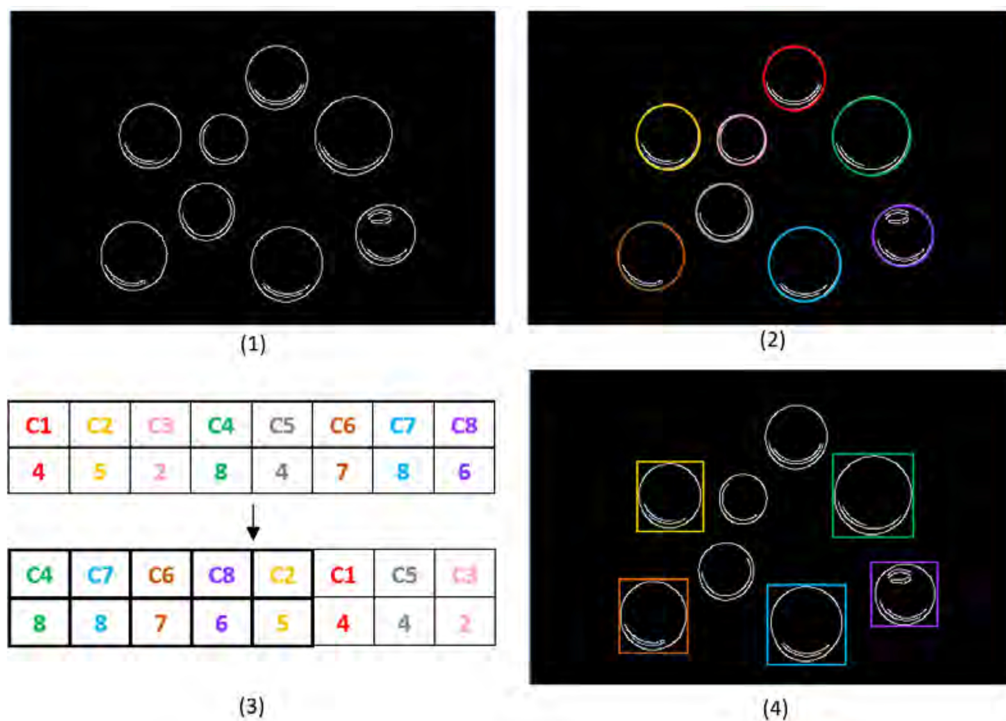


Figura 4.10. Técnica de encontrar contornos y calcular áreas. (1) Imagen tras haber aplicado detección de bordes de Canny. (2) Imagen con los contornos externos encontrados. (3) La variable superior almacena el índice de cada contorno asociado a su área y la variable inferior es el resultado de la ordenación de las áreas de mayor a menor y la selección de los cinco contornos con mayor área. (4) Rectángulos de menor área que contienen los puntos de los contornos seleccionados. Modificado de [Wwwae].

Una vez se poseen los contornos de la imagen almacenados en una variable, se van a intentar cerrar los mismos mediante la función `cv::appoxPolyDP`. Esta función aproxima una curva poligonal con una precisión especificada [Wwwac]. Su declaración es la siguiente:

```
void approxPolyDP(InputArray curve, OutputArray approxCurve, double epsilon, bool
    closed)
```

Donde `curve` es el vector de puntos de entrada, en este caso es el vector que almacena los contornos, `approxCurve` es el resultado de la aproximación, `epsilon` se corresponde con el parámetro que indica la precisión de la aproximación (máxima distancia entre la curva original y su aproximación) y `closed` que, si su valor es `true`, la aproximación cerrará la curva. En el algoritmo diseñado el parámetro de precisión tendrá un valor 50 ya que, tras hacer varias pruebas, éste es el valor que proporciona mejores resultados. Por otro lado, `closed` será `true` puesto que el siguiente paso será calcular el área de los contornos encontrados, para lo que necesitamos que sean cerrados. En la imagen (2) de la Figura 4.10 los contornos detectados se aproximarán a círculos cerrados con la precisión indicada.

Dado que ya se poseen los contornos cerrados almacenados en una variable, se procede a calcular el área que encierran los mismos. Esto se lleva a cabo mediante la función `cv::contourArea` [Wwwac]:

```
double contourArea(InputArray contour, bool oriented=false)
```

Esta función toma como parámetros de entrada `contour`, el contorno cuya área desea calcularse y `oriented`, un valor que sirve para deducir la orientación del contorno si su valor es `true`. En este caso tomará el valor `false`. La función devuelve una variable tipo `double` que es el valor del área.

Una vez calculadas todas las áreas de los contornos encontrados, éstas se ordenan de menor a mayor y se seleccionan los cinco contornos cuyas áreas sean las más elevadas puesto que, tras varias pruebas, el contorno que rodea al código de barras, suele encontrarse entre esas áreas. Además, se especifica que el área de los contornos seleccionados sea mayor que un tamaño mínimo, en este caso 10. Esto se hace con el fin de evitar que contornos cuya área sea cero (lo cual es muy corriente) sean seleccionados. Esto puede deberse o bien, a que los contornos no han sido cerrados, lo que no ocurre en este caso ya que antes se ha usado la función `approxPolyDP`, o que al aproximar los contornos a polígonos, los más pequeños hayan quedado aproximados como puntos. Los puntos no son polígonos cuadrados con un borde de un píxel, sino objetos de un solo píxel, por lo que no encierran área [Wwwc]. La imagen (3) de la Figura 4.10 muestra el proceso de calcular y almacenar en una variable los índices de los contornos asociados a sus áreas para luego ordenarlas de mayor a menor y seleccionar los cinco que poseen áreas mayores.

Finalmente, para acabar con esta fase, se hace uso de la función `cv::minAreaRect`, la cual devuelve el rectángulo (posiblemente rotado respecto a la horizontal) de menor área que contiene los puntos de los contornos. De esta forma, se consigue aproximar cada uno de los contornos seleccionados a un rectángulo, siendo esta la forma del código de barras. Esta función toma como parámetro de entrada el contorno y devuelve el rectángulo que contiene los puntos de dicho contorno [Wwwac]. En la imagen (4) de la Figura 4.10 se puede observar cómo, para los contornos seleccionados se han calculado los rectángulos de menor área que los contienen. En este caso particular, al ser los contornos círculos, dichos rectángulos son, en realidad, cuadrados sin rotación respecto a la horizontal.

En la Figura 4.11 puede apreciarse que el algoritmo ha encontrado dos posibles contornos que pueden contener al código, uno de los cuales efectivamente lo contiene . El otro se corresponde con un borde cerrado que fue detectado mediante Canny en la Figura 4.9 y, al aproximarlo a

un rectángulo, queda de esa forma. Además, como se ha especificado anteriormente, ambos contornos poseen forma rectangular.

Figura 4.11. Ejemplo de imagen tras haber encontrado y seleccionado los contornos cerrados con la mayor área.

4.2.3. Recortar y rotar

En primer lugar, cabe destacar que recortar y rotar podrían identificarse como técnicas de corrección ya que éstas se aplican una vez se ha localizado la posición del código. La técnica de recortar aparece explicada en la subsección 3.1.5 del capítulo 3 (Estado del arte) encajándose dentro de la fase de preprocesamiento. Por otro lado, rotar se clasifica propiamente dentro de la fase de corrección del código y su funcionamiento está especificado en la sección 3.3.1.

Volviendo al caso del algoritmo desarrollado, una vez se poseen los contornos de mayor área aproximados por rectángulos, se procede a recortar y rotar dichas áreas. Tomando como referencia [Wwwb], para la rotación lo primero que debe hacerse es calcular el ángulo que debe rotarse la imagen. Para ello, inicialmente se obtiene, por un lado, el ángulo que posee el rectángulo (en el sentido de las agujas del reloj) y, por otro, su anchura y altura. Tras esto, se comprueba si el ángulo es menor de -45°. En caso afirmativo, se le suman 90°, y se intercambia la anchura del rectángulo por su altura y viceversa, consiguiendo que este rote otros 90°. De esta forma, si el ángulo del rectángulo es -90°, al intercambiar la anchura por la altura, quedará en la posición deseada y su ángulo de giro será de 0°, tal y como se muestra en la Figura 4.12. Si el ángulo es mayor de -45°no se altera, como puede apreciarse en el ejmplo de la Figura 4.13. A continuación, mediante la función `cv::getRotationMatrix2D` se obtiene una matriz afín de rotación en dos dimensiones, la cual será empleada para realizar una transformación afín mediante la función `cv::warpAffine` que da como resultado la imagen rotada el ángulo deseado [Wwwy].

La función `cv::getRotationMatrix2D` toma como parámetros de entrada `center` que es el centro de rotación de la imagen origen (el centro del rectángulo en este caso), `angle` identificado como el ángulo de rotación en grados, el cual ha sido calculado como se ha explicado anteriormente; y `scale` es el factor de escala, que en este caso tomará el valor 1.0. Es importante destacar, además, que valores positivos del parámetro `angle` significan que la rotación se realiza en el sentido contrario a las agujas del reloj. Además, se toma como coordenada origen la esquina superior izquierda de la imagen. De esta forma, si el ángulo del rectángulo es 10°, (en el sentido de las agujas del reloj) ese es el valor que se le pasa a la

función, y ésta corrige la posición de la imagen girándola 10 grados en sentido contrario a las agujas del reloj. La declaración de la función se muestra a continuación [Wwwy]:

```
Mat getRotationMatrix2D(Point2f center, double angle, double scale)
```

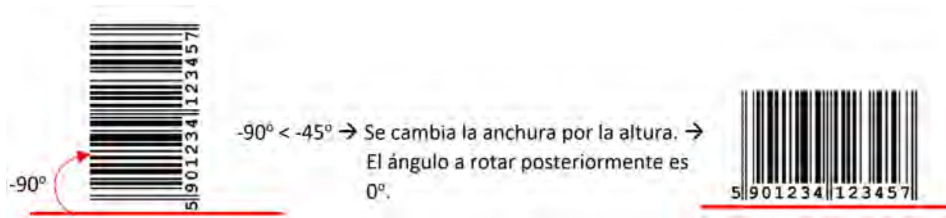


Figura 4.12. Ejemplo de cálculo de un ángulo de rotación negativo. Modificado de [Wwwap].

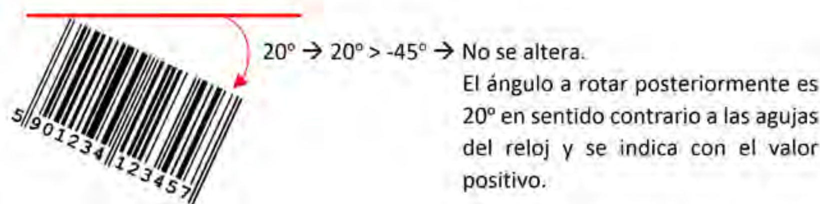


Figura 4.13. Ejemplo de cálculo de un ángulo de rotación positivo. Modificado de [Wwwap].

Por otra parte, la función `cv::warpAffine` recibe de parámetros de entrada `src` como la imagen origen, `dst` como la imagen resultado de la rotación, `M` que es la matriz afín de rotación obtenida mediante la función `cv::getRotationMatrix2D`, `dsize` identificado como el tamaño de la imagen resultante (en este caso el tamaño será el mismo que el de la imagen original) y otra serie de flags y valores que especifican métodos para realizar la transformación. Para el algorimo diseñado únicamente se incluirá el parámetro `flags` con valor `INTER_CUBIC` (interpolación bicúbica sobre una cuadrícula de 4x4 píxeles vecinos) omitiendo el resto. A continuación se incluye la declaración de la función [Wwwy]:

```
void warpAffine(InputArray src, OutputArray dst, InputArray M, Size dsize, int
    flags=INTER_LINEAR, int borderMode=BORDER_CONSTANT, const Scalar& borderValue=
    Scalar())
```

Figura 4.14. Ejemplo de imagen tras haber corregido la rotación del código.

La Figura 4.14 muestra el ejemplo de la imagen explicada hasta ahora rotada de forma que se corrige la rotación del código, dejando sus barras perpendiculares a la horizontal.

Una vez se ha corregido la rotación de la imagen, el último proceso a realizar para finalizar con la fase de localización es recortar el código de barras. Para ello se emplea la función `cv::getRectSubPix` la cual extrae un rectángulo de píxeles de una imagen con precisión de subpíxel. Su declaración aparece a continuación [Wwwy]:

```
void getRectSubPix(InputArray image, Size patchSize, Point2f center, OutputArray
    patch, int patchType=-1)
```

La función recibe como variables de entrada `image` la imagen origen, que en este caso será la imagen rotada, `patchSize` es el tamaño del área que se desea extraer de la imagen (en este caso el rectángulo), `center`; el centro del rectángulo que se extrae y `patch` es la imagen cortada, resultado de haber aplicado la función. `patchType` es la profundidad de los píxeles extraídos y puede omitirse, como se hace en este caso.

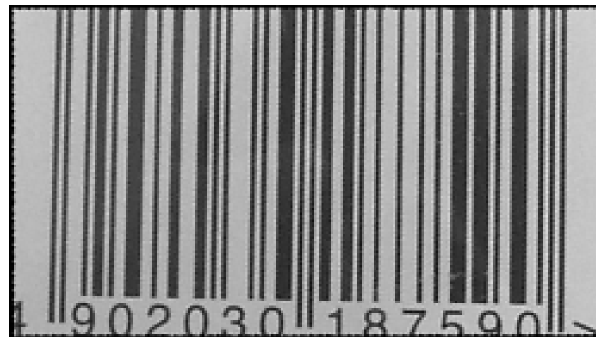


Figura 4.15. Ejemplo de imagen tras haber recortado el código.

La Figura 4.15 muestra la imagen del código de barras extraída de la imagen original. Puede apreciase, además, que aunque algunos de los números del código aparecen cortados esto no supondrá un problema a la hora de realizar la lectura ya que el lector únicamente necesita un fragmento del código en el que aparezcan todas sus barras para poder decodificarlo.

Figura 4.16. Ejemplo de las dos rotaciones: la del ángulo calculado en azul y la del ángulo suplementario en amarillo. Modificado de [Wwwap].

Cabe destacar que, el otro rectángulo detectado en la imagen pasará por el mismo proceso de rotar y girar si, tras realizar la fase de lectura sobre el primer rectángulo, no se consigue decodificar ningún código. Por otro lado, se debe tener en cuenta que el código de barras puede necesitar una rotación con el ángulo suplementario al empleado por el algoritmo. Esto se debe a que el código puede estar en una posición con sus barras perpendiculares a la horizontal pero

del revés (con los números en la parte superior). En un principio se planteó la posibilidad de rotar el código con ambos ángulos: el calculado y su suplementario tal y como se muestra en la Figura 4.16, ya que existen casos en los que únicamente girando el ángulo calculado el código no quedaría en la posición por defecto. Sin embargo, más tarde se comprobó que el lector empleado realiza la decodificación correctamente de ambas formas, por lo que se decidió que el algorimo realizase una única rotación para ahorrar así tiempo de procesamiento y recursos de CPU. Esto se explica más detalladamente en la sección 4.2.5.

Al localizar, recortar y girar el código, se elimina el resto de información no necesaria para la lectura del mismo y se reduce el tamaño y peso de la imagen. De esta forma, la fase de lectura se realizará de una manera más rápida y eficiente.

4.2.4. Otras técnicas probadas

En esta sección se describen otra serie de técnicas con las que se experimentó sobre diferentes imágenes para comprobar su efectividad pero fueron descartadas finalmente por una serie de motivos que se exponen a continuación.

4.2.4.1. Sobel

El operador de Sobel es otra de las técnicas empleadas para la detección de bordes y cuyo funcionamiento se especificó en el apartado 3.2.1.8. Para aplicar este algoritmo hacemos uso de la función `cv:Sobel` [Wwwz] mediante la cual se calcula el gradiente en la dirección del eje X y del eje Y. Tras esto, se resta al gradiente de X el gradiente de Y. De esta forma, se obtienen las partes de la imagen con gradientes horizontales grandes y verticales bajos, indicativo de que en dichas zonas se encuentra un código de barras. Un ejemplo de una imagen tras haber aplicado esta técnica se presenta en la Figura 4.17.

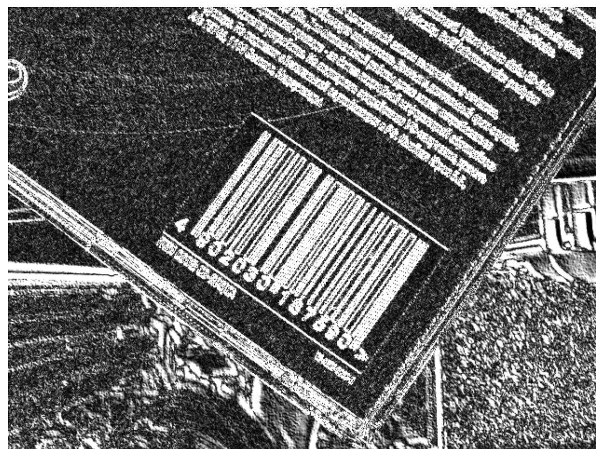


Figura 4.17. Ejemplo de imagen tras haber aplicado el operador de Sobel.

