UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

INGENIERÍA DE TELECOMUNICACIÓN

DETECCIÓN Y RECONOCIMIENTO DE BALIZAS PARA ORIENTAR UN ROBOT MÓVIL

PROYECTO FIN DE CARRERA

Julio – 2013

AUTOR: Francisco Manuel Follana Maciá

DIRECTOR: César Fernández Peris

VISTO BUENO Y CALIFICACIÓN DEL PROYECTO

Título proyecto:

Proyectante: _____

_____

Director/es: _____

VºBº director/es del proyecto:

Lugar y fecha: _____

CALIFICACIÓN NUMÉRICA ☐ MATRÍCULA DE HONOR ☐

Méritos justificativos en el caso de conceder Matrícula de Honor:

_____

_____

_____

_____

Conforme presidente:	Conforme secretario:	Conforme vocal:

Lugar y fecha: _____

AGRADECIMIENTOS

Quiero dedicar y agradecer la realización de este trabajo que culmina el esfuerzo de muchos años a mis padres, Vicente y Margarita, a mi hermano Vicente, a Miriam y a mis sobrinos Rodrigo y Jimena. También quiero dedicárselo especialmente a mi amor Marian por todo el apoyo y comprensión durante estos últimos meses.

Tampoco olvido a mi tutor César por la paciencia y apoyo que me ha dado durante la realización del mismo.

RESUMEN

En este trabajo se lleva a cabo el desarrollo de balizas que nos permitan orientar un robot móvil así como los algoritmos para su detección y reconocimiento. Todo el desarrollo se realiza en el entorno de programación de Matlab, creando los algoritmos de detección y haciendo uso de las funciones predefinidas de las cuales dispone.

En el primer capítulo tratamos la imagen digital como elemento básico de nuestro sistema, etapas de procesamiento, elementos que la componen y como se representa comunmente. También se tratan diversas técnicas de realzado de imágenes digitales que nos serán útiles más adelante en el desarrollo de este proyecto así como el diseño de balizas para su posterior detección

En el segundo capítulo se detallan los elementos con los que llevaremos a cabo el proyecto, haciendo hincapié en las características del robot SURVEYOR SRV1B y en la adquisición de imágenes a través de su sistema de visión.

En el tercer capítulo se desarrollan los algoritmos de detección de bordes y líneas en imágenes, extracción de las líneas más significativas, desarrollo de balizas codificadas y técnicas para el recorte e interpretación correcta de dichas balizas.

En el último capítulo se llevan a la práctica todos los algoritmos desarrollados anteriormente. Se detectan balizas a través de las cámaras del SURVEYOR SRV1B y se extraen estadísticas de los resultados obtenidos.

Para finalizar se presentan las conclusiones obtenidas tras la realización de este proyecto, posibles mejoras y líneas futuras para continuar con su desarrollo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICAS

1. INTRODUCCIÓN

Para cualquier dispositivo móvil es de vital importancia la habilidad de moverse por su entorno. Un robot móvil se caracteriza por realizar una serie de desplazamientos y llevar a cabo una interacción con distintos elementos de su hábitat de trabajo que implican el cumplimiento de una serie de objetivos impuestos según cierta especificación.

Las tareas involucradas en la navegación de un robot móvil son: la percepción del entorno a través de sus sensores, de modo que le permita crear una abstracción del mundo; la planificación de una trayectoria libre de obstáculos, para alcanzar el punto destino seleccionado; y el guiado del vehículo a través de la referencia construida.

La orientación implica el total conocimiento de la posición actual del mismo con respecto al sistema de coordenadas que estemos utilizando. Entre los métodos de localización nos encontramos con la localización basada en balizas visuales, estos son marcas inmersas en el entorno cuya posición es conocida.

Estas balizas visuales conforman la representación del medio que rodea al robot y puede ser en forma de puntos y líneas, patrones más complejos o modelos tridimensionales de los objetos en el medio ambiente. Estos patrones deben ser fácilmente reconocibles e interpretables por el robot, de tal modo que sea capaz de conocer en todo momento su posición y orientación.

Objetivos

Este proyecto pretende que un robot móvil pueda reconocer un conjunto de marcas definidas utilizando visión artificial. Para ello, se utilizarán balizas (marcas impresas) codificadas, fácilmente reconocibles por la cámara del robot.

Los principales objetivos son:

- Diseñar las balizas de modo que su detección sea rápida y fiable.

- Desarrollar algoritmos de visión artificial que permitan detectar las balizas e interpretar su codificación.

Desarrollo

Para cumplir con los objetivos impuestos realizaremos las siguientes tareas:

- Estudio de documentación sobre navegación de robots mediante balizas artificiales.
- Diseño de balizas adecuadas a nuestro experimento.
- Estudio de documentación sobre la transformada de Hough para la detección de líneas.
- Desarrollo de algoritmos basados en la transformada de Hough para la detección de las balizas en la imagen.
- Análisis de la precisión y eficiencia computacional de los algoritmos anteriores.

2. FUNDAMENTOS PREVIOS

En este primer apartado se sientan las bases del procesado digital de imágenes y la creación de balizas que más adelante se pondrá en práctica en el desarrollo del proyecto. En una breve introducción definiremos qué es una imagen digital estableciendo las etapas del procesamiento, la conversión de espacio de color RGB a niveles de grises, de niveles de grises a imágenes binarizadas (blanco y negro), filtrado espacial y detección de bordes, balance de blancos y transformada de Hough. Para terminar se describe la detección de balizas para su posterior decodificación.

2.1 PROCESADO DE IMÁGENES DIGITALES

2.1.1 MODELO DE IMAGEN DIGITAL

El modelo más simple de una imagen digital consiste en una imagen monocroma definida por una función bidimensional de intensidad de luz, **f(x, y)**, donde x e y representan las coordenadas espaciales y **f** es proporcional al brillo, o nivel de gris, de la imagen en ese punto.

Una imagen digital es una imagen **f(x, y)** que procede del muestreo espacial y en intensidad de una imagen óptica. Está formada por una matriz de elementos (píxeles). El píxel es el valor de color o intensidad asociado a cada elemento de la matriz y puede tomar valores dentro del rango [0, 255] (en el caso de una codificación de 8 bits).

2.1.2 ETAPAS DEL PROCESAMIENTO DE IMÁGENES

A continuación se presentan las etapas fundamentales para realizar el procesamiento de una imagen [1].

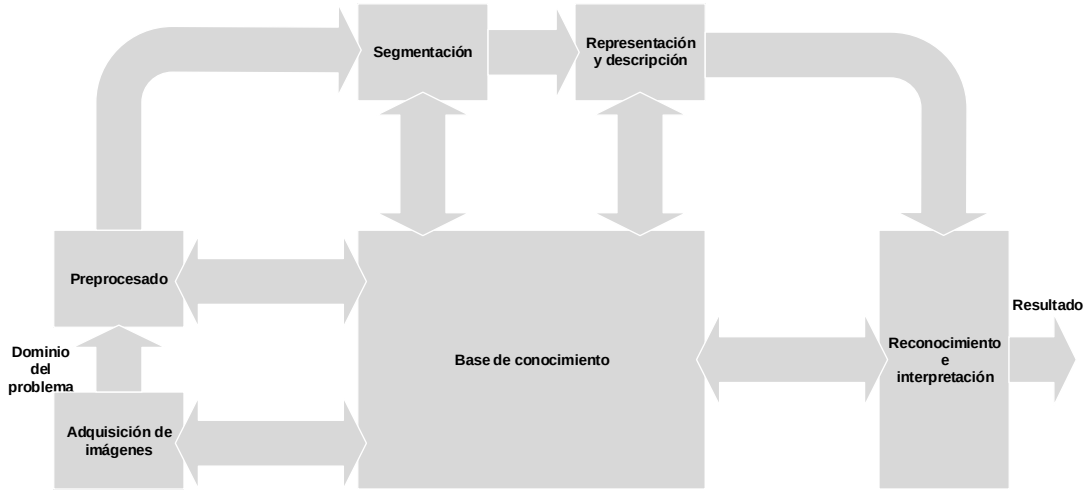


Figura 2.1. Etapas del procesamiento de imágenes.

Atendiendo a la **Figura 2.1** podemos dividir las etapas del procesamiento de imágenes como:

- *Adquisición de la imagen*: obtención de la imagen digital a través de un sensor que nos posibilite la posterior digitalización de la señal.
- *Preprocesado*: la función básica en este punto es la de mejorar la imagen de forma que se aumenten las posibilidades de éxito en los procesos a realizar posteriormente.
- *Segmentación*: esta etapa consiste en dividir una imagen de entrada en sus partes constituyentes u objetos.
- *Representación y descripción*: seleccionar la representación es parte de la solución para transformar los datos obtenidos a una forma adecuada para su posterior tratamiento. En cuanto a la descripción, consiste en extraer información de interés o que sea fundamental para diferenciar una clase de objetos de otra.
- *Reconocimiento e interpretación*: entendemos el reconocimiento como el proceso por el que asignamos significado a un grupo de objetos reconocidos. La interpretación asigna un significado al conjunto de entidades reconocidas.
- *Base de conocimiento*: la necesidad de conocimiento previo es vital para abordar el problema. El conocimiento está codificado en un sistema de procesamiento de imágenes como una base de datos donde se detallen las regiones de una imagen donde se sabe que se ubica información de

interés, acotando la búsqueda que ha de realizarse para hallar esta información.

De esta forma podemos establecer una cadena genérica de elementos de los sistemas de procesamiento digital de imágenes [1]:

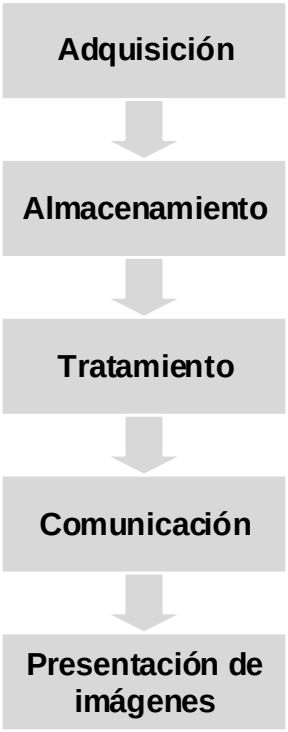


Figura 2.2. Elementos de un sistema de procesado digital.

De forma general las técnicas de procesado pretenden la mejora y realzado de las imágenes para una mejor extracción de detalles para una aplicación específica.

En este apartado se presentan técnicas en el dominio espacial, tratando a la imagen como una señal en dos dimensiones. Estas técnicas abarcan desde transformaciones geométricas (traslación, rotación, escalado…), realzado en el dominio espacial y frecuencial o correcciones de color entre otros.

2.1.3 CONVERSIÓN DE IMÁGENES EN COLOR RGB A ESCALA DE GRISES

Las imágenes en espacio de color RGB están compuestas de 3 matrices del mismo tamaño en x e y, que representan cada una de las componentes de los 3 colores primarios (**Figura 2.3**).

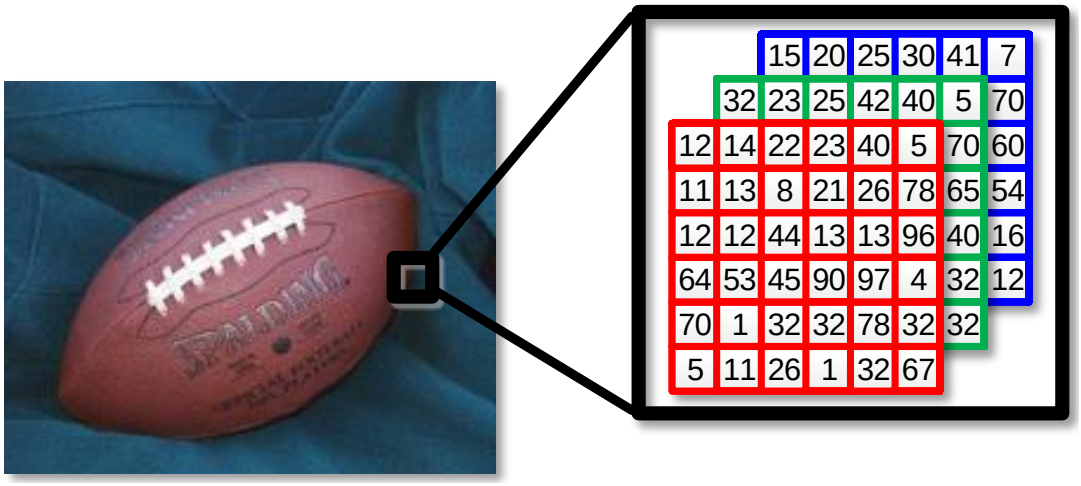


Figura 2.3. Ejemplo de los 3 espacios de color en una imagen.

A partir de las 3 matrices mostradas correspondientes a nuestros colores primarios podemos obtener el equivalente en escala de grises como la media de los tres planos representados para cada píxel de la imagen. A continuación se redondea a un valor discreto:

$$\text{Imagen} = \text{redondeo}\left(\frac{1}{3}\cdot(R+G+B)\right) \qquad (2.1)$$

Una vez obtenida la media de las 3 componentes de color tenemos un resultado como se muestra en la siguiente figura.

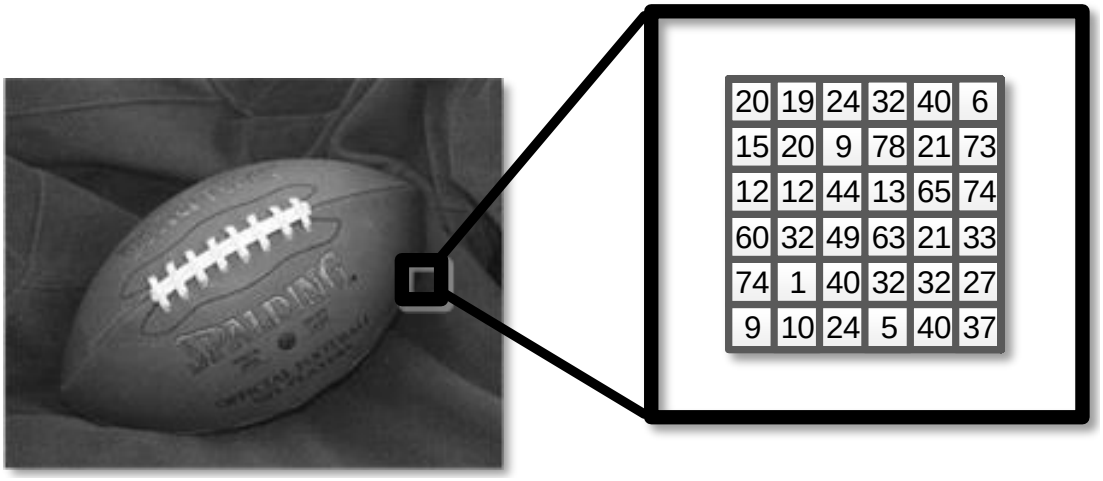


Figura 2.4. Ejemplo en escala de grises.

2.1.4 CONVERSIÓN DE ESCALA DE GRISES A BLANCO Y NEGRO

Esta conversión se realiza una vez obtenida la escala de grises comentada anteriormente. Para ello necesitamos fijar un umbral para decidir qué valores de grises los aproximamos a 1 o a 0. Obtendremos la imagen umbral **f'(x, y)** a partir de la imagen en escala de grises **f(x, y)** como se representa en la siguiente ecuación:

$$f'(x,y) = \begin{cases} 1, & f(x,y) \geq \text{umbral} \\ 0, & f(x,y) < \text{umbral} \end{cases} \qquad (2.2)$$

Siendo x e y las coordenadas de cada píxel y el umbral un valor entre 0 y 255, normalmente se asume como umbral la media de todo el espacio de valores, es decir 128. En la **Figura 2.5** podemos ver el ejemplo de una imagen umbralizada.

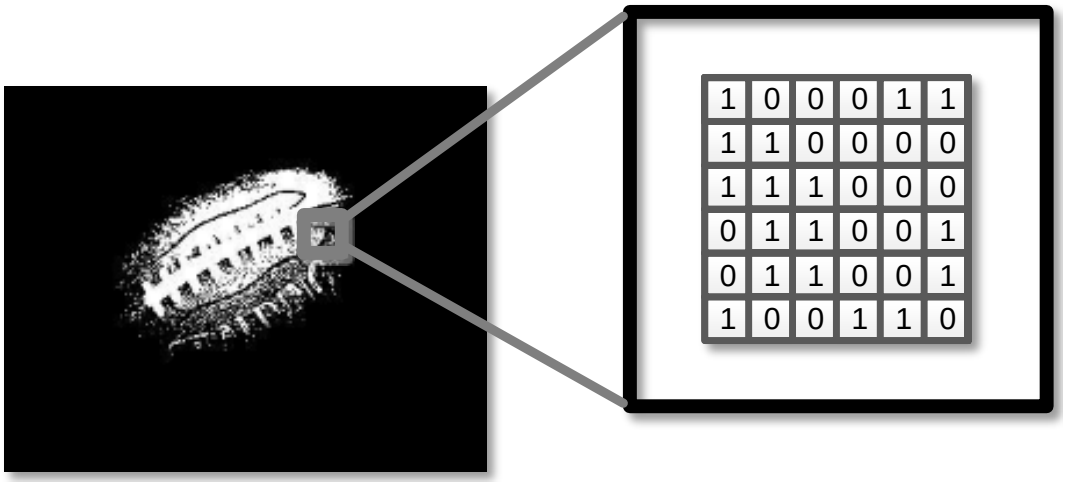


Figura 2.5. Ejemplo de imagen umbralizada.

2.1.5 FILTRADO ESPACIAL. DETECCIÓN DE BORDES

Entendemos como filtrado espacial a la operación que se realiza sobre el conjunto de píxeles de la imagen, por lo que operan directamente con los píxeles. Las funciones de procesamiento de la imagen en el dominio espacial pueden expresarse como:

$$g(x,y) = T[f(x,y)] \qquad (2.3)$$

Donde **f(x, y)** es la imagen de entrada, **T** el operador espacial (máscara) sobre **f** y **g(x, y)** la imagen procesada.

El empleo de máscaras espaciales para el procesamiento de las imágenes es lo que denominamos filtrado espacial (frente al filtrado en el dominio de la frecuencia empleando la transformada de Fourier), y las propias máscaras se denominan filtros espaciales. Lo más común es utilizar máscaras de 3x3, siendo posible aplicar máscaras del tamaño que se desee.

A partir del ejemplo de la **Figura 2.6** la aplicación de la máscara sería la siguiente: escogemos el píxel central de la máscara 3x3 (celda roja), a continuación se realiza una operación con cada uno de los píxeles colindantes con el central, para a continuación promediar el resultado y aplicarlo sobre este mismo.

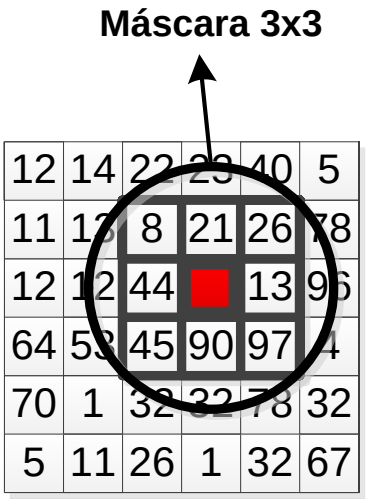


Figura 2.6. Ejemplo de aplicación de una máscara 3x3.

Mientras que la aplicación de una máscara de difuminación de la imagen consistiría en un filtro paso bajo, para destacar los bordes se realiza un filtrado paso alto.

Los filtros paso alto atenúan o eliminan las componentes de baja frecuencia. Como estas componentes son responsables de las características lentamente variables de la imagen, como el contraste global y la intensidad media, el resultado neto de un filtrado paso alto es la reducción de estas características

y, en correspondencia, una aparente agudización de los bordes y de los restantes detalles finos. En la **Figura 2.7** tenemos algunos ejemplos de máscaras 3x3 paso alto [2] [3] [4].

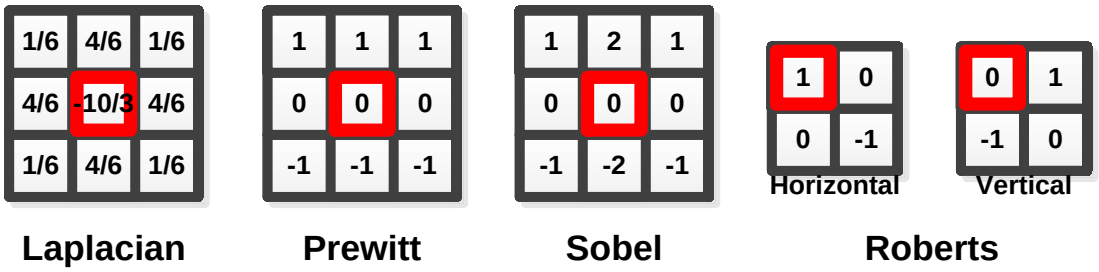


Figura 2.7. Ejemplos de máscaras 3x3.

La aplicación de la máscara se realiza sobre la celda roja, obteniendo el filtrado en sentido horizontal y en vertical con la máscara transpuesta.

En la **Figura 2.8** tenemos un ejemplo del efecto de la detección de bordes en una imagen cualquiera utilizando la máscara “sobel” [5].

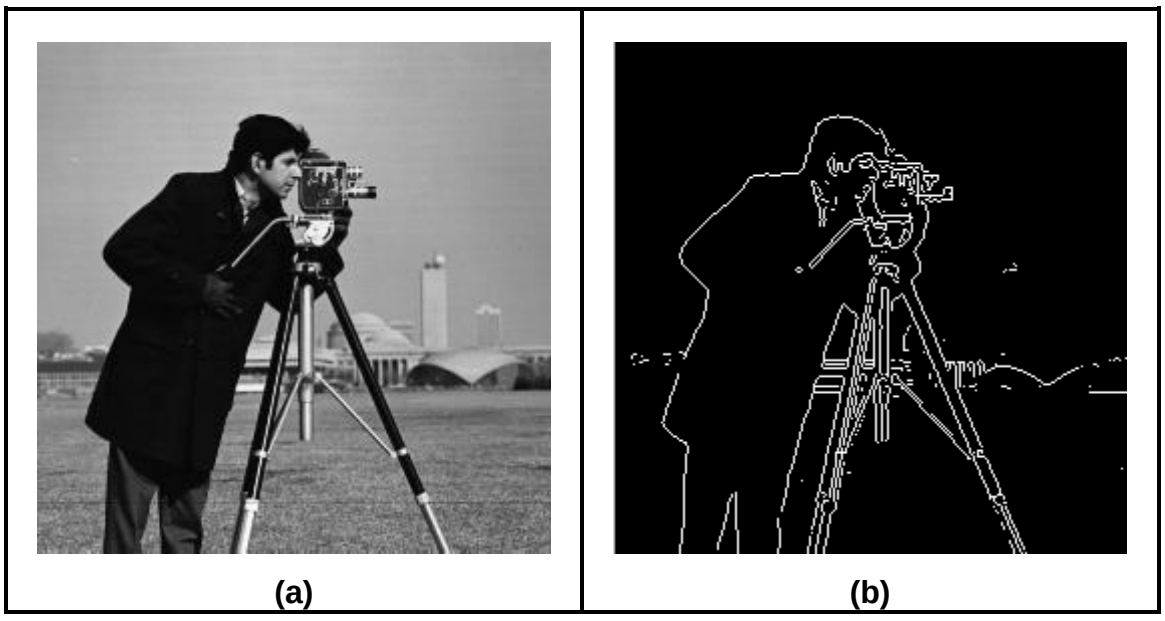
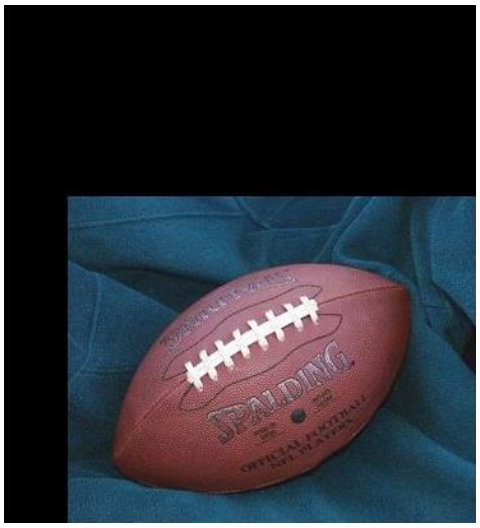


Figura 2.8. Detección de bordes con máscara “sobel”.

2.1.6 TRANSFORMACIONES GEOMÉTRICAS

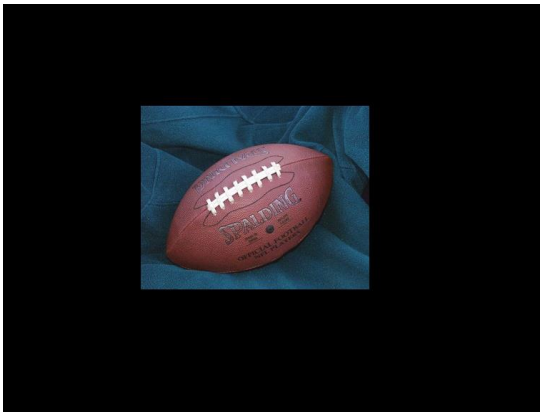
En esta sección se representan numerosas transformaciones geométricas que podemos realizar en las imágenes, estas nos permiten modificar la imagen para una mejor extracción de sus características.

En la **Figura 2.9** tenemos 4 ejemplos de transformaciones geométricas, estas son: traslación, cambio de escala, rotación y cambio de perspectiva [5].



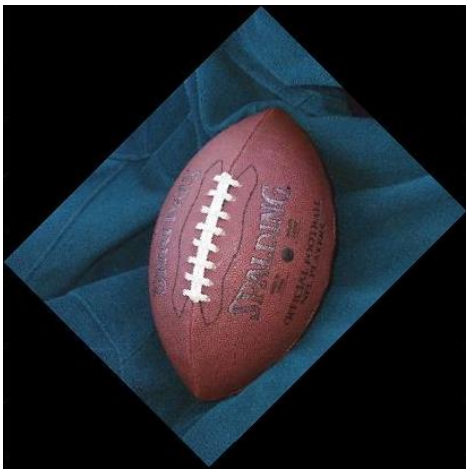
(a)

Traslación



(b)

Cambio de escala



(c)

Rotación

(d)

Cambio de perspectiva

Figura 2.9. Distintas transformaciones geométricas en la imagen.

2.1.7 BALANCE DE BLANCOS

Más adelante cuando veamos la aplicación real de este proyecto veremos cómo ante imágenes tomadas en ambientes reales la iluminación puede no ser

uniforme o deficiente en muchos casos, una forma de compensar estos efectos es realizar un balance de blancos.

La aplicación de un balance de blancos pretende compensar el brillo en los tres colores básicos RGB, es decir, compensar sus valores para que el blanco aparezca como blanco ya que en escenas con iluminaciones artificiales el balance para estas tres componentes puede no ser el adecuado, dando lugar a coloraciones indeseadas en las imágenes. Si extendemos este principio a imágenes en escala de grises la conclusión sería muy parecida, pero en este caso tendríamos aparentemente el efecto de iluminar la escena. El proceso por el cual realizamos el balance de blancos en este trabajo es el siguiente:

Una vez obtenida la imagen obtenemos los máximos de cada columna de las matrices RGB, por lo que tendremos tantos máximos como columnas tenga la imagen. Una vez encontrados estos máximos realizamos la media para cada componente, sumamos y dividimos entre el número de columnas, y así obtendremos tres valores, uno para R, para G y para B. Una vez calculado estos tres valores dividimos cada plano de color con su correspondiente valor y multiplicamos por 255 (para imágenes codificadas con 8 bits/pixel) cada uno. Los resultados que se obtienen se pueden ver representados en la **Figura 2.10** donde el ajuste no es un simple aumento del brillo, sino que es una ponderación sobre los valores originales, acercando los tonos más claros al blanco mientras que los más oscuros apenas sufren modificación.

Figura 2.10. Ejemplo del funcionamiento del balance de blancos en una imagen en escala de grises.

2.1.8 ALGORITMO DE HOUGH

La transformada de Hough [6] es una herramienta computacional que nos permite detectar figuras simples en una imagen, estas figuras pueden ser desde rectas, hasta círculos o elipses [7].

La versión más simple nos permite encontrar líneas. Como primer paso para esta detección se puede aplicar a la imagen un detector de bordes, con lo que obtendremos los puntos pertenecientes a la frontera de la figura que buscamos. Cada punto de la imagen lo podemos representar con la ecuación de la recta:

$$y = m \cdot x + n \tag{2.4}$$

Asignándole una cierta pendiente dentro del espacio que queremos comprobar obtendremos para cada punto y cada posible pendiente sus parámetros **(m, n)** (**Figura 2.11**).

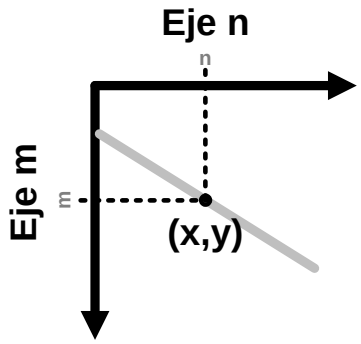


Figura 2.11. Representación de los parámetros (m, n) de un punto de la imagen.

Conocidos el punto **(x, y)** y la pendiente **m**, obtendremos el valor de **n** como:

$$n = -m \cdot x + y \tag{2.5}$$

Utilizando los parámetros **(m, n)** podemos encontrarnos con ciertos problemas, aparecen singularidades cuando las rectas son verticales. Para evitarlo se representan estas rectas en coordenadas polares, con el par **(ρ, θ)**, donde **ρ** representa la distancia del origen de coordenadas al punto **(x, y)** y **θ** es el

ángulo del vector director de la recta perpendicular a la recta original y que pasa por el origen de coordenadas (**Figura 2.12**).