Sin embargo, el operador de Sobel no resultaba tan efectivo como el algoritmo de Canny a la hora de detectar los bordes de la imagen ya que, una vez calculados los gradientes de la imagen, se debía filtrar el ruido mediante el cierre y apertura de la misma. Al aplicar el cierre de la imagen, incluso empleando el mismo elemento estructural que se usa en el algoritmo desarrollado, la zona del código prácticamente desaparecía, junto con el resto de bordes de la imagen. Además, la función `cv::findContours` se aplica normalmente tras Canny. Es por esto que se decidió emplear esta última técnica.

4.2.4.2. Transformada Jerárquica de Hough (HHT)

Otra técnica que también puede emplearse para la detección de bordes la Transformación Jerárquica de Hough. Su funcionamiento quedó especificado en la subsección 3.2.1.4 y su implementación consiste en básicamente traducir las operaciones matemáticas de cálculo de las coordendas polares a código. En la Figura 4.18 se muestra un ejemplo del resultado.

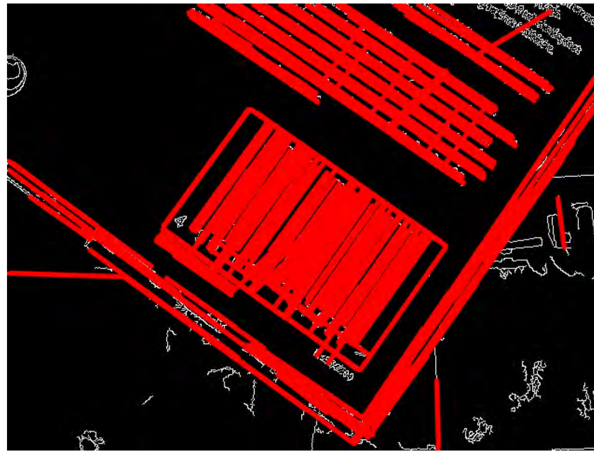


Figura 4.18. Ejemplo de imagen tras haber aplicado la transformada jerárquica de Hough.

Si bien esta transformada detecta con gran precisión las líneas de la imagen, su ejecución no es muy eficiente. Esto se debe a que el método que emplea conlleva una elevada cantidad de cálculo matemático, lo que eleva la carga de trabajo del procesador y el tiempo que el código tardará en ejecutarse.

4.2.5. Resultados de la versión previa del algoritmo

Al igual que se hizo en el apartado 4.1.6 con la fase de preprocesamiento, para las técnicas aplicadas en la localización también se han llevado a cabo una serie de pruebas con el objetivo de medir el tiempo de ejecución. Las imágenes escogidas son las mismas que las empleadas en dicho apartado, al igual que las características del *hardware* donde se ejecutan las pruebas. La Tabla 4.2 muestra los resultados recogidos.

En primer lugar, puede apreciarse que el algortimo de Canny es bastante ligero, siendo 2,73 milisegundos de media lo que tarda en ejecutarse, con lo que se reafirma como una buena opción para la detección de bordes frente a la Transformada Jerárquica de Hough, comentada anteriormente en el apartado 4.2.4.2, la cual conlleva una carga de cálculo más elevada. En cuanto a la etapa de encontrar contornos y calcular las áreas de los mismos, así como seleccionar de entre aquellas las cinco de mayor valor, su tiempo de ejecucción es el menor de esta fase, 1,12 milisegundos

Estas dos primeras técnicas deben su eficiencia a la labor realizada en la fase de preprocesamiento de filtrar la imagen para eliminar la información que no sea necesaria para las fases posteriores. El proceso de seleccionar las áreas consiste simplemente en un bucle `for` con una condición a verificar, el cual se detiene cuando se poseen las cinco áreas mayores, por lo que conlleva un tiempo de procesamiento mínimo.

Finalmente, cabe destacar que en esta versión previa del algoritmo, la principal diferencia respecto a la versión definitiva, es que, una vez se ha detectado uno de los rectángulos, éste

se rota no solamente el ángulo calculado para dejar sus barras perpendiculares a la horizontal sino también su ángulo suplementario. De esta forma, si el código se encontraba más cerca de la posición por defecto (barras perpendiculares a la horizontal) pero del revés (con los números situados en la parte superior), al rotarlo tanto el ángulo normal como el suplementario, se obtendrían las dos posibles posiciones del código con sus barras perpendiculares a la horizontal. Sin embargo, más tarde se comprobó que este proceso no era necesario puesto que la librería empleada para realizar la lectura de los códigos, era capaz de decodificarlos correctamente en ambas posiciones.

Teniendo en cuenta todo lo anterior, en el algoritmo final se decidió eliminar la segunda etapa de recortar y rotar la imagen, la cual, tarda de media aproximadamente lo mismo que la primera etapa: 8,98 milisegundos. Con esto, el tiempo total medio empleado por la fase de localización pasaría de 22,07 milisegundos, en la versión previa del algoritmo, a 13,09 milisegundos aproximadamente, en su versión definitiva. Se consigue reducir en un 40,7 % aproximadamente el tiempo de ejecución de esta fase, aunque esto se examinará con mayor detalle en el capítulo 6.

Imagen	**Tiempo localización (ms)**				
	Canny	**Encontrar contornos y calcular áreas**	**Cortar y rotar (1)**	**Cortar y rotar (2)**	**Total**
05102009082	3,22	2,35	7,29	8,00	**20,86**
05102009115	2,19	0,61	7,52	7,01	**17,33**
05102009139	2,23	0,77	8,85	6,11	**17,96**
05102009104	2,45	1,09	9,67	8,55	**21,76**
05102009108	3,33	1,20	11,00	8,00	**23,53**
05102009122	4,70	2,04	12,23	15,57	**34,54**
05102009170	2,09	0,55	11,75	10,04	**24,43**
05102009192	2,32	0,98	7,78	9,06	**20,14**
05102009201	2,05	0,53	7,01	8,51	**18,10**
Promedio	**2,73**	**1,12**	**9,23**	**8,98**	**22,07**

Tabla 4.2. Tiempo del algoritmo previo al definitivo en ejecutar las diferentes etapas de la fase de localización. Todas las imágenes poseen formato *.png*. Los valores máximos de cada columna aparecen sombreados en gris.

5
Lectura

No existe la felicidad sino cuando
comprendemos que hemos logrado algo.
Henry Ford
(1863–1947)

El quinto capítulo abarca la realización de la fase de lectura del algoritmo propuesto. Expone tanto el funcionamiento de la librería empleada como su aplicación específica en dicho algoritmo. Al igual que en el capítulo anterior, se muestran una serie de resultados de pruebas parciales realizadas sobre una versión del algoritmo previa al definitivo.

Una vez que se ha realizado la localización del código y se ha corregido alguna característica del mismo, únicamente queda por llevar a cabo la última fase el algoritmo: la lectura. Al igual que ocurre con el preprocesamiento y la localización, para realizar la lectura de los códigos se emplea una librería: ZBar [Wwwbc]. ZBar es un paquete de *software* de código abierto para leer códigos de barras de diversas fuentes, como transmisiones de video, archivos de imágenes y sensores. Soporta la lectura múltiples tipos de códigos de barras distintos, que incluyen EAN-13, Code 128, Code 39 y QR, entre otros.

Para realizar la lectura de los códigos, ZBar utiliza un enfoque más cercano al utilizado por los escáneres *wand* y *láser*, cuya mecánica es similar solo que el primero hace uso de un único LED y el segundo de un haz de luz láser [Wwwaj]. Ambos basan su funcionamiento en pasar un punto de luz sobre el símbolo del código de barras impreso, de forma que, las barras oscuras absorben la fuente de luz y las barras blancas la relejan. El sensor mide la reflexión de la luz y es capaz de determinar la anchura y color (blanco o negro) de cada barra [Wwwaj]. Aprovechando esto, la implementación ZBar realiza pasadas de escaneo lineal sobre una imagen, tratando cada píxel como una muestra de un único sensor de luz [Wwwba].

Siguiendo el ejemplo de los paradigmas de procesamiento actuales, ZBar se basa en un modelo estructurado en capas. El procesamiento se divide en capas independientes con interfaces bien definidas, que se pueden usar juntas o individualmente conectadas a cualquier otro sistema como se muestra en la Figura 5.1.

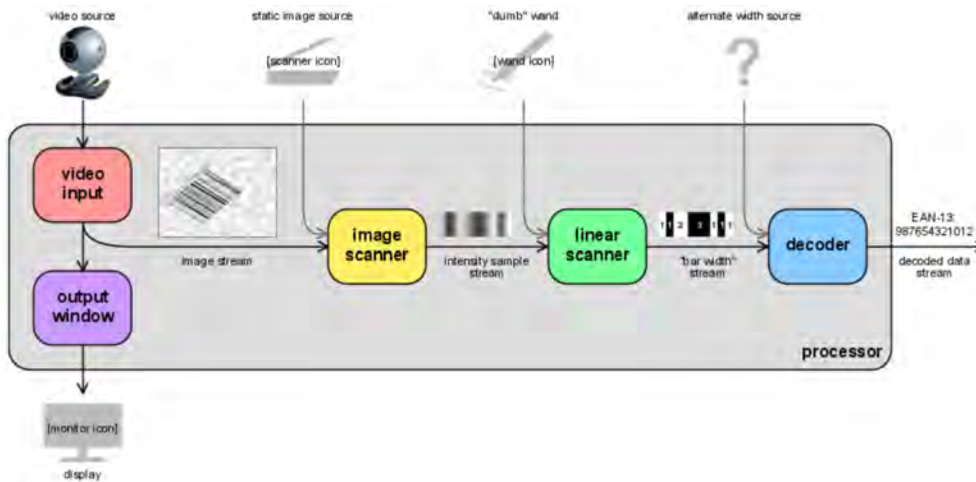


Figura 5.1. Diagrama de bloques del funcionamiento de ZBar [Wwwba].

Siguiendo el diagrama de la Figura 5.1, en primer lugar, el lector recibe un *stream* de vídeo de entrada, el cual está compuesto por una serie de imágenes. De esta forma, el mismo proceso se seguiría si se introdujese una única imagen. Opcionalmente, tras esto, puede mostrarse la imagen de entrada por pantalla y recoger parámetros introducidos por el usuario [Wwwba]. En el caso que nos concierne este último paso se omite.

Una vez se posee el conjunto de bits que forman la imagen, se realiza un escaneo de la misma. El escáner realiza pasadas sobre la imagen bidimensional para obtener un flujo lineal de muestras de intensidad de los píxeles. Es decir, lo que consigue es para cada línea de escaneo obtener el valor de cada píxel que es atravesado por dicha línea. La imagen introducida al lector debe estar en escala de grises ya que el escaneo guarda valores de intensidad.

Dado que ya ha sido obtenido el flujo de muestras de intensidad, se aplica a éste un escaneo lineal para generar un nuevo flujo que contenga la anchura de cada una de las barras. Este proceso es similar al realizado por los lectores convencionales como el lector *wand*, solo que éstos emplean la cantidad de luz reflejada que recoge el sensor y, en el caso de ZBar, posee exactamente el valor de las intensidades lumínicas de cada píxel. Finalmente, el decodificador busca patrones reconocibles en el flujo de anchura de las barras y los identifica con los distintos símbolos que representan, generando así la secuencia de datos decodificados.

El módulo de alto nivel *procesador* hace posible conectar el resto de módulos ya que, un inconveniente del enfoque modular en el que las capas son completamente independientes es que puede llevar algo de codificación unir todos los módulos, lo que complica incluso las aplicaciones más simples.

5.1. Lectura del código

Para realizar la lectura del código se tomará la imagen resultado de la fase de localización. Dicha imagen únicamente posee el código con sus barras en posición perpendicular a la horizontal y se encuentra en escala de grises. Tomando como referencia [Wwwl], en primer lugar se crea un objeto del tipo `zbar::ImageScanner` el cual nos permitirá realizar el escaneo sobre la imagen. Es necesario establecer cierta configuración para el escáner, lo cual se hace a mediante la función `zbar::set_config` cuya definición es la siguiente [Wwwbe]:

```
int zbar::ImageScanner::set_config(zbar_symbol_type_t symbology, zbar_config_t
    config, int value)
```

Donde `symbology` es el tipo del código de barras decodificado, `config` son las opciones de configuración y `value` que es 1 para opciones booleanas como es en este caso [Wwwbe]. El algoritmo usa `ZBAR_NONE` para el primer parámetro, con lo que no se especifica exactamente el tipo de código que se va a escanear (pudiendo decodificar cualquiera) y `ZBAR_CFG_ENABLE` para el segundo parámetro, permitiendo la obtención de dicha simbología en el tiempo de lectura del código.

Una vez que se posee el escáner, se obtiene la información de la imagen haciendo el uso del atributo `data` de la clase `cv::Mat`, así como su número de filas y de columnas. Con estos parámetros se crea un objeto `zbar::Image`, el cual almacena muestras de datos de imagen junto con los metadatos de formato y tamaño asociados [Wwwbd]. Sobre este objeto se realiza el escaneo mediante la función `zbar::scan` cuya declaracion es [Wwwbe]:

```
int zbar::ImageScanner::scan(Image &image)
```

Esta función simplemente recibe como parámetro de entrada el objeto `zbar::Image` y busca símbolos en la dicha imagen proporcionada. Finalmente, el algoritmo se introduce en un bucle en el que para cada uno de los símbolos detectados por el escáner, se extraen los datos decodificados y se muestran por pantalla, junto con el tipo de código que ha sido leído, tal y como se muestra en la Figura 5.2.

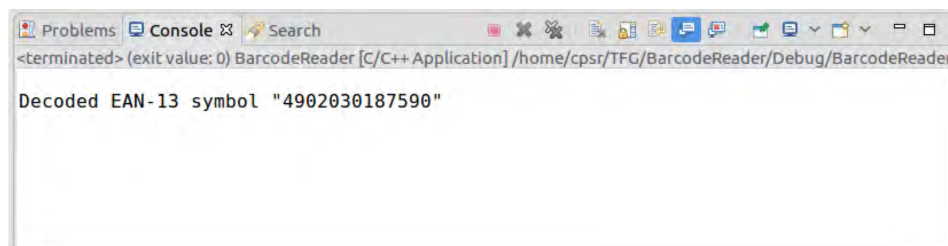


Figura 5.2. Salida del programa tras la lectura del código de barras.

En la Figura 5.2 se observa que el código ha sido leído y decodificado correctamente, puesto que se ha seguido el ejemplo tratado durante todo el documento, partiendo de la imagen original, (1) de la Figura 4.3. Además, se detecta que la simbología del código de barras es EAN-13, cuya estructura y características están definidas en el capítulo 2.

Cabe destacar que el algoritmo definitivo en cuanto detecta y lee un código barras en uno de los rectángulos extraídos de la imagen original, no lee el resto de ellos. De esta forma, se evitar realizar la lectura de todos los contornos detectados en la imagen y que no poseen ningún código de barras, disminuyendo el tiempo de ejecución del algoritmo.

Una vez que se han llevado a cabo las tres fases del algoritmo, puede darse el caso de que éste no haya sido capaz de detectar y leer el código de barras. En estas ocasiones se ha optado por aplicar directamente la lectura con ZBar a toda la imagen. Se omitirían así, las fases de preprocesamiento y localización, únicamente realizando la carga de la imagen y su transformación a escala de grises. Como se comprobará más adelante, en el capítulo 6, aplicar directamente ZBar a toda la imagen conlleva un mayor tiempo de procesamiento pero mejora el ratio de códigos leídos.

Finalmente, es importante recalcar que el algoritmo cuyo diseño e implementación han sido explicados a lo largo del capítulo 4 y del presente, es el mismo que se ha aplicado para el reconocimiento y decodificación de los códigos QR. Un ejemplo de una imagen original con un

código QR se muestra en la Figura 5.3 y la lectura de dicho código, resultado de la ejecución del algoritmo, se muestra en la Figura 5.4.

Figura 5.3. Ejemplo de imagen original con código QR. Obtenida de la base de datos *qrcode-dataset* [Wwwi].

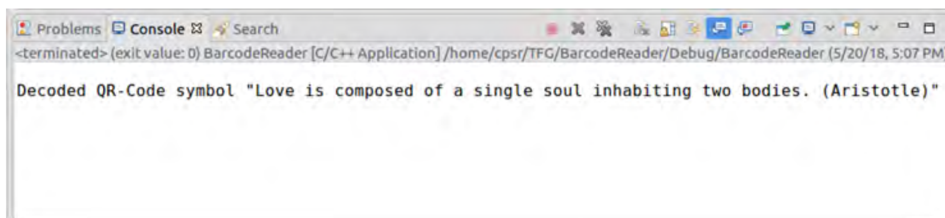


Figura 5.4. Salida del programa tras la lectura del código QR.

5.2. Resultados de la versión previa del algoritmo

Una vez que se posee una compresión completa del funcionamiento del algoritmo y de las modificaciones que las fases de preprocesamiento y localización realizan sobre la imagen, en esta sección se pretenden analizar los resultados de tiempo de lectura obtenidos para la versión previa a la definitiva y compararlos con una lectura realizada aplicando directamente ZBar a la imagen, sin el tratamiento previo.

Al igual que en las pruebas parciales de las fases anteriores, mostradas en los apartados 4.1.6 y 4.2.5, se emplean las mismas imágenes (con diferentes rotaciones y condiciones de iluminación) de la base de datos *no.2 plain* extraída de [Wwwp]. El *hardware* en el que han sido realizadas las pruebas también es idéntico al empleado en dichos apartados: una máquina virtual con el sistema operativo Ubuntu 16.04. Sus recursos asignados son 4 GB de memoria RAM y 2 hilos de procesador. La máquina virtual se ejecuta sobre un ordenador con procesador Intel® Core™ i7-4500U CPU @ 1.80 GHz (Turbo Boost hasta 2.40 GHz), el sistema operativo Windows 8.1 de 64 bits y 8 GB de memoria RAM. En la Tabla 5.1 pueden apreciarse los resultados obtenidos.

Se observa en dicha tabla que, de media, el tiempo que tarda el algoritmo en realizar la lectura del código es aproximadamente 4,2 veces menor de lo que emplea ZBar aplicado directamente sobre la imagen, siendo 17,6 milisegundos el tiempo empleado por esta versión previa del algoritmo y 73,94 milisegundos el de ZBar.

Esta reducción de tiempo se debe al tratamiento aplicado a la imagen durante las fases de preprocesamiento y localización, las cuales consiguen eliminar toda la información innecesaria para la lectura dejando únicamente el código en una posición en la que se facilita su detección

al realizar el escaneo, con sus barras perpendiculares a la horizontal. Por lo tanto, tras dicho tratamiento, ZBar no tarda mucho tiempo en escanear la imagen y encontrar el código. Sin embargo, al aplicar ZBar directamente a la imagen, el lector se ve en la obligación de escanear un mayor número de píxeles al tener que buscar los símbolos del código en la imagen completa, lo que incrementa considerablemente el tiempo de lectura.

Imagen	**Tiempo lectura (ms)**	
	Algoritmo	**ZBar**
05102009082	15,35	95,27
05102009115	13,01	86,10
05102009139	11,04	50,47
05102009104	13,07	109,54
05102009108	17,75	75,64
05102009122	32,76	87,76
05102009170	22,85	47,35
05102009192	17,82	62,89
05102009201	14,78	53,46
Promedio	**17,60**	**73,94**

Tabla 5.1. Tiempo del algoritmo previo al definitivo en ejecutar la fase de lectura. Todas las imágenes poseen formato *.png*. Los valores máximos de cada columna aparecen sombreados en gris.

6 Resultados

Si buscas resultados diferentes,
no hagas siempre lo mismo.
Albert Einstein
(1879–1955)

Este capítulo recoge los resultados de las pruebas realizadas sobre el algoritmo definitivo con el fin de analizar el funcionamiento del mismo. Se ha medido tanto el tiempo de ejecución del algoritmo como el número de códigos que es capaz de leer correctamente con distintas orientaciones y condiciones de iluminación.

Para realizar las pruebas de la versión definitiva del algoritmo, al igual que se ha hecho con las pruebas parciales llevadas a cabo en los capítulos 4 y 5 se han utilizado las imágenes de la base de datos *no.2 plain*, la cual puede encontrarse en [Wwwp]. Además, dichas pruebas han sido ejecutadas sobre dos *hardwares* distintos: en el ordenador donde se ha desarrollado el código y en el microcontrolador sobre el que se implementará el lector. De esta forma pueden analizarse las diferencias de tiempos existentes entre ambos. Al igual que en los capítulos anteriores, los resultados han sido obtenidos tras la ejecución del programa en una máquina virtual creada con VirtualBox con el sistema operativo Ubuntu 16.04. Sus recursos asignados son 4 GB de memoria RAM y 2 hilos de procesador. La máquina virtual se ejecuta sobre un ordenador con procesador Intel® Core™ i7-4500U CPU @ 1.80 GHz (Turbo Boost hasta 2.40 GHz), el sistema operativo Windows 8.1 de 64 bits y 8 GB de memoria RAM. El tipo de tarjeta es Raspberry Pi Zero W, el cual posee un procesador mononúcleo de 1 GHz modelo ARM11 Broadcom y 512 GB de RAM. Su sistema operativo es Raspbian 9.4 (Stretch).

Las imágenes han sido divididas en dos conjuntos: entrenamiento y test. El grupo de entrenamiento sirve para ajustar los parámetros del algoritmo y ha ido empleando durante el desarrollo del mismo para probar su funcionamiento. Por otro lado, las imágenes de test se emplean únicamente para evaluar el rendimiento del algoritmo definitivo.

Se han llevado a cabo las pruebas para cada uno de estos conjuntos y, en función de los mismos, se han dividido los resultados obtenidos. Además, las imágenes han sido subdivididas a su vez en grupos atendiendo a sus características. Así, el conjunto de entrenamiento contiene

imágenes con buenas condiciones de iluminación y en las que la orientación del código es distinta de la de por defecto (esto es, las barras perpendiculares a la horizontal y los números situados de forma que se puedan leer de izquierda a derecha), otras imágenes con buenas condiciones de iluminación y la orientación del código por defecto y, el último grupo, con malas condiciones de iluminación y diversas orientaciones del código. Para el conjunto de test se dispone de los mismos tres grupos y, además, se añade un cuarto formado por imágenes que poseen algún defecto en el código.

Por otro lado, es importante tener en cuenta que todas las imágenes tomadas para realizar las pruebas poseen un único código y su resolución es idéntica, 640x480 píxeles. Los resultados se han clasificado en dos grandes secciones: eficacia y tiempo, atendiendo a los objetivos que se pretende que cumpla el algoritmo. Para la sección de eficacia se realizará el análisis de los resultados obtenidos de los conjuntos de imágenes de entrenamiento y test, con sus respectivas subdivisiones por características. Para las pruebas de tiempo, únicamente se han clasificado los resultados por las características de las imágenes. Esto se debe a que los parámetros temporales del algoritmo no se han ajustado primero para un conjunto concreto de imágenes sino que se ha hecho de manera general.

6.1. Resultados de eficacia

Los resultados que se presentan en esta sección son el producto de una serie de pruebas realizadas para medir la eficacia del algoritmo, es decir, la cantidad de imágenes en las que se consiguen leer los códigos frente a la cantidad de las mismas en las que no se realiza la lectura. Para realizar estas pruebas se ha ejecutado la versión final del algoritmo y se ha contabilizado el número de códigos leídos para cada conjunto. Como cada imagen posee un solo código de barras, cada lectura de una de ellas equivale a un código leído. Cada conjunto posee un total de trece imágenes a excepción de las que poseen un defecto en el código, cuyo total es seis.

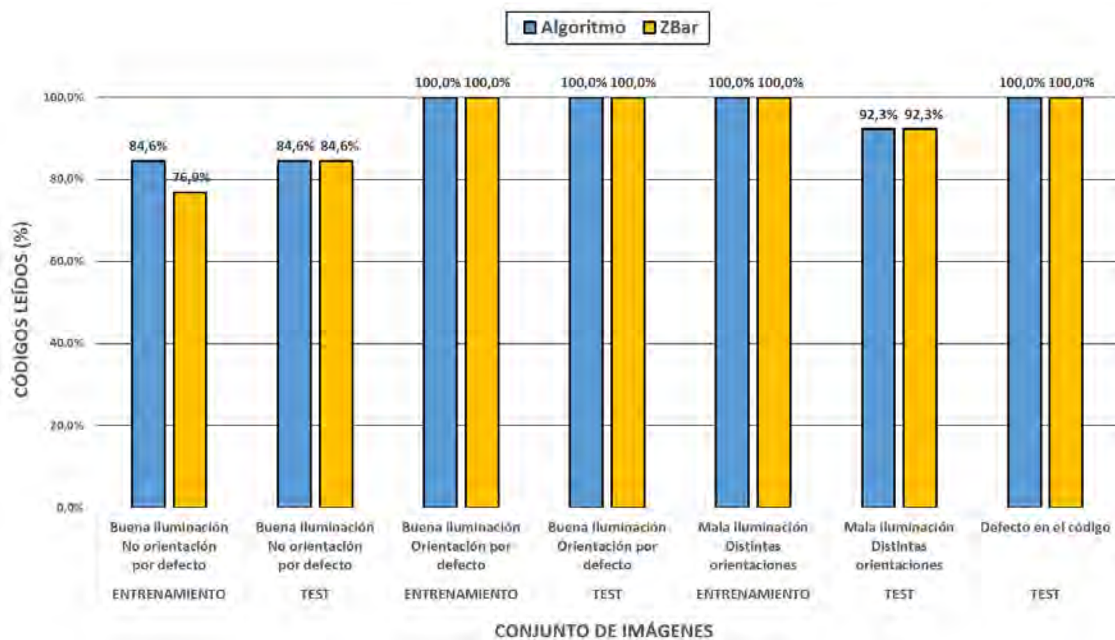


Figura 6.1. Histograma comparativo del porcentaje de códigos leídos para cada conjunto de imágenes por el algoritmo diseñado y por el algoritmo que únicamente aplica ZBar.

Cabe destacar, antes de analizar los resultados, que ZBar maneja automáticamente la verificación de los códigos, es decir, el decodificador nunca devolverá resultados incorrectos si al realizar la lectura del código y calcular la suma de verificación o *checksum*, ésta no coincide con el dígito de control.

La Figura 6.1 muestra un histograma cuyas barras se encuentran agrupadas por parejas atendiendo a los distintos conjuntos de imágenes, ya sean de entrenamiento o test y las características de dichas imágenes. Además, por cada pareja una de las barras es de color azul, representando el algoritmo desarrollado y otra de color amarillo, representando el algoritmo que aplica directamente ZBar. La altura de las barras indica el porcentaje total de códigos leídos para cada conjunto por el algoritmo correspondiente.

Figura 6.2. Ejemplos de imágenes con distintas condiciones de iluminación y orientación de los códigos. (1) Buena iluminación y orientación distinta a la de por defecto. (2) Buena iluminación y orientación por defecto. (3) Mala iluminación y orientación por defecto. (4) Imagen con defecto en el código.

Para las imágenes que poseen buenas condiciones de iluminación y una orientación diferente a la de por defecto como la (1) de la Figura 6.2, la primera y segunda pareja de barras del histograma muestran los resultados obtenidos en los conjuntos de entrenamiento y test, respectivamente. En el entrenamiento, se consigue la lectura del 84,6 % de los códigos con el algoritmo diseñado, porcentaje que se mantiene durante las pruebas de validación o test. Sin embargo, el algoritmo de ZBar consigue una eficacia un tanto menor durante el entrenamiento, de 76,9 % aunque en la validación alcanza el mismo porcentaje que el diseñado. En comparación con el resto de conjuntos de imágenes con otras características, este es el que presenta unos índices menores de códigos leídos. Podría deducirse, por tanto, que se posee una mayor dificultad al leer códigos que poseen distintas rotaciones que aquellos que poseen condiciones de mala iluminación, aunque la diferencia es muy baja. Esto puede deberse, por un lado, a que la

inclinación del código respecto a la horizontal es demasiado elevada con lo que ZBar al realizar el escaneo de la imagen en línea recta, no encuentra ningún patrón correspondiente al código de barras. En el caso del algoritmo desarrollado puede que el filtrado de ruido realizado mediante el cierre y la apertura de la imagen no haya conseguido eliminar toda la información innecesaria de la imagen con lo que no se detecta el contorno del código de barras. Para solucionar esto puede optarse por emplear un elemento estructural que se ajuste mejor a la posición del código en esta imágenes concretas, sin embargo, tras distintas pruebas, se comprobó que el elemento estructural elegido era el que mejor se adaptaba a la mayoría de las imágenes. Por lo tanto, una posible solución sería estudiar los casos concretos en los que no se consigue realizar la lectura y probar distintos tamaños de elementos estructurales hasta conseguir el más efectivo para dichas imágenes. El algoritmo entonces debería realizar dos iteraciones del proceso completo: la primera aplicando el elemento estructural más genérico, y, si no se consigue leer el código, una segunda con el elemento estructural más específico para dichos casos.

En cuanto a los resultados de las imágenes con buenas condiciones de iluminación y la orientación del código por defecto (imagen (2) de la Figura 6.2), éstos aparecen representados por la tercera y cuarta pareja de barras del histograma de la Figura 6.1. Es importante tener en cuenta que dichas condiciones de la imagen son las óptimas para realizar la lectura, con lo que se espera que el número de códigos leídos sea alto. Efectivamente, los resultados obtenidos tanto para el conjunto de imágenes de entrenamiento como para el de test, muestra una eficacia del 100 % en la lectura, tanto para el algoritmo diseñado, como para el que únicamente aplica ZBar.

La quinta y sexta pareja de barras del histograma de la Figura 6.1 representan el porcentaje de códigos leídos para imágenes que poseen malas condiciones de iluminación y cuyos códigos poseen distintas orientaciones. Un ejemplo de este tipo de condiciones se aprecian en la imagen (3) de la Figura 6.2. Los resultados obtenidos son similares para el algoritmo diseñado y para ZBar, tanto en el conjunto de entrenamiento como en el de test, obteniéndose en el primero un 100 % de códigos leídos y en el segundo un 92,3 %. Es este segundo resultado el que se toma como valor real de la eficiencia para imágenes con estas características, ya que en las pruebas de validación se mide la eficiencia del algoritmo definitivo. Además, es razonable que con malas condiciones de iluminación los algoritmos no consigan detectar y leer el código en el total de las situaciones, al no ser dichas condiciones las óptimas.

El último par de barras del histograma (Figura 6.1) muestra los resultados para un grupo de imágenes que poseen algún defecto en el código. Esto es, algunas líneas trazadas sobre los códigos con algún tipo de lápiz de color, como se muestra en la imagen (4) de la Figura 6.2. Este conjunto de imágenes se ha probado directamente sobre el algoritmo definitivo, puesto que no es uno de los casos principales a los que se someterá el algoritmo en su entorno de trabajo. Tanto el algoritmo diseñado como el que emplea ZBar directamente consiguen leer el total de los códigos en estos casos. Es un porcentaje alto debido a que no supone una elevada dificultad para ZBar este tipo de lectura, ya que al realizar el escaneo mediante varias líneas el código podría leerse código por una de las líneas que no atraviese la zona afectada.

Como observaciones generales, el algoritmo diseñado posee un elevado índice de códigos leídos para distintas condiciones de iluminación de la imagen y rotaciones del código, detectando y decodificando más del 80 % de los códigos del total de imágenes probadas. El algoritmo que aplica directamente el lector ZBar también supera dicho porcentaje en prácticamente todas las situaciones salvo para uno de los conjuntos de imágenes con buena iluminación y con la orientación del código distinta a la por defecto, en la que su eficacia es levemente menor. Podría

afirmarse, por lo tanto, que ambos algoritmos poseen una eficacia elevada, pues cumplen su objetivo: la detección y lectura correcta de los códigos, para la mayoría de los casos.

6.2. Resultados de tiempo

Los resultados que se presentan en esta sección son el producto de una serie de pruebas realizadas para medir el tiempo que tarda el algoritmo en ejecutar cada una de las fases en las que se divide. Para obtenerlos se ha ejecutado la versión final del algoritmo y se han recogido los tiempos en milisegundos que éste emplea en aplicar las distintas técnicas a cada una de las imágenes que forman los conjuntos. Además, se han realizado dichas pruebas sobre un algoritmo que aplica directamente el lector ZBar. De esta forma se pueden comparar los resultados obtenidos y extraer conclusiones más desarrolladas.

Tanto la ejecución del algoritmo definitivo como el algoritmo que únicamente aplica ZBar se ha realizado, en primer lugar, sobre el *hardware* del ordenador cuyas características se han sido especificadas anteriormente y, tras esto, en la Raspberry Pi Zero W. De esta forma se puede apreciar la diferencia de tiempos existente al ejecutar ambos algoritmos en distintos dispositivos y se obtienen resultados del funcionamiento del algoritmo en la Raspberry, donde será finalmente implementado.

6.2.1. Resultados en ordenador

Los tiempos parciales contabilizados para el algoritmo diseñado se han dividido en cuatro: carga de la imagen, preprocesamiento, localización y lectura. Para la lectura, en los casos que ésta se realiza primero mediante ZBar aplicado a la imagen recortada y rotada y, posteriormente, aplicando ZBar a toda la imagen, el tiempo de ejecución se ha desglosado en una suma. El primer sumando indica el tiempo empleado en realizar la primera lectura y el segundo sumando el empleado en realizar la segunda lectura. El algoritmo que aplica directamente ZBar posee dos fases previas a la lectura: la carga de la imagen y el escalado de grises. Estas también han sido contabilizadas individualmente de forma que se conozcan los tiempos parciales que dan lugar al tiempo total de ejecución de este algoritmo.