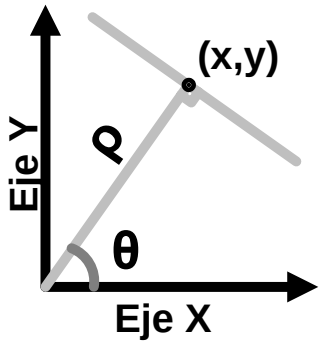


Figura 2.12. Representación de las coordenadas polares de un punto.

Trabajando en este espacio podemos reescribir la ecuación de la recta de la siguiente manera [8]:

$$y = -\frac{\cos\theta}{\sin\theta}\cdot x + \frac{\rho}{\sin\theta} \tag{2.6}$$

En la cual despejamos **ρ**:

$$\rho = x\cdot\cos\theta + y\cdot\sin\theta \tag{2.7}$$

Con lo que a cada punto representado por el par **(ρ, θ)** le podemos asociar una recta que es única para valores de $\boldsymbol{\theta} \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)$ y $\boldsymbol{\rho} \in \mathbf{R}$. A partir de la altura y longitud de la imagen a tratar podemos definir el valor máximo de **ρ**:

$$\text{Distancia Máxima (DM)} = \sqrt{\text{altura}^2 + \text{longitud}^2} \tag{2.8}$$

De esta forma $\boldsymbol{\rho} \in$ **[-DM, DM)**. El punto más alejado del origen es el que se encuentra en la diagonal de la imagen.

2.1.8.1 DISTANCIA ENTRE RECTAS

A partir del par en coordenadas polares descrito anteriormente podemos establecer la similitud entre dos rectas utilizando el teorema del coseno (**Figura 2.13**).

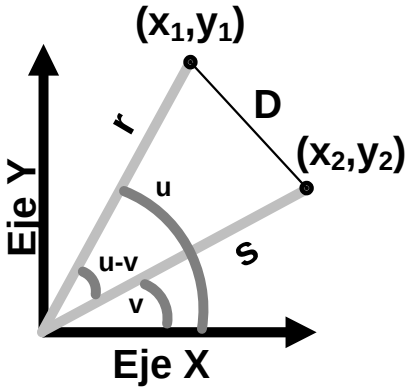


Figura 2.13. Distancia entre dos puntos en coordenadas polares.

Esta similitud la obtenemos con la distancia **D**, que la podemos calcular a partir de la ecuación:

$$D = \sqrt{r^2 + s^2 + 2 \cdot r \cdot s \cdot \cos(u-v)} \qquad (2.9)$$

La cual recibe el nombre de teorema del coseno o teorema de Pitágoras generalizado.

2.2 BALIZAS ARTIFICIALES

El diseño de balizas es una de las motivaciones de este trabajo. El propósito es que las balizas sean unívocamente detectables y que se puedan extraer de ellas su codificación correspondiente.

Las marcas que se propondrán más adelante en el desarrollo se basan en el trabajo descrito en la referencia bibliográfica número [9]. En ella, haciendo uso de la transformada de Hough se detectan dos tipos de marcas, una en forma de flecha (**Figura 2.14(a)**) y una línea (**Figura 2.14(b)**)).

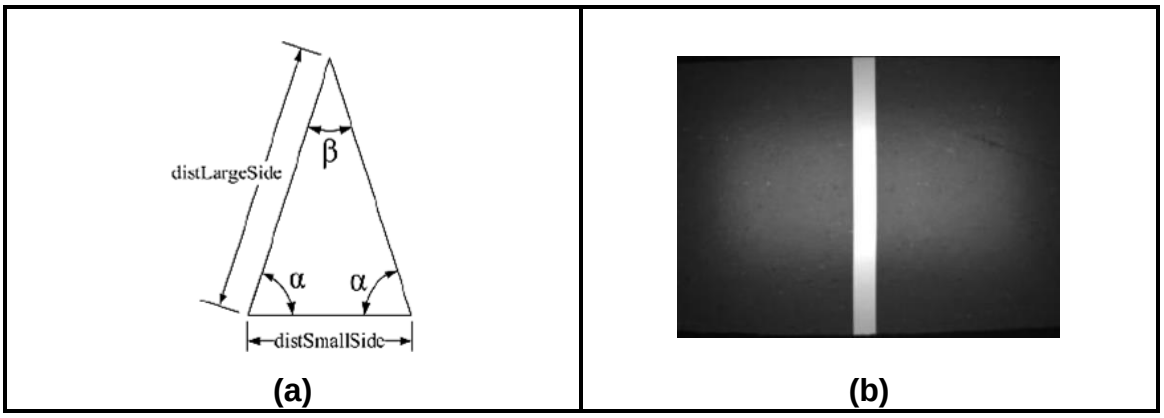


Figura 2.14. Marcas a detectar.

A continuación se lee el código interior, compuesto por una serie de puntos (**Figura 2.15**), los cuales definen los bits de la imagen, en total son 6 bits, cuya codificación a decimal se muestra en la ecuación (2.10) y nos proporcionan un total de 64 balizas diferentes.

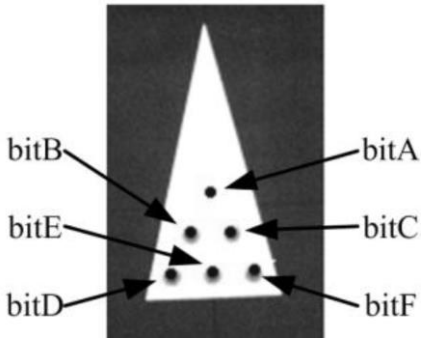


Figura 2.15. Codificación de las balizas.

$$\text{Código} = \begin{bmatrix} 2^0 & 2^1 & 2^2 & 2^3 & 2^4 & 2^5 \end{bmatrix} \begin{bmatrix} \text{bit A} \\ \text{bit B} \\ \text{bit C} \\ \text{bit D} \\ \text{bit E} \\ \text{bit F} \end{bmatrix} \quad (2.10)$$

Por lo que la detección de marcas se hace en tres pasos esenciales:

- Detección de bordes y binarización de la imagen.
- Detección de líneas utilizando la transformada de Hough.
- Orientación de la línea o flecha y código interior.

3. MATERIAL EMPLEADO

En este apartado se presenta el material que utilizaremos en este proyecto, características del PC utilizado, entorno de trabajo y robot móvil.

3.1 PC UTILIZADO: ACER ASPIRE 5735Z

Todos los algoritmos que se describirán en las sucesivas secciones de este proyecto han sido programados y testeados en un ordenador personal portátil, el ACER Aspire 5735Z (**Figura 3.1**).

Figura 3.1. ACER Aspire 5735Z.

En la **Tabla 3.1** tenemos algunas características del ACER Aspire 5735Z.

Sistema operativo	Microsoft Windows 8 Pro
Procesador	Intel Pentium Dual CPU T3200 a 2.00GHz
Memoria RAM	3 GB
Disco duro	Hitachi HTS543225L9A300, 232,88 GB

Tabla 3.1. Características ACER Aspire 5735Z.

3.2 SOFTWARE: MATLAB

Este proyecto está desarrollado íntegramente en Matlab, concretamente en la versión R2010a.

MATLAB es un entorno de cálculo técnico de altas prestaciones para cálculo numérico y visualización. Integra: Análisis numérico, cálculo matricial, procesamiento de señales y gráficos.

Mediante MATLAB crearemos las funciones específicas que se ajusten a nuestras necesidades.

3.3 ROBOT SURVEYOR SRV1B

Se trata de un robot autónomo tipo tanque de dimensiones reducidas, controlado remotamente a través de WiFi (**Figura 3.2**).

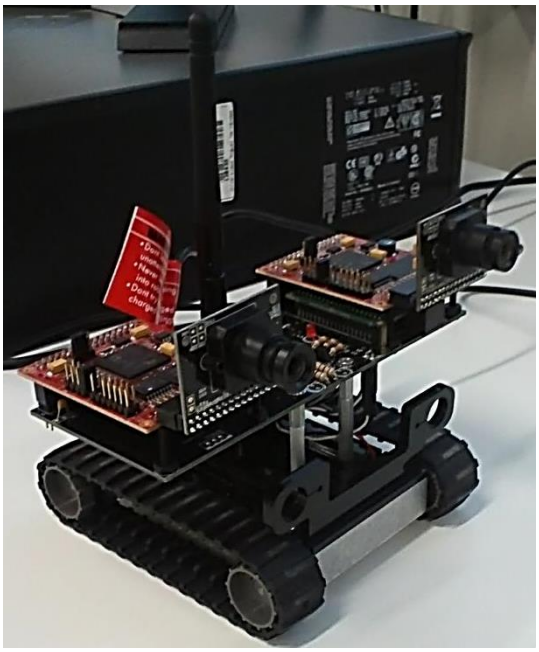


Figura 3.2. Surveyor SRV1B.

Está diseñado con fines educativos para la investigación y exploración, mediante el control a través de internet o red inalámbrica. El robot Surveyor SRV1 emplea el SRV1 Blackfin Camera Board con un procesador Blackfin BF537 a 500 MHz con 32 MB de memoria SDRAM, 4MB de memoria flash, una cámara digital con una resolución desde los 160x128 hasta los 1280x1024 píxeles (**Figura 3.3**) y un total de 8 calidades distintas de compresión JPEG. También puede ser controlado vía red inalámbrica (WiFi 802.11b/g) con buenos resultados incluso para largos alcances, del orden de 100 metros en interior y 1000 metros en el exterior [10] [11].

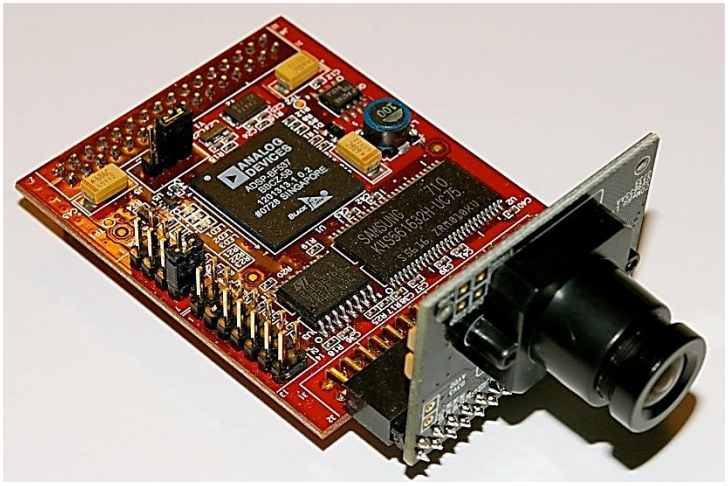


Figura 3.3. Una de las cámaras montadas en el SURVEYOR SRV1B.

En la **Tabla 3.2** se muestran las 4 resoluciones soportadas por la cámara del Surveyor.

	Resoluciones soportadas
0	160x120
1	320x240
2	640x480
3	1280x1024

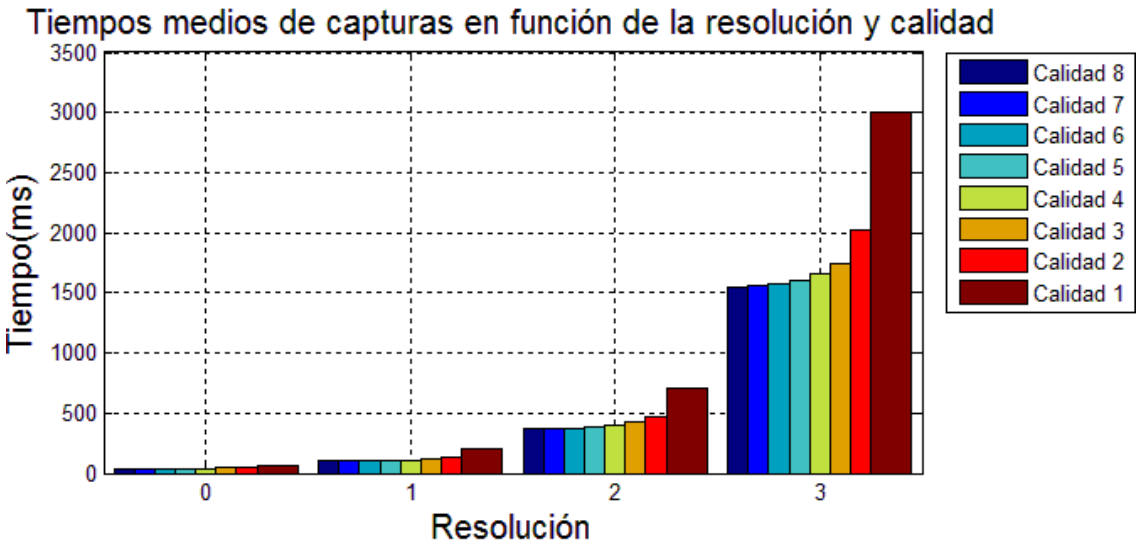
Tabla 3.2. Resoluciones admitidas.

Para tener una medida de los tiempos de procesado invertidos para la captura, compresión y envío de las imágenes captadas vía WiFi tenemos en la **Tabla 3.3** los tiempos medios para todas las resoluciones y calidades de compresión JPEG en milisegundos [11].

		Resolución (tiempo en milisegundos)			
		0	**1**	**2**	**3**
Calidad	**8**	36,80	98,24	369,20	1550,50
	7	36,71	102,20	375,40	1563,70
	6	37,93	104,50	375,20	1578,00
	5	40,42	106,50	387,50	1608,10
	4	41,22	111,00	402,40	1656,50
	3	44,65	125,80	430,90	1743,80
	2	48,81	135,80	474,60	2022,40
	1	68,86	207,80	709,40	3001,30

Tabla 3.3. Tiempos medios para la captación y envío de imágenes.

Representando los datos de la **Tabla 3.3** en la **Gráfica 3.1** observamos como el aumento de resolución y calidad va asociado a un aumento del tiempo de obtención de la imagen



Gráfica 3.1. Tiempos medios de capturas en función de la resolución y calidad.

3.3.1 SURVEYOR TOOLBOX DE MATLAB

El manejo del robot así como la obtención de las imágenes se realiza a través de una ToolBox en Matlab, a continuación se describen las funciones de las cuales dispone [11]:

- *addSurveyorJavaPath.m*: Comando interno utilizado por *initializeRobot.m* para añadir el path de las clases de Java.
- *initializeRobot.m*: Establece la conexión con el robot.
- *sendDriveCommand.m*: Envía comandos para manejar el robot.
- *Get_srv_image.m*: Devuelve una imagen de la cámara del robot.
- *setLasers.m*: Activa o desactiva el láser.
- *setImageResolution.m*: Establece la resolución de la imagen que es enviada por el robot.
- *setImageQuality.m*: Establece la calidad de la imagen JPEG enviada por el robot.
- *setImageCaption.m*: Activa o desactiva la captación de imágenes.
- *shutdownRobot.m*: Cierra la conexión con el robot.

Para inicializar el robot, debemos poner en la línea de comandos:

>> srv1robot = initializeRobot('192.168.1.xx')

Donde “xx” indica el número de identificación del robot. Una vez iniciado, ya está listo para enviar diferentes comandos al robot usando las funciones de Matlab comentadas anteriormente.

3.3.2 CORRECCIÓN EFECTO FISHEYE

Un efecto degradante que se da en las cámaras de dimensiones reducidas (como la que porta el Surveyor SRV1B) es el de fisheye. Consiste en una deformación de la imagen captada dando sensación de espejo convexo o de “barril” convirtiendo líneas rectas de la imagen en curvas.

Una forma de compensar estos efectos consiste en aplicar un filtrado que realice la operación inversa, esto es, un efecto “cojín” sobre la imagen captada.

En la **Figura 3.4** se representa el efecto cojín sobre la imagen de una retícula.

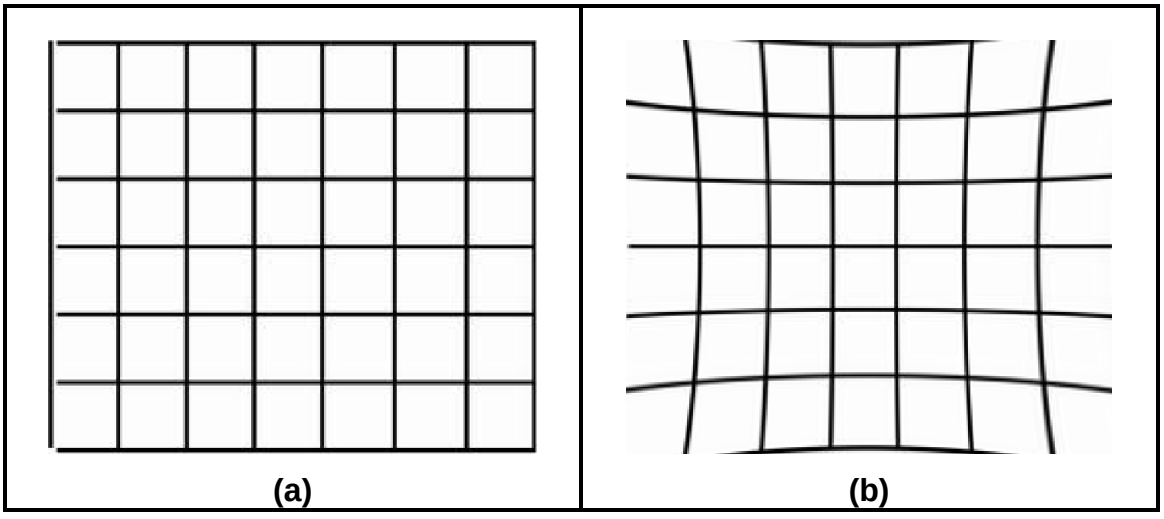


Figura 3.4. Retícula de prueba y efecto cojín.

Si aplicamos este efecto a las imágenes captadas por la cámara de nuestro robot obtenemos los resultados que se ven en la **Figura 3.5**.

Figura 3.5. Corrección del efecto ojo de pez en las imágenes captadas por el robot.

En las imágenes anteriores observamos como las líneas curvas que tenemos en la primera imagen se compensan en la segunda, a cambio sufrimos un recorte en los extremos, consecuencia de compensar el efecto “ojo de pez”.

4. DESARROLLO

En este tercer apartado se desarrollan los algoritmos que nos permitan detectar líneas en imágenes a partir de la transformada de Hough.

En primer lugar se presentan algoritmos de detección de bordes, paso previo para el desarrollo de la transformada de Hough. A continuación se desarrollan dichos algoritmos, testeando la calidad de los mismos en cuanto a semejanza con las líneas reales y tiempos de cómputo utilizando como imagen de prueba la que se representa en la **Figura 4.1**. En la sección 0 hablamos de la extracción de las líneas más significativas de la imagen. También se diseñan los códigos detectables y se comprueba, poniendo en práctica los apartados desarrollados, la detección de marcas, tanto simples como múltiples.

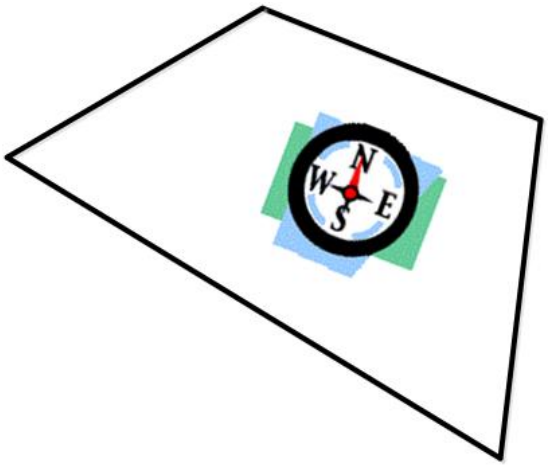
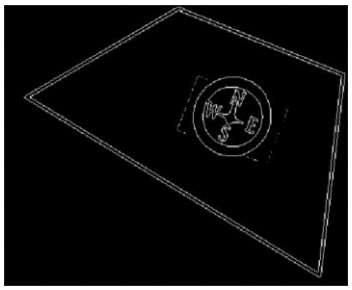


Figura 4.1. Polígono de prueba.

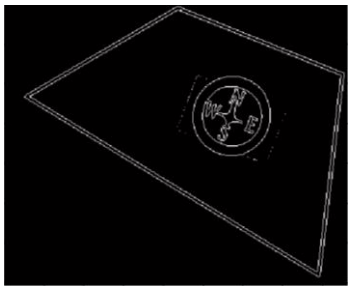
4.1 COMPARACIÓN DE MÁSCARAS PARA DETECCIÓN DE BORDES

Como paso previo a la implementación de la transformada de Hough necesitamos detectar los bordes que se encuentran en la imagen. Durante este proceso obtendremos una imagen binaria donde se representarán todos los puntos frontera que se encuentren en la imagen. En la **Figura 4.2** tenemos los resultados de la detección de bordes de la **Figura 4.1** utilizando 3 máscaras diferentes, en concreto “Sobel”, “Prewitt” y “Roberts” [5].



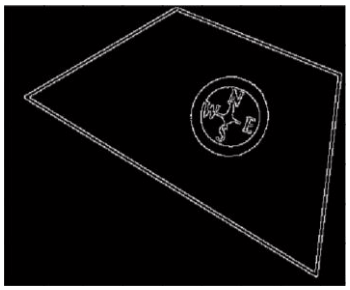
(a)

Sobel



(b)

Prewitt



(c)

Roberts

Figura 4.2. Efectos utilizando las máscaras de “Sobel”, “Prewitt” y “Roberts”.

Ante la futura aplicación de una de estas tres máscaras para la detección de líneas en la imagen podemos ver como “Roberts” agudiza más los puntos frontera, en concreto tenemos aproximadamente un tercio más de puntos frontera utilizando “Roberts” en comparación con las demás máscaras (**Tabla 4.1**). Este número de puntos está directamente relacionado con el tiempo de cómputo de los algoritmos que desarrollaremos en los siguientes apartados, pero en este caso nos interesa obtener las líneas de la imagen lo más definidas posible por lo que será el que utilizaremos en el futuro desarrollo.

Número de puntos frontera encontrados		
Sobel	**Prewitt**	**Roberts**
2995	2981	4121

Tabla 4.1. Número de puntos frontera con cada máscara.

Veamos que sucede con los tiempos de cómputo que podemos ver en la siguiente tabla (**Tabla 4.2**).

		Tiempos de cómputo en milisegundos		
		Sobel	**Prewitt**	**Roberts**
Dimensiones de la imagen	200x150	79	63	67
	430x352	62	67	62
	430x352	69	59	63
	394x394	65	59	62
	736x611	64	62	60
	400x264	60	59	65
	1024x768	86	67	69
	500x375	58	59	58
	800x335	59	59	59
Tiempo medio		67,02	61,57	62,93

Tabla 4.2. Tiempos de cómputo de distintas imágenes para los algoritmos detectores de bordes.

Atendiendo a la tabla anterior no encontramos grandes diferencias entre las 3 máscaras, siendo la más rápida “Prewitt” y la más lenta “Sobel” pero estando los 3 casos entre los 62 y 67 milisegundos de procesado, por lo que el uso de cualquiera de estas máscaras no significa una mejora significativa de tiempo con respecto a las otras.

4.2 IMPLEMENTACIÓN DEL ALGORITMO DE HOUGH

Una vez realizado el filtrado previo se binariza la imagen y obtenemos una matriz donde representamos con un uno los puntos frontera y con cero los que no pertenecen a ningún borde.

A continuación pasamos a obtener la matriz de Hough, esta tendrá las dimensiones descritas en la introducción teórica, recorreremos cada punto encontrado calculando **ρ** para todo el conjunto de ángulos **θ**, realizamos una discretización de estos valores e incrementaremos el acumulador en la posición **(ρ_i, θ_j)** (**Figura 4.3**).

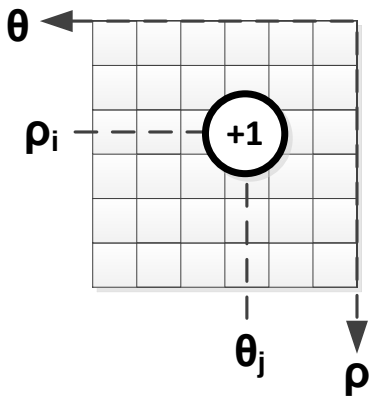
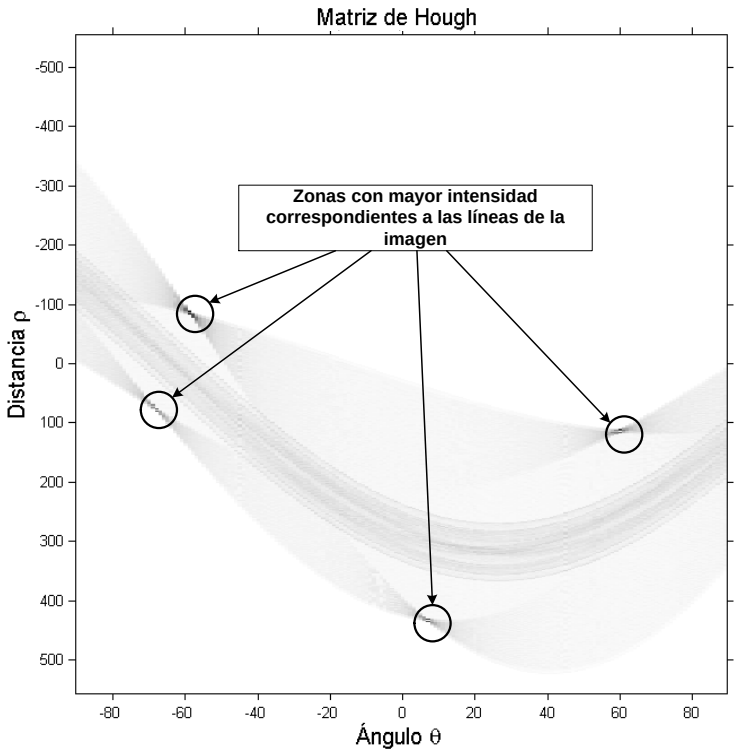


Figura 4.3. Acumulación en una posición determinada de la matriz de Hough.

Una vez recorridos todos los puntos de la imagen pasamos a guardar las coordenadas de las celdas con un mayor número acumulado, para a continuación poder obtener las rectas de la imagen.

En la **Gráfica 4.1** representamos la matriz de Hough de la **Figura 4.1**, para todo el rango posible de valores de distancia **ρ** y ángulos **θ** (en grados) como se ha establecido en la introducción.



Gráfica 4.1. Representación de la Matriz de Hough de un polígono de 4 lados.

Podemos observar en la gráfica anterior como tenemos 4 zonas de mayor intensidad, estas corresponden a las líneas de la imagen.

4.2.1 VERSIÓN 1: ALGORITMO DE HOUGH

Esta versión del algoritmo corresponde a la implementación literal del algoritmo de Hough. A continuación podemos ver el pseudocódigo del script realizado en Matlab:

1: cargamos la imagen
2: detectamos los bordes en la imagen
3: por cada punto detectado:
4: para todos los posibles ángulos :
5: calculamos ρ para el punto actual con un ángulo θ
6: se incrementa la posición (ρ, θ) en el acumulador
7: buscamos las posiciones con los mayores valores en el acumulador
8: se devuelven las ecuaciones de las rectas cuyos valores son los mayores en el acumulador

4.2.2 VERSIÓN 2: ALGORITMO SEMIALEATORIO

Este algoritmo realiza los mismos pasos que el anterior, pero en lugar de computar todos los puntos de la imagen sólo lo hacemos de un número aleatorio, en primer lugar establecemos el porcentaje de puntos que queremos evaluar y comenzamos a ejecutar el algoritmo:

1: cargamos la imagen
2: detectamos los bordes en la imagen
3: escogemos un número aleatorio de puntos
4: por cada punto aleatorio:
5: para todos los posibles ángulos :
6: calculamos ρ para el punto actual con un ángulo θ
7: se incrementa la posición (ρ, θ) en el acumulador
8: buscamos las posiciones con los mayores valores en el acumulador
9: se devuelven las ecuaciones de las rectas cuyos valores son los mayores en el acumulador.

4.2.3 VERSIÓN 3: ALGORITMO ALEATORIO

En esta versión escogemos al azar un tanto por ciento de los puntos obtenidos y los comparamos todos con todos, a partir de cada par de puntos obtenemos el ángulo **θ** y la distancia **ρ** del punto medio del par actual.

1: cargamos la imagen
2: detectamos los bordes en la imagen
3: escogemos un número aleatorio de pares de puntos
4: por cada par de puntos aleatorios:
5: obtenemos el ángulo *θ* de la recta que pasa por esos dos puntos
6: calculamos *ρ* para el punto medio y el ángulo *θ* obtenido
7: se incrementa la posición *(ρ, θ)* en el acumulador
8: buscamos las posiciones con los mayores valores en el acumulador
9: se devuelven las ecuaciones de las rectas cuyos valores son los mayores en el acumulador.

A partir de la implementación de los tres algoritmos podemos llegar a la conclusión de que el tiempo de cómputo de los mismos desciende con el número de puntos que utilizamos para obtener la matriz de Hough. No es necesario obtener las coordenadas polares de todos ellos, a partir de cierto número de puntos los resultados convergen al resultado óptimo, que serían los obtenidos en la versión 1. De esta forma, los tiempos utilizados por cada algoritmo se ordenan de la siguiente forma:

Versión 1 > Versión 2 > Versión 3

Donde el porcentaje de puntos utilizados en la versión 2 y 3 es el mismo.