En este apartado se presentan cuatro tablas (Tabla 6.1-6.4) que muestran los resultados de los tiempos de ejecución del algoritmo final y de la aplicación directa de ZBar para un conjunto de imágenes con distintas características. Todas las tablas poseen el mismo formato: la primera columna recoge el nombre de los archivos de imagen, las siguientes cinco columnas corresponden al tiempo de ejecución de cada una de las fases del algoritmo diseñado junto con el tiempo total de ejecución, las cuatro sucesivas presentan los tiempos parciales y total de ejecución para el algoritmo que únicamente aplica ZBar y la columna final recoge la diferencia de tiempo en porcentaje entre ambos algoritmos. Es importante tener en cuenta que algunas celdas de dichas tablas contienen el símbolo “-”. Esto quiere decir que dichas imágenes no han sido leídas por el algoritmo con lo que, al no completar el proceso, su tiempo de ejecución no va a contabilizarse. Además, dichas imágenes tampoco se tienen en cuenta a la hora de realizar el promedio de los tiempos de ejecución de cada etapa de los algoritmos.

La Tabla 6.1 muestra los resultados obtenidos para un conjunto de imágenes que poseen buenas condiciones de iluminación y en las que el código se encuentra en una posición diferente a la por defecto. Destaca, en primer lugar, el hecho de que el código de la primera imagen del conjunto (05102009081) es leído por el algoritmo desarrollado pero no cuando se aplica únicamente ZBar. Por otro lado, los códigos de las imágenes 05102009103, 05102009105,

05102009159 y 05102009160 no son detectados por ninguno de los algoritmos. Estos resultados se corresponden con los analizados en la sección 6.1 del presente capítulo.

Por otro lado, destaca el caso de la imagen 05102009111 puesto que el tiempo que tarda en realizar la localización es muy bajo. Este hecho se da porque el algoritmo no ha encontrado ningún contorno con un área significativa con lo que no entra en el bucle que permite recortar dichas áreas, rotarlas y leerlas. Al no haber detectado ningún código, se aplica directamente la lectura con ZBar sobre toda la imagen, lo que, como ya se comentó en el apartado 5.2 conlleva una cantidad de tiempo más elevada que para la lectura del código extraído de la imagen. Existen tres casos en los que, además de intentar realizar la lectura de los contornos extraídos de la imagen, finalmente se debe aplicar ZBar a la imagen completa, al no ser posible la detección del código mediante la primera lectura.

El tiempo promedio de ejecución total del algoritmo diseñado de 49,74 milisegundos, aproximadamente 1,5 veces inferior al tiempo promedio total que emplearía una aplicación que solo utilizase ZBar, siendo este de 74,77 milisegundos. La mediana del total de tiempo empleado por el primer algoritmo es de 41,70, medida que es más exacta a la hora de representar la tendencia central de los resultados, puesto que no se ve tan afectada como la media por los valores atípicos. El valor máximo alcanzado por el algoritmo diseñado es 116,61. Para el segundo algoritmo la mediana es 71,92, muy cercano al valor de la media, y su máximo es 119,58. En la gráfica de la Figura 6.3 se pueden apreciar de manera visual las diferencias entre ambos algoritmos. En dicha gráfica no se ha incluido la imagen 05102009081 debido a que únicamente es leída por uno de los mismos. Efectivamente el algoritmo diseñado tarda menos tiempo en ejecutarse que el otro para la mayoría de los casos.

La Tabla 6.2 muestra los resultados de tiempo obtenidos para el conjunto de imágenes con buenas condiciones de iluminación y la orientación del código por defecto. Para dichas características de la imagen, el tiempo medio empleado por el algoritmo diseñado para realizar las fases de localización y la lectura aumenta ligeramente con respecto al caso anterior: 2,7 milisegundos para la primera y 3,36 para la segunda. Esto hace que el tiempo promedio total sea un poco más elevado.

Además, existen cuatro casos en los que el algoritmo realiza tanto la lectura de los rectángulos recortados como de toda la imagen con ZBar. Esto se debe a que se encuentran una serie de contornos, se recortan, se rotan y se leen, pero ninguno de ellos contenía al código de barras por lo que finalmente debe aplicarse ZBar a toda la imagen. Se aprecia también que para estos casos el tiempo empleado en realizar la lectura es más elevado, debido a que se aplica sobre varios o sobre todos los contornos con áreas más elevadas en busca de que alguno de ellos contenga al código. En la gráfica de la Figura 6.4 se observa cómo para los puntos que se corresponden con dichas imágenes aparecen picos que superan el tiempo empleado por ZBar, a excepción de la imagen PICT0001.

A pesar de lo comentado anteriormente que hace que el total de tiempo medio de ejecución del algoritmo aumente, este sigue siendo inferior al del algoritmo que aplica directamente ZBar, siendo 55,20 milisegundos lo que tarda el primero frente a los 78,01 milisegundos del segundo. Con esto puede decirse que el algoritmo tarda 1,4 veces menos en ejecutarse de media que el solo emplea ZBar, para este tipo de situaciones. El valor de la mediana para el primer algoritmo es de 45,06 y el máximo es 128.03 milisegundos. Para el segundo algoritmo la mediana es 72,18 milisegundos y el 124,64 el máximo.

BUENA ILUMINACIÓN - NO ORIENTACIÓN POR DEFECTO										
Imagen	Tiempo algoritmo (ms)					Tiempo ZBar (ms)				Diferencia (%)
	Carga	Preproces.	Localización	Lectura	**Total**	Carga	Escalado	Lectura	**Total**	
05102009081	12,00	4,47	14,14	12,07	**42,68**	-	-	-	**-**	-
05102009082	9,89	5,52	12,00	7,06	**34,47**	12,81	3,71	84,19	**100,71**	65,77
05102009103	-	-	-	-	**-**	-	-	-	**-**	-
05102009105	-	-	-	-	**-**	-	-	-	**-**	-
05102009110	11,21	4,96	10,95	3,46	**30,58**	10,76	3,99	104,83	**119,58**	74,43
05102009111	11,09	4,05	2,55	98,93	**116,61**	10,02	3,43	75,86	**89,31**	-30,57
05102009115	10,32	5,67	10,26	11,14	**37,39**	11,42	3,23	85,03	**99,68**	62,49
05102009117	11,32	3,69	8,59	13,03	**36,63**	11,44	3,16	86,19	**100,79**	63,66
05102009119	11,03	5,55	11,56	12,53	**40,67**	12,05	4,66	79,16	**95,86**	57,58
05102009120	10,55	5,87	10,79	10,40	**37,60**	9,99	3,34	74,93	**88,25**	57,39
05102009121	10,86	5,46	13,84	11,47	**41,64**	11,25	3,60	82,01	**96,86**	57,01
05102009123	12,15	4,51	9,04	19,28	**44,97**	10,37	3,32	66,09	**79,78**	43,64
05102009139	11,79	4,29	9,45	14,04	**39,57**	12,36	5,62	53,62	**71,60**	44,73
05102009145	8,88	4,49	10,71	24,70	**48,77**	11,00	3,68	49,53	**64,22**	24,05
05102009146	11,60	6,26	12,71	22,72	**53,29**	8,87	3,42	42,27	**54,55**	2,31
05102009155	9,35	4,62	11,64	35,75 (7,72+28,03)	**61,36**	8,91	3,60	35,78	**48,29**	-27,05
05102009156	11,33	4,50	34,55	24,53	**74,91**	9,44	5,66	30,89	**45,98**	-62,90
05102009157	9,36	4,61	11,47	14,86	**40,31**	9,33	3,19	44,51	**57,03**	29,33
05102009159	-	-	-	-	**-**	-	-	-	**-**	-
05102009160	-	-	-	-	**-**	-	-	-	**-**	-
05102009161	12,25	5,66	13,52	10,27	**41,70**	11,88	3,73	56,62	**72,23**	42,26
05102009223	9,14	4,26	11,55	41,55 (8,39+33,16)	**66,50**	13,38	4,24	49,68	**67,30**	1,19
05102009224	10,41	3,81	12,69	57,86 (17,70+40,16)	**84,77**	10,53	4,64	39,15	**54,32**	-56,06
06102009249	9,62	4,48	10,51	11,76	**36,36**	11,54	5,16	49,60	**66,30**	45,16
06102009250	8,24	4,36	10,45	13,83	**36,89**	8,15	3,35	35,71	**47,21**	21,87
06102009252	12,46	4,35	9,35	20,48	**46,63**	9,22	3,42	37,58	**50,23**	7,16
Promedio	**10,67**	**4,79**	**11,92**	**22,35**	**49,74**	**10,70**	**3,91**	**60,15**	**74,77**	**33,47**
Mediana	**10,86**	**4,51**	**11,47**	**14,04**	**41,70**	**10,73**	**3,64**	**55,12**	**71,92**	**42,01**

Tabla 6.1. Resultados de tiempo del algoritmo definitivo y de la aplicación directa de ZBar para imágenes con buenas condiciones de iluminación y orientaciones distintas a la de por defecto. *Preproces.* se refiere a la fase de preprocesamiento. La diferencia está calculada de acuerdo a la expresión $\frac{ZBar - Algoritmo}{ZBar} \cdot 100$. Todas las imágenes poseen formato *.png*. Los paréntesis indican que se ha tenido que leer dos veces y las celdas sombreadas contienen los máximos de cada columna.

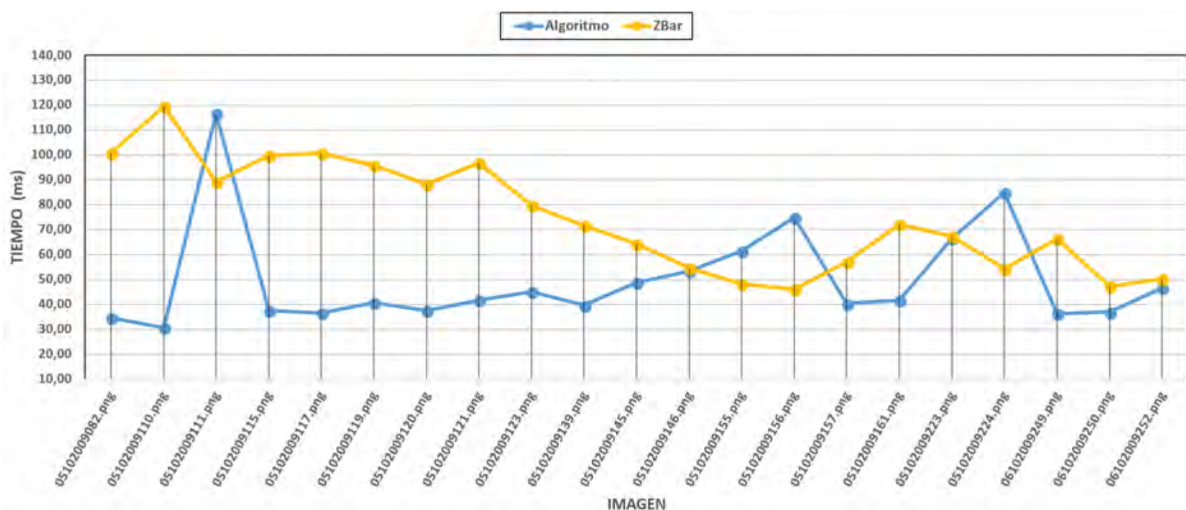


Figura 6.3. Gráfico comparativo del tiempo total de ejecución de las imágenes con buenas condiciones de iluminación y orientaciones del código distintas a la por defecto por el algoritmo diseñado y por el algoritmo que únicamente aplica ZBar.

BUENA ILUMINACIÓN - ORIENTACIÓN POR DEFECTO										
Imagen	**Tiempo algoritmo (ms)**					**Tiempo ZBar (ms)**				Diferencia (%)
	Carga	Preproces.	Localización	Lectura	**Total**	Carga	Escalado	Lectura	**Total**	
05102009083	11,93	5,74	15,61	9,60	**42,89**	10,24	3,44	58,50	**72,18**	40,59
05102009101	9,98	4,51	11,61	2,73	**28,84**	10,10	3,31	81,69	**95,10**	69,68
05102009102	10,24	4,37	4,62	89,86	**109,09**	9,91	3,39	99,37	**112,68**	3,19
05102009104	9,76	4,40	10,78	4,73	**29,67**	9,73	3,41	109,25	**122,39**	75,75
05102009106	10,79	4,45	25,21	87,58 (9,59+77,99)	**128,03**	9,56	3,39	81,52	**94,47**	-35,52
05102009108	12,41	5,66	14,33	7,69	**40,08**	10,53	3,70	87,90	**102,12**	60,75
05102009114	11,18	3,66	11,40	12,36	**38,60**	10,46	3,41	67,11	**80,98**	52,33
05102009118	10,25	4,29	10,42	8,54	**33,49**	11,34	3,52	94,35	**109,20**	69,33
05102009122	10,15	4,34	11,38	15,38	**41,24**	10,34	3,37	86,71	**100,42**	58,93
05102009142	8,14	4,44	17,42	36,98 (3,75+33,23)	**66,98**	7,93	3,32	33,34	**44,59**	-50,21
05102009153	8,12	4,33	16,20	10,86	**39,51**	10,07	4,03	30,65	**44,75**	11,72
05102009164	9,98	5,61	23,17	11,77	**50,53**	12,50	5,29	52,23	**70,02**	27,84
05102009178	9,64	4,56	10,42	17,92	**42,54**	9,69	3,58	46,62	**59,89**	28,98
05102009158	11,36	5,82	13,97	9,59	**40,74**	10,56	3,54	80,25	**94,35**	56,82
05102009206	10,50	4,85	30,54	57,77	**103,65**	9,70	3,51	73,77	**86,98**	-19,17
05102009207	9,15	4,36	10,48	51,72 (1,13+50,59)	**75,70**	10,65	3,43	50,46	**64,54**	-17,29
05102009208	8,77	4,31	11,56	8,44	**33,08**	9,14	3,40	42,05	**54,59**	39,41
05102009225	9,98	4,47	3,42	40,78	**58,65**	11,62	5,20	43,04	**59,86**	2,02
06102009240	9,07	5,64	17,41	13,88	**46,00**	10,15	4,15	34,30	**48,61**	5,36
06102009247	10,31	4,72	10,27	9,36	**34,66**	9,95	3,19	43,92	**57,06**	39,27
IMAG0397	11,19	5,59	15,49	12,80	**45,06**	8,80	3,27	26,18	**38,26**	-17,78
PICT0001	9,89	4,36	9,72	78,92 (3,06+75,86)	**102,89**	11,45	3,42	109,77	**124,64**	17,45
PICT0002	10,97	4,93	20,82	19,77	**56,49**	11,91	3,96	55,14	**71,01**	20,44
PICT0002a	9,54	4,49	16,33	21,26	**51,62**	11,01	9,23	70,86	**91,11**	43,44
PICT0003	10,19	4,81	20,61	16,14	**51,75**	10,14	3,36	50,68	**64,18**	19,37
PICT0004	10,07	4,35	17,00	11,93	**43,35**	10,18	3,67	50,41	**64,26**	32,54
Promedio	**10,14**	**4,73**	**14,62**	**25,71**	**55,20**	**10,29**	**3,87**	**63,85**	**78,01**	**29,24**
Mediana	**10,14**	**4,49**	**14,33**	**13,88**	**45,06**	**10,18**	**3,44**	**58,50**	**72,18**	**37,57**

Tabla 6.2. Resultados de tiempo del algoritmo definitivo y de la aplicación directa de ZBar para imágenes con buenas condiciones de iluminación y orientaciones iguales a la de por defecto. *Preproces.* se refiere a la fase de preprocesamiento. La diferencia está calculada de acuerdo a la expresión $\frac{ZBar-Algoritmo}{ZBar}\cdot 100$. Todas las imágenes poseen formato *.png*. Los paréntesis indican que se ha tenido que leer dos veces y las celdas sombreadas contienen los máximos de cada columna.

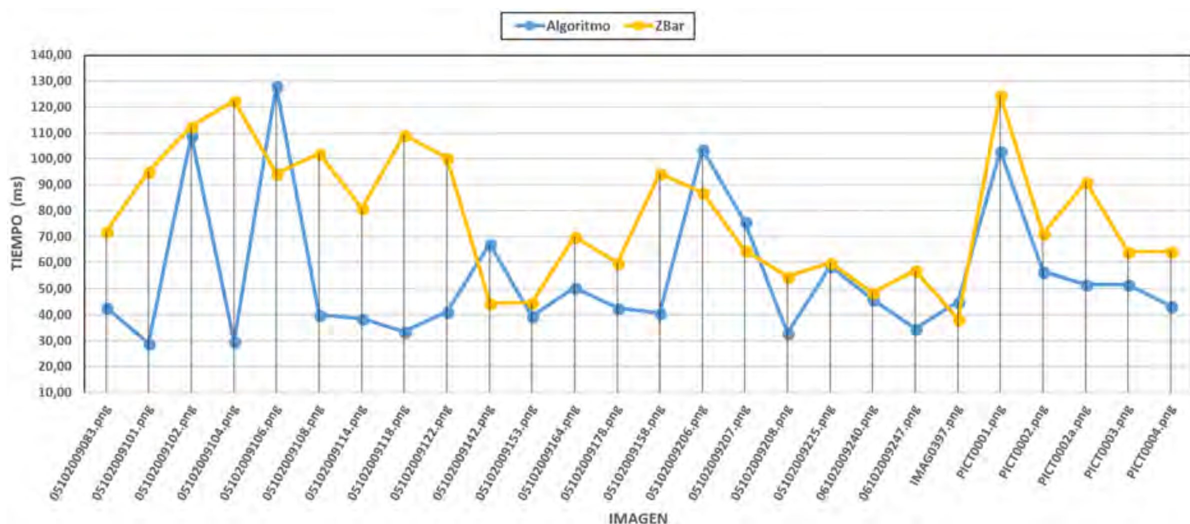


Figura 6.4. Gráfico comparativo del tiempo total de ejecución de las imágenes con buenas condiciones de iluminación y orientaciones del código por defecto por el algoritmo diseñado y por el algoritmo que únicamente aplica ZBar.

La Tabla 6.3 recoge los resultados del conjunto de imágenes con mala iluminación y distintas orientaciones de los códigos. Para estas condiciones el algoritmo aplica las dos lecturas en diez ocasiones. Esto nuevamente se traduce en un aumento del tiempo total de ejecución del algoritmo, siendo el promedio en este caso de 69.90 milisegundos. Es un aumento considerable respecto a los casos en los que las imágenes poseen buenas condiciones de iluminación si tenemos en cuenta que se ha incrementado en torno a 14,70 milisegundos respecto a cuando el la posición del código es la de por defecto y 20,16 milisegundos respecto a las ocasiones en las que la posición es distinta a la de por defecto. Observando la gráfica de la Figura 6.5 se aprecia que para la mayoría de las imágenes, el algoritmo diseñado tarda más en ejecutarse que el otro.

Así, el algoritmo desarrollado tarda más ejecutarse que el algoritmo que aplica ZBar de manera directa, cuyo tiempo se ha visto a su vez reducido considerablemente, empleando 59,27 milisegundos para completar su ejecución. De esta forma, el algoritmo diseñado en este caso tarda de media 1,18 veces más en ejecutarse que el otro. Esto supone una pérdida en la eficacia obtenida hasta el momento pero no demasiado elevada. El valor de la mediana para el algoritmo diseñado es de 67,21 y el máximo es 125,38 milisegundos. Para el otro algoritmo la mediana es 57,37 y 91,36 milisegundos el valor máximo.

La Tabla 6.4 muestra los resultados obtenidos de las pruebas de tiempo para imágenes con algún defecto en el código como un trazo horizontal sobre el mismo. De este conjunto únicamente se poseen seis imágenes, de las cuales, la mitad debe realizar dos lecturas para conseguir realizar la decodificación. Esto sitúa a dicho conjunto en el que es más costoso detectar y leer los códigos, lo que se traduce en un tiempo total medio de 74,42 milisegundos, el más elevado de todos. En la Figura 6.6 puede apreciarse gráficamente dicha pérdida en la eficiencia puesto que los valores temporales han aumentado en la escala. Destaca, sin embargo, el caso de la imagen 05102009214 en la que el algoritmo diseñado tarda un 77,42 % menos en ejecutarse que el otro, siendo particularmente eficiente.

Por otro lado, el tiempo total promedio empleado por el algoritmo que únicamente aplica ZBar es también el más elevado, siendo de 78,78 milisegundos. A pesar de estos incrementos, el algoritmo diseñado sigue siendo un poco más eficiente que el otro, en torno a 1,06 veces aproximadamente. El valor de la mediana es de 74,00 para el algoritmo diseñado y su máximo es 129,44 milisegundos. Para el otro algoritmo la mediana es 65,90 y el valor máximo 160,19 milisegundos.

Como conclusiones generales, las condiciones en las que el algoritmo encuentra más dificultades a la hora de leer los códigos son cuando la iluminación es mala y cuando dicho código posee algún defecto. Para estos casos se necesita con más frecuencia realizar las dos lecturas por lo que el tiempo medio total aumenta. Por otro lado, si se calcula la media de los tiempos totales empleados para la ejecución de todas las imágenes independientemente del conjunto al que pertenezcan, se obtiene un valor de 62,01 milisegundos para el algoritmo desarrollado frente a 74,14 milisegundos para el algoritmo que aplica directamente ZBar sobre la imagen, siendo este 1,2 veces superior. De estos resultados puede afirmarse que el algoritmo desarrollado es más eficiente, existiendo una diferencia media aproximada de 12,13 milisegundos en cada ejecución o de un 16,36 %.

MALA ILUMINACIÓN - DISTINTAS ORIENTACIONES										
Imagen	**Tiempo algoritmo (ms)**					**Tiempo ZBar (ms)**				Diferencia (%)
	Carga	Preproces.	Localización	Lectura	**Total**	Carga	Escalado	Lectura	**Total**	
05102009169	11,10	6,50	22,63	21,35	**61,57**	9,03	3,34	38,30	**50,68**	-21,49
05102009170	10,44	4,40	11,24	29,50	**55,57**	9,74	3,39	44,83	**57,96**	4,13
05102009171	10,58	4,05	15,74	56,53 (12,52+44,01)	**86,90**	9,40	3,42	38,54	**51,36**	-69,21
05102009172	11,34	7,31	7,77	46,27	**72,69**	10,52	3,31	48,03	**61,86**	-17,51
05102009173	9,98	4,32	10,77	9,44	**34,51**	10,14	3,83	46,53	**60,50**	42,97
05102009190	9,36	4,46	10,60	68,91 (14,01+54,9)	**93,33**	10,60	3,48	53,04	**67,13**	-39,04
05102009191	10,63	4,50	10,61	77,65 (14,05+63,60)	**103,39**	12,51	3,53	66,41	**82,46**	-25,38
05102009192	10,64	4,44	10,95	73,70 (16,69+57,01)	**99,73**	10,47	3,43	59,18	**73,09**	-36,46
05102009193	10,01	4,41	10,55	18,05	**43,02**	11,23	4,52	75,32	**91,07**	52,77
05102009195	9,20	4,54	25,23	68,20 (24,68+43,52)	**107,17**	9,51	3,42	43,84	**56,77**	-88,79
05102009201	9,64	5,85	13,38	30,92	**59,79**	10,55	3,34	51,39	**65,28**	8,41
05102009202	8,88	4,74	16,76	29,90	**60,28**	9,98	3,40	42,11	**55,49**	-8,63
IMAG0399	9,88	4,65	16,67	17,90	**49,10**	9,87	3,97	24,92	**38,76**	-26,66
IMAG0400	10,03	5,67	31,96	20,65	**68,31**	10,58	4,78	37,05	**52,41**	-30,34
IMAG0403	10,68	4,95	10,74	5,75	**32,12**	11,13	3,42	38,33	**52,88**	39,27
IMAG0404	9,19	4,37	9,48	10,03	**33,06**	9,17	3,45	28,67	**41,29**	19,92
IMAG0405	-	-	-	-	**-**	-	-	-	**-**	-
IMAG0408	8,60	5,51	12,87	11,76	**38,74**	8,67	3,34	28,56	**40,57**	4,52
IMAG0411	10,57	4,89	29,67	24,68	**69,81**	11,13	6,05	51,22	**68,40**	-2,06
IMAG0413	9,75	5,35	19,12	45,69 (10,05+35,64)	**79,91**	9,77	4,62	34,59	**48,98**	-63,13
IMAG0426	11,81	3,76	35,79	74,03 (16,49+57,54)	**125,38**	13,26	3,54	51,56	**68,36**	-83,42
IMAG0427	10,50	3,88	8,88	65,44 (3,36+62,08)	**88,69**	10,05	4,46	76,85	**91,36**	2,92
IMAG0428	9,37	4,73	18,60	29,71	**62,41**	10,09	3,33	47,68	**61,10**	-2,14
IMAG0429	9,86	4,46	28,61	69,88 (21,8+48,08)	**112,81**	9,88	3,72	40,72	**54,33**	-107,64
IMAG0432	8,53	4,99	16,83	35,76 (4,54+31,22)	**66,10**	8,63	3,29	41,23	**53,15**	-24,37
IMAG0442	11,29	4,82	16,95	10,05	**43,11**	10,92	4,77	38,36	**54,05**	20,25
Promedio	**10,07**	**4,86**	**16,90**	**38,07**	**69,90**	**10,27**	**3,81**	**45,89**	**59,27**	-16,56
Mediana	**10,02**	**4,69**	**16,21**	**30,41**	**67,21**	**10,12**	**3,46**	**44,34**	**57,37**	-17,16

Tabla 6.3. Resultados de tiempo del entrenamiento del algoritmo definitivo y de la aplicación directa de ZBar para imágenes con malas condiciones de iluminación y distintas orientaciones. *Preproces.* se refiere a la fase de preprocesamiento. Todas las imágenes poseen formato *.png*. La diferencia está calculada de acuerdo a la expresión $\frac{ZBar - Algoritmo}{ZBar} \cdot 100$. Los paréntesis indican que se ha tenido que leer dos veces y las celdas sombreadas contienen los máximos de cada columna.

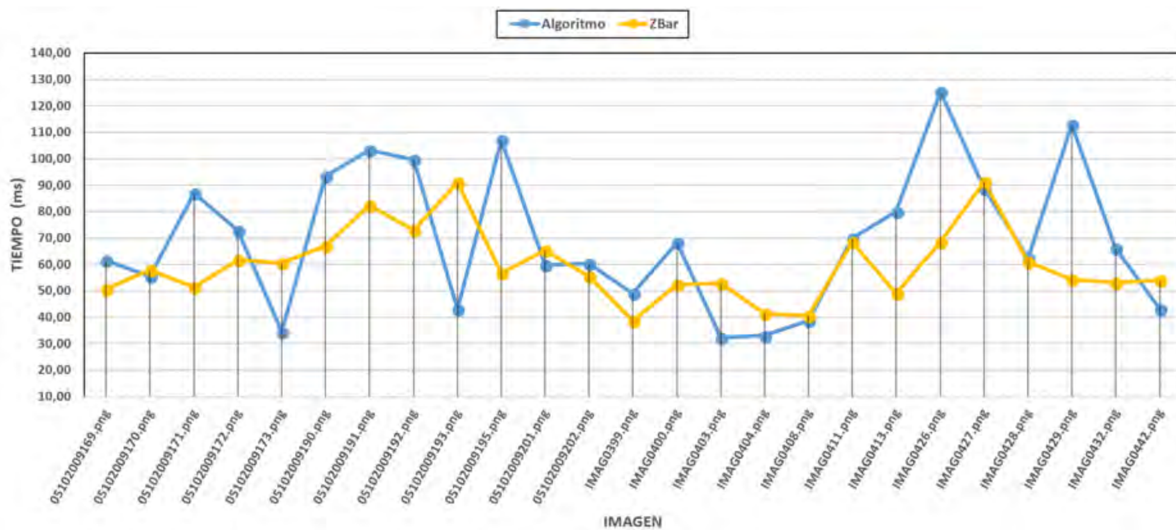


Figura 6.5. Gráfico comparativo del tiempo total de ejecución de las imágenes con malas condiciones de iluminación y distintas orientaciones del código por el algoritmo diseñado y por el algoritmo que únicamente aplica ZBar.

DEFECTO EN EL CÓDIGO										
Imagen	Tiempo algoritmo (ms)					Tiempo ZBar (ms)				Diferencia (%)
	Carga	Preproces.	Localización	Lectura	**Total**	Carga	Escalado	Lectura	**Total**	
05102009212	8,57	4,49	10,54	11,18	**34,77**	8,68	4,53	44,03	**57,24**	39,25
05102009213	9,90	3,72	15,88	99,94 (31,95+67,99)	**129,44**	9,15	3,48	56,09	**68,71**	-88,38
05102009214	9,09	4,47	12,57	10,05	**36,18**	10,65	5,03	144,51	**160,19**	77,42
05102009219	9,80	4,29	16,62	63,46 (22,47+40,99)	**94,16**	9,28	3,78	41,67	**54,74**	-72,03
05102009220	9,80	3,81	9,32	75,18 (10,23+64,95)	**98,11**	9,14	4,03	51,24	**64,41**	-52,32
05102009222	9,33	4,37	16,33	23,79	**53,83**	9,11	9,49	48,79	**67,39**	20,12
Promedio	**9,41**	**4,19**	**13,54**	**47,27**	74,42	**9,33**	**5,06**	**64,39**	**78,78**	**5,54**
Mediana	**9,56**	**4,33**	**14,22**	**43,63**	74,00	**9,14**	**4,28**	**50,01**	**65,90**	**-12,29**

Tabla 6.4. Resultados de tiempo del algoritmo definitivo y de la aplicación directa de ZBar para imágenes que poseen un defecto en el código de barras. *Preproces.* se refiere a la fase de preprocesamiento. La diferencia está calculada de acuerdo a la expresión $\frac{ZBar-Algoritmo}{ZBar} \cdot 100$. Todas las imágenes poseen formato *.png*. Los paréntesis indican que se ha tenido que leer dos veces y las celdas sombreadas contienen los máximos de cada columna.

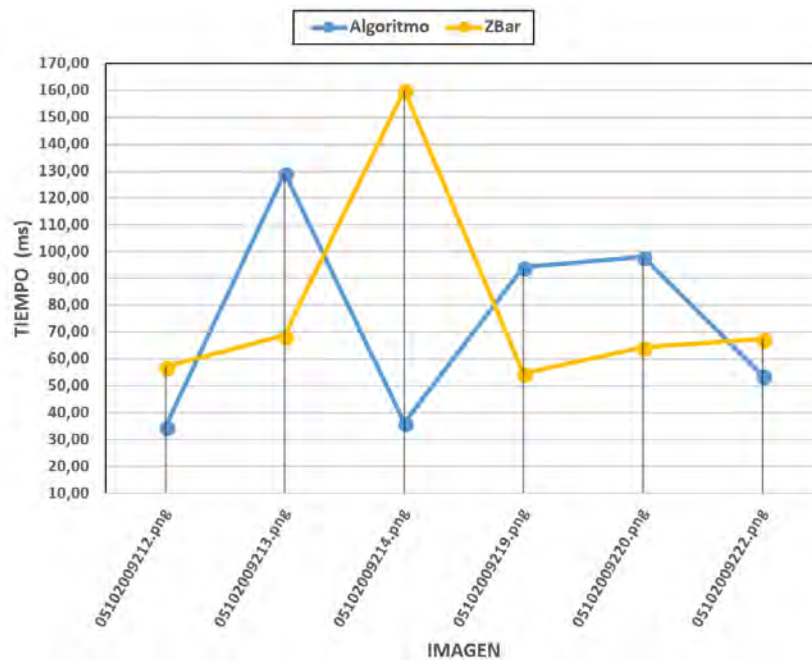


Figura 6.6. Gráfico comparativo del tiempo total de ejecución de las imágenes con defectos en el código por el algoritmo diseñado y por el algoritmo que únicamente aplica ZBar.

6.2.2. Resultados en la Raspberry Pi Zero W

Los resultados que se presentan en esta sección han sido obtenidos realizando la misma serie de pruebas que en el apartado anterior (6.2.1) solo que en lugar de ejecutarlas en el ordenador, han sido ejecutadas en la Raspberry Pi Zero W. De esta forma, se puede observar el tiempo real que tardará en ejecutarse el algoritmo diseñado ya que su implementación final será sobre Raspberry y compararlo con el que emplea el algoritmo que únicamente aplica ZBar. Dichos resultados se muestran en el gráfico de la Figura 6.7 en cuyo eje horizontal aparece el nombre de la imagen y en el eje vertical el tiempo total que tardan ambos algoritmos en ejecutarse para cada imagen.

En primer lugar, cabe destacar que la gráfica se limita a mostrar todas aquellas imágenes que son leídas correctamente por ambos algoritmos. De esta forma, la gráfica muestra los resultados para un total de 78 imágenes leídas por ambos algoritmos, de las cuales, 53 poseen un mayor tiempo de ejecución total del algoritmo diseñado, dejando 25 en las que el algoritmo de solo ZBar tarda más en ejecutarse. Es decir, aproximadamente en un 32,1 % de los casos, el algoritmo diseñado es más eficiente, mientras que para un 67,9 % de las imágenes, resulta más óptimo el algoritmo que únicamente aplica ZBar.