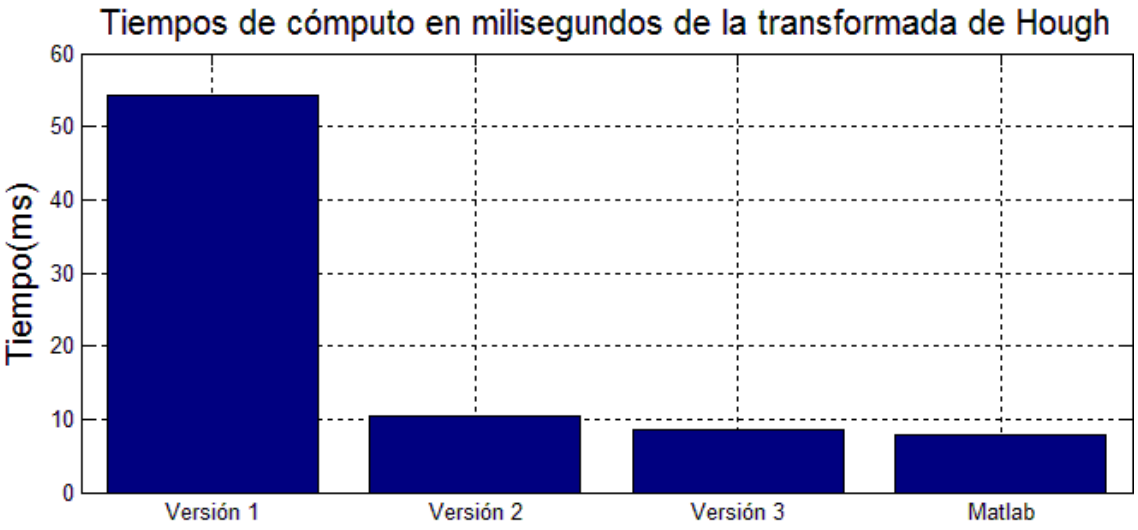
4.2.4 TIEMPOS DE PROCESAMIENTO

En la **Tabla 4.3** se muestra un ejemplo de los tiempos medios de procesamiento de cada algoritmo para distintas imágenes, también se obtiene el tiempo que utiliza la transformada de Matlab para poder compararlos con nuestros propios programas.

		Tiempos de cómputo en milisegundos de la transformada de Hough			
		Versión 1	**Versión 2**	**Versión 3**	**Matlab**
Dimensiones de la imagen	200x150	1,7911	1,0567	1,7347	1,672
	430x352	52	9	6	8
	430x352	43	7	4	6
	394x394	48	8	11	7
	736x611	100	17	13	13
	400x264	27	5	7	5
	1024x768	119	23	20	17
	500x375	24	6	3	4
	800x335	73	18	9	10
Tiempo medio		54,21	10,49	8,49	7,96

Tabla 4.3. Tiempos de cómputo de distintas imágenes para la transformada de Hough.

Podemos observar de una forma más sencilla los tiempos medios de cómputo de la **Tabla 4.3** en la **Gráfica 4.2**.



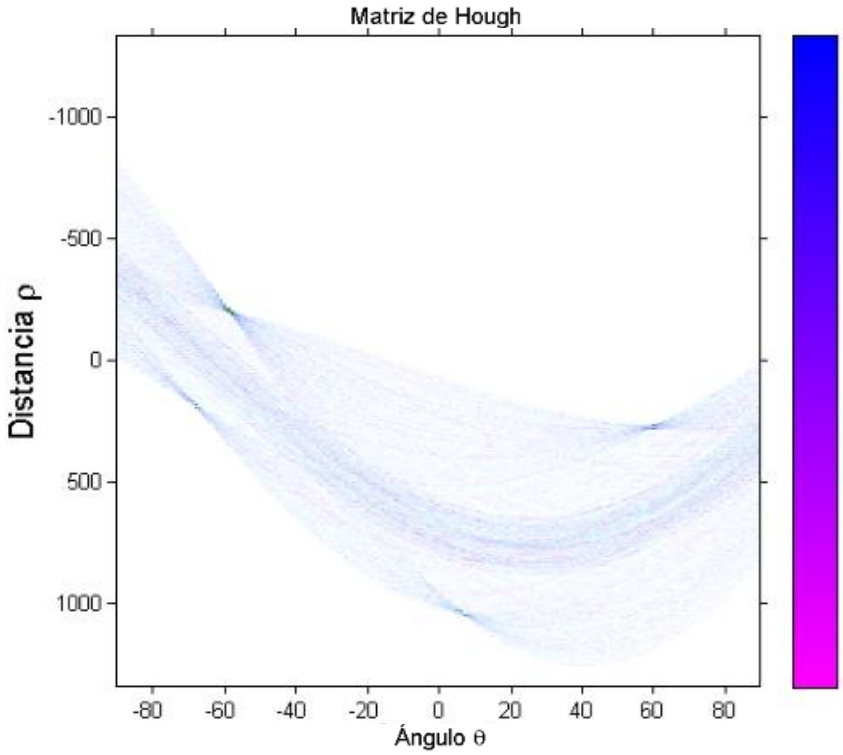
Gráfica 4.2. Tiempos medios de cómputo en milisegundos de la transformada de Hough.

Analizando los datos podemos observar que el algoritmo que mejor se comporta es el aleatorio (Versión 3), obteniendo generalmente los tiempos más reducidos. Comparando la transformada normal con la de Matlab vemos que esta última es más eficiente haciendo el mismo número de cálculos. En el caso de la transformada semialeatoria los tiempos los podemos equiparar a los obtenidos por la transformada de Hough de Matlab.

También cabe destacar que el tiempo de procesamiento no está directamente relacionado con el tamaño de la imagen y sí con el número de píxeles frontera que podemos encontrar en el procesado previo.

En capítulos posteriores se compararán los tiempos de cada algoritmo reduciendo el tamaño de la imagen de entrada y modificando la resolución del ángulo **θ**.

Para finalizar esta parte en la **Gráfica 4.3** se representa una comparación de la matriz de Hough obtenida con las 3 versiones del algoritmo para la **Figura 4.1**.



Gráfica 4.3. Representación de la matriz de Hough para las 3 versiones del algoritmo.

En la gráfica anterior se hace una representación superponiendo las matrices de Hough de los tres algoritmos implementados. De esta forma podemos ver que en las zonas con color azul más oscuro es donde encontramos una mayor semejanza entre los resultados de los 3 algoritmos, por lo que nos da a entender que aunque utilicemos un número menor de puntos en las versiones 2 y 3 podremos encontrar igualmente las líneas más importantes de la imagen.

4.2.5 COMPARACIÓN DE ALGORITMOS

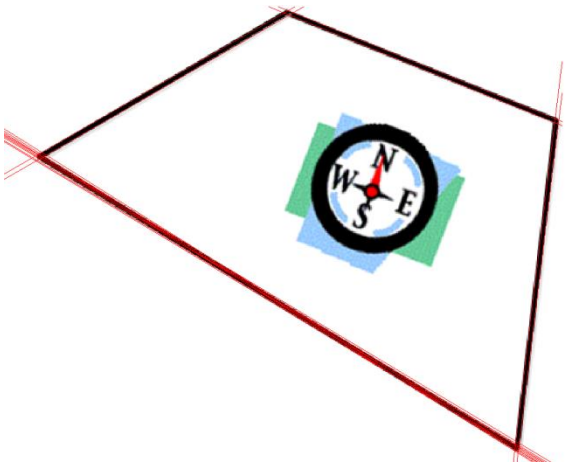
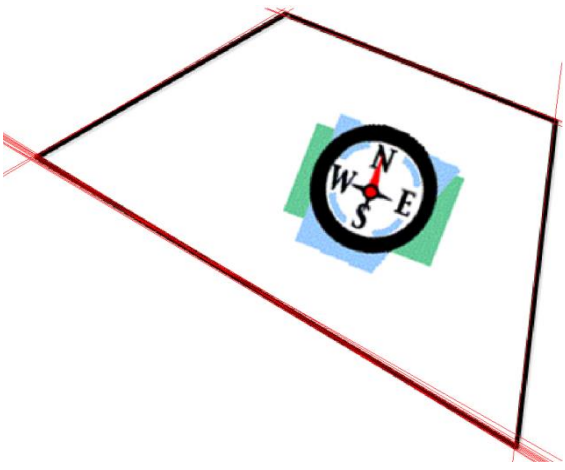
4.2.5.1 REDUCCIÓN DE TIEMPO EN LAS VERSIONES 1 Y 2

Como se ha representado en la **Tabla 4.3** el algoritmo aleatorio es con el que obtenemos unos procesados más rápidos, en esta sección se intenta reducir los tiempos de procesado de las versiones 1 y 2 con el fin de obtener tiempos similares a los de la versión 3. Esta reducción de tiempo la llevaremos a cabo reduciendo el tamaño de la imagen y la resolución del ángulo theta. En el caso de la versión 3 un cambio de la resolución del ángulo no afecta al algoritmo.

El tiempo objetivo por tanto es acercarnos a 6 ms, que es el tiempo utilizado por la versión aleatoria para procesar la **Figura 4.1**.

A continuación se presentan 6 ejemplos en los que reduciendo los parámetros comentados anteriormente intentamos reducir los tiempos de procesado. En primer lugar se presenta la resolución del ejemplo, a la izquierda la imagen obtenida con el algoritmo normal y a la derecha la del algoritmo semialeatorio. Para terminar se hace un breve comentario.

<table>
<tr><td colspan="2">Resolución de la imagen: 100%

Resolución de θ: 1º</td></tr>
<tr><td> 
(a)</td><td> 
(b)</td></tr>
<tr><td>52 ms</td><td>10 ms</td></tr>
</table>

La detección de líneas es buena, aunque los tiempos de procesado están lejos del algoritmo aleatorio.

Figura 4.4. Reducción de tiempo en las versiones 1 y 2. Ejemplo 1.

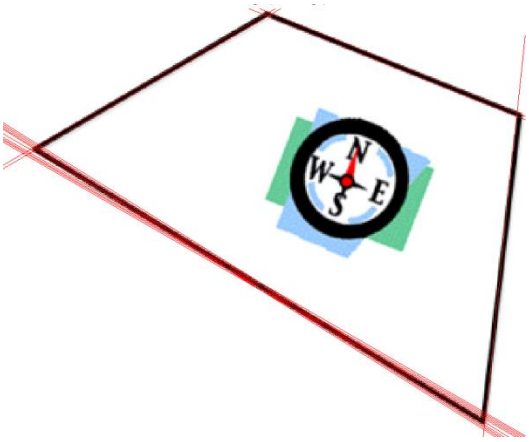
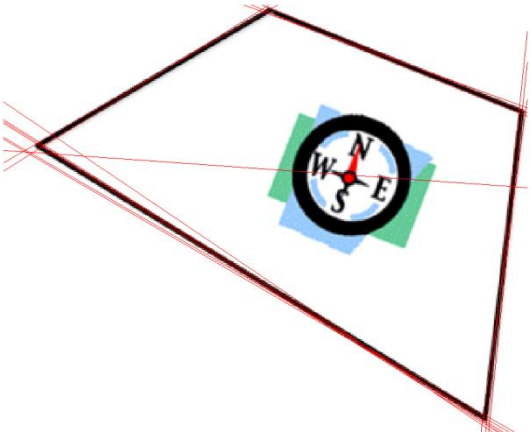
Resolución de la imagen: 75% Resolución de **θ**: 2º	
 **(a)**	 **(b)**
18 ms	3 ms
Aparece alguna recta que no pertenece a ninguna línea, pero los tiempos de computo de la transformada semialeatoria empiezan a ser menores que la transformada aleatoria.	

Figura 4.5. Reducción de tiempo en las versiones 1 y 2. Ejemplo 2.

Resolución de la imagen: 50% Resolución de **θ**: 5º

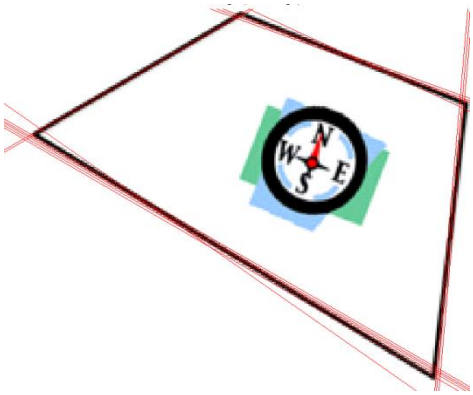
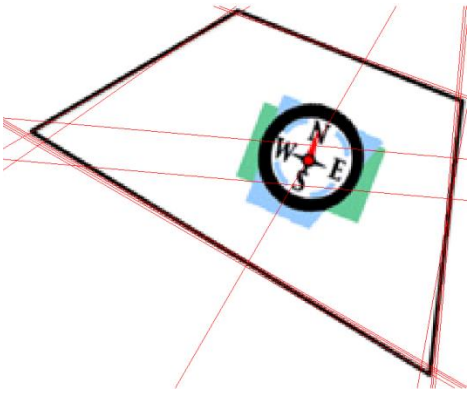
 **(a)**	 **(b)**
7 ms	2 ms
Obtenemos resultados aceptables en la transformada normal en calidad de resultados y reducción de tiempo. En el caso de la transformada aleatoria aunque obtenemos tiempos muy reducidos los resultados empiezan a ser de mala calidad.	

Figura 4.6. Reducción de tiempo en las versiones 1 y 2. Ejemplo 3.

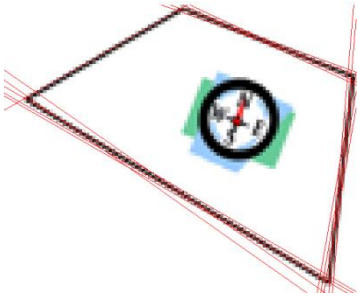
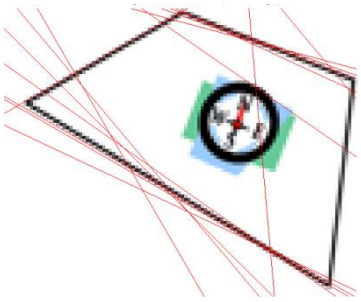
Resolución de la imagen: 25% Resolución de **θ**: 5º	
 **(a)**	 **(b)**
2.42 ms	0.56 ms
Tiempos de ejecución muy reducidos, pero resultados muy pobres en el caso de la versión 2 del algoritmo.	

Figura 4.7. Reducción de tiempo en las versiones 1 y 2. Ejemplo 4.

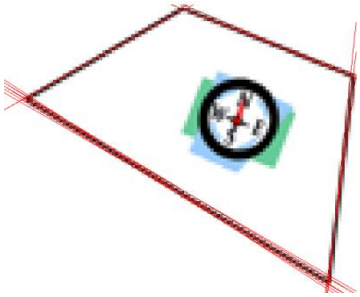
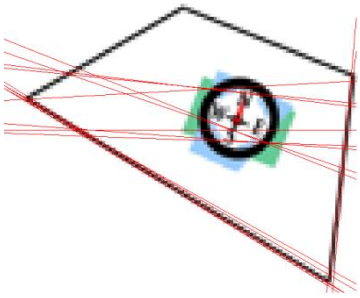
Resolución de la imagen: 25% Resolución de **θ**: 1º	
 (a)	 **(b)**
11 ms	2 ms
Reducimos el tiempo (en el caso de la semialeatoria) y obtenemos resultados satisfactorios en la transformada normal.	

Figura 4.8. Reducción de tiempo en las versiones 1 y 2. Ejemplo 5.

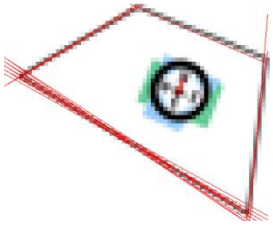
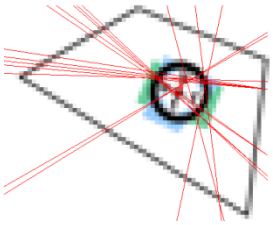
Resolución de la imagen: 15% Resolución de **θ**: 1º	
 **(a)**	 **(b)**
6 ms	1 ms
Una vez más reducimos los tiempos, la transformada normal se comporta aceptablemente, pero la semialeatoria no.	

Figura 4.9. Reducción de tiempo en las versiones 1 y 2. Ejemplo 6.

Para obtener una reducción del tiempo de ejecución en las transformadas de Hough normal y semialeatoria tenemos que reducir mucho las resoluciones de

la imagen y del ángulo theta, con lo que también se dificulta la correcta detección de líneas. En determinadas circunstancias, una buena elección del tamaño y resolución de theta nos puede dar mejores resultados la transformada semialeatoria que la aleatoria. Pero en definitiva, podemos decir que no funcionan mejor que la versión 3 aun sin reducir el tamaño de la imagen en este algoritmo. No se representan los resultados para ángulos menores e imágenes menos reducidas ya que obtenemos buenos resultados, pero tiempos muy alejados de la transformada aleatoria.

4.2.5.2 REDUCCIÓN DE TIEMPO EN LAS VERSIÓN 3

A continuación representamos los resultados de la versión 3 reduciendo el tamaño de la imagen.

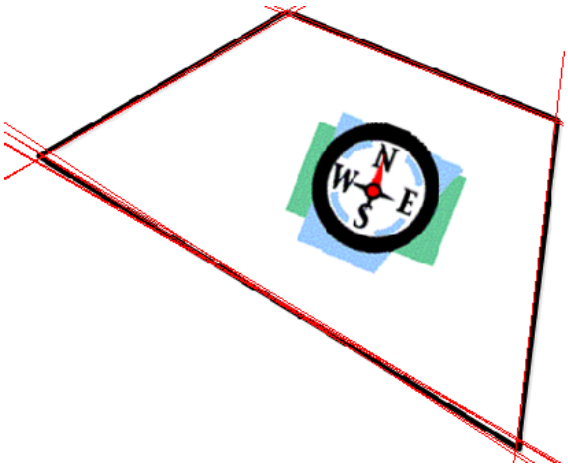
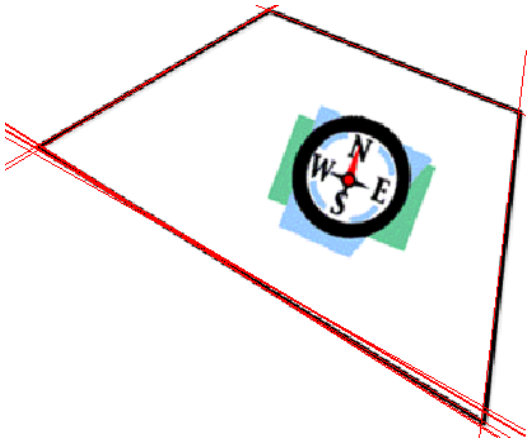
Resolución de la imagen: 100%

6 ms
Imagen resultante sin reducción de tamaño.

Figura 4.10. Reducción de tiempo en las versión 3. Ejemplo 1.

Resolución de la imagen: 75%

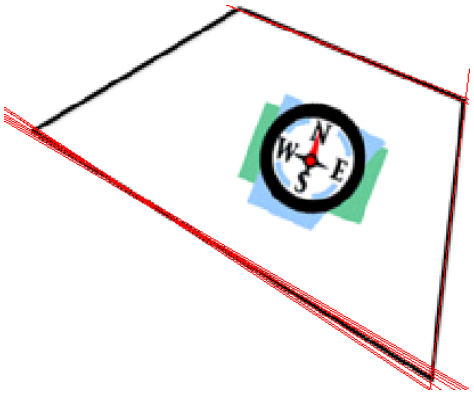


3 ms

Reducimos el tiempo a la mitad, obteniendo buenos resultados.

Figura 4.11. Reducción de tiempo en las versión 3. Ejemplo 2.

Resolución de la imagen: 50%



2.5 ms

Resultados similares a los de antes, tiempo algo inferior.

Figura 4.12. Reducción de tiempo en las versión 3. Ejemplo 3.

Resolución de la imagen: 25%

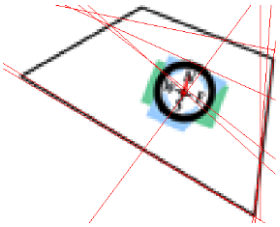

0.64 ms
Un tiempo de ejecución muy rápido, pero resultados muy pobres.

Figura 4.13. Reducción de tiempo en las versión 3. Ejemplo 4.

De forma general imágenes de resoluciones bajas afectan más a la versión aleatoria que a los algoritmos 1 y 2.

4.2.5.3 PROBLEMAS EN LA VERSIÓN 3, ALGORITMO ALEATORIO

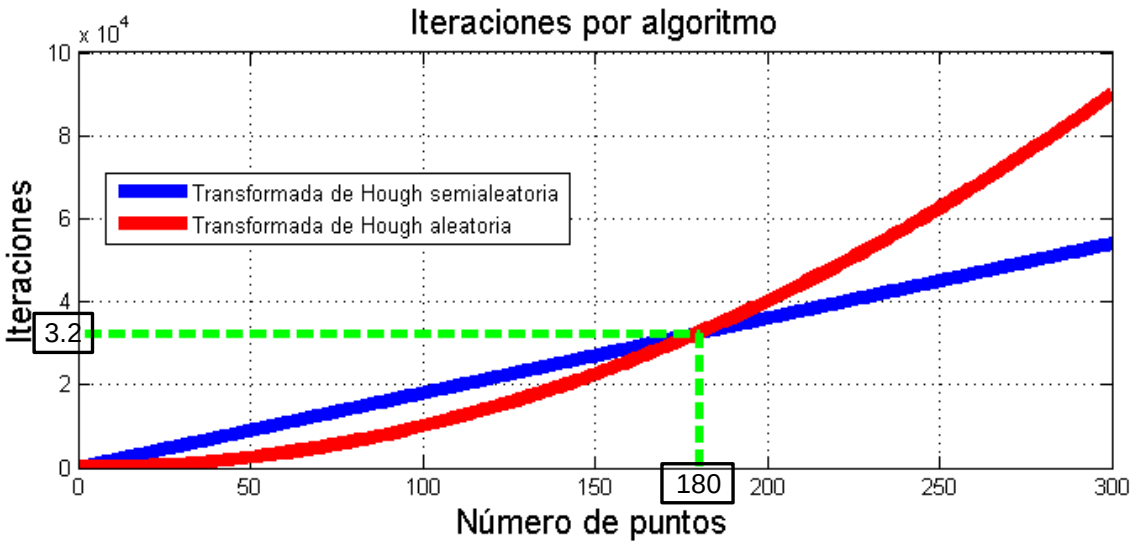
Como hemos visto en la **Tabla 4.3** la versión 3 del algoritmo de Hough es el que nos proporciona mejores resultados en cuanto al tiempo de cómputo, pero es un dato engañoso ya que este tiempo puede elevarse con respecto al que emplea la versión 2 (semialeatoria) en el caso de imágenes con un gran número de puntos frontera. Esto se debe a que el número de iteraciones que realiza la versión 3 es:

$$Iter_{versión\ 3} = N \cdot N = N^2 \qquad (4.1)$$

Donde N es el número de puntos de la imagen que utilizamos para obtener las rectas de la imagen. En el caso de la versión semialeatoria el número de iteraciones es:

$$Iter_{versión\ 2} = N \cdot 180 \qquad (4.2)$$

Igual que antes N es el número de puntos de la imagen y 180 son los grados para una resolución de theta igual a uno. Cuando N es menor que 180 el algoritmo aleatorio se comporta de forma más rápida. Pero en el caso de que el número de puntos se eleve por encima de 180 el número de iteraciones aumenta exponencialmente (**Gráfica 4.4**).

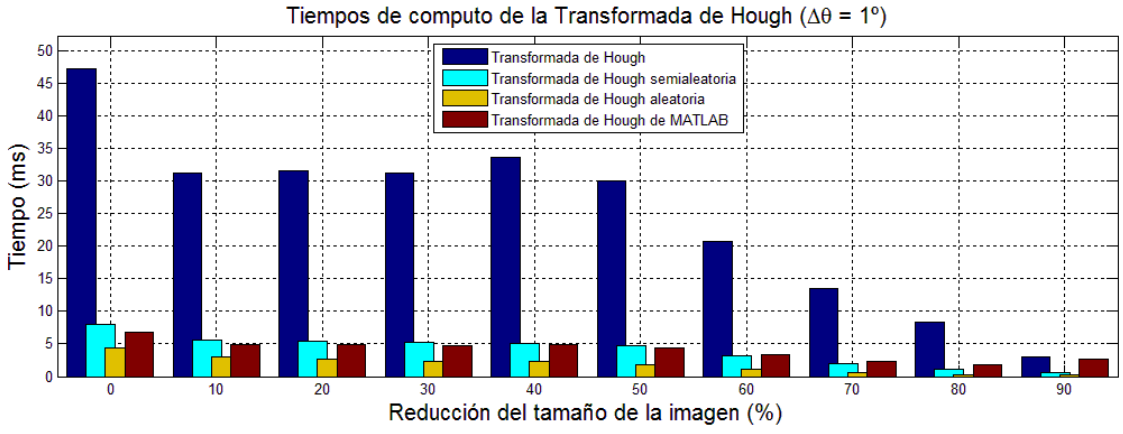


Gráfica 4.4. Comparación del número de iteraciones por algoritmo.

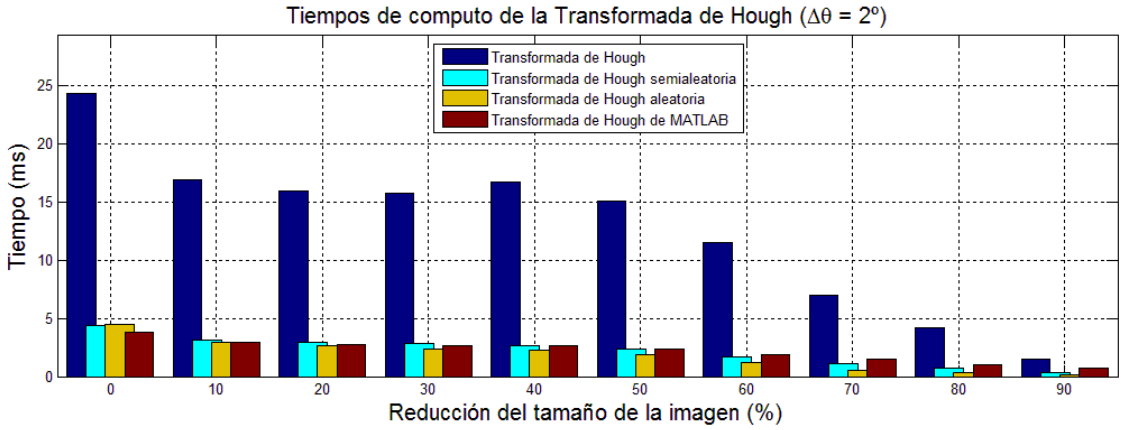
Podemos ver en la gráfica anterior como a partir de 180 puntos en la imagen el número de iteraciones para la transformada aleatoria aumenta rápidamente con respecto a las iteraciones realizadas por el algoritmo semialeatorio.

4.2.5.4 TIEMPOS DE CÓMPUTO

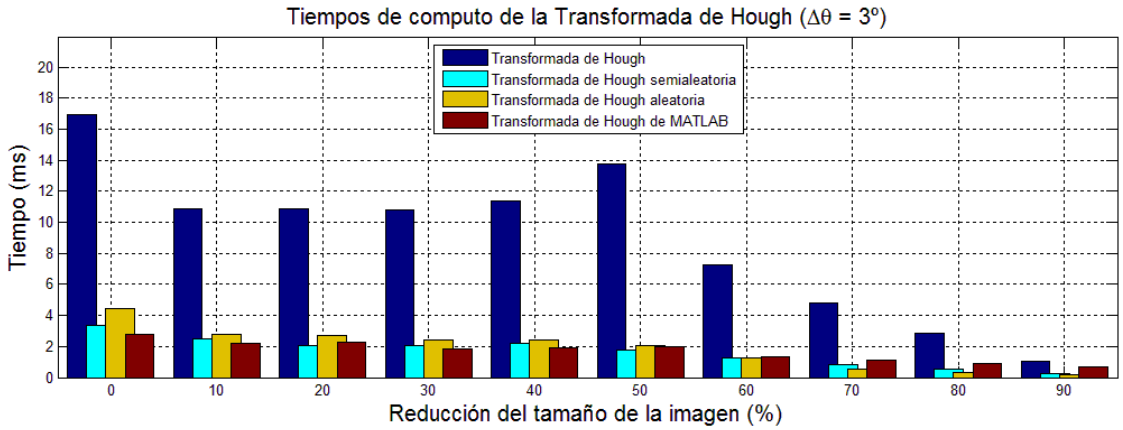
En este apartado se representan los tiempos de ejecución para reducciones de las imágenes de 10% en 10% y resoluciones de theta de 1, 2, 3, 5 y 10 grados. Recordar que la resolución de theta no afecta al algoritmo aleatorio. La imagen en todas las pruebas es la misma que la anterior (**Figura 4.1**).



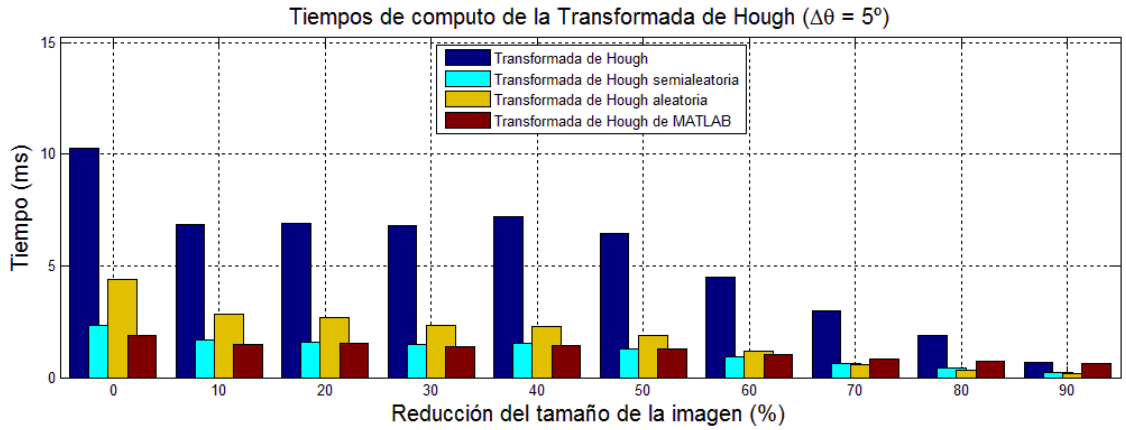
Gráfica 4.5. Tiempo de cómputo para Δθ = 1°.