La mediana del tiempo total del algoritmo diseñado es de 661,70 y su valor máximo es 1621,20 milisegundos, mientras que para el otro algoritmo la mediana es 616,84 y 1172,16 milisegundos el máximo alcanzado. Por otro lado, el promedio del tiempo total de ejecución del algoritmo desarrollado es de 787,00 milisegundos frente a 654,83 milisegundos del otro algoritmo. Puede apreciarse por tanto, que al realizar la ejecución en el microcontrolador, el lector diseñado pierde eficiencia, tardando 1,2 veces más en ejecutarse que el que solo emplea ZBar. Dicha pérdida se debe a un aumento considerable del tiempo de ejecución en las fases de preprocesamiento y localización, pero sobre todo, a la primera de ellas. El tiempo medio de la fase de preprocesamiento para el algoritmo diseñado es de 146,30 milisegundos frente a 25,88 milisegundos que tarda de media el otro algoritmo en realizar el escalado de grises. Este dato resulta bastante llamativo ya que en las pruebas realizadas por ordenador, ambas magnitudes eran similares. La fase de localización posee un promedio de tiempo de ejecución de 241,98 milisegundos. Estos notables aumentos podrían estar causados por la librería OpenCV, puesto que ésta podría estar aprovechando que en el ordenador se posee más de un núcleo de procesador, mientras que en la Raspberry es mononúcleo. Por lo tanto, las fases que realizan un mayor número de operaciones con OpenCV son las que más afectadas se ven al ejecutarse en el otro *hardware*. Sin embargo, la fase de lectura del algoritmo diseñado, tarda de media 314,21 milisegundos frente a 544,04 milisegundos que emplea el algoritmo de ZBar en leer el código. Este resultado sí coincide con lo observado hasta el momento en los resultados obtenidos: la lectura realizada por el algoritmo diseñada es más óptima que la lectura aplicando directamente ZBar a toda la imagen. Además, este hecho confirma lo deducido anteriormente, al ser el proceso de lectura uno en el que no se emplea OpenCV, el aumento en el tiempo de ejecución del ordenador al microcontrolador no es tan elevado.

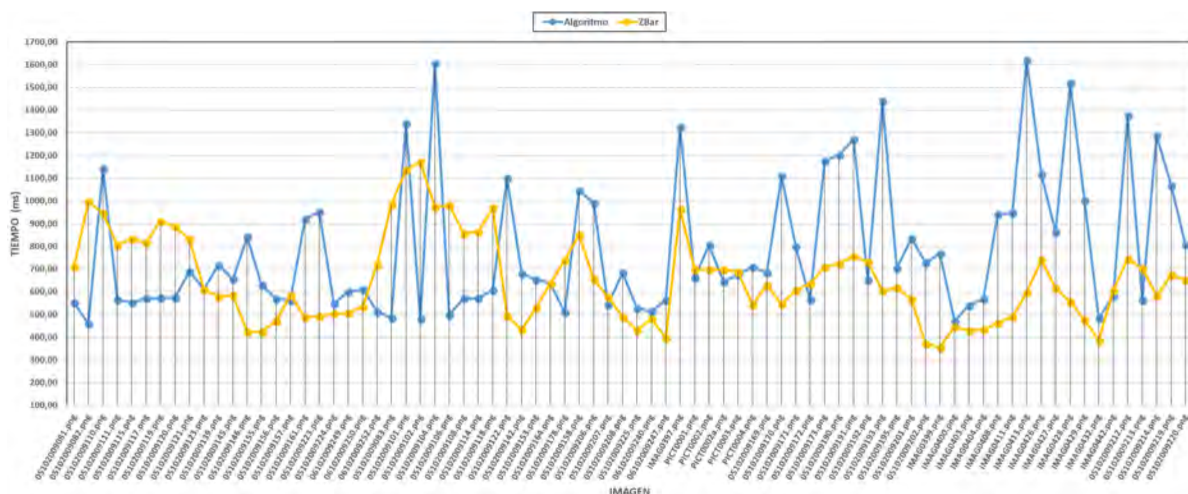


Figura 6.7. Gráfico comparativo del tiempo total de ejecución de cada imagen en la Raspberry Pi Zero W por el algoritmo diseñado y por el algoritmo que únicamente aplica ZBar.

6.3. Pruebas con códigos QR

Para la lectura de los códigos QR se ha aplicado el mismo algoritmo que ha sido diseñado para los códigos de barras con el fin de comprobar su funcionamiento en estos casos. Se ha tomado una muestra de seis imágenes, tres de las cuales poseen buenas condiciones de iluminación y distintas orientaciones del código y las tres restantes malas condiciones de iluminación e igualmente diferentes orientaciones. Las imágenes poseen formato .jpg y han sido extraídas de la librería *qr-dataset* descargada de [Wwwi]. Igualmente, dichas pruebas han sido ejecutadas en el mismo ordenador en el que fueron realizadas las anteriores. La Tabla 6.5 muestra los resultados obtenidos.

En cuanto a la eficacia del algoritmo, este consigue leer cinco de las seis imágenes, lo que supone un 83,33 % del total, aproximadamente similar al porcentaje de códigos de barras leídos. En cuanto al tiempo de ejecución empleado, éste aumenta considerablemente siendo el promedio 908,37 milisegundos para el algoritmo diseñado y 1273,46 para el otro algoritmo. Esto supone una pérdida significativa de eficiencia al compararlo con el tiempo de ejecución del código de barras, ya que éste último es 14,64 veces menor para el algoritmo diseñado y 16,7 veces menor para el algoritmo que aplica ZBar directamente. Cabe destacar que, para el caso de esta librería el tiempo promedio de la carga de las imágenes aumenta considerablemente ya que éstas poseen una resolución de 2560x1440 píxeles, mayor que las de la librería de los códigos de barras, con lo que es más costoso leerlas completas. A pesar de ello, observado las medias de los tiempos totales de cada fase, se aprecia que la pérdida de eficacia se produce en todas ellas. La mediana para el algoritmo diseñado es 935,79 y su máximo es 1007,27 milisegundos mientras que para el otro, estas estadísticas toman los valores 1554,80 y 1973,18.

Por lo tanto, la eficacia para leer códigos QR del algoritmo desarrollado es bastante elevada pero no es demasiado eficiente. Sin embargo, para ejecutarlo en el ordenador es preferible a otro que aplique directamente ZBar. En la Raspberry Pi Zero W, sería indistinto emplear un algoritmo a otro ya que no existe una gran diferencia entre ambos, pero quizás se comprometería el funcionamiento del dron ya que el valor máximo de ambos supera el tiempo de un segundo.

BUENA/MALA ILUMINACIÓN Y DISTINTAS ORIENTACIONES										
	Tiempo algoritmo (ms)					Tiempo ZBar (ms)				
Imagen	Carga	Preproces.	Localización	Lectura	**Total**	Carga	Escalado	Lectura	**Total**	Diferencia (%)
IMG_2710	170,95	138,68	605,31	20,85	**935,79**	165,14	66,74	1322,92	**1554,80**	39,81
IMG_2716	267,98	100,06	505,86	65,20	**939,09**	216,72	79,51	1363,88	**1660,11**	43,43
IMG_2717	197,64	128,20	575,62	105,82	**1007,27**	180,50	67,07	1725,61	**1973,18**	48,95
20110817_096	42,18	41,00	199,86	628,05 (5,48+622,57)	**911,10**	39,04	19,73	530,18	**588,95**	54,70
20110817_106	40,37	30,96	142,80	534,44 (10,84+523,60)	**748,57**	39,23	19,44	531,61	**590,29**	-26,82
20110817_143	-	-	-	-	-	-	-	-	-	-
Promedio	**143,82**	**87,78**	**405,89**	**270,87**	**908,37**	**128,13**	**50,50**	**1094,84**	**1273,46**	**28,67**
Mediana	**170,95**	**100,06**	**505,86**	**105,82**	**935,79**	**165,14**	**66,74**	**1322,92**	**1554,80**	**39,81**

Tabla 6.5. Resultados de tiempo del algoritmo definitivo y de la aplicación directa de ZBar para imágenes con códigos QR y buena iluminación y distintas orientaciones del código y para imágenes con mala iluminación e igualmente diferentes orientaciones. *Preproces.* se refiere a la fase de preprocesamiento. La diferencia está calculada de acuerdo a la expresión $\frac{ZBar - Algoritmo}{ZBar} \cdot 100$. Todas las imágenes poseen formato *.jpg*. Los paréntesis indican que se ha tenido que leer dos veces y las celdas sombreadas contienen los máximos de cada columna.

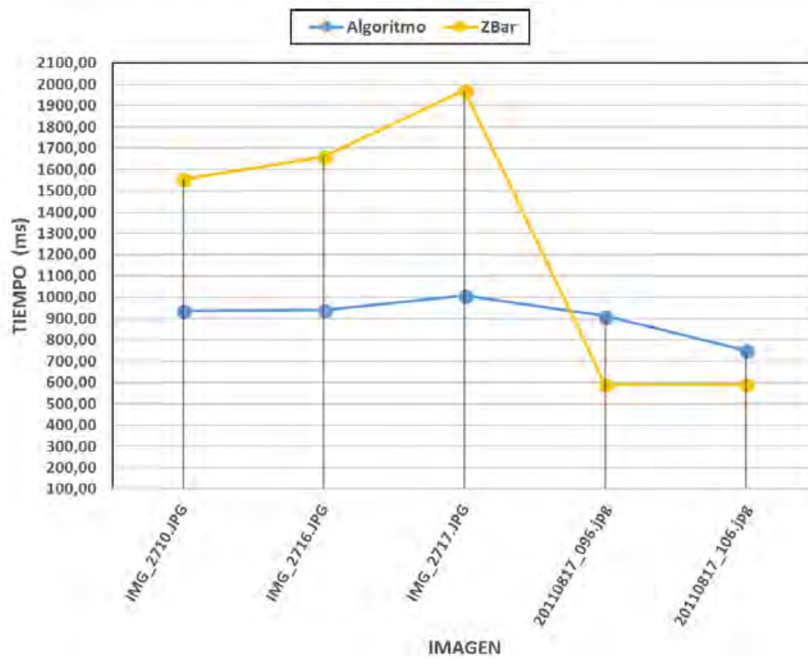


Figura 6.8. Gráfico comparativo del tiempo total de ejecución de las imágenes con buena/mala iluminación y distintas orientaciones del código QR por el algoritmo diseñado y por el algoritmo que únicamente aplica ZBar.

7 Conclusiones

Es mejor debatir una cuestión sin llegar a concluirla, que llegar a una conclusión sin debatirla.
Joseph Joubert
(1754–1824)

El presente capítulo expone las conclusiones finales del proyecto extraídas a partir de los resultados obtenidos de las diferentes pruebas realizadas tanto del algoritmo desarrollado como del algoritmo que realiza la lectura con ZBar directamente.

Se ha desarrollado un algoritmo de detección de códigos de barras que evita tener que leer la imagen completa para encontrar el código, el cual mejora ligeramente la eficacia en la decodificación. El algoritmo diseñado consigue leer más de un 80 % del total de imágenes para las distintas condiciones de iluminación y orientaciones del código de barras, lo que supone un porcentaje bastante elevado. Además, dicho porcentaje siempre será similar o superior al que se consiga con el algoritmo que aplica ZBar directamente (en este caso, consigue leer un 3,1 % más del total de las imágenes que el otro). Esto se debe al hecho de que el algoritmo diseñado primero realiza un preprocesamiento y una localización que ayudan a la detección del código de barras y, en el caso de que no se consiga leer el código, aplica directamente ZBar a toda la imagen. Por lo tanto, puede apreciarse que el algoritmo diseñado, toma el otro algoritmo y lo mejora añadiéndole un proceso previo de detección y lectura, con lo que puede concluirse que conseguirá leer, como mínimo, todas las imágenes que lea un algoritmo que directamente aplique ZBar. Esto hace al algoritmo diseñado más eficaz que cualquiera que aplique ZBar directamente.

Se ha demostrado que dicho algoritmo es más rápido en sistemas con más de un núcleo pero no así en aquellos con un solo núcleo. De esta forma, el segundo algoritmo tarda aproximadamente 1,2 veces más en ejecutarse en el ordenador que el algoritmo diseñado mientras que éste tarda 1,2 veces más en ejecutarse en la Raspberry que el primero. Por ello, puede afirmarse que el algoritmo es más óptimo si ejecuta en dispositivos que cuenten con más de un núcleo de procesador mientras que un algoritmo que aplica directamente ZBar es más

eficiente para dispositivos mononúcleo. El algoritmo diseñado tarda 62,01 milisegundos de media en ejecutarse en el ordenador y 787,00 milisegundos en ejecutarse en Raspberry. A pesar de que se pierda eficiencia al pasar el código al microcontrolador, éste se sigue ejecutando en un tiempo razonable que no compromete el funcionamiento del dron, puesto que éste no volará a una velocidad excesivamente elevada. Además, la Raspberry es un *hardware* ligero que no consume una gran cantidad de batería: 1,19 W y 230 mA a pleno rendimiento y 0,6 W y 120 mA en reposo [Wwwah].

En cuanto a los códigos QR, la eficacia del algoritmo se mantiene en el 83,33 % con lo que sigue siendo bastante elevada. Por otro lado, se pierde eficiencia, ya que tarda 908,35 milisegundos de media en ejecutarse en el ordenador y se estima que tardará más tiempo en la Raspberry, con lo que podría no ser conveniente su instalación en dicha tarjeta. Sin embargo, al compararlo con otro algoritmo que use únicamente ZBar, el diseñado tarda menos en ejecutarse en el ordenador y aproximadamente lo mismo en ejecutarse en Raspberry, siendo, además, los valores de sus máximos menores. Por lo tanto, a la hora de elegir entre ambos algoritmos, sería conveniente escoger el desarrollado en el presente proyecto.

Podría concluirse, por tanto, que el objetivo del proyecto ha sido cumplido para los códigos de barras puesto que se ha descrito e implementado un algoritmo que los localiza y lee a partir de imágenes tomadas con diferentes orientaciones y condiciones de iluminación, ejecutándose en un tiempo razonable sin comprometer el funcionamiento del dron. Para los códigos QR, dicho objetivo se ha completado parcialmente puesto que éstos son detectados y leídos pero en un tiempo que podría comprometer el funcionamiento del dron en ciertas ocasiones.

8 Futuros desarrollos

El amor al estudio es, de todas las pasiones, la que más contribuye a nuestra felicidad.

Émilie du Châtelet
(1706–1749)

En el último capítulo se expone el trabajo que podría realizarse en el futuro partiendo del ya desarrollado en el presente proyecto. Se hace un análisis tanto de las posibles mejoras y funcionalidades que se añadirían al algoritmo así como de una nueva forma de implementar el mismo.

Como desarrollos a realizar en un futuro sería posible, por un lado, realizar mejoras del algoritmo diseñado. En primer lugar, deberían buscarse procedimientos alternativos para realizar el preprocesamiento y la localización de los códigos de forma más eficiente. Es importante disminuir el tiempo de ejecución en sistemas mononúcleo, dado que el algoritmo diseñado debería ser más rápido que el que aplica ZBar directamenta a toda la imagen. Una opción sería desarrollar sin emplear OpenCV sino otra librería de tratamiento de imágenes. Otra posibilidad es experimentar con técnicas alternativas dichas fases que empleasen una menor cantidad de tiempo en ejecutarse.

Por otra parte, se podrían implementar una serie de funcionalidades nuevas:

- Hacer que el algoritmo sea capaz de localizar y leer imágenes en las que existe más de un código de barras, ya que, en el desarrollo actual, cuando se lee correctamente un código acaba la ejecución.
- Seguir probando técnicas de localización que hagan al algoritmo más eficaz en cuanto al número de códigos leídos.
- Mejorar la localización y lectura de los códigos QR puesto que el trabajo llevado a cabo con en este tipo de códigos únicamente ha consistido en aplicar el algoritmo desarrollado para los códigos de barras. Además, el análisis debería centrarse en los casos en los que

existen malas condiciones de iluminación puesto que es cuando el algoritmo posee mayor dificultad para realizar la lectura.

Por otro lado, una nueva forma de implementar el algoritmo podría realizarse mediante el entrenamiento de una red neuronal convolucional. Las Redes Neuronales Convolucionales (CNN de sus siglas en inglés) son un tipo de redes neuronales artificiales donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico [Wwway]. Es interesante emplear este tipo de red debido a que su aplicación es realizada en matrices bidimensionales, con lo que son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otros usos.

Un posible futuro desarrollo consistiría en tomar una red neuronal convolucional que ya haya sido entrenada para reconocer un tipo de objetos o animales por medio de visión artificial como, por ejemplo, perros. Así, únicamente sería necesario cambiar el patrón de dicho animal por el de los códigos de barras o QR, de forma que se ahorrarían tiempo y recursos a la hora de entrenar el sistema. Esto se conoce como transferencia de aprendizaje (*transfer learning*).

Instalación de herramientas

Nunca en mi vida había utilizado una herramienta, más con el tiempo, con trabajo, empeño e ingenio descubrí que no había nada que no pudiera construir, en especial, si tenía herramientas.

Daniel Defoe
(1660–1731)

En este apéndice se detalla el procedimiento seguido para instalar las herramientas *software* indicadas en el apartado 1.3 del capítulo 1 en el microcontrolador Raspberry Pi Zero W, así como el proceso seguido para implementar el código del algoritmo en dicho *hardware*.