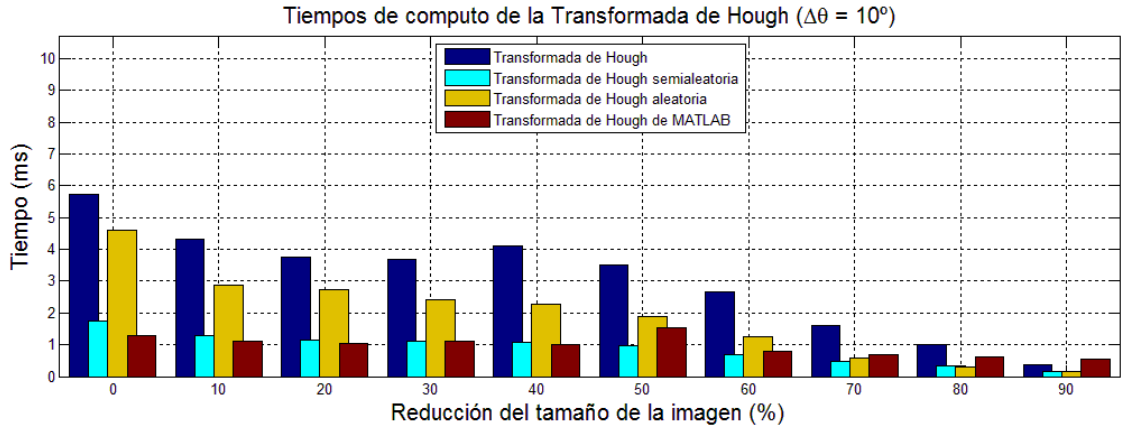
Gráfica 4.6. Tiempo de cómputo para Δθ = 2°.



Gráfica 4.7. Tiempo de cómputo para Δθ = 3°.



Gráfica 4.8. Tiempo de cómputo para Δθ = 5°.



Gráfica 4.9. Tiempo de cómputo para Δθ = 10°.

Al reducir el tamaño de la imagen y la resolución de theta somos capaces de acercarnos a los tiempos de procesado que utiliza la transformada aleatoria.

4.2.5.5 CONCLUSIONES

Para obtener unos resultados óptimos la mejor opción es utilizar la versión 3 ya que esta transformada aleatoria obtiene un buen balance en cuanto a tiempo de ejecución y resultados. Si queremos reducir más los tiempos, reducimos la imagen, pero sin superar un 50%. Pero como ya se ha comentado antes, reduciendo la calidad el algoritmo semialeatorio se puede acercar al aleatorio en resultados y tiempo.

4.2.6 CALIDAD DE LOS RESULTADOS

En este apartado se pretende establecer una medida de la calidad de los resultados proporcionados por los algoritmos, es decir, la semejanza de las líneas encontradas con las rectas reales de la imagen.

Esta medida de calidad consiste en obtener las componentes (**ρ, θ**) de las rectas de la imagen y calcular su distancia con las obtenidas en cada uno de los algoritmos. Este cálculo se realiza con el teorema del coseno explicado en el apartado teórico, una vez obtenida esta distancia realizamos una normalización de los resultados que consiste en dividirlos por la máxima distancia posible que pueden haber entre dos rectas de la misma imagen. Para terminar adecuamos los resultados para obtener las gráficas haciendo el cálculo:

$$\text{Distancia} = 1 - \text{Distancia}_{norm} \quad (4.3)$$

Por lo que si los resultados se acercan a uno, es la máxima igualdad que puede haber entre dos rectas.

En la **Figura 4.14** tenemos numerada cada línea de la imagen por lo que en las gráficas que representaremos podremos ver la semejanza de cada línea obtenida con las reales de la imagen.

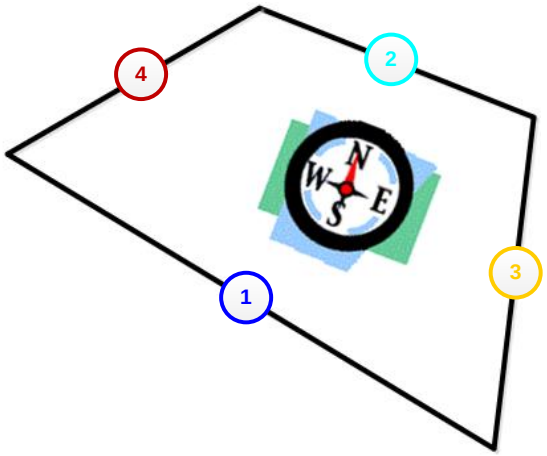
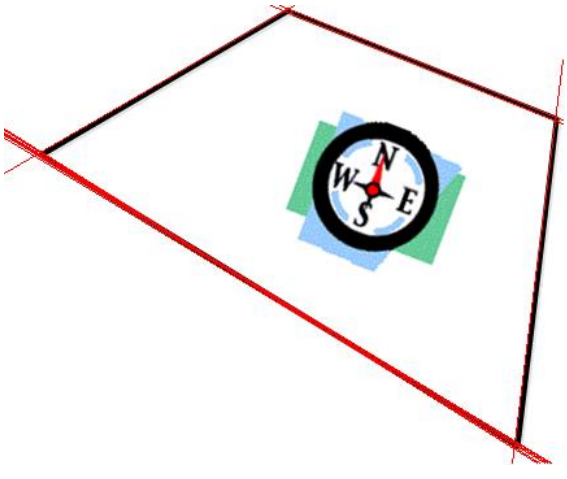
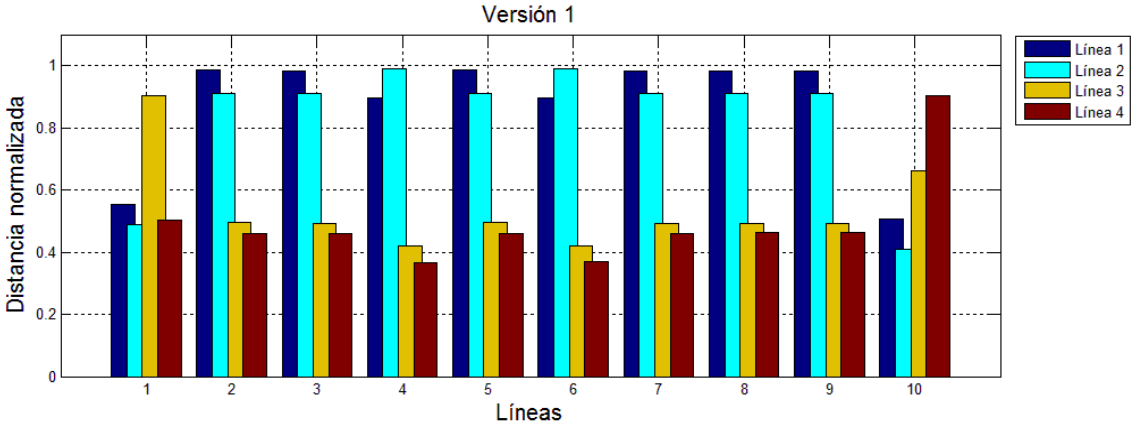


Figura 4.14. Polígono de prueba con las líneas numeradas.

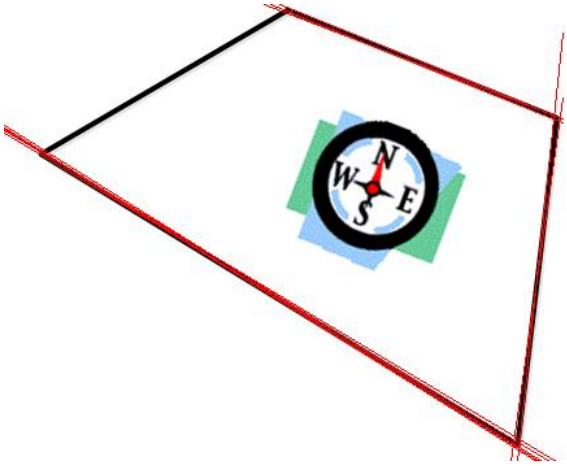
Se representan 10 líneas obtenidas con cada algoritmo y se hace un breve comentario de los resultados.

	Número de rectas pertenecientes a cada línea del polígono: Línea 1: 6 rectas. Línea 2: 2 rectas. Línea 3: 1 recta. Línea 4: 1 recta.



Todas las rectas encontradas pertenecen a alguna de las líneas de la imagen. En la gráfica el número de rectas encontradas sobre cada línea corresponde con las que se han dibujado en la imagen.

Figura 4.15. Versión 1 del algoritmo de Hough.



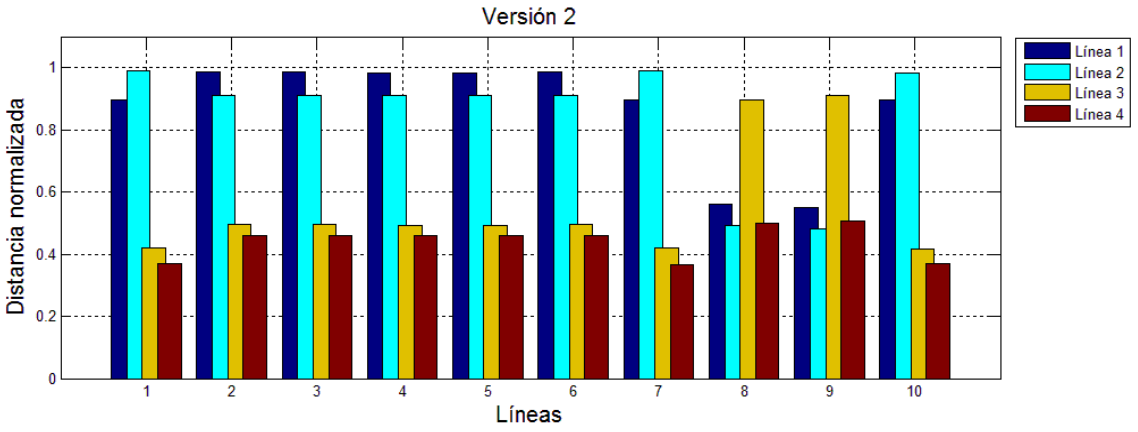
Número de rectas pertenecientes a cada línea del polígono:

Línea 1: 5 rectas.

Línea 2: 3 rectas.

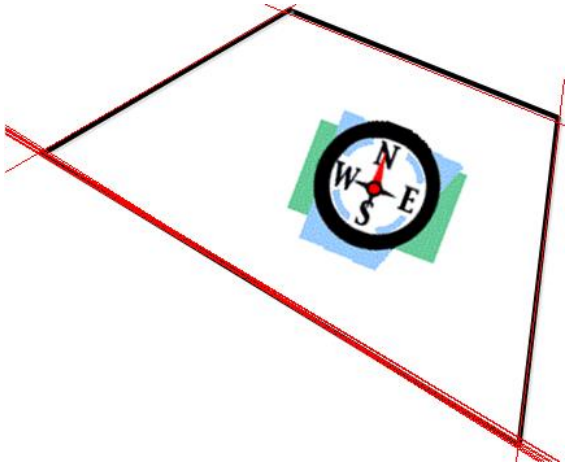
Línea 3: 2 recta.

Línea 4: 0 rectas.

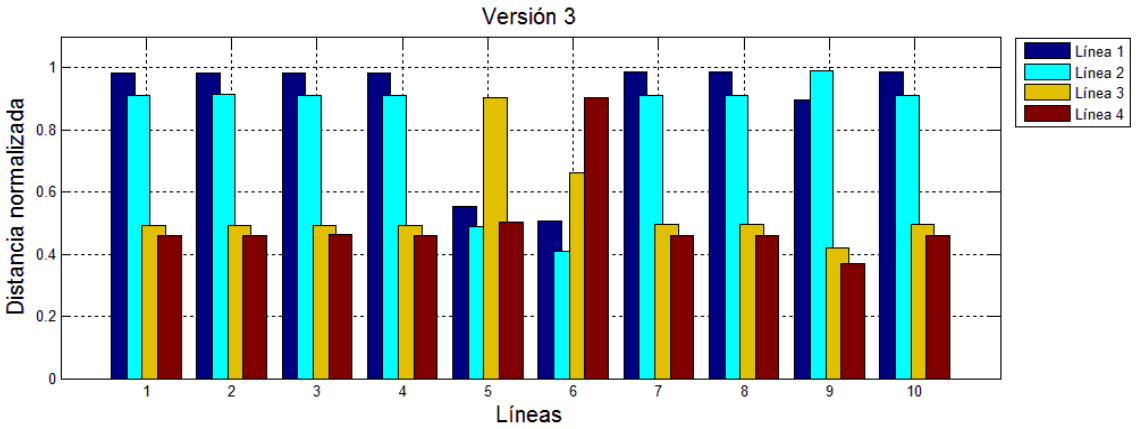


En este caso se comprueba que efectivamente en la gráfica no vemos ninguna línea perteneciente a la 4, ya que no se ha encontrado.

Figura 4.16. Versión 2 del algoritmo de Hough.



Número de rectas pertenecientes a cada línea del polígono:
Línea 1: 7 rectas.
Línea 2: 1 rectas.
Línea 3: 1 recta.
Línea 4: 1 rectas.



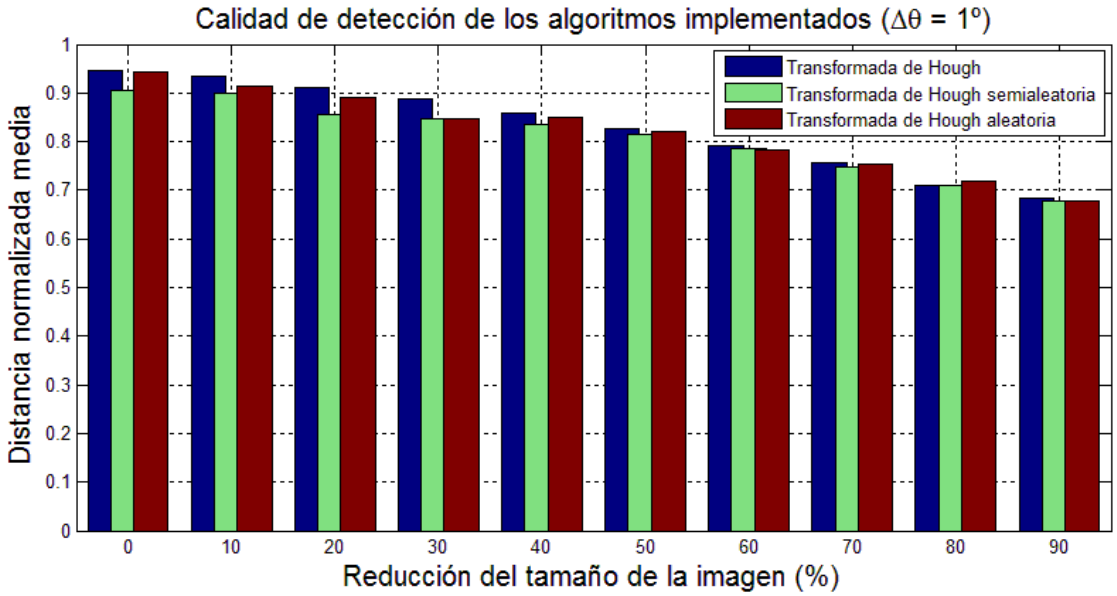
Con la versión 3 también podemos ver como las líneas de la gráfica se corresponden con las encontradas.

Figura 4.17. Versión 3 del algoritmo de Hough.

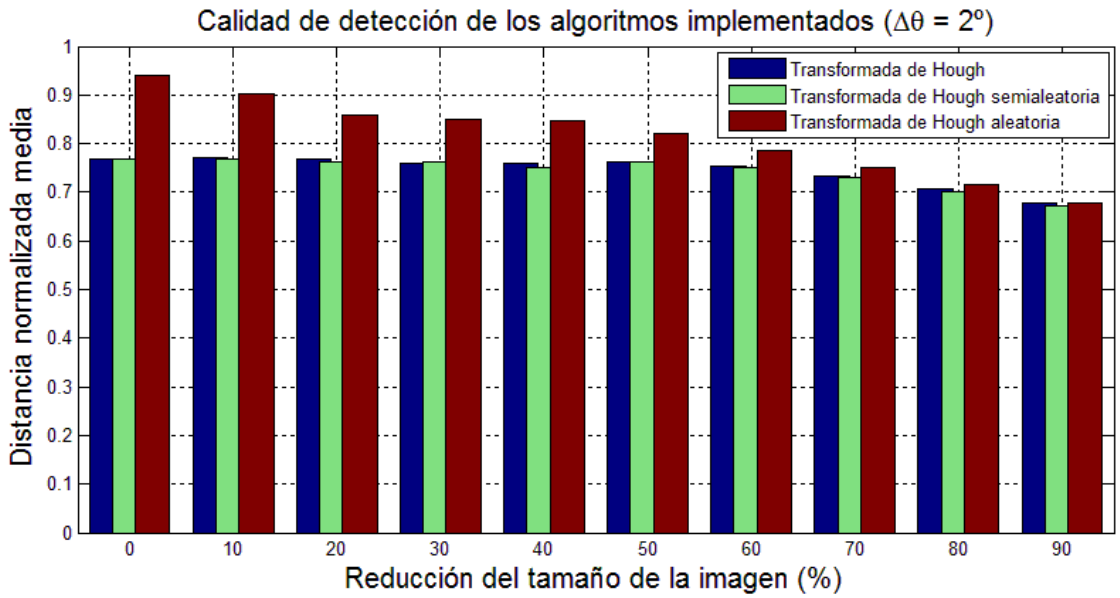
La calidad de los resultados en los 3 algoritmos es muy similar, acercándose los valores de cada recta a la máxima igualdad, teniendo en cuenta que todas las líneas dibujadas pertenecen a alguna de las rectas. Como detalle se observa que las rectas que pertenecen a las líneas 1 y 2 tienen grados de igualdad muy parecidos, esto se debe a que estas dos líneas son casi paralelas, por lo que tendrán un ángulo theta muy similar.

4.2.6.1 DISTANCIAS MEDIAS NORMALIZADAS

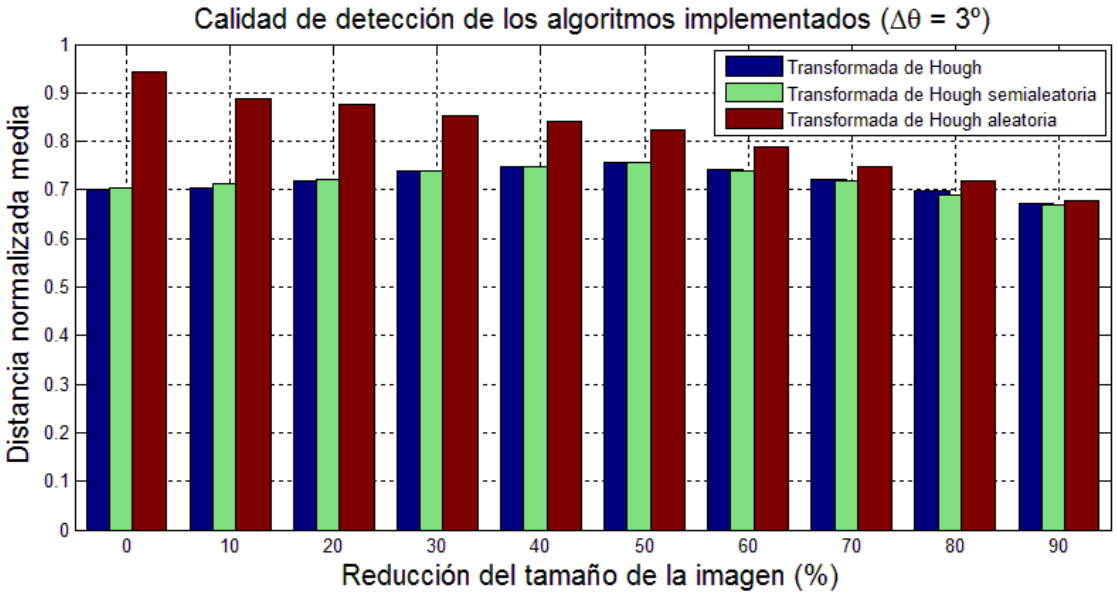
Para finalizar este apartado mostramos una medida de la calidad de los resultados a partir de la media de las distancias de las líneas buscadas con las reales de la imagen. De esta forma se representa en las siguientes gráficas dichas distancias en función de las resoluciones del ángulo theta y de la imagen.



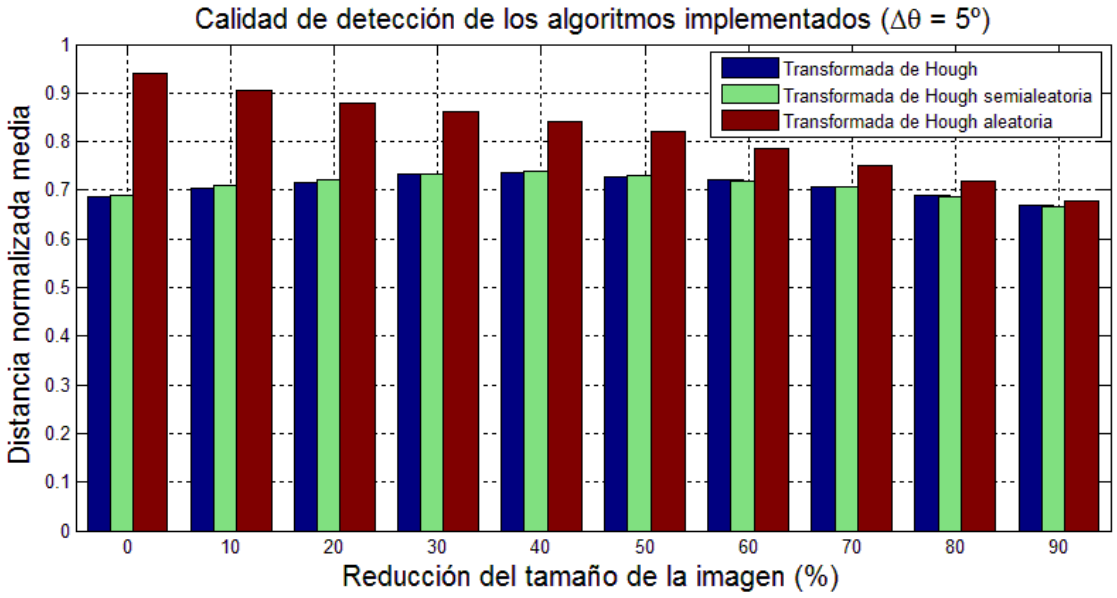
Gráfica 4.10. Distancia normalizada media para Δθ = 1°.



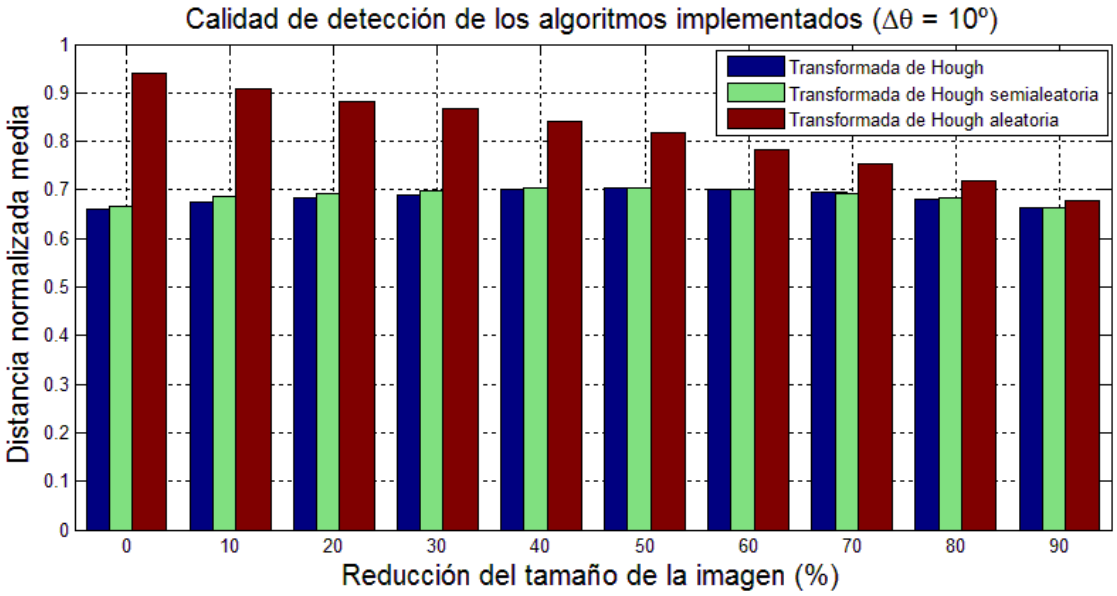
Gráfica 4.11. Distancia normalizada media para Δθ = 2°.



Gráfica 4.12. Distancia normalizada media para Δθ = 3°.



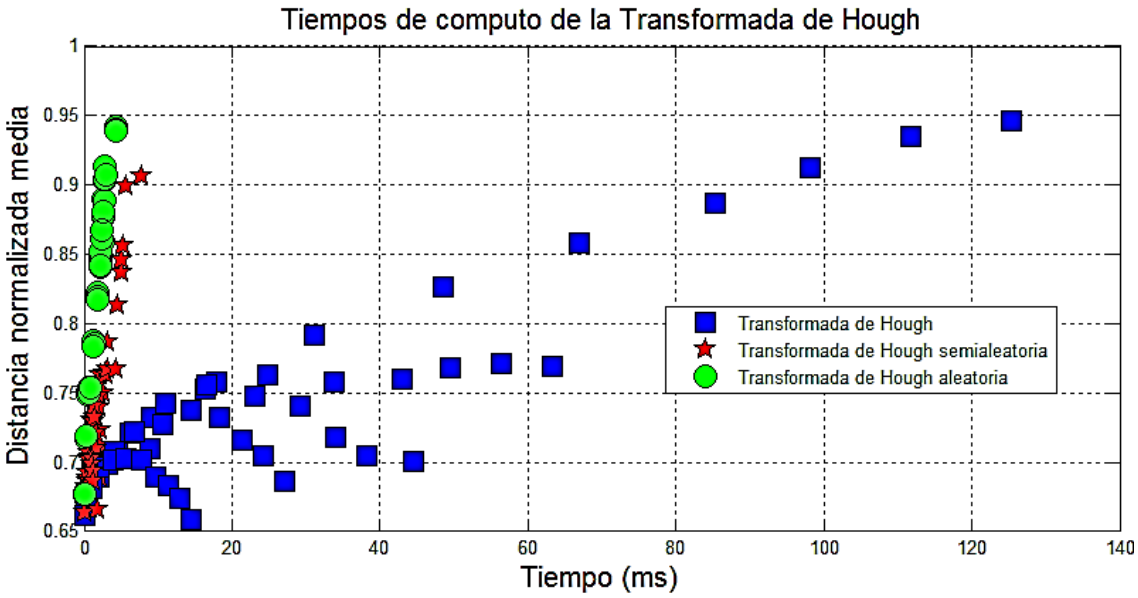
Gráfica 4.13. Distancia normalizada media para Δθ = 5°.



Gráfica 4.14. Distancia normalizada media para Δθ = 10°.

A modo de comentario general la distancia normalizada media disminuye conforme reducimos la resolución de la imagen y aumentamos el ángulo theta. Podemos considerar que cuando la distancia media no supera valores de 0.75 las rectas encontradas no tienen semejanza con las líneas de la imagen. También podemos ver que la variación en la resolución de θ no afecta al algoritmo de la transformada de Hough aleatoria (versión 3).

En la **Gráfica 4.15** podemos ver la distancia media que se ha representado anteriormente en función del tiempo de cómputo en milisegundos. Los puntos corresponden con cada experimento mostrado en las gráficas anteriores.



Gráfica 4.15. Distancias medias en función de sus tiempos de cómputo.

Como era de esperar, cuando la distancia normalizada media disminuye (al reducir las resoluciones) también lo hace el tiempo de cómputo, y por ello podemos observar una mayor concentración de puntos cerca del origen de coordenadas.

4.3 EXTRACCIÓN DE LÍNEAS

Hasta ahora se ha hecho una representación de las líneas obtenidas a partir de los algoritmos implementados, pero el fin de este trabajo es la detección de marcas por lo tanto debemos de ser capaces de extraer las líneas únicas que componen la imagen, es decir, discriminar las líneas que hemos obtenido y son muy parecidas entre sí.

Para ello se han implementado 3 algoritmos, el primero se basa en las ecuaciones obtenidas de las rectas, los dos siguientes se centran en extraer información directamente de la matriz de Hough.

4.3.1 SEMEJANZA ENTRE RECTAS

Con este primer algoritmo la idea fundamental es eliminar las rectas que tengan un parecido muy grande entre sí. A partir del cálculo de distancias entre rectas representado en el apartado anterior se calculan las distancias entre sí de un número alto de rectas obtenidas a partir de la transformada de Hough en

cualquiera de sus versiones, se dividen las rectas en grupos a partir de la distancia entre ellas, extrayendo una recta única de cada grupo, la recta media de todas. A continuación podemos ver el pseudocódigo:

1: obtenemos las ecuaciones de un número alto de líneas (>20) con cualquier algoritmo de la transformada de Hough
2: calculamos las distancias entre ellas
3: discriminamos las distancias menores que cierto umbral
4: agrupamos las rectas que se parecen entre sí
5: se obtiene la recta media de cada grupo
6: representamos las rectas únicas obtenidas

En el **Figura 4.18** tenemos las rectas obtenidas a partir de las ecuaciones de 30 líneas con cada uno de los algoritmos.

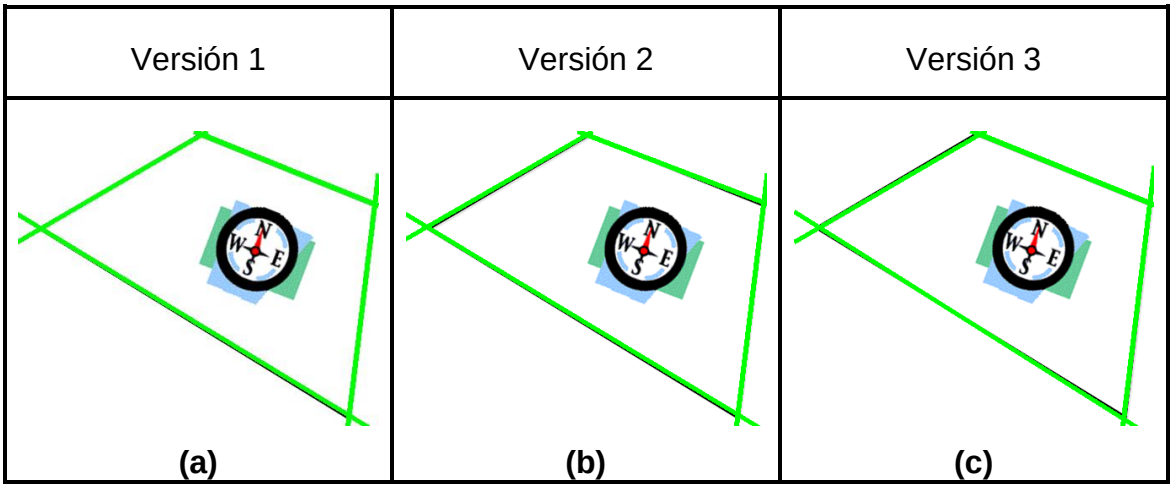


Figura 4.18. Ejemplo de semejanza entre rectas.

Como se puede comprobar las rectas obtenidas son las que corresponden con las reales de la imagen, por lo que con las tres versiones del algoritmo de Hough funciona correctamente. Los tiempos de ejecución obtenidos van en proporción con los representados en los apartados anteriores, por lo que no es necesario representarlos aquí.

4.3.2 RECTAS A PARTIR DE LA MATRIZ DE HOUGH

Tal y como se puede ver en la **Gráfica 4.1** en la que se representa la matriz de Hough de una imagen, los máximos de esta se concentran en un número de puntos que es igual al número de líneas que componen la imagen en cuestión.

Si conocemos a priori el número de rectas que tiene la imagen que queremos analizar nos podemos centrar en extraer la línea correspondiente con cada uno de los máximos de la matriz de Hough. A continuación se muestra el pseudocódigo de este método:

```
1: obtenemos la matriz de Hough con cualquier algoritmo de la transformada de Hough
2:    para el número de rectas que queremos obtener
3:        extraemos el máximo
4:        eliminamos los puntos colindantes a esta recta
5:    representamos las rectas únicas obtenidas
```

Al eliminar los puntos colindantes a la recta encontrada nos aseguramos que en la siguiente búsqueda de un máximo no encontremos este en una recta muy similar a la obtenida.

En la **Figura 4.19** tenemos los resultados obtenidos con este algoritmo, previamente indicamos el número de líneas que queremos obtener y con cada uno de los algoritmos de Hough.

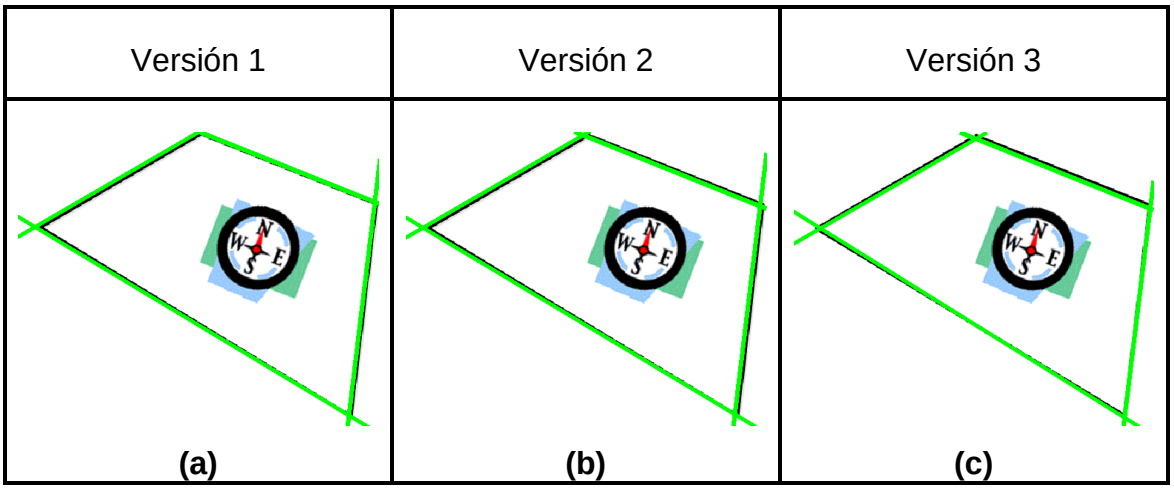


Figura 4.19. Ejemplo de extracción de rectas a partir de la matriz de Hough.

Los resultados son correctos, y muy similares con el algoritmo de semejanza entre líneas.

4.3.3 DETECCIÓN AUTOMÁTICA A PARTIR DE LA MATRIZ DE HOUGH

La idea en este algoritmo es la misma que en el algoritmo anterior, pero detectando el número de líneas de una forma automática.

Para ello extraeremos los máximos cuyo valor supere cierto umbral. El resto del algoritmo es igual que el que se ha explicado anteriormente:

1: obtenemos la matriz de Hough con cualquier algoritmo de la transformada de Hough
2: mientras existan máximos por encima del umbral
3: extraemos el máximo
4: eliminamos los puntos colindantes a esta recta
5: representamos las rectas únicas obtenidas

En **Figura 4.20** y **Figura 4.21** podemos ver las respuestas con este algoritmo, tanto para la imagen que utilizamos habitualmente como para otra imagen con más líneas. Los resultados son satisfactorios, obteniendo todas las rectas que forman las imágenes con cada uno de los algoritmos de la transformada de Hough.

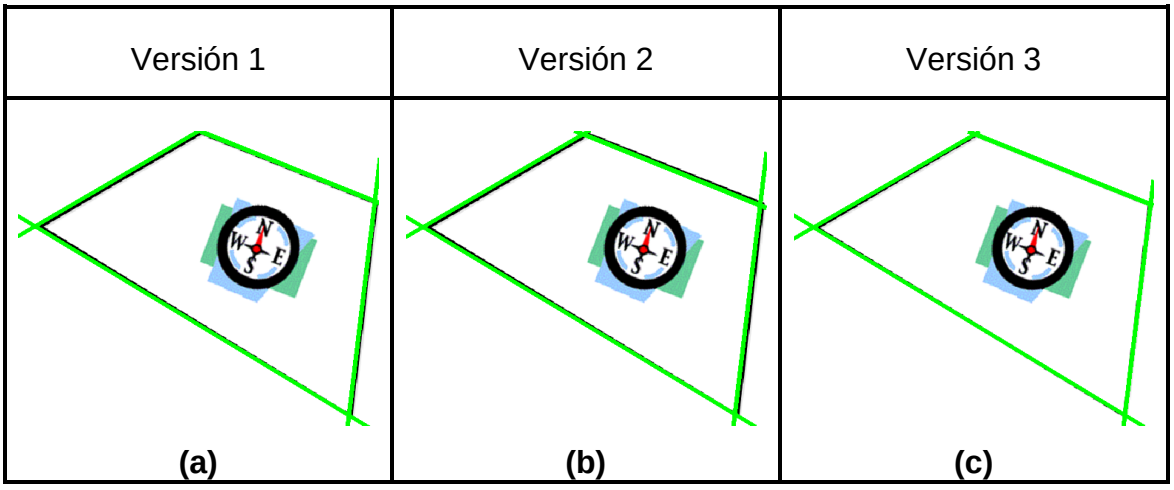


Figura 4.20. Ejemplo de detección automática a partir de la transformada de Hough.

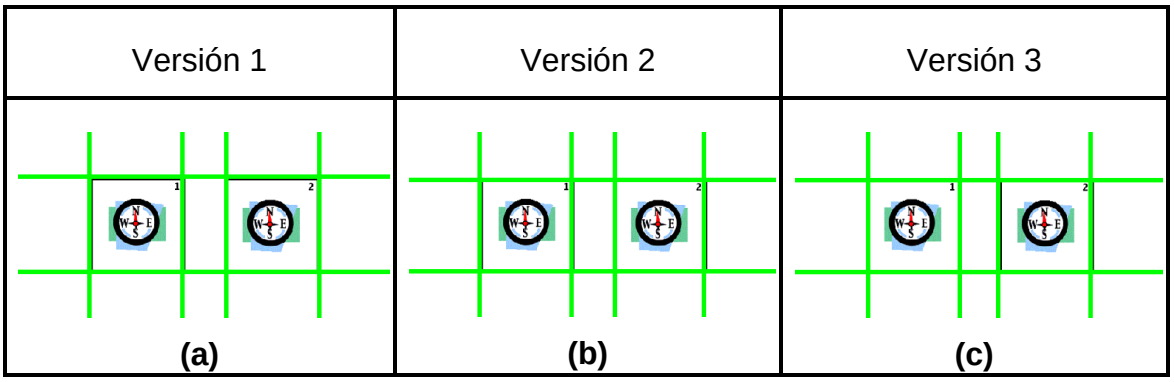


Figura 4.21. Ejemplo de detección automática a partir de la transformada de Hough con varias imágenes.

Para el correcto funcionamiento de este algoritmo es de vital importancia hacer un ajuste de los umbrales en función del tipo de imagen que vamos a captar y la versión de la transformada de Hough que vamos a utilizar. Teniendo en cuenta estos aspectos previos se pueden obtener resultados muy satisfactorios.

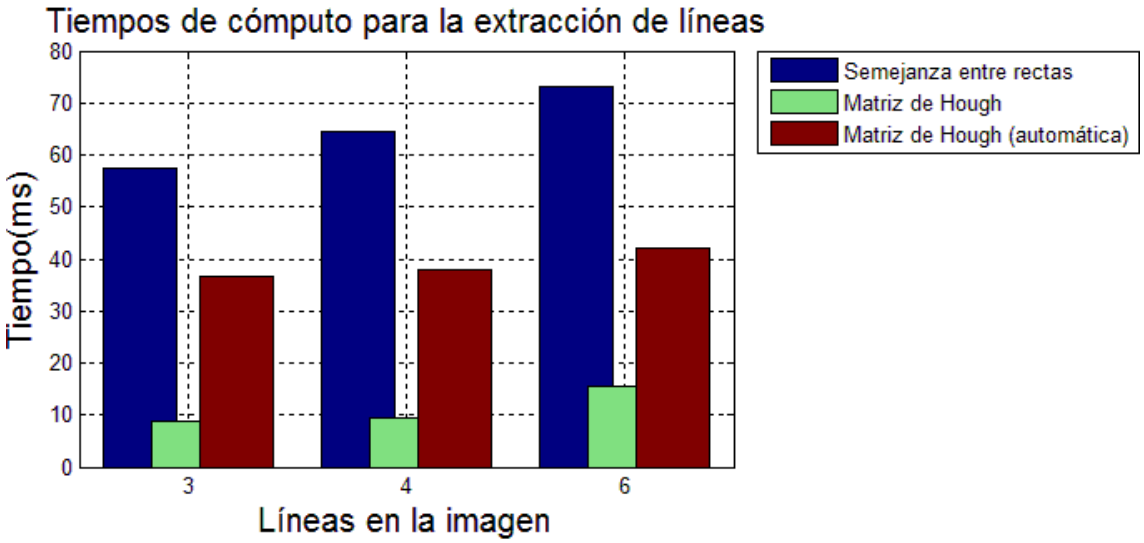
4.3.3.1 COMPARACIÓN DE ALGORITMOS

Una forma de comparar la eficiencia de los 3 algoritmos anteriores es computando los tiempos medios de procesado con varias imágenes. En concreto se obtiene el tiempo para 3 imágenes con 3, 4 y 6 líneas respectivamente después de 20 repeticiones en cada experimento. Estos datos se representan en la **Tabla 4.4**.

		Método de extracción de líneas (milisegundos)		
		Semejanza entre rectas	**Matriz de Hough**	**Matriz de Hough (automática)**
Líneas en la imagen	3	57,58	8,87	36,72
	4	64,52	9,54	37,95
	6	73,27	15,54	41,97
Tiempo medio		65,12	11,32	38,88

Tabla 4.4. Tiempos de cómputo para cada algoritmo de extracción de líneas.

La representación de la **Tabla 4.4** la tenemos en la siguiente gráfica (**Gráfica 4.16**).



Gráfica 4.16. Tiempos de cómputo para cada algoritmo de extracción de líneas.

Podemos ver como el algoritmo que más tiempo utiliza es el de semejanza entre líneas. Entre los dos métodos utilizando la matriz de Hough el automático es el que peor se comporta, pero hay que recordar que con este no necesitamos saber previamente el número de líneas de la imagen.

4.4 DISEÑO, CODIFICACIÓN Y DECODIFICACIÓN DE CÓDIGOS

4.4.1 DISEÑO Y CODIFICACIÓN

En las secciones anteriores se han establecido las bases para la detección de las líneas que forman parte de una imagen cualquiera. En este apartado hablaremos de las balizas que deberán de ser detectadas por la cámara del robot móvil.

Estas balizas tienen que ser fáciles de detectar y para ello una de las premisas más importante es que estas tengan acentuado el marco, de tal forma que la detección de las líneas que la conforman sea fiable independientemente de la distancia a la que estemos de la marca.

Otro punto importante es la codificación de cada baliza para que sea única. Para ser capaces de diferenciarlas codificaremos en su interior un código binario consistente en unos puntos degradados sobre fondo blanco, de tal forma que la ausencia o no de estos puntos nos determinará el nivel de cada bit (1 o 0).

En la **Figura 4.22** tenemos varios ejemplos de balizas con distintos tamaños, grosor del marco, codificación interior y agrupación de varias marcas.

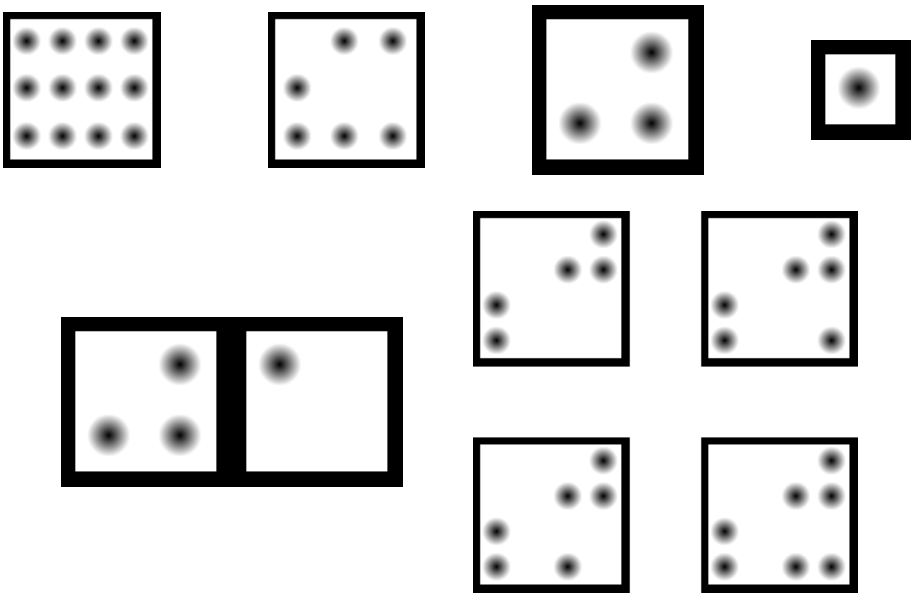


Figura 4.22. Ejemplos de balizas.

El código binario mostrado en las balizas se codifica como se muestra en la **Figura 4.23** en la que podemos ver concretamente la codificación de una baliza de 3x4 [9].

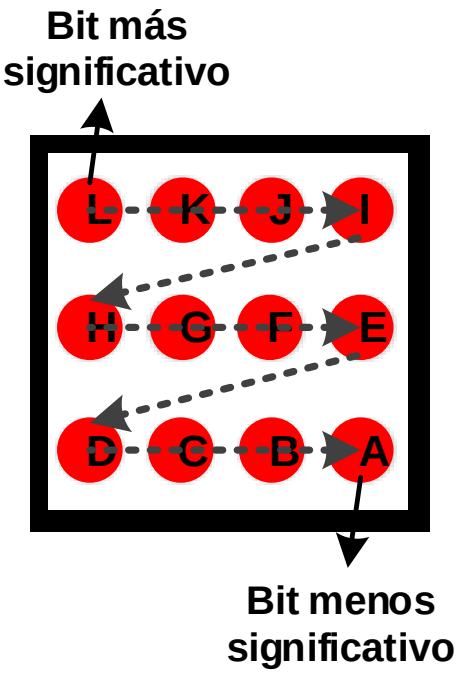


Figura 4.23. Codificación de una baliza 3x4.

Si se quiere obtener cualquier tipo de código NxM el orden será el mismo, el bit menos significativo será el situado en el extremo inferior derecho y el más significativo el del extremo superior izquierdo.

4.4.2 DECODIFICACIÓN

Para detectar cada bit de la baliza el procedimiento que seguimos es el siguiente (**Figura 4.24**):

- Comenzamos la detección en el extremo superior izquierdo.
- Situamos una máscara en el centro de cada zona donde se debe alojar un bit de nuestro código.
- Obtenemos la media de los valores de gris de la zona circular y de la zona representada en negro.
- Si la diferencia de estos valores medios exceden de cierto umbral se asume que se ha representado un bit, luego guardaremos el valor 1. En caso contrario el valor del bit será igual a 0.

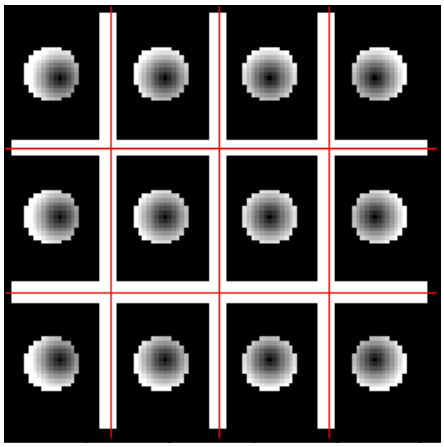


Figura 4.24. Detección de bits.

Para conocer el código detectado podemos obtenerlo en forma binaria siguiendo el orden representado en la **Figura 4.23** o a partir de su codificación decimal a partir de la siguiente ecuación:

$$\text{Código} = \begin{bmatrix} 2^0 & 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 & 2^7 & 2^8 & 2^9 & 2^{10} & 2^{11} \end{bmatrix} \begin{bmatrix} \text{bit A} \\ \text{bit B} \\ \text{bit C} \\ \text{bit D} \\ \text{bit E} \\ \text{bit F} \\ \text{bit G} \\ \text{bit H} \\ \text{bit I} \\ \text{bit J} \\ \text{bit K} \\ \text{bit L} \end{bmatrix} \quad (4.4)$$

4.5 DETECCIÓN DE MARCAS A PARTIR DE IMÁGENES REALES

4.5.1 MARCAS INDIVIDUALES

A continuación se va a mostrar la detección de balizas individuales en imágenes reales tomadas mediante cámara de fotos.

El proceso por el que detectaremos estas balizas se representa en el siguiente pseudocódigo:

1: obtenemos la matriz de Hough con cualquier algoritmo de la transformada de Hough
2: extraemos las líneas más significativas de la imagen
3: calculamos los puntos de corte de las líneas encontradas
4: ordenamos los puntos en sentido horario empezando por el extremo superior izquierdo
5: recortamos y transformamos la imagen
6: leemos el código representado en la imagen

Como ejemplo podemos ver en la **Figura 4.25** una baliza, en la que se marcan los puntos de corte de las rectas encontradas, las enumeramos y se hace una operación geométrica en la imagen, en concreto cambiamos la perspectiva [5] y a continuación efectuamos el recorte de la marca.

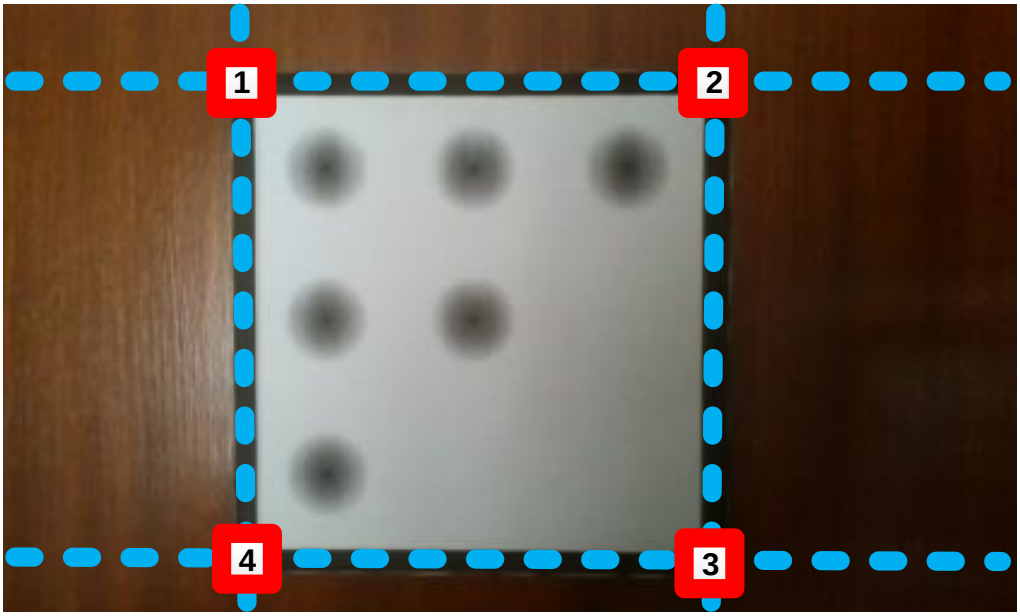


Figura 4.25. Puntos de corte para una baliza.

A la imagen recortada (**Figura 4.26(a)**) le aplicamos la máscara mostrada en la sección anterior (**Figura 4.24**), en esta ocasión de 3x3, obteniendo como resultado la **Figura 4.26(b)** y finalmente el código que representa la baliza.

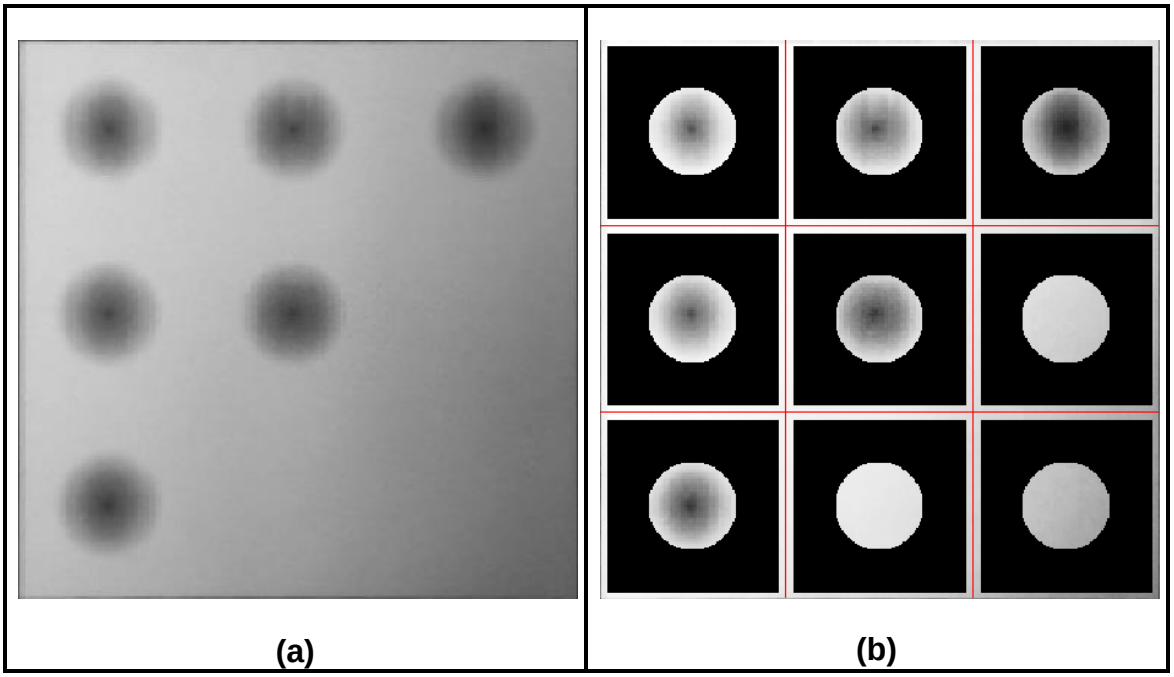
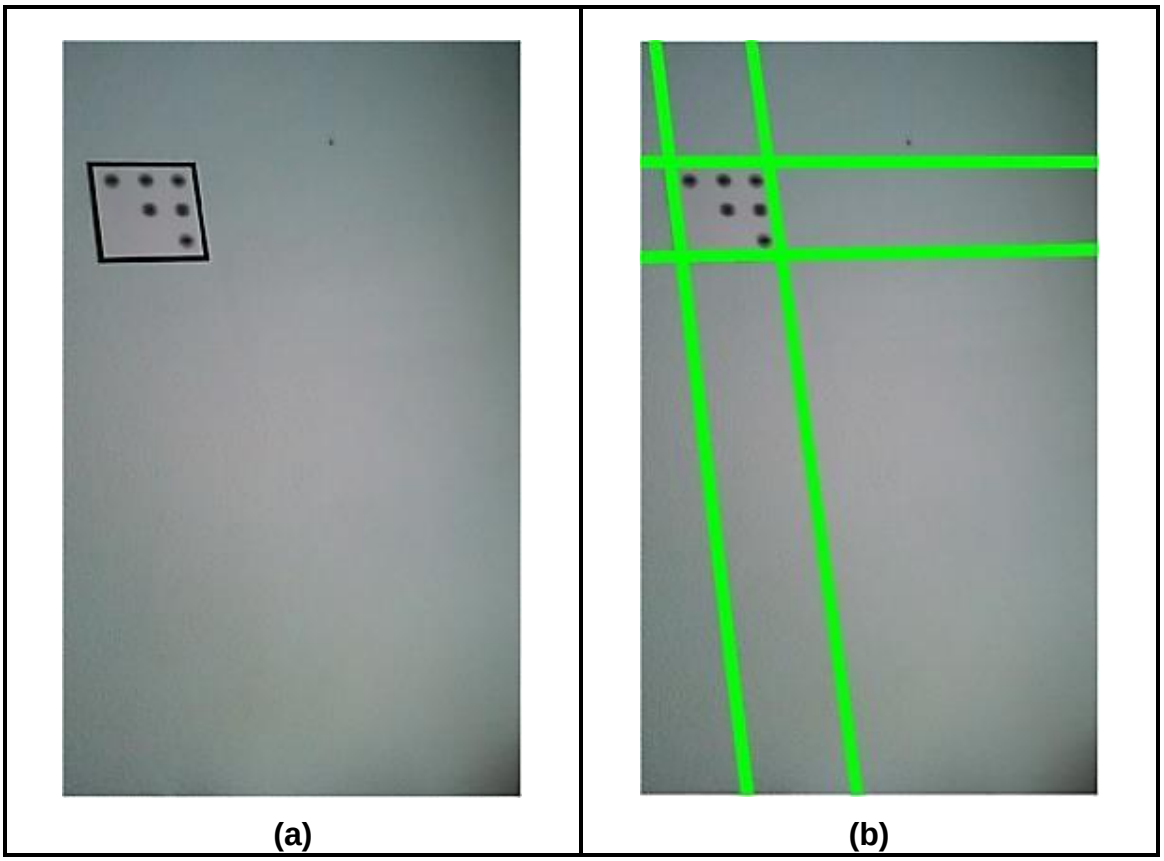


Figura 4.26. Aplicación de máscara para detectar los bits en la imagen.

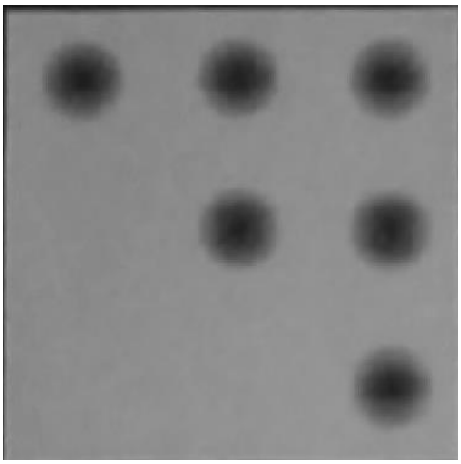
El código obtenido es el 111.110.100 con lo que la detección es correcta.