Antes de comenzar la instalación de herramientas en Raspberry (y tras haber alimentado esta con corriente eléctrica) es importante conocer cómo se accede a dicho *hardware*. Para ello existen dos opciones: acceso físico y acceso remoto. El acceso remoto posee, a su vez, dos nuevas posibilidades de conexión. Para el acceso físico, puede conectarse al microcontrolador un monitor mediante un cable HDMI y un ratón y un teclado mediante un conector USB. Este acceso físico permite visualizar el entorno gráfico del sistema operativo Raspbian en el terminal y manejar Raspberry con la ayuda del ratón y el teclado como si fuese cualquier ordenador común. Por otro lado, con el acceso remoto se puede conectar el microcontrolador al ordenador mediante un cable USB y acceder, a través de un nuevo terminal de línea de comandos de Linux, al terminal de Raspberry. Esta conexión se realiza mediante SSH (*Secure Shell*) tras ejecutar el comando `ssh pi@raspberry.local` en el terminal del ordenador. La otra opción de acceso remoto también se realiza mediante SSH desde la línea de comandos de Linux pero de forma inalámbrica puesto que este modelo de Raspberry posee antena Wi-Fi, lo que permite acceder al terminal del microcontrolador sin necesidad de poseer una conexión física. En este caso, el comando a ejecutar es similiar al que se emplearía en cualquier terminal para realizar una conexión ssh a otro dispositivo, por lo que se debe conocer la dirección IP de la Raspberry: `ssh dirección_IP`, por ejemplo, `ssh 192.162.10.4`.

Por lo tanto, puede observarse que el acceso físico permite acceder directamente al microcontrolador mientras que el acceso remoto posibilita la conexión a través del terminal de otro dispositivo. Para que el la Raspberry quede lista para su instalación en el dron, debe realizarse tanto la instalación de librerías necesarias como la compilación y ejecución del código del propio algoritmo para comprobar que este funciona de forma correccta.

A.1. Instalación de librerías en Raspberry

Al igual que en el ordenador en el que se ha desarrollado el código, en el microcontrolador deben instalarse, la librería de tratamiento de imágenes OpenCV y la librería de lectura de códigos ZBar. Los procedimientos seguidos para realizar dichas instalaciones se detallan a continuación.

A.1.1. Instalación de OpenCV

Para la instalación de la librería OpenCV en Raspberry basta con seguir las instrucciones de [Wwwag], donde se explica el procedimiento en detalle. Este proceso puede llevar una cantidad considerable de tiempo ya que OpenCV tarda aproximadamente once horas en compilar en la Raspberry Pi Zero W.

A.1.2. Instalación de ZBar

El proceso de instalación de la librería ZBar es más sencillo que el necesario para instalar OpenCV. En primer lugar, se descargan los archivos de la librería de la página web oficial [Wwwbb] y se descomprimen. Este paso podría hacerse directamente en Raspberry ya que el modelo empleado en el presente proyecto posee antena Wi-Fi, lo que permite su conexión a Internet. Sin embargo, en este caso se realizó la descarga de la librería desde el ordenador y, tras esto, se descomprimieron los archivos y se pasaron al microcontrolador ya que es el proceso más rápido, al ser la velocidad de descarga empleada por el ordenador mayor que la de Raspberry.

Una vez se posee la carpeta con los archivos de ZBar, el siguiente paso es compilar la librería. Para ello, se abre una nueva consola y, una vez se ha cambiado el directorio actual al de la carpeta de ZBar, en línea de comandos se escribe:

```
./configure --without-qt --without-gtk --without-python --without-java CFLAGS=-
    DNDEBUG --disable-video --without-imagemagick
```

Mediante este comando se ejecuta el script de configuración, que se asegura de que todas las dependencias para el resto del proceso de compilación e instalación estén disponibles y descubre todo lo que necesita saber para usar esas dependencias [Wwwai]. Además, con este comando se establecen las distintas opciones de configuración del proceso de compilación e instalación de la librería. Tras esto, simplemente se ejecutan los comandos:

```
make
sudo make install
```

El comando `make` compila el software, una vez que `configure` ha hecho su trabajo. Ejecuta una serie de tareas definidas en un archivo *Makefile* para compilar el programa a partir de su código fuente. El archivo comprimido de la librería que se descarga generalmente no incluye un archivo *Makefile* finalizado. En cambio, viene con una plantilla llamada *Makefile.in* y el

comando `configure` produce un *Makefile* final personalizado específicamente para el sistema en el que se ejecuta, incluyendo las opciones que se indicaron anteriormente en dicho comando [Wwwbb].

Finalmente, ahora que el software está compilado y listo para ejecutarse, los archivos se pueden copiar en sus destinos finales. El comando `make install` copiará el programa creado y sus bibliotecas y documentación en las ubicaciones correctas. Esto generalmente significa que el binario del programa se copiará en un directorio en su PATH, la página del manual del programa se copiará en un directorio en su MANPATH, y cualquier otro archivo del que dependa se almacenará de manera segura en el lugar apropiado. Añadir ZBar al PATH del microcontrolador será muy útil posteriormente para compilar el código del lector, ya que se le indicará al compilador dónde encontrar las librerías de forma más sencilla, tal y como detalla en la sección A.2.2.

Dado que el paso de instalación también se define en el archivo Makefile, el lugar en el que es instalado el software puede cambiar en función de las opciones que se pasen al script de configuración o de lo que el script de configuración haya descubierto sobre el sistema en el que se instala. Dependiendo de dónde se esté instalando el software, es posible que sean necesarios permisos para este paso, de modo que pueda copiar archivos en los directorios del sistema. Es por ello que se añade `sudo` al comando `make install` [Wwwbb].

A.2. Compilación del código para Raspberry

Existen varias manera de compilar el código del lector de códigos para Raspberry. Se puede realizar una compilación cruzada del código, compilar directamente el código sobre Raspberry o generar manualmente el archivo *makefile*. En el presente proyecto se han probado las dos primeras técnicas para realizar la compilación, ya que escribir el código del archivo makefile es un proceso más complejo y requiere de un conocimiento más avanzado de este tipo de archivos.

La técnica que finalmente se emplea para compilar el código es la segunda, debido a que, a pesar de conseguir generar un código compilado mediante la compilación cruzada, no se logró ejecutar este en Raspberry libre de errores.

A.2.1. Compilación cruzada sobre Linux

La compilación cruzada consiste en generar código ejecutable bajo una determinada arquitectura para una arquitectura diferente [Wwwn]. En este caso se va a generar código ejecutable en Linux para Raspbian, que es el sistema operativo de Raspberry.

Para realizar la compilación cruzada en la máquina virtual Linux donde se ha desarrollado el código del algoritmo, se siguen los pasos indicados por la referencia [Wwwj], en la cual, además, especifica el procedimiento si se está empleando el entorno de desarrollo Eclipse, como es el presente caso. Antes de comenzar, es necesario descargar lo que se conoce como una cadena de herramientas o *toolchain* de [Wwwm] específica para Raspberry. Una cadena de herramientas es un conjunto de programas software (herramientas de programación) que se emplean para llevar a cabo un desarrollo software complejo. Una cadena de herramientas posee, entre otros elementos, un compilador y enlazador para transformar el código fuente en un programa ejecutable y un depurador para probar el código y eliminar errores.

Una vez se posee la cadena de herramientas, se crea un nuevo proyecto vacío y se selecciona en el apartado de *Toolchains* la opción *Cross GCC* como se muestra en la Figura A.1. De esta forma se indica que se va a utilizar un compilador cruzado. Tras esto, cuando aparezca la ventana que pide configurar los comandos de *Cross GCC*, se añade `arm-linux-gnueabihf-` al campo que indica el prefijo del compilador cruzado. En el segundo campo, cuyo nombre es ruta del compilador cruzado, se debe introducir la ruta hasta el directorio bin de la cadena de herramientas: `RUTA_A_LA_CARPETA/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin` (reemplazando `RUTA_A_LA_CARPETA` por la ruta a la carpeta donde se halla descomprimido el archivo de la cadena de herramientas). En la Figura A.2 aparece gráficamente cómo quedaría la configuración del compilador.

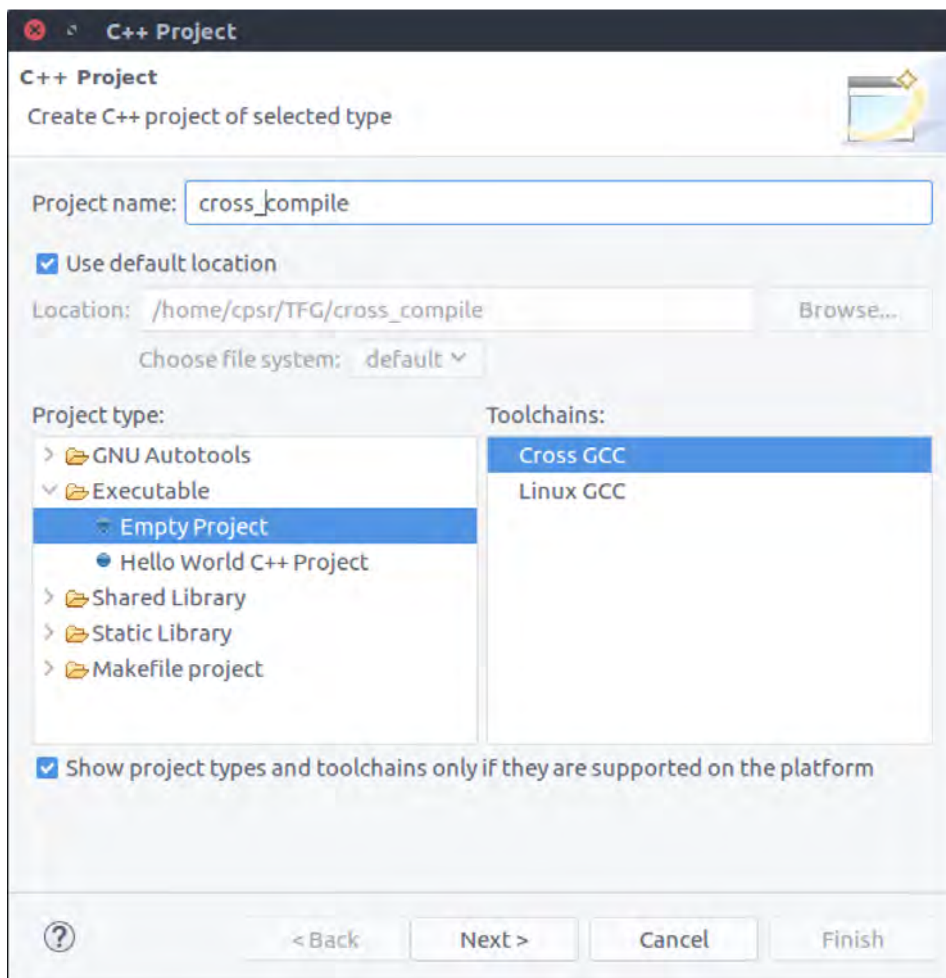


Figura A.1. Selección del compilador cruzado al crear el proyecto.

Tras esto, se descarga y se descomprime el archivo de [Wwwk] el cual contiene una serie de librerías que se deben incluir al proyecto. Para ello, en Eclipse se selecciona *Proyecto*, después *Propiedades* y, dentro de la opción *Cross GCC Compiler*, en *Includes*, se añade la ruta al directorio de la carpeta *include* que se encuentra dentro de la que se acaba de descomprimir, tal y como se muestra en la Figura A.3.

A continuación, dentro de la opción *Librerías* de *Cross GCC Linker* se añaden las librerías *xenomai*, *native* y *rtdm*. Así como la ruta al directorio *lib*, situado dentro de la carpeta que acaba de ser descomprimida (ver Figura A.4) . Finalmente, se pega el código del algoritmo desarrollado en un nuevo archivo de extensión .cpp dentro del proyecto y se compila seleccionando la opción

del menú del proyecto *Build project*. De esta forma, se crea un fichero ejecutable obtenido a partir de la compilación cruzada, el cual puede transferirse a la Raspberry y ejecutarlo allí.

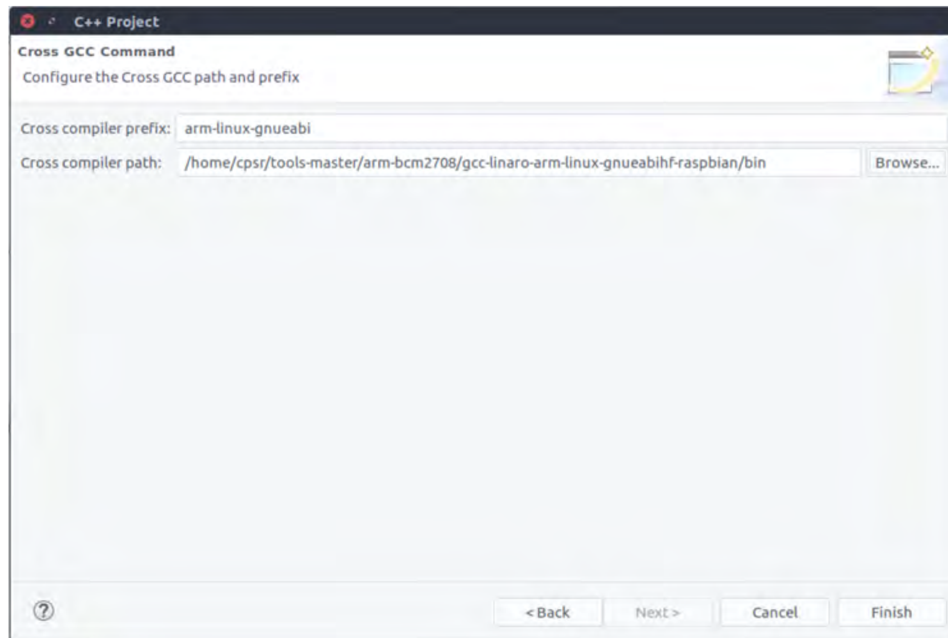


Figura A.2. Configuración del prefijo y la ruta del compilador cruzado.

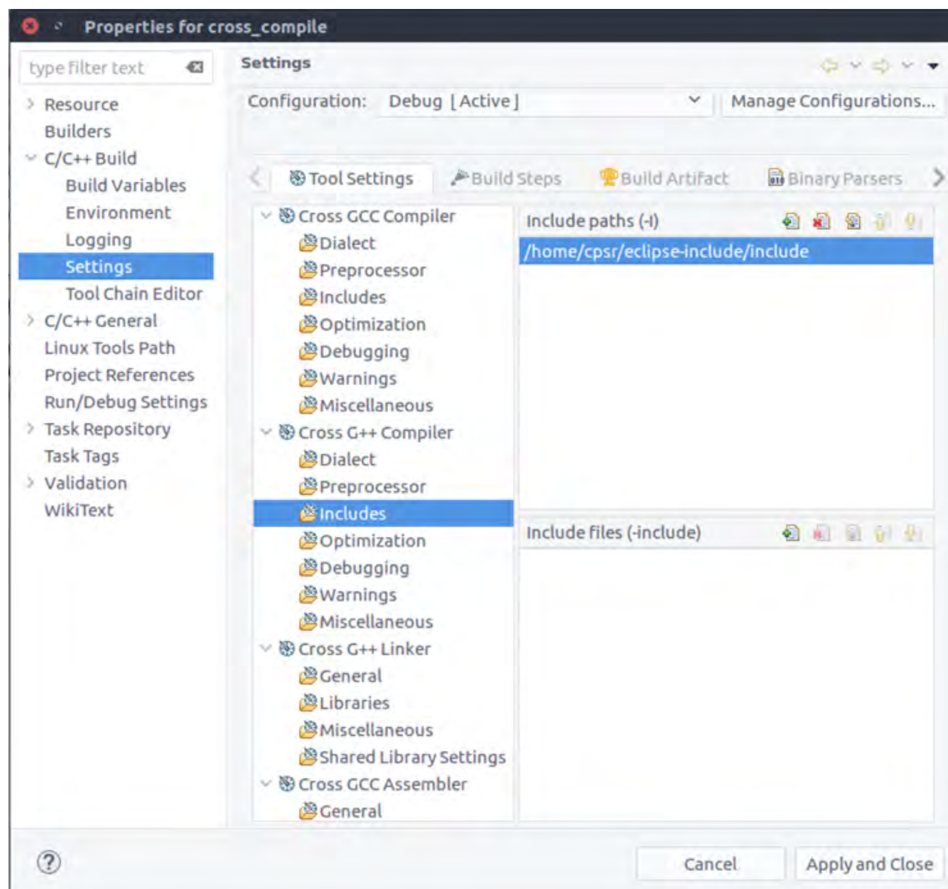


Figura A.3. Incluir ruta al directorio donde se encuentran las librerías del toolchain para Raspberry.