En los ejemplos que se muestran a continuación la detección del código de la baliza siempre es correcto. Para que esta detección sea correcta es muy importante que las líneas obtenidas sean las 4 que pertenecen a la baliza, en el caso de que las líneas encontradas no sean correctas, superior o inferior a 4 esta detección no se podrá llevar acabo.



(a) **(b)**

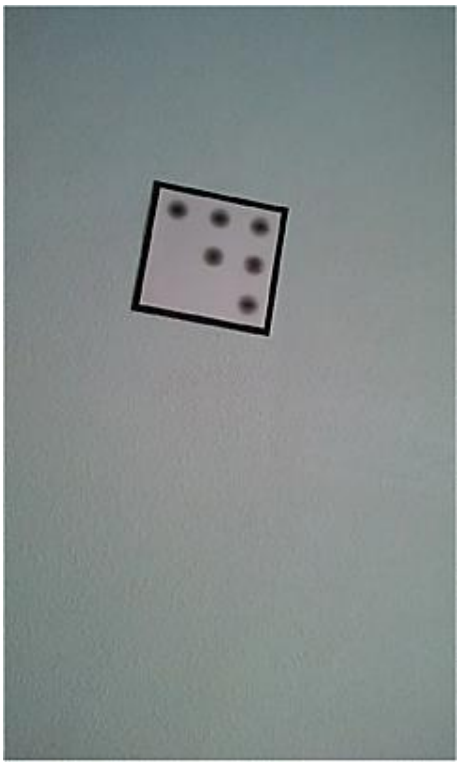
(c)



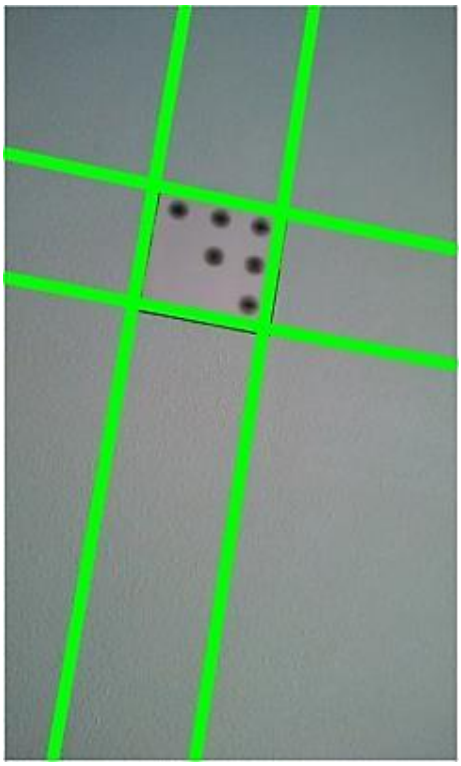
(d)

Código obtenido: 111.011.001

Figura 4.27. Detección de marcas simples. Ejemplo 1.

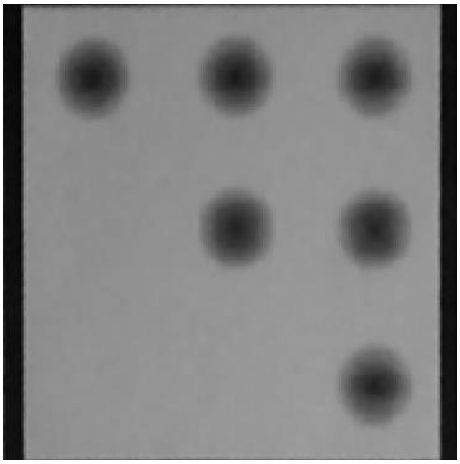


(a)



(b)

(c)



(d)

Código obtenido: 111.011.001

Figura 4.28. Detección de marcas simples. Ejemplo 2.

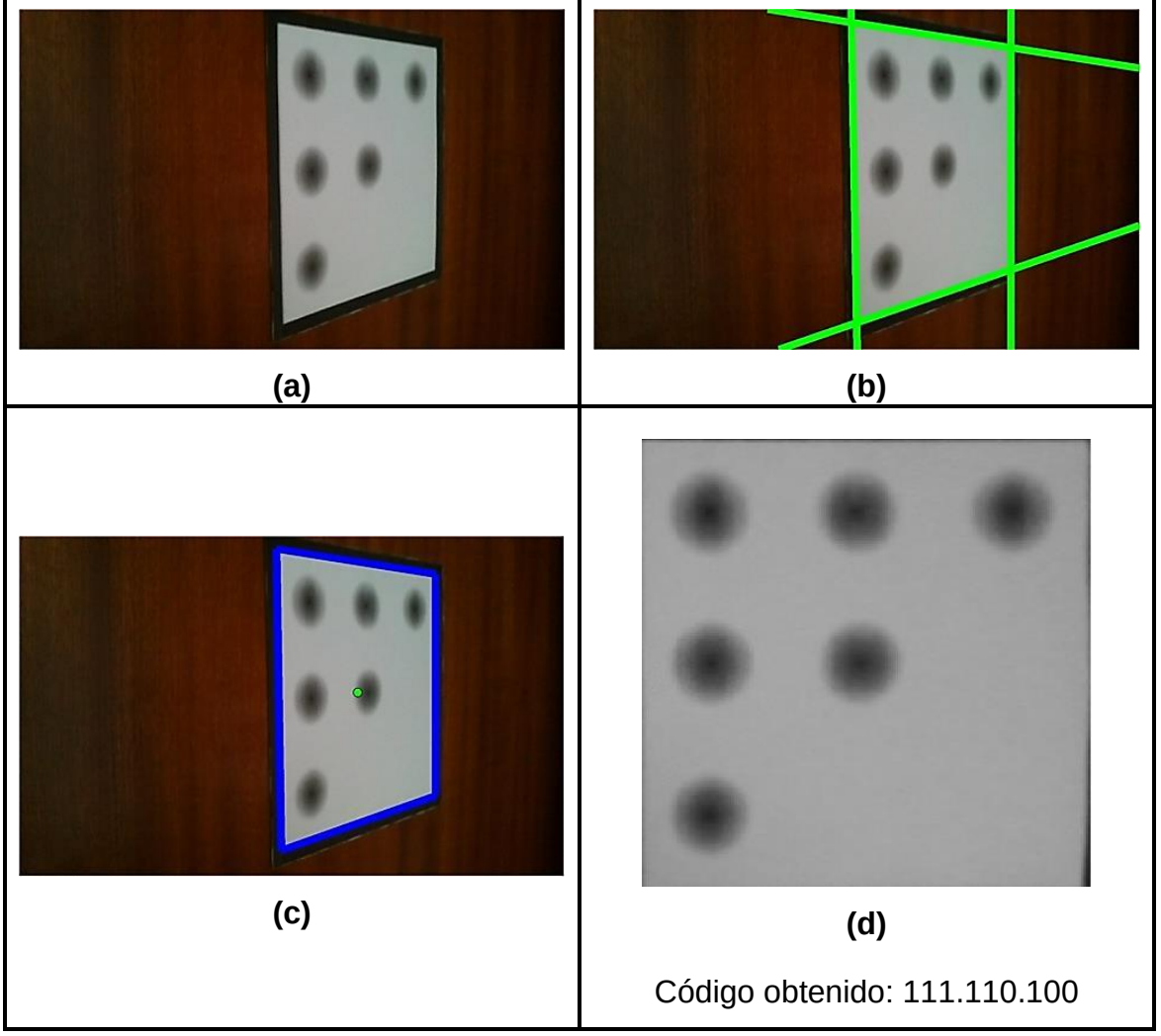


Figura 4.29. Detección de marcas simples. Ejemplo 2.

En el siguiente ejemplo el número de líneas detectadas es superior a las que pertenecen a la baliza, por lo que no es posible la detección de la marca.

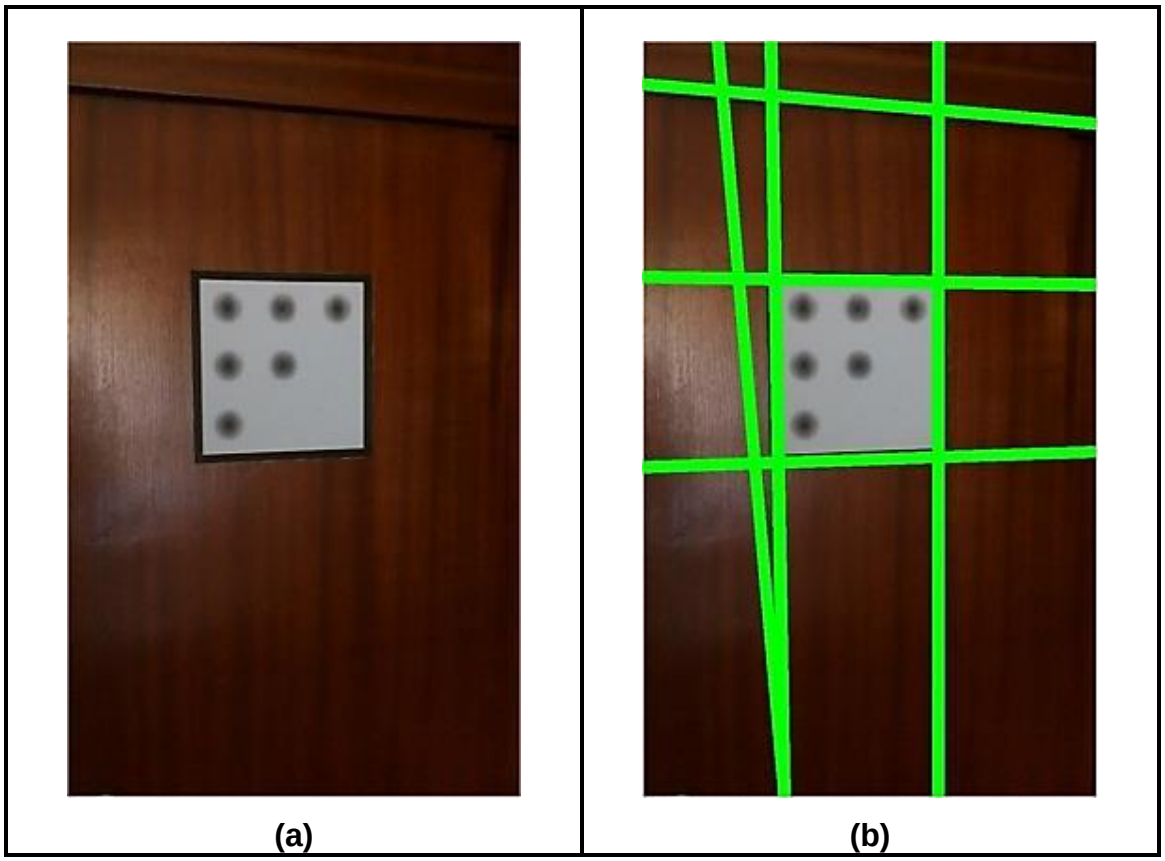


Figura 4.30. Detección de marcas simples. Ejemplo 3.

En capítulos posteriores se hará un estudio estadístico del número de códigos detectados correctamente, pero como se puede ver, siempre y cuando se haga un recorte correcto de la baliza se interpretará correctamente el código.

4.5.2 MARCAS MÚLTIPLES

En el caso de la detección de imágenes con múltiples marcas deberemos obtener un número mayor de líneas, una vez extraídas con alguno de los algoritmos en la sección 0.

Dado que en estos casos tendremos un gran número de cruces entre las líneas encontradas deberemos decidir cuando el recorte realizado se corresponde con una baliza. Esta decisión la tomaremos detectando la primera fila de puntos de la imagen, si en la imagen recortada encontramos algún punto en esta línea asumiremos que se trata de un código correcto. Cuantos más puntos utilicemos

para esta comprobación más seguros estaremos de haber detectado una baliza. El problema que nos podemos encontrar es la disminución del número de bits de nuestros códigos, pero ampliando el número de filas, columnas o ambas de bits se soluciona este problema.

A continuación mostramos el pseudocódigo para la detección de balizas múltiples:

1: obtenemos la matriz de Hough con cualquier algoritmo de la transformada de Hough
2: extraemos las líneas más significativas de la imagen
3: separamos las líneas en horizontales y verticales y se ordenan de mayor a menor valor en función de la componente ρ
4: para cada par de líneas verticales comprobamos los puntos de corte con cada par de líneas horizontales
5: recortamos, transformamos la imagen y comprobamos si es un código válido
6: leemos el código representado en la imagen

En las siguientes imágenes podemos comprobar la detección de balizas, en esta ocasión para discernir si es una marca correcta detectamos si existe bit en la fila 1 columna 2. En todos los casos la interpretación del código fue correcta.

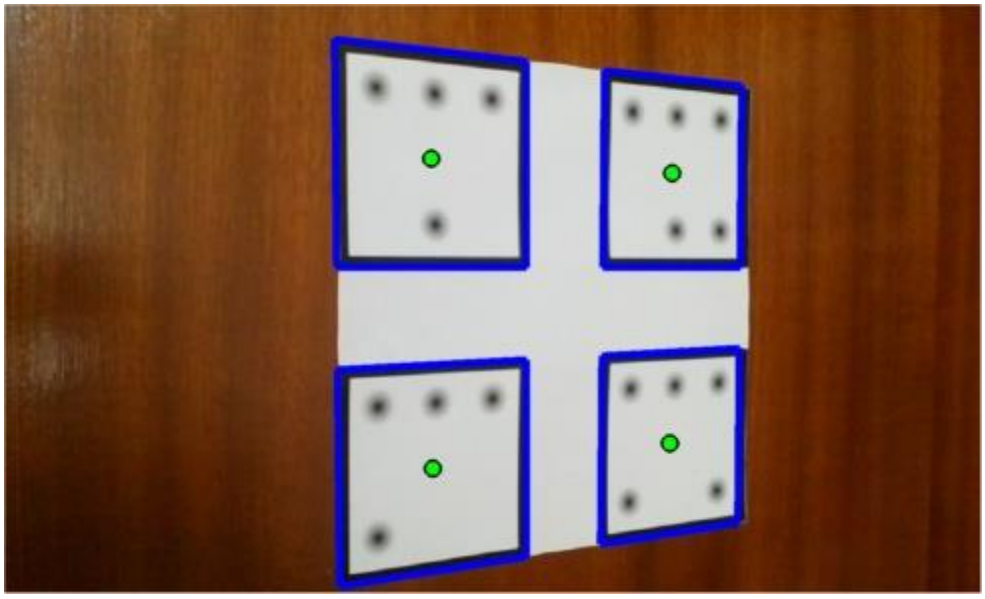


Figura 4.31. Detección de marcas múltiples. Ejemplo 1.

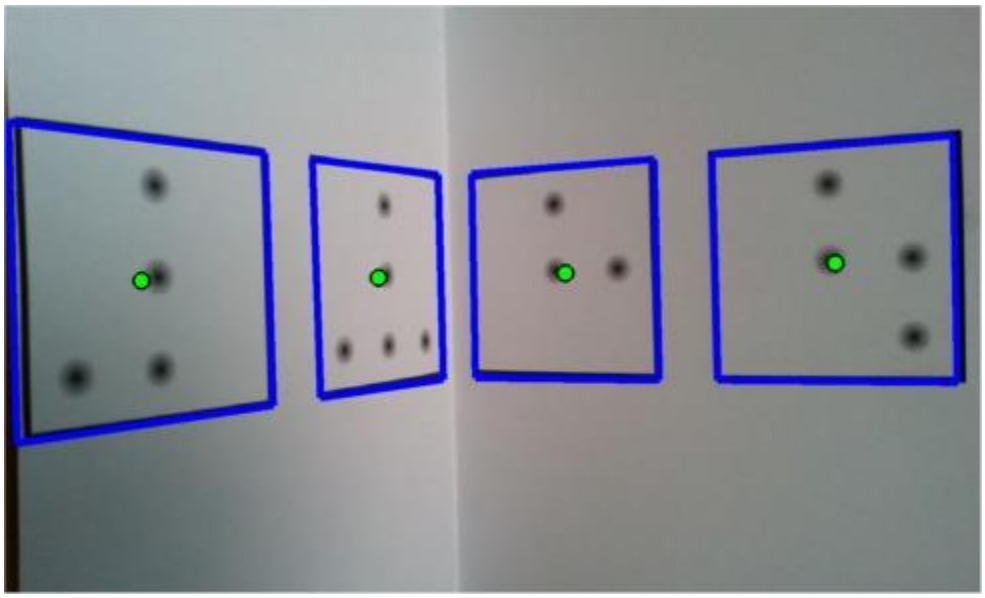


Figura 4.32. Detección de marcas múltiples. Ejemplo 2.

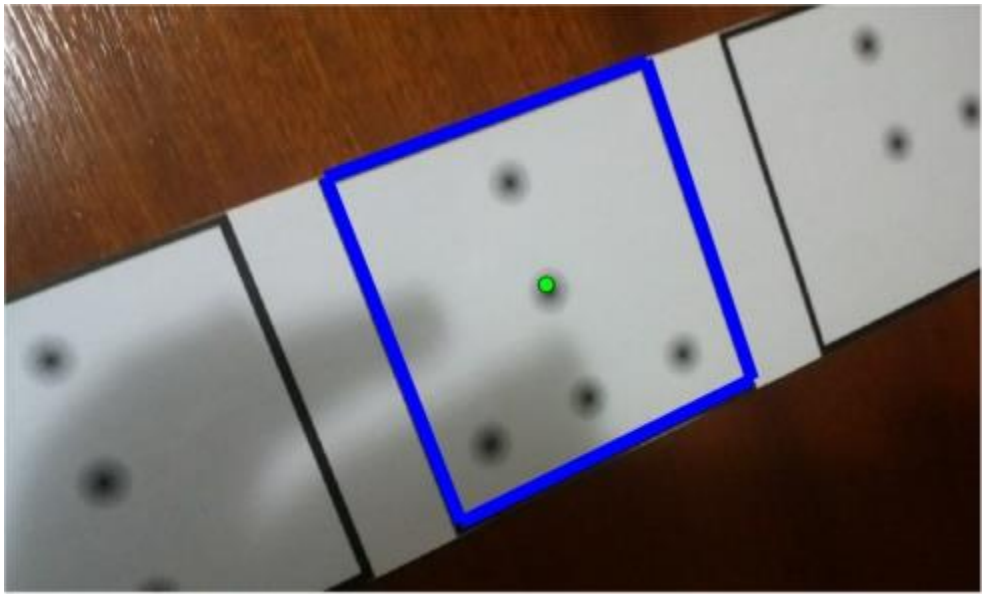


Figura 4.33. Detección de marcas múltiples. Ejemplo 3.

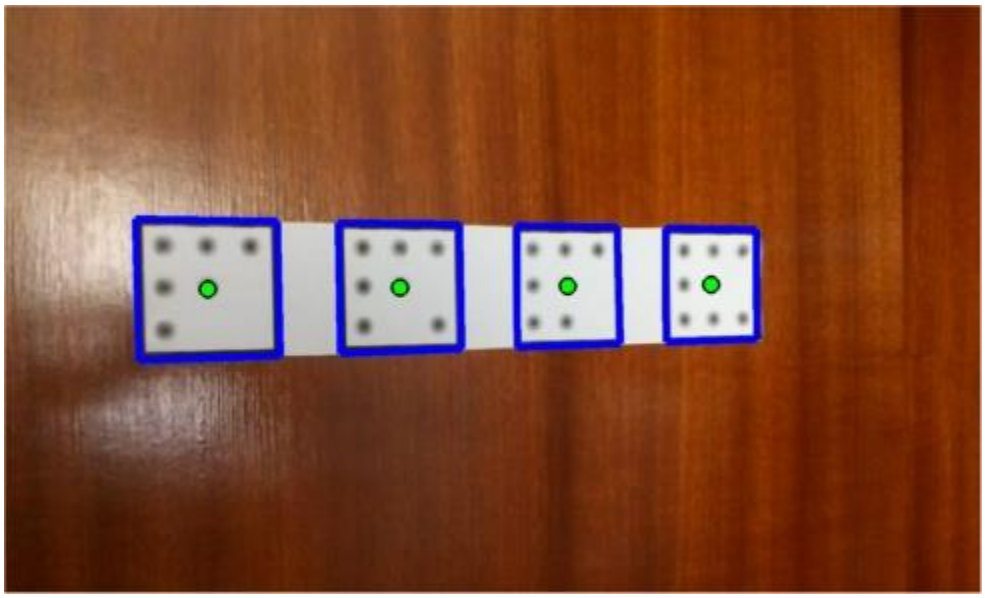


Figura 4.34. Detección de marcas múltiples. Ejemplo 4.

En la **Figura 4.35** tenemos una mala detección de las balizas, en concreto no se han podido recortar las dos últimas, esto se debe a que las líneas que pertenecen a estas balizas no han sido detectadas.

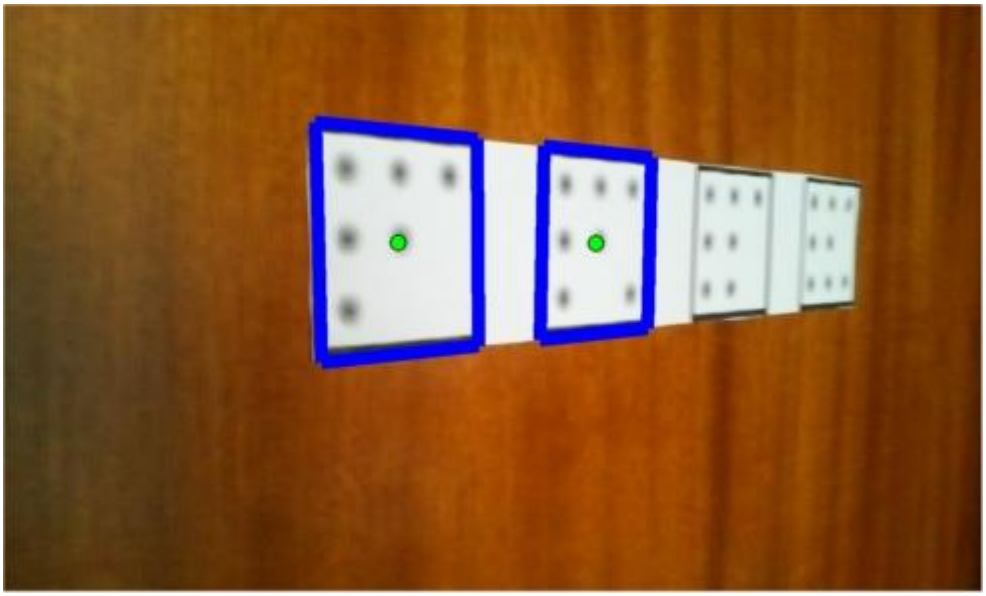


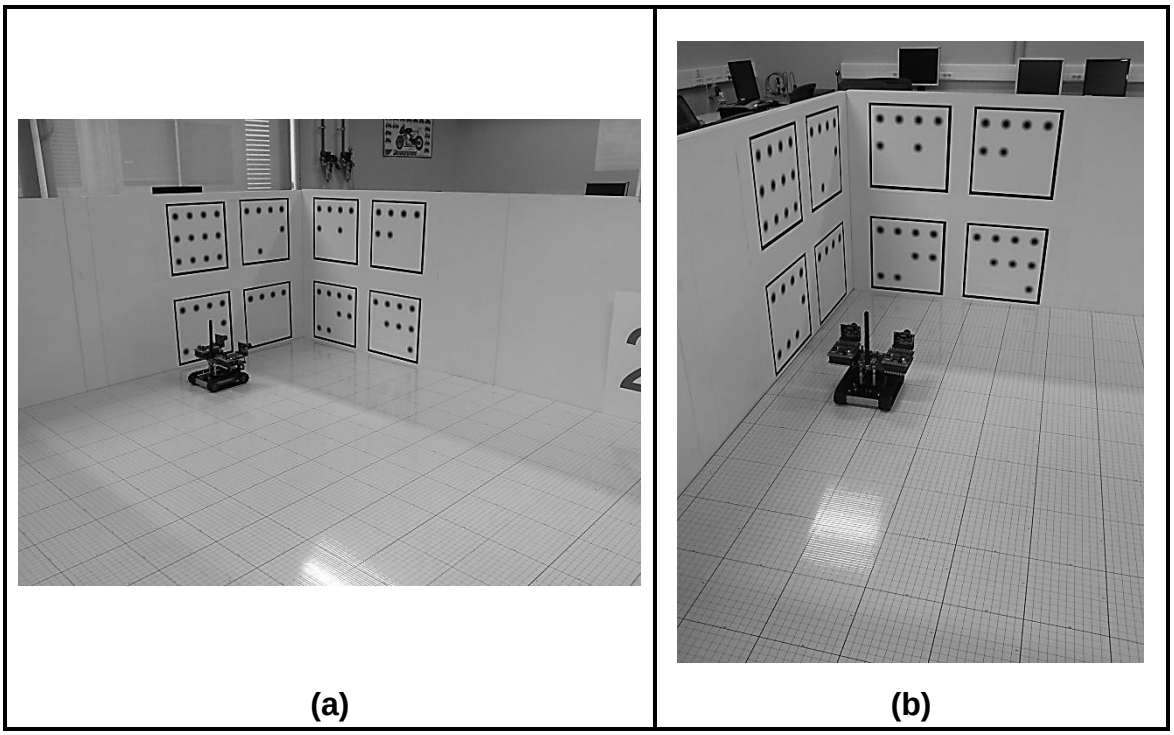
Figura 4.35. Detección de marcas múltiples. Ejemplo 5.

5. APLICACIÓN REAL

En esta sección se muestran los resultados y estadísticas de la captación y reconocimiento de balizas con la cámara del robot SURVEYOR SRV1B. En primer lugar se presentan las imágenes de prueba captadas con el robot y se obtienen las marcas detectadas con los algoritmos de la transformada de Hough y se extraen unas conclusiones a la vista de los resultados. En el último apartado podremos ver los códigos que somos capaces de detectar en el mejor de los casos.

5.1 IMÁGENES DE PRUEBA

El robot deberá de ser capaz de interpretar las marcas existentes en las paredes de un recinto rectangular (**Figura 5.1**) en el Laboratorio de Control de Sistemas Inteligentes, ubicado en el edificio QUORUM 5, Universidad Miguel Hernández de Elche, campus de Elche.

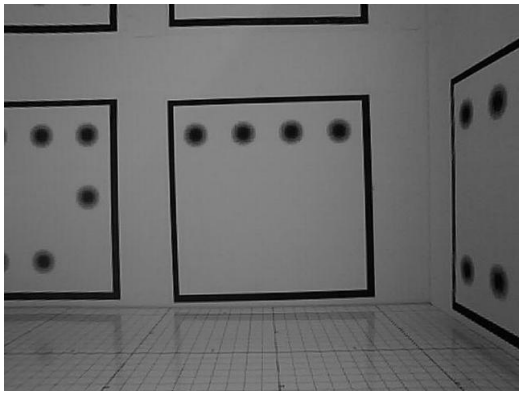
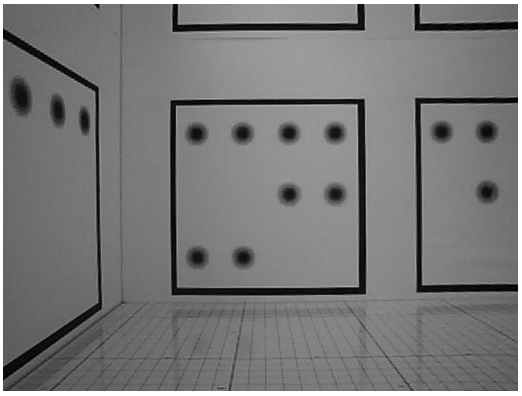
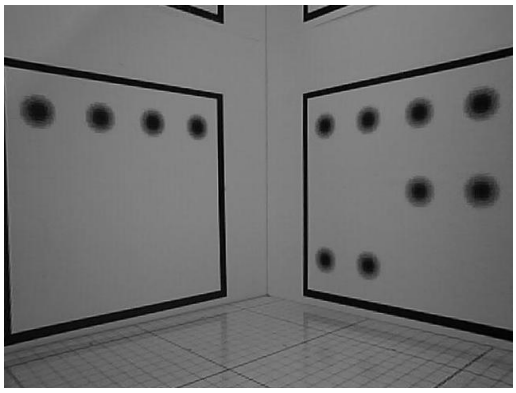
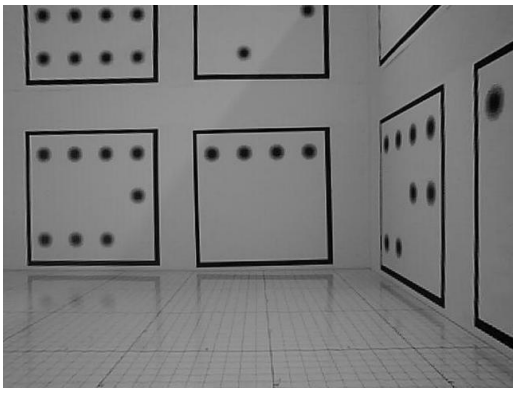
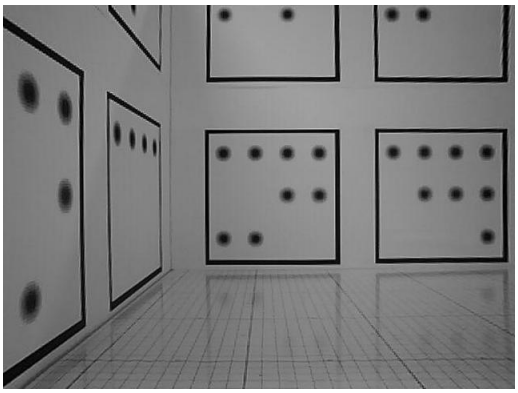
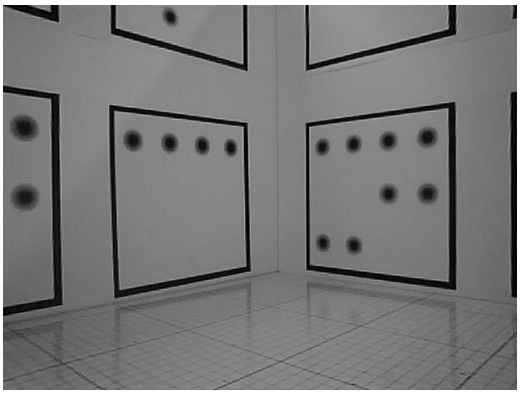


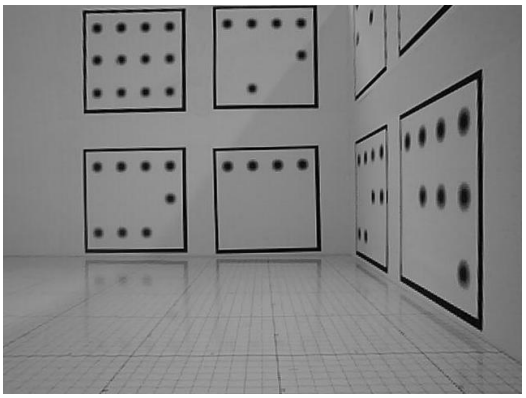
(a) **(b)**

Figura 5.1. Recinto en el que se mueve el SURVEYOR SRV1B.

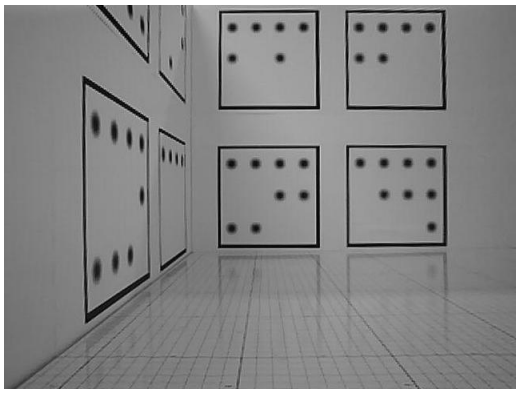
Captaremos un total de 11 imágenes (**Figura 5.2**) que utilizaremos como pruebas para la extracción e interpretación de nuestras balizas. Obtendremos

cada una para todas las combinaciones de resoluciones y calidad del formato JPEG que nos permite la cámara del robot, lo que hacen un total de 352 imágenes.

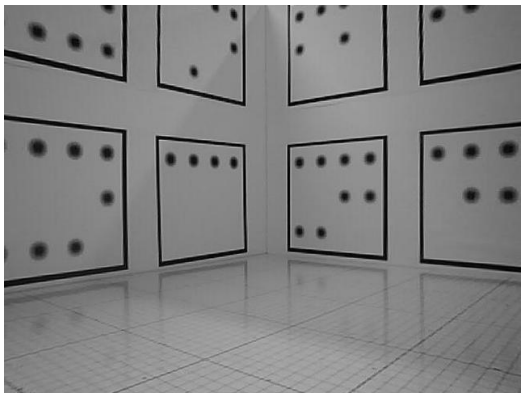
	
Posición 1	Posición 2
	
Posición 3	Posición 4
	
Posición 5	Posición 6



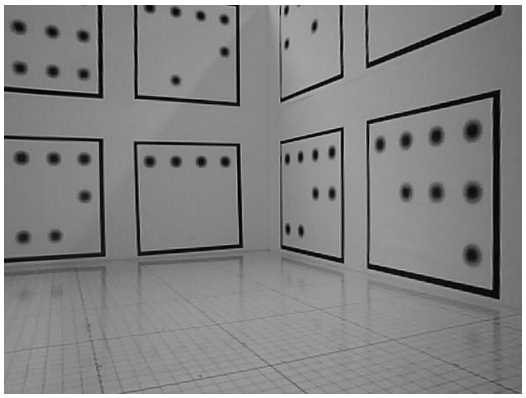
Posición 7



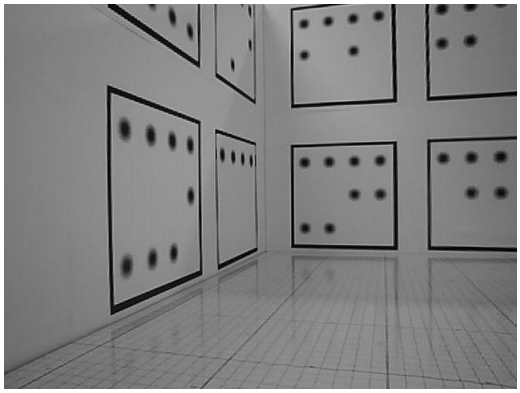
Posición 8



Posición 9



Posición 10



Posición 11

Figura 5.2. Muestra de cada una de las posiciones del robot.

A cada una de las imágenes se les ha aplicado el filtro corrector del efecto ojo de pez descrito en la sección 3.3.2.

5.2 DETECCIÓN DE BALIZAS

Para la detección de las marcas en las 11 imágenes tendremos en cuenta los códigos detectables, es decir, aquellos que se muestran completamente. Por ejemplo en la posición 3 no tenemos ningún código detectable ya que las dos balizas que aparecen no están completas, por lo que no se debería detectar ninguna marca en esta posición.

En la **Tabla 5.1** se muestra el número de códigos detectables por cada posición.

Imágenes	Códigos
Posición 1	1
Posición 2	1
Posición 3	0
Posición 4	3
Posición 5	3
Posición 6	2
Posición 7	6
Posición 8	6
Posición 9	2
Posición 10	3
Posición 11	3
CÓDIGOS TOTALES	**30**

Tabla 5.1. Número de códigos detectables por posición.

Para tener una medida de la calidad en la detección de nuestras balizas mostraremos los resultados para las 30 marcas existentes en las 11 posiciones, pertenecientes a:

- Porcentaje de aciertos por resolución y calidad

$$\text{Aciertos} = \frac{\text{códigos detectados correctamente}}{30} \cdot 100\% \qquad (5.1)$$

- Porcentaje de fallos por resolución y calidad.

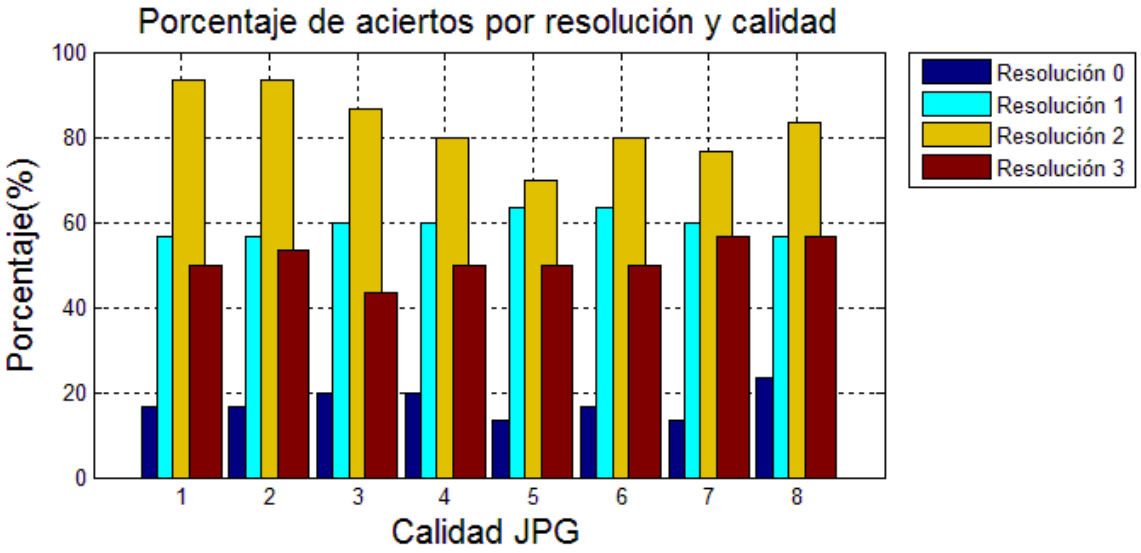
$$\text{Fallos} = \frac{\text{códigos detectados erróneos}}{\text{total códigos detectados}} \cdot 100\% \qquad (5.2)$$

- Porcentaje medio de aciertos, códigos no detectados y fallos por resolución.
- Porcentaje medio de aciertos, códigos no detectados y fallos por calidad.
- Tiempos de procesado por resolución.
- Tiempos de procesado por calidad.
- Por último se representan los resultados para la resolución y calidad más efectivas.

En primer lugar veremos los resultados de la detección con la transformada de Hough, a continuación se expondrán los resultados con la transformada semialeatoria. Finalizaremos mostrando la detección sobre las imágenes de prueba para el mejor algoritmo, mejor resolución y mejor calidad.

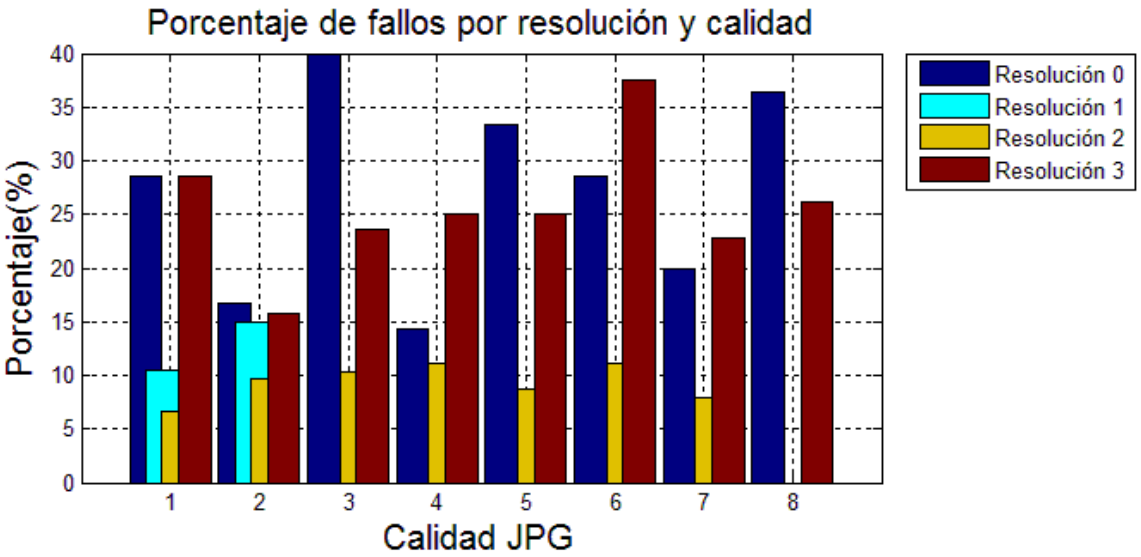
5.2.1 DETECCIÓN CON EL ALGORITMO DE LA TRANSFORMADA DE HOUGH

En primer lugar podemos ver en la **Gráfica 5.1** el porcentaje de aciertos por resolución y calidad, en ella observamos que los mayores niveles de aciertos en la detección de las balizas nos los da la resolución 2, llegando a niveles superiores al 90%. Las calidades 1 y 2 son las que mejor se comportan.



Gráfica 5.1. Porcentaje de aciertos por resolución y calidad para la transformada de Hough.

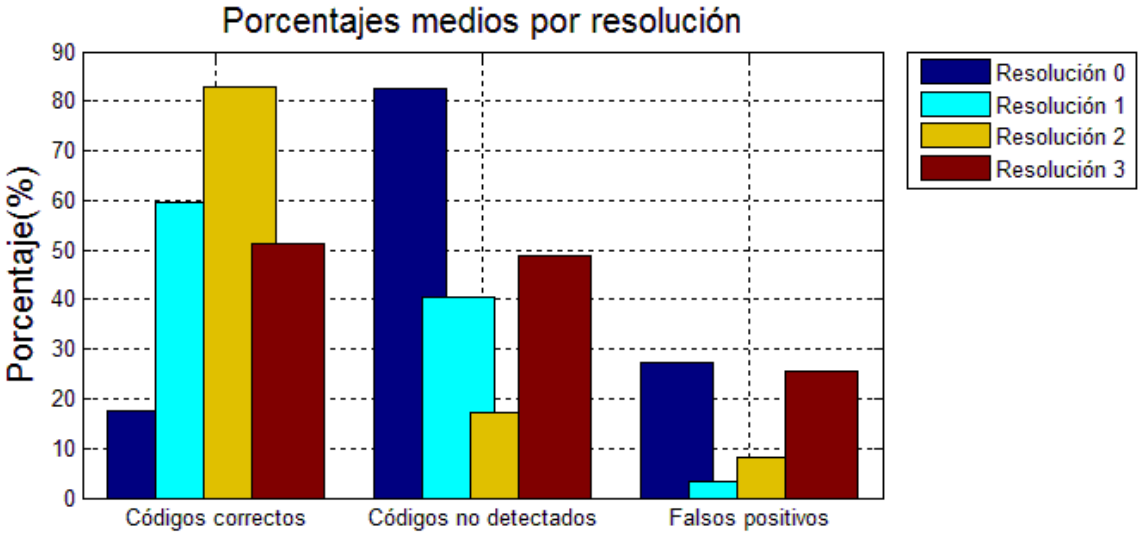
En la **Gráfica 5.2** tenemos el porcentaje de fallos por resolución y calidad. Los peores resultados los obtenemos para la resolución 0 y 3. En la mayoría de calidades de imagen jpeg no se obtienen fallos con la resolución 1.



Gráfica 5.2. Porcentaje de fallos por resolución y calidad para la transformada de Hough.

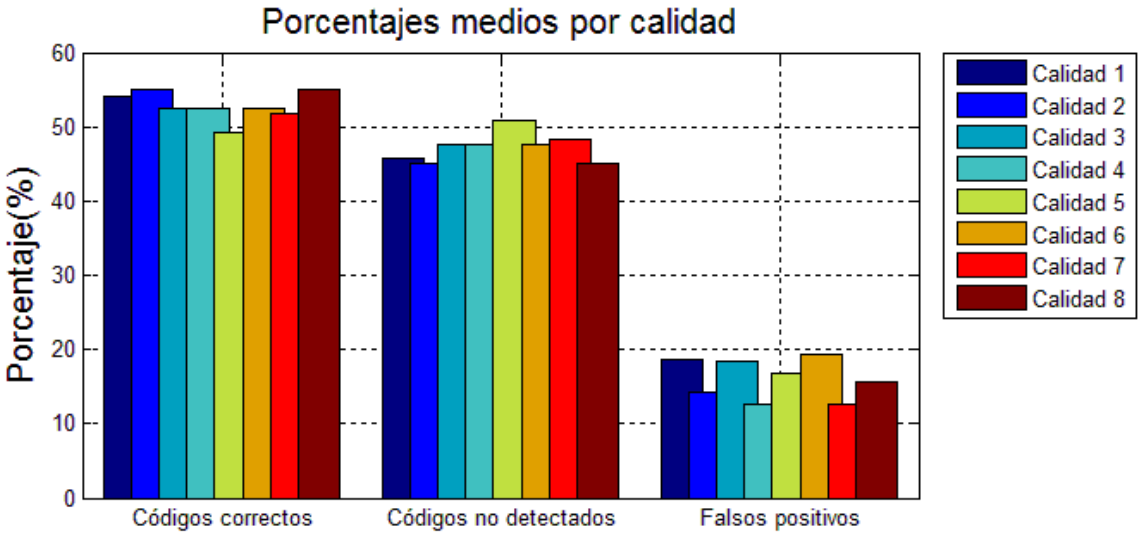
En la **Gráfica 5.3** se representan los porcentajes medios por resolución para los códigos detectados correctamente (aciertos), códigos existentes en la imagen no detectados y falsos positivos, es decir, códigos obtenidos que no se encuentran en la imagen. Como era de esperar conforme a lo observado en la **Gráfica 5.1** la resolución 2 nos proporciona los mayores porcentajes de

aciertos, y los menores niveles de códigos no detectados. En cuanto a los fallos o falsos positivos tanto la resolución 0 como la 3 son las que nos dan los mayores niveles.



Gráfica 5.3. Porcentajes medios por resolución para la transformada de Hough.

Para la **Gráfica 5.4** obtenemos los porcentajes medios de los 3 ítems descritos en el párrafo anterior en función de la calidad. En este caso no se puede determinar que la calidad de la codificación jpeg afecte en nuestra detección ya que ninguna de estas tiene niveles destacables comparados con las demás.



Gráfica 5.4. Porcentajes medios por calidad para la transformada de Hough.

5.2.1.1 CONCLUSIONES

A partir de las gráficas mostradas anteriormente podemos extraer las siguientes conclusiones:

- La resolución 2 es con la que se obtienen mejores resultados, ésta es la que mejor se ajusta a nuestros algoritmos consiguiendo los mayores porcentajes de aciertos y niveles muy reducidos de falsos positivos. Esto se debe a que a resoluciones bajas no somos capaces de detectar todas las líneas de las marcas reduciendo los niveles de aciertos y aumentando los falsos positivos. En cuanto a la baja efectividad de la resolución más alta se debe a la aparición de líneas extra, sobre todo las que forman parte del suelo del recinto, que dificulta la correcta obtención de marcas correctamente y aumenta los falsos positivos.
- La detección de falsos positivos es debido a la manera en la que decidimos si existe o no baliza. En esta ocasión la decisión se basa en la existencia de la primera fila de puntos, por lo que pueden existir recortes que nos proporcionen niveles de negro sobre estos puntos, se comprueba si cumplen con el umbral impuesto entre el centro y las zonas adyacentes y se toma como marca correcta. En la **Figura 5.3** podemos ver una detección incorrecta, en la cual el código leído no era el correcto, y un falso positivo, en ambos casos al existir la primera fila de puntos se cumplen los umbrales y se toman como baliza correcta.

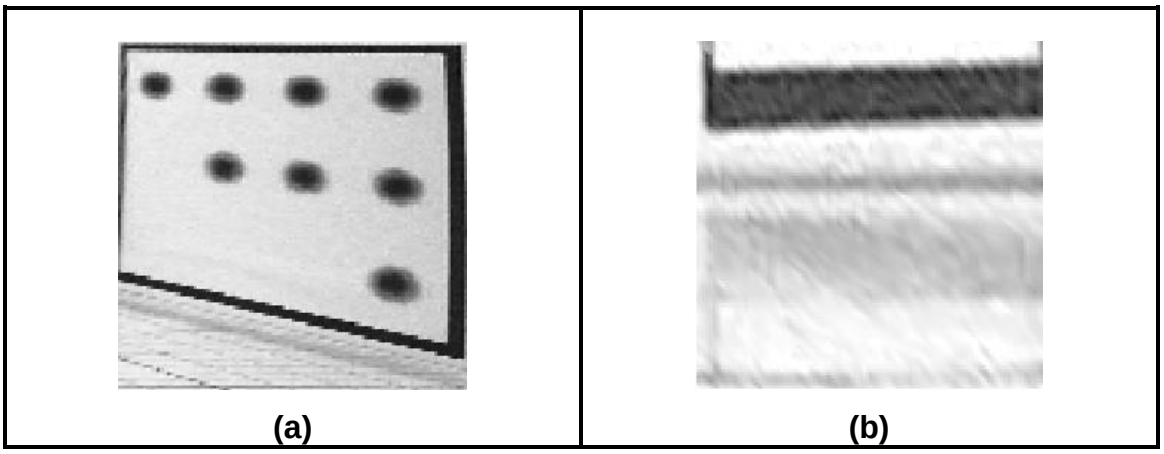
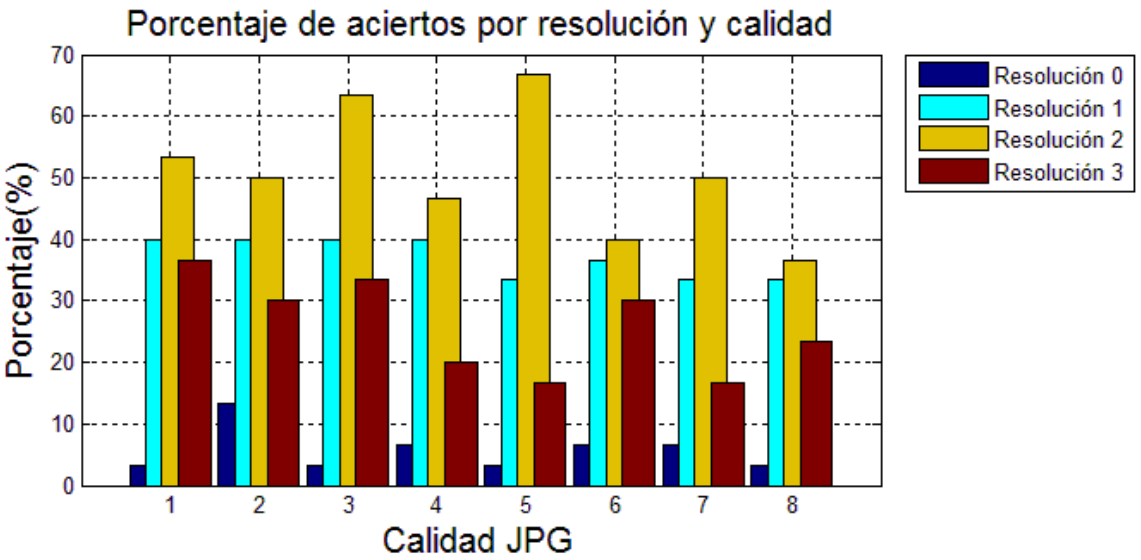


Figura 5.3. Ejemplo de detección incorrecta **(a)** y falso positivo **(b)**.

- En cuanto a la calidad de la codificación JPEG no podemos destacar cuál de ellas funciona de forma más eficiente. Mientras que para la resolución 2 aparentemente funcionan mejor calidades altas (1 y 2) en otras resoluciones no podemos decir lo mismo. En un principio la calidad del JPEG debería afectar a la detección de líneas y al reconocimiento de las balizas pero en este caso las imágenes codificadas son muy parecidas entre sí, los bloques característicos creados por la codificación y el color no afectan a nuestras imágenes (siempre se convierten a escala de grises).

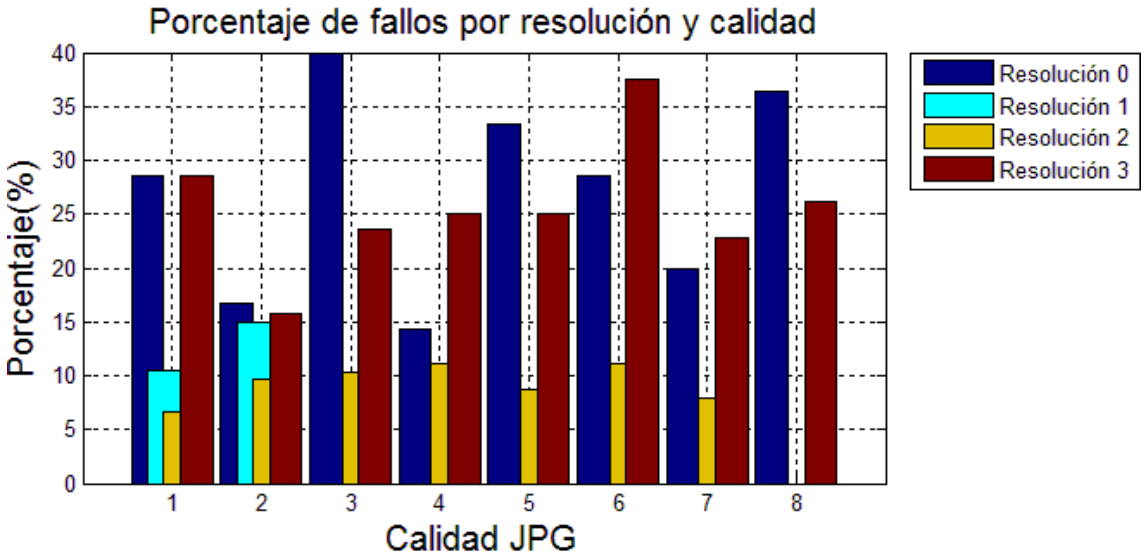
5.2.2 DETECCIÓN CON EL ALGORITMO DE LA TRANSFORMADA DE HOUGH SEMIALEATORIA

Ahora pasaremos a comentar las gráficas obtenidas para la transformada de Hough semialeatoria. El porcentaje de aciertos por resolución y calidad que se representa en la **Gráfica 5.5** nos destaca, como pasara con la transformada de Hough, que la resolución 2 es con la que se obtienen mejores resultados, en concreto con una calidad 5 nos acercamos a un 70% de códigos detectados correctamente.



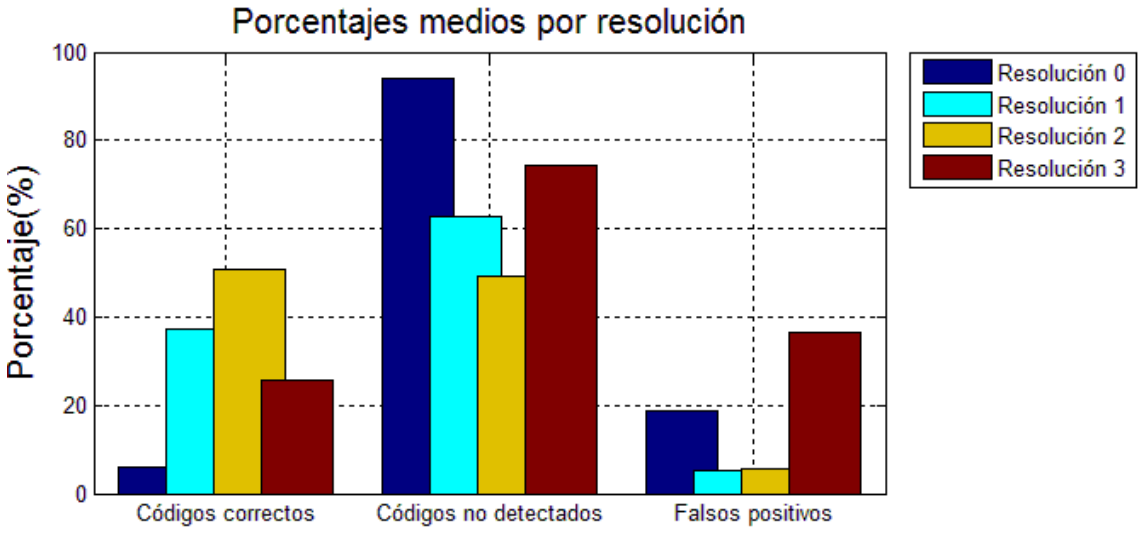
Gráfica 5.5. Porcentaje de aciertos por resolución y calidad para la transformada de Hough semialeatoria.

En cuanto a los fallos, las resoluciones 0 y 3 son las que peor funcionan (**Gráfica 5.6**).



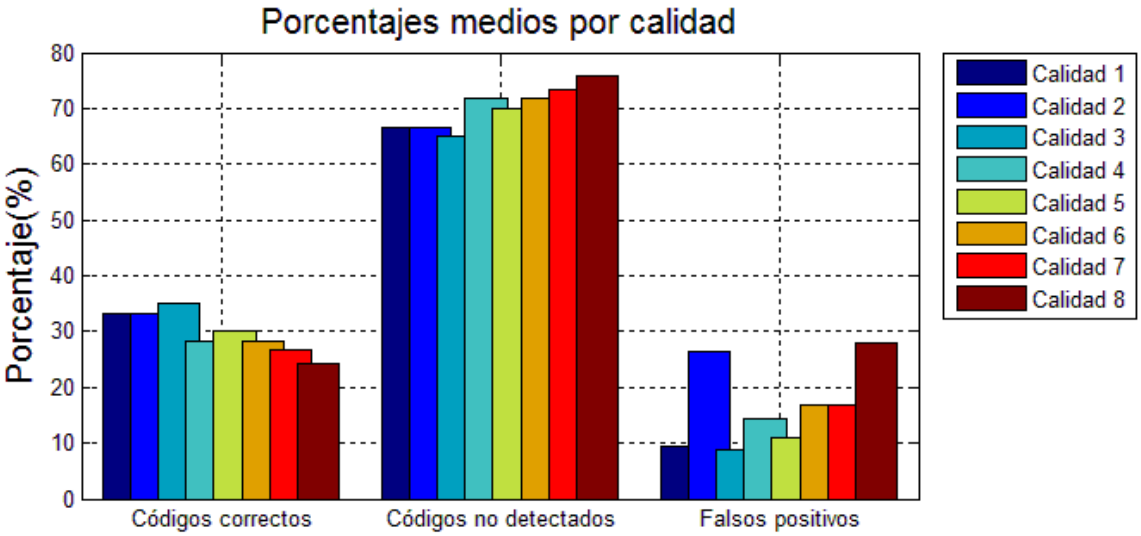
Gráfica 5.6. Porcentaje de fallos por resolución y calidad para la transformada de Hough semialeatoria.

Observando los porcentajes generales (**Gráfica 5.7**), la resolución 2 es la que mayor porcentaje de códigos correctos consigue, entorno a un 50 %, en cuanto a códigos no detectados en el caso de la resolución 0 tenemos niveles muy altos cercanos al 100%. La resolución 3 es la que mayores porcentajes de falsos positivos nos da (40%).



Gráfica 5.7. Porcentajes medios por resolución para la transformada de Hough semialeatoria.

En cuanto a los resultados por calidad (**Gráfica 5.8**) como sucediera con la transformada de Hough ninguna calidad de compresión de las imágenes jpeg se comporta mejor que otras.