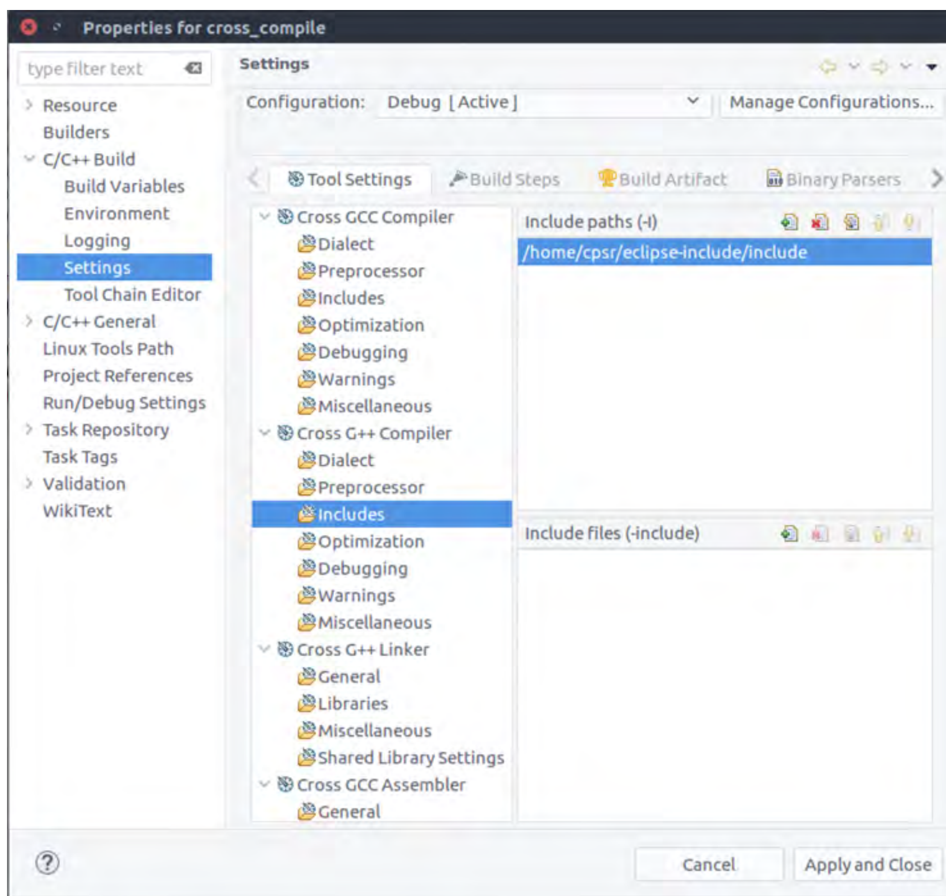


Figura A.4. Añadir al proyecto las librerías del toolchain para Raspberry.

Existen numerosas cadenas de herramientas que pueden emplearse para realizar la compilación cruzada aunque ésta es la oficial de Raspberry. Sin embargo, no se consiguió que el código compilado se ejecutase correctamente en Raspberry, a pesar de haber seguido el procedimiento indicado anteriormente. Por ello, fue necesario buscar una solución alternativa, la cual se detalla en el siguiente apartado (A.2.2). Con dicha solución se consiguió finalmente ejecutar el código en Raspberry de forma adecuada.

A.2.2. Compilación directa sobre Raspberry

Para compilar el código directamente sobre Raspberry, una vez que se ha realizado la instalación de las librerías, es necesario el archivo de extensión .cpp que contiene el código. Por otro lado, para realizar la ejecución del programa se necesita el conjunto de ficheros de imágenes que vayan a ser leídos por el algoritmo. En el caso en que el algoritmo recibe las imágenes directamente del vídeo tomado por la cámara del dron, únicamente será necesario el archivo .cpp.

Una vez que se posee el fichero con el código en el microcontrolador, el siguiente paso es abrir un nuevo terminal de línea de comandos y cambiar el directorio actual al directorio en el que se encuentra dicho fichero. Tras esto, se ejecuta el comando:

```
g++ $(pkg-config --libs --cflags opencv zbar) -o NombreEjecutable
   NombreFicheroACompilar.cpp
```

En el cual se le indica al compilador g++ qué librerías emplea el programa que va a compilar, el nombre que se desea que posea el fichero ejecutable que se generará y el nombre del fichero

que se desea compilar. Es importante tener en cuenta que a la hora de realizar la instalación de dichas librerías, éstas han sido añadidas al PATH del microcontrolador, por lo que no es necesario incluir en el comando la ruta hacia las mismas, sino simplemente indicar su nombre. La ejecución de este comando genera un archivo ejecutable que, al igual que se ha indicado en la sección anterior, se ejecuta con `./NombreEjecutable`.

Bibliografía

[CH05] D. Chai y F. Hock, «Locating and Decoding EAN-13 Barcodes from Images Captured by Digital Cameras», *2005 5th International Conference on Information Communications & Signal Processing*, n.º 01010, págs. 1595-1599, 2005. dirección: http://ieeexplore.ieee.org/document/1689328/.

[Cor11] P. Corke, *Robotics, vision and control. Fundamental algorithms in MATLAB*, 1st Ed. Springer, 2011.

[FK07] T. Falas y H. Kashani, «Two-dimensional bar-code decoding with camera-equipped mobile phones», *Proceedings - Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2007*, págs. 597-600, 2007.

[GZ11] Y. Gu y W. Zhang, «QR code recognition based on image processing», *International Conference on Information Science and Technology*, págs. 733-736, 2011. dirección: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5765349.

[Liu+08] Y. Liu, J. Yang y L. Mingjun, «Recognition of QR Code with mobile phones», *2008 Chinese Control and Decision Conference*, págs. 203-206, 2008. dirección: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4597299.

[Liu+10] F. Liu, J. Yin, K. Li y Q. Liu, «An improved recognition method of PDF417 barcode», *2010 Chinese Conference on Pattern Recognition, CCPR 2010 - Proceedings*, págs. 450-454, 2010.

[LL06] Y. Liu y M. Liu, «Automatic Recognition Algorithm of Quick Response Code Based on Embedded System», *Sixth International Conference on Intelligent Systems Design and Applications*, vol. 2, págs. 783-788, 2006. dirección: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4021764.

[Lu+06] X. Lu, G. Fan e Y. Wang, «A robust barcode reading method based on image analysis of a hierarchical feature classification», *IEEE International Conference on Intelligent Robots and Systems*, págs. 3358-3362, 2006.

[LZ10] Y. M. Li y L. Zeng, «Research and application of the EAN-13 barcode recognition on Iphone», *2010 International Conference on Future Information Technology and Management Engineering, FITME 2010*, vol. 2, págs. 92-95, 2010.

[Muñ+99] R. Muñiz, L. Junco y A. Otero, «A robust software barcode reader using the Hough transform», *Proceedings 1999 International Conference on Information Intelligence and Systems (Cat. No.PR00446)*, págs. 313-319, 1999. dirección: http://ieeexplore.ieee.org/document/810282/.

[Ohb+04] E. Ohbuchi, H. Hanaizumi y L. A. Hock, «Barcode readers using the camera device in mobile phones», *Proceedings - 2004 International Conference on Cyberworlds, CW 2004*, págs. 260-265, 2004.

[Sze11] R. Szeliski, *Computer Vision : Algorithms and Applications*, 1st Ed., Springer, ed. 2011.

[Wwwa] *30 cosas que debes saber sobre los códigos QR*. dirección: https://uqr.me/es/blog/cosas-que-debes-saber-sobre-codigos (visitado 04-01-2018).

[Wwwb] *Answers OpenCV: Extract a RotatedRect area*. dirección: http://answers.opencv.org/question/497/extract-a-rotatedrect-area/ (visitado 18-05-2018).

[Wwwc] *Area of a single pixel object in OpenCV*. dirección: http://answers.opencv.org/question/58/area-of-a-single-pixel-object-in-opencv/.

[Wwwd] *Blog: Lenguajes interpretados vs. compilados*. dirección: https://otroespacioblog.wordpress.com/2012/09/02/lenguajes-de-programacion-compilados-vs-interpretados/ (visitado 11-05-2018).

[Wwwe] *Calibración de cámara*. dirección: http://www.hugorodriguez.com/blog/servicios-de-gestion-de-color/calibracion-de-camaras/ (visitado 21-01-2018).

[Wwwf] *Code Pool: Finding contours in images with OpenCV*. dirección: http://www.codepool.biz/opencv-finding-image-contours.html (visitado 14-05-2018).

[Wwwg] *Code Project: Drawing Barcodes in Windows Part 4 – Code 93*. dirección: https://www.codeproject.com/Articles/2437/Drawing-Barcodes-in-Windows-Part-Code (visitado 04-01-2018).

[Wwwh] *Cognex: Código de barras Code 93*. dirección: http://www.cognex.com/symbologies/1-d-linear-barcodes/code-93-barcodes/?langtype=1034&locale=co (visitado 23-01-2018).

[Wwwi] *Fast detection and recognition of QR codes in high-resolution images*. dirección: http://www.fit.vutbr.cz/research/groups/graph/pclines/pub_page.php?id=2012-SCCG-QRtiles (visitado 09-06-2018).

[Wwwj] *GitHub: Crosscompile a program from your computer to your Raspberry Pi*. dirección: https://github.com/awesomebytes/xenorasp/wiki/Crosscompile-a-program-from-your-computer-to-your-Raspberry-Pi (visitado 06-06-2018).

[Wwwk] *GitHub: Includes for Raspberry toolchain*. dirección: https://github.com/awesomebytes/xenorasp/raw/xenorasp3.10/eclipse-include.tar.gz (visitado 07-06-2018).

[Wwwl] *GitHub: Lectura con ZBar*. dirección: https://github.com/rportugal/opencv-zbar/blob/master/main.cpp (visitado 18-05-2018).

[Wwwm] *GitHub: Raspberry toolchain*. dirección: https://github.com/raspberrypi/tools (visitado 07-06-2018).

[Wwwn] *GNU/Linux embebido: Compilación cruzada*. dirección: http://linuxemb.wikidot.com/tesis-c3 (visitado 07-06-2018).

[Wwwo] *Group T - Master Thesis Blog: Edge detection and Sobel operator*. dirección: https://gtms1337.wordpress.com/2013/03/16/edge-detection-and-sobel-operator/ (visitado 22-01-2018).

[Wwwp] *Hough transformation barcode collection*. dirección: http://artelab.dista.uninsubria.it/downloads/datasets/barcode/hough_barcode_1d/hough_barcode_1d.html (visitado 20-05-2018).

[Wwwq] *Informática hoy: Tipos de código de barras y sus ventajas*. dirección: https://www.informatica-hoy.com.ar/informatica-tecnologia-empresas/Tipos-de-codigo-de-barras-y-sus-ventajas.php (visitado 21-01-2018).

[Wwwr] *Matlab Tricks: Understanding the Hough transform*. dirección: http://matlabtricks.com/post-39/understanding-the-hough-transform (visitado 07-01-2018).

[Wwws] *Mega Label: ¿Qué tipos de códigos de barras existen?* Dirección: http://megalabel.com.mx/tipos-de-codigos-de-barras/ (visitado 21-01-2018).

[Wwwt] *Monografías.com: Introducción a los códigos de barras*. dirección: http://www.monografias.com/trabajos42/codigo-de-barras/codigo-de-barras3.shtml (visitado 03-01-2018).

[Wwwu] *OpenCV: about*. dirección: https://opencv.org/about.html (visitado 17-05-2018).

[Wwwv] *OpenCV: Camera calibration*. dirección: https://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html (visitado 06-01-2018).

[Wwww] *OpenCV: Canny Edge Detector*. dirección: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html (visitado 09-01-2018).

[Wwwx] *OpenCV: Feature detection*. dirección: https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html (visitado 14-05-2018).

[Wwwy] *OpenCV: Geometric Image Transformations*. dirección: https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html (visitado 15-05-2018).

[Wwwz] *OpenCV: Image filtering*. dirección: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=morphologyex#morphologyex (visitado 12-05-2018).

[Wwwaa] *OpenCV: Image Thresholding*. dirección: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html (visitado 12-05-2018).

[Wwwab] *OpenCV: Miscellaneous Image Transformations*. dirección: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html (visitado 11-05-2018).

[Wwwac] *OpenCV: Structural analysis and shape descriptors*. dirección: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html (visitado 14-05-2018).

[Wwwad] *Operaciones Logísticas: Códigos de barra*. dirección: http://logisticaenchile.blogspot.com.es/2009/06/codigos-de-barra.html (visitado 03-01-2018).

[Wwwae] *Programar fácil: Detector de bordes con Canny*. dirección: https://programarfacil.com/blog/vision-artificial/detector-de-bordes-canny-opencv/.

[Wwwaf] *Raspberry Pi*. dirección: https://www.raspberrypi.org/.

[Wwwag] *Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi*. dirección: https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/ (visitado 05-06-2018).

[Wwwah] *Redes Zone: Energía consumida por cada modelo de Raspberry*. dirección: https://www.redeszone.net/2017/03/02/energia-consumida-raspberry-pi/.

[Wwwai] *The magic behind configure, make, make install*. dirección: https://robots.thoughtbot.com/the-magic-behind-configure-make-make-install (visitado 06-06-2018).

[Wwwaj] *Thomas: Four types of barcode scanners*. dirección: https://www.thomasnet.com/articles/automation-electronics/barcode-scanner-types (visitado 18-05-2018).

[Wwwak] *Tutor de Programación. Clase Mat*. dirección: http://acodigo.blogspot.com.es/2017/04/conociendo-la-clase-cvmat-de-opencv.html (visitado 11-05-2018).

[Wwwal] *Wikipedia: Adaptive histogram ecualization*. dirección: https://en.wikipedia.org/wiki/Adaptive_histogram_equalization#Contrast_Limited_AHE (visitado 12-11-2018).

[Wwwam] *Wikipedia: Code 128*. dirección: https://es.wikipedia.org/wiki/Code_128 (visitado 21-01-2018).

[Wwwan] *Wikipedia: Code 39*. dirección: https://es.wikipedia.org/wiki/Code_39 (visitado 21-01-2018).

[Wwwao] *Wikipedia: Code 93*. dirección: https://es.wikipedia.org/wiki/Code_93 (visitado 21-01-2018).

[Wwwap] *Wikipedia: Código de barras*. dirección: https://es.wikipedia.org/wiki/Codigo_de_barras (visitado 22-01-2018).

[Wwwaq] *Wikipedia: Código QR*. dirección: https://es.wikipedia.org/wiki/Codigo_QR (visitado 21-01-2018).

[Wwwar] *Wikipedia: Ecualización del histograma*. dirección: https://es.wikipedia.org/wiki/Ecualizacion_del_histograma (visitado 12-05-2018).

[Wwwas] *Wikipedia: European Article Number*. dirección: https://es.wikipedia.org/wiki/European_Article_Number (visitado 21-01-2018).

[Wwwat] *Wikipedia: Imagen CLAHE*. dirección: https://en.wikipedia.org/wiki/Adaptive_histogram_equalization#/media/File:Clahe-redist.svg (visitado 12-05-2018).

[Wwwau] *Wikipedia: OpenCV*. dirección: https://en.wikipedia.org/wiki/OpenCV#Programming_language (visitado 11-05-2018).

[Wwwav] *Wikipedia: PDF417*. dirección: https://es.wikipedia.org/wiki/PDF417 (visitado 04-01-2018).

[Wwwaw] *Wikipedia: Píxel*. dirección: https://es.wikipedia.org/wiki/Pixel (visitado 11-05-2018).

[Wwwax] *Wikipedia: QR File*. dirección: https://commons.wikimedia.org/wiki/File:QR-code-OA-catalogue.png (visitado 04-01-2018).

[Wwway] *Wikipedia: Redes neuronales convolucionales*. dirección: https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales (visitado 07-06-2018).

[Wwwaz] *Youtube: Image processing: Dilation*. dirección: https://www.youtube.com/watch?v=xO3ED27rMHs (visitado 05-01-2018).

[Wwwba] *ZBar bar code reader: about*. dirección: http://zbar.sourceforge.net/about.html (visitado 17-05-2018).

[Wwwbb] *ZBar bar code reader: download*. dirección: http://zbar.sourceforge.net/download.html (visitado 05-06-2018).

[Wwwbc] *ZBar bar code reader: home*. dirección: http://zbar.sourceforge.net/index.html (visitado 17-05-2018).

[Wwwbd] *ZBar Documentation: Image*. dirección: http://zbar.sourceforge.net/api/classzbar_1_1Image.html#_details (visitado 18-05-2018).

[Wwwbe] *ZBar Documentation: Image Scanner*. dirección: http://zbar.sourceforge.net/api/classzbar_1_1ImageScanner.html#bd75786c6fdcfd91fe46957ff6ee1ac3 (visitado 18-05-2018).

[YS07] S. M. Youssef y R. M. Salem, «Automated barcode recognition for smart identification and inspection automation», *Expert Systems with Applications*, vol. 33, n.º 4, págs. 968-977, 2007.