Gráfica 5.8. Porcentajes medios por calidad para la transformada de Hough semialeatoria.

5.2.2.1 CONCLUSIONES

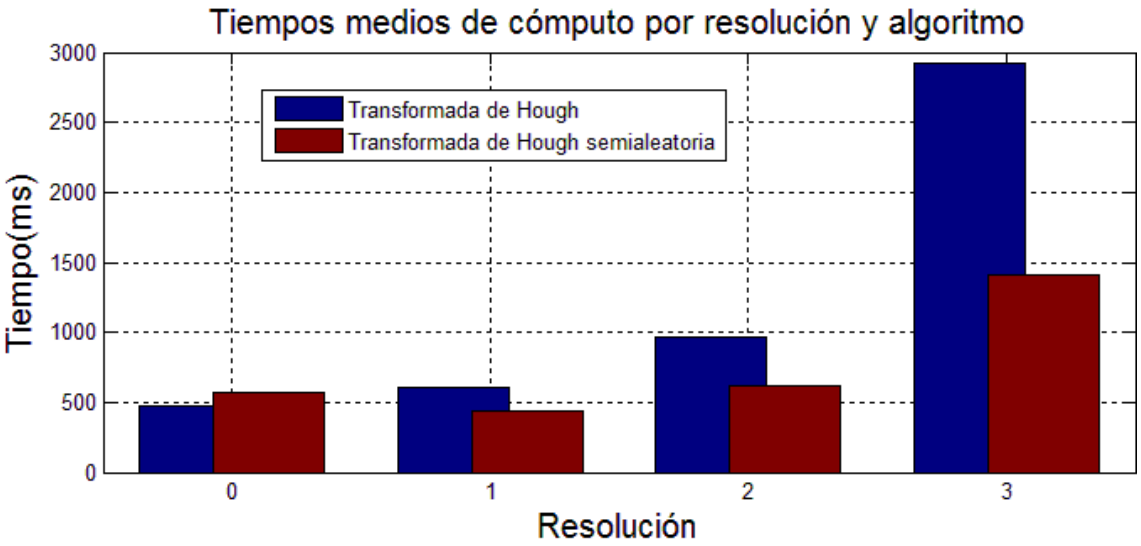
Las conclusiones a las que podemos llegar en el caso de la transformada de Hough semialeatoria son muy similares a las comentadas en la sección 5.2.1.1. La resolución que mejor se comporta es la número 2, así como la 0 y la 3 son las que peores resultados en cuanto a códigos no detectados y falsos positivos nos dan.

El hecho de que obtengamos resultados más pobres es debido al número de líneas que somos capaces de detectar con la transformada semialeatoria. Como se explicó en la sección 4.3.3 en la que se comenta la forma de extraer las líneas más significativas a través de la matriz de Hough, estas líneas se obtienen siempre y cuando superen cierto umbral, en el caso de la transformada semialeatoria el número de líneas que superen este umbral puede ser inferior, o incluso superior (matriz normalizada al máximo), por lo que la obtención de rectas y la posterior extracción de códigos se ve dificultada.

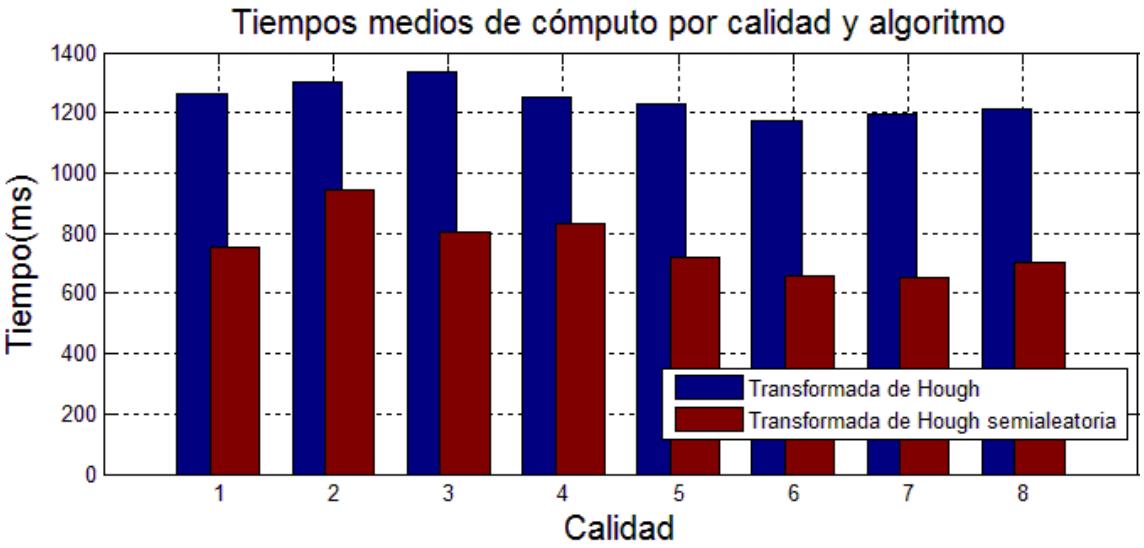
5.2.3 COMPARATIVA DE TIEMPOS DE CÓMPUTO ENTRE LA TRANSFORMADA DE HOUGH Y LA TRANSFORMADA DE HOUGH SEMIALEATORIA

En las siguientes gráficas (**Gráfica 5.9** y **Gráfica 5.10**) tenemos los tiempos de cómputo por resolución y calidad respectivamente para ambos algoritmos,

mientras que la calidad no afecta al tiempo de procesado sí lo hace la resolución, como era de esperar, obteniendo peores resultados conforme aumentamos la resolución de las imágenes.



Gráfica 5.9. Tiempos medios de cómputo para la resolución y algoritmo.



Gráfica 5.10. Tiempos medios de cómputo por calidad y algoritmo.

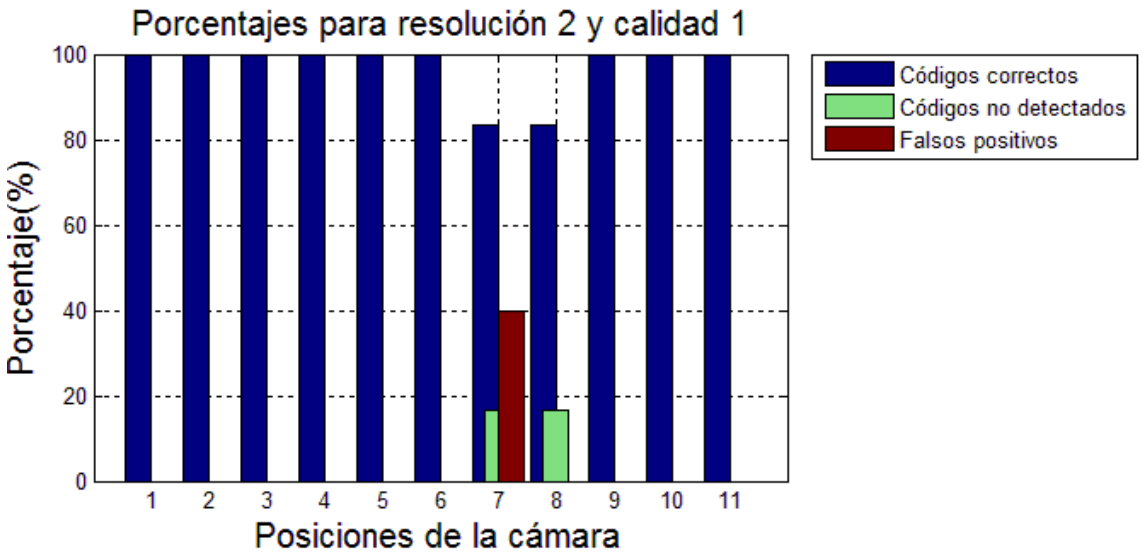
Los tiempos de cómputo son relevantes en función de la resolución. Un aumento de calidad conlleva también un aumento de líneas detectadas en la imagen elevando en última instancia el número de comprobaciones de las posibles balizas. La calidad sin embargo no afecta a estos tiempos, como se ha comentado en secciones anteriores las imágenes de una misma resolución son muy similares, por lo que su procesado es prácticamente el mismo.

En cuanto a la comparativa de algoritmos los tiempos de cómputo de la transformada de Hough semialeatoria siempre son menores que los de la transformada normal.

5.2.4 IMÁGENES RESULTADO

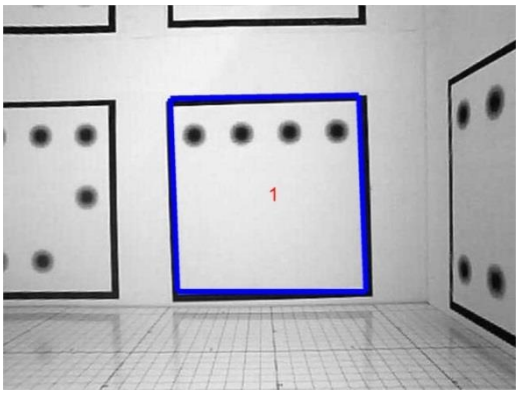
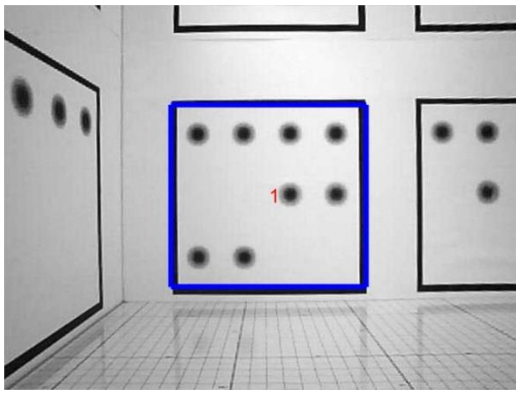
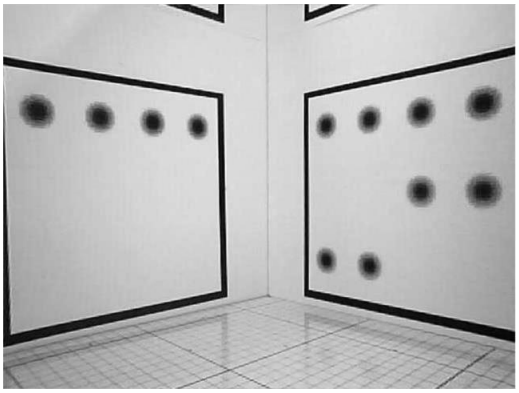
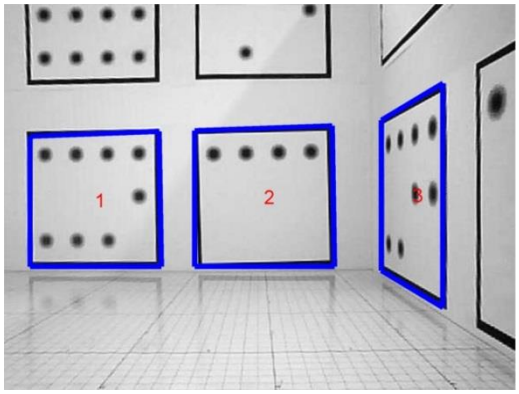
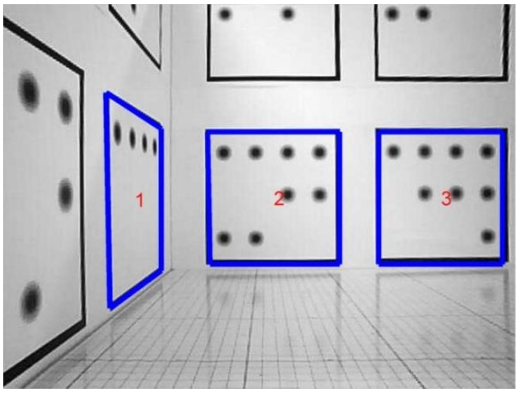
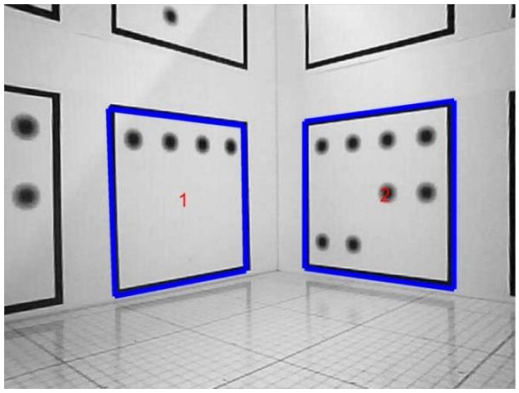
Una vez repasados los resultados obtenidos con ambos algoritmos los mejores datos se consiguen para la transformada de Hough con resolución 2 y calidad 1.

Los errores que obtenemos están localizados en las imágenes de las posiciones 7 y 8 (**Gráfica 5.11**), en ambos tenemos una marca muy ladeada que dificulta su detección por nuestro algoritmo.

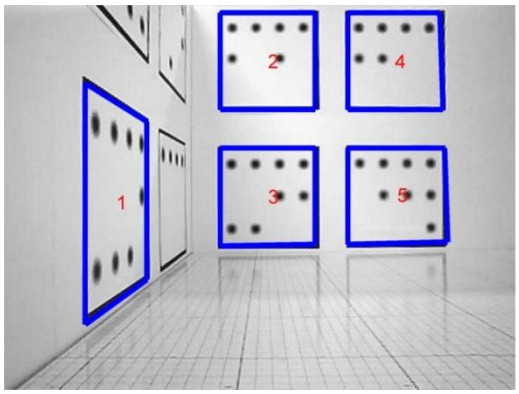


Gráfica 5.11. Porcentaje de códigos correctos, no detectados y falsos positivos por posición.

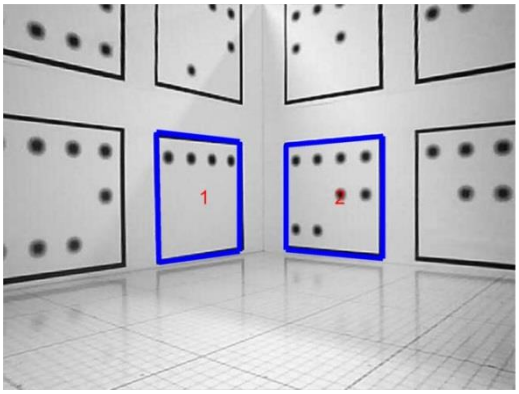
En la **Figura 5.4** podemos ver las imágenes obtenidas para una resolución 2 y calidad 1.

	
Posición 1	Posición 2
	
Posición 3	Posición 4
	
Posición 5	Posición 6

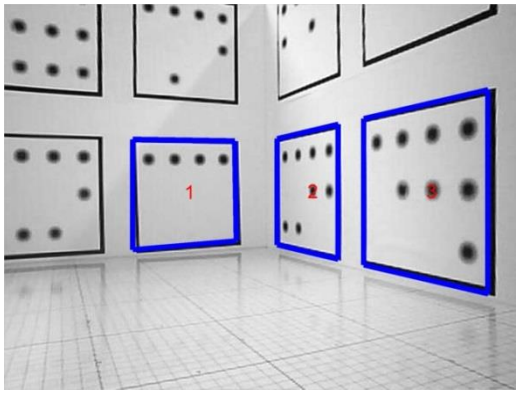
Posición 7



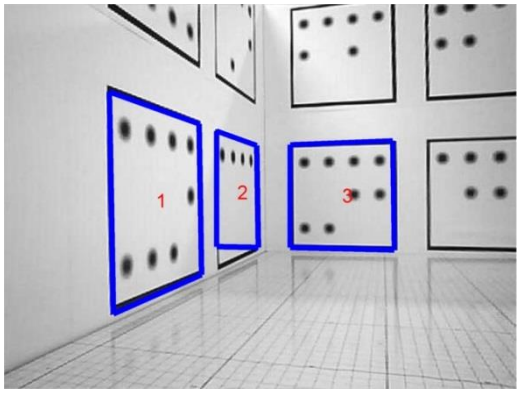
Posición 8



Posición 9



Posición 10



Posición 11

Figura 5.4. Imágenes resultado en cada posición.

6. CONCLUSIONES Y LÍNEAS FUTURAS

Conclusiones

En este trabajo se han presentado las herramientas para la detección de balizas para la orientación de un robot móvil. Para ello se ha hecho un estudio de las técnicas de procesado de imagen: máscaras de detección de bordes, algoritmos para la transformada de Hough, selección de líneas y codificación de balizas.

Algoritmos de detección de bordes

Se ha hecho un estudio de 3 máscaras diferentes (Sobel, Prewitt y Roberts). Con todas podemos encontrar las líneas de la imagen satisfactoriamente, pero en concreto nos interesa obtener el mayor número de puntos frontera que nos permitan definir con la mayor precisión posible las líneas de la imagen. Es por ello que optamos por utilizar la máscara tipo Roberts. En cuanto a los tiempos utilizados para filtrar la imagen no se aprecian mejoras significativas de ninguna máscara sobre las demás. En definitiva se pueden obtener los bordes con cualquier filtrado paso alto de la imagen, ya sea en el dominio espacial (con máscaras) o bien con métodos basados en la transformada de Fourier.

Algoritmos para la transformada de Hough

Ante las 3 versiones de los algoritmos de la transformada de Hough que se han desarrollado podemos extraer las siguientes conclusiones:

- Versión 1: es el algoritmo que realiza la transformada de Hough sobre todos los puntos frontera detectados en la imagen. Es el más lento pero también el que nos proporcionará resultados más fiables, dando más peso a las rectas más importantes de la imagen. Ante una posible aplicación será el que se elegirá en primera instancia.

- Versión 2: la implementación es la misma que la realizada en la versión 1 pero obteniendo la matriz de Hough a partir de un porcentaje de los puntos que detectamos en la imagen. Disminuye considerablemente el tiempo de procesado de la versión 1 y los resultados son aceptables en cuanto a las líneas detectadas en las imágenes probadas.
- Versión 3: se obtiene la matriz de Hough a partir de pares de puntos escogidos aleatoriamente de la imagen. Su tiempo de cómputo es menor que los algoritmos anteriores con buenos resultados, siempre y cuando el número de puntos detectados en la imagen sea reducido, ya que el tiempo de procesado aumenta exponencialmente cuando los puntos crecen.

Selección de líneas

Una vez obtenida la matriz de Hough con cualquiera de los algoritmos descritos anteriormente deberemos seleccionar las líneas más importantes que definan nuestra imagen. Para ello se desarrollan 3 algoritmos:

- Semejanza entre rectas: obteniendo un número elevado de líneas de la matriz de Hough somos capaces de extraer las líneas más importantes a partir de la distancia entre ellas. Con este método nos pueden surgir problemas, como que obtengamos líneas repetidas debido a que su distancia con las demás no sea inferior a cierto umbral y se tome como línea única. Es el que más tiempo de cómputo necesita.
- Rectas a partir de la matriz de Hough: si conocemos el número de rectas que tiene nuestra imagen podemos, a partir de la matriz de Hough, extraerlas atendiendo a los máximos de la matriz y obviando las rectas que estén en las celdas adyacentes de la matriz de Hough. El inconveniente de este algoritmo es que debemos conocer el número de rectas que queremos obtener. Es el que menos tiempo utiliza pero tenemos que saber previamente el número de líneas que queremos extraer de la imagen.
- Detección automática a partir de la matriz de Hough: la idea es la misma que la versión anterior pero ahora el número de líneas que obtendremos dependerán de si superan cierto umbral impuesto. Este algoritmo es el

que mejor se comporta, consiguiendo siempre las rectas de más peso. Utiliza más tiempo que el anterior pero en contraposición es automático, por lo que no tenemos por qué saber previamente las líneas que tiene la imagen.

Codificación de balizas

La primera premisa para el desarrollo de las balizas era dotarlas de líneas para poder detectarlas mediante la transformada de Hough. En segundo lugar la codificación interior no debe interferir en esta detección de líneas es por ello que se eligen círculos degradados. Por otra parte utilizando un total de 12 bits nos permite tener un total de 4096 codificaciones diferentes y por lo tanto 4096 balizas distintas.

Resultado sobre imágenes reales

Para la detección de las balizas en imágenes reales necesitamos una forma de asegurarnos que el recorte que hacemos en función de las líneas encontradas es el de una marca real. Para ello elegimos uno o varios de los bits codificados para asegurarnos que estamos ante una baliza. Esto nos reduce el número de marcas que podemos codificar ya que se reduce el número de bits pero es necesario para reducir la detección de falsas balizas.

En cuanto a la detección de estas, se hace una buena decodificación siempre y cuando encontremos correctamente las rectas y por lo tanto la baliza. La aparición de rectas de más dificulta la detección y en muchos casos la hace imposible.

Aplicación sobre las imágenes captadas por el robot

En este apartado es importante la resolución y calidad jpeg elegida de las imágenes captadas por el robot. Se utilizan 2 algoritmos de la transformada de Hough, versiones 1 y 2 y una extracción de líneas automática. Los mejores resultados los obtenemos para una resolución de 640x480 píxeles y máxima calidad y versión 1 del algoritmo de Hough, de esta forma detectamos correctamente más del 90% de los códigos. El uso de la versión 2 reduce el

tiempo de cómputo pero con un número de códigos detectados inferior, no llegamos al 70% en el mejor de los casos.

Por lo tanto, si prima la detección correcta de códigos, deberemos elegir la versión 1.

Líneas futuras

Para las posibles líneas futuras hacemos especial énfasis en los siguientes puntos:

- Mejora de la detección: se podrían mejorar las balizas añadiéndoles elementos que ayuden a su detección. Actualmente se usan bits de su codificación para detectarlos, pero se podrían añadir otros elementos como colores o marcas distintas a los círculos (por ejemplo cuadrados) degradados para que su detección se mejore.
- Otros entornos de programación: realizar la implementación en otros entornos, en concreto en C se puede hacer uso de las librerías de procesado de imagen “Open CV” las cuales son de libre uso.
- Orientar el robot móvil: A partir de la visualización de las balizas deberíamos ser capaces de posicionar nuestro robot, obtener la distancia y ángulo sobre las marcas que es capaz de ver.
- Programación de tareas simples: Una vez conocida la posición del robot móvil podríamos establecer unos objetivos al robot en función de la codificación de la baliza que es capaz de ver. Estas tareas pueden ser la de seguir las marcas en orden ascendente según su numeración, que cambie su dirección, etc.

ANEXO

APLICACIÓN EN GUI DE MATLAB

En esta sección se presenta una pequeña aplicación que prueba los algoritmos diseñados, los 3 algoritmos de la transformada de Hough y la detección de balizas.

Para lanzar la aplicación escribiremos la palabra “prog” en la línea de comandos de Matlab, una vez tengamos como directorio actual la carpeta donde se aloja el programa.

En la **Figura A.1** podemos ver la pantalla principal de nuestra aplicación y a partir de la **Figura A.2** se detalla la funcionalidad de cada parte.

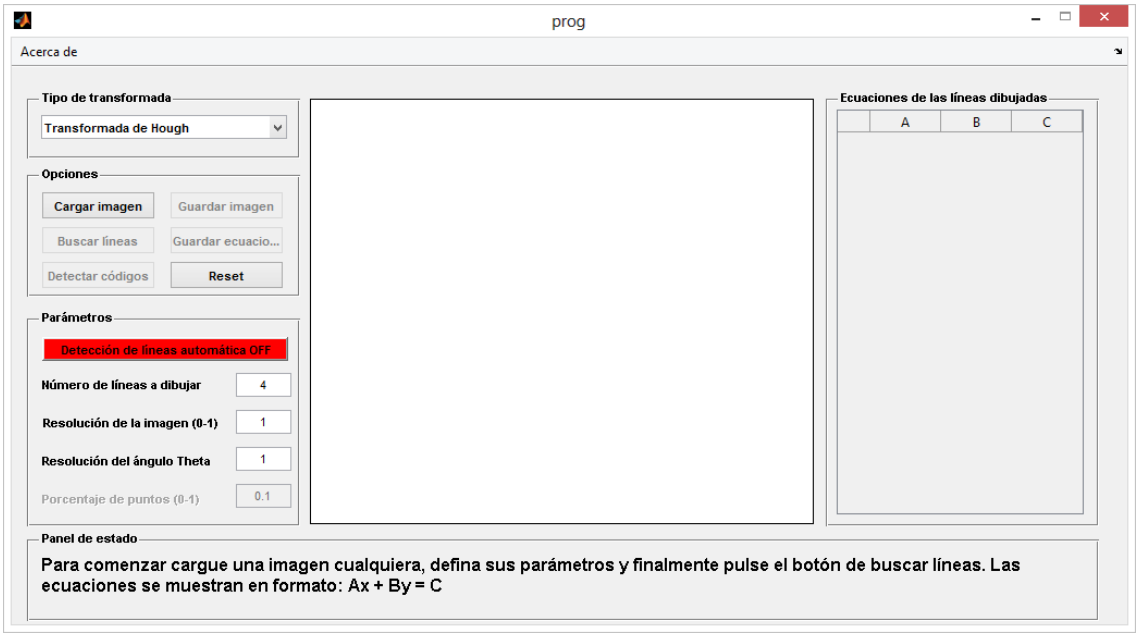


Figura A.1. Pantalla principal de la aplicación.

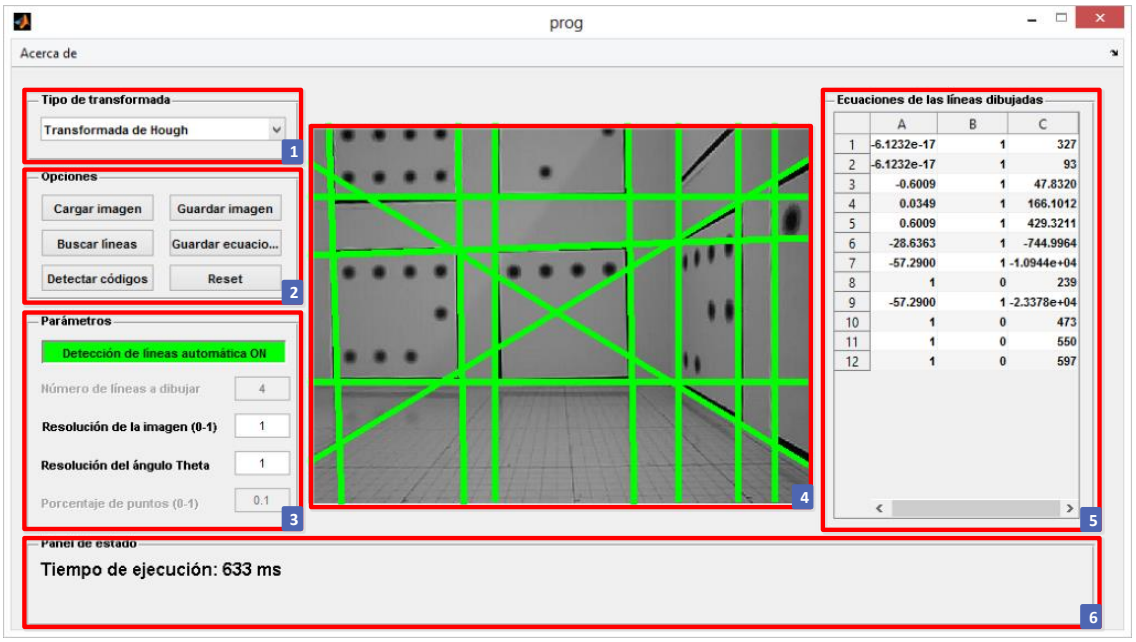


Figura A.2. Aplicación en funcionamiento.

- *Sección 1*. En esta parte podemos seleccionar el tipo de transformada de Hough que queremos utilizar para obtener las líneas de la imagen (**Figura A.3**).

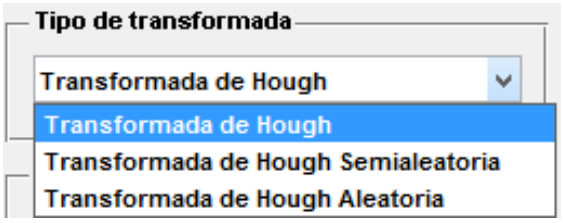


Figura A.3. Detalle de los tipos de transformadas de Hough.

- *Sección 2*. Desde este punto podemos cargar la imagen de prueba, guardar la representación con las líneas, guardar las ecuaciones en formato .txt, resetear los valores introducidos por el usuario y detectar los códigos existentes en la imagen, esta opción nos llevará a otra pantalla que explicaremos más adelante.

- *Sección 3*. Aquí introducimos los parámetros para los algoritmos, en función del elegido la aplicación nos permitirá introducir unos u otros. Podemos elegir si queremos que la detección de líneas sea automática (**Figura A.4**), el número de rectas que queremos representar, las resoluciones tanto de la

imagen como del ángulo theta y el porcentaje de puntos a computar en las transformadas aleatorias.

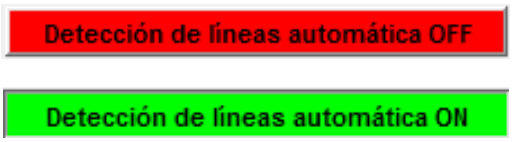


Figura A.4. Detalle del botón que nos permite seleccionar el tipo de detección.

- *Sección 4*. Se representa la imagen cargada y los resultados obtenidos al buscar sus rectas.

- *Sección 5*. Se representan las ecuaciones de las rectas obtenidas en formato Ax + By = C. Este es en el formato en que son guardadas con la opción de la sección 2.

- *Sección 6*. En este cuadro se muestran mensajes tiempo de cómputo, errores etc.

Si elegimos la opción de detectar códigos de la sección 2 se abrirá una segunda pantalla (**Figura A.5**). A continuación mostramos las secciones de las que está compuesta.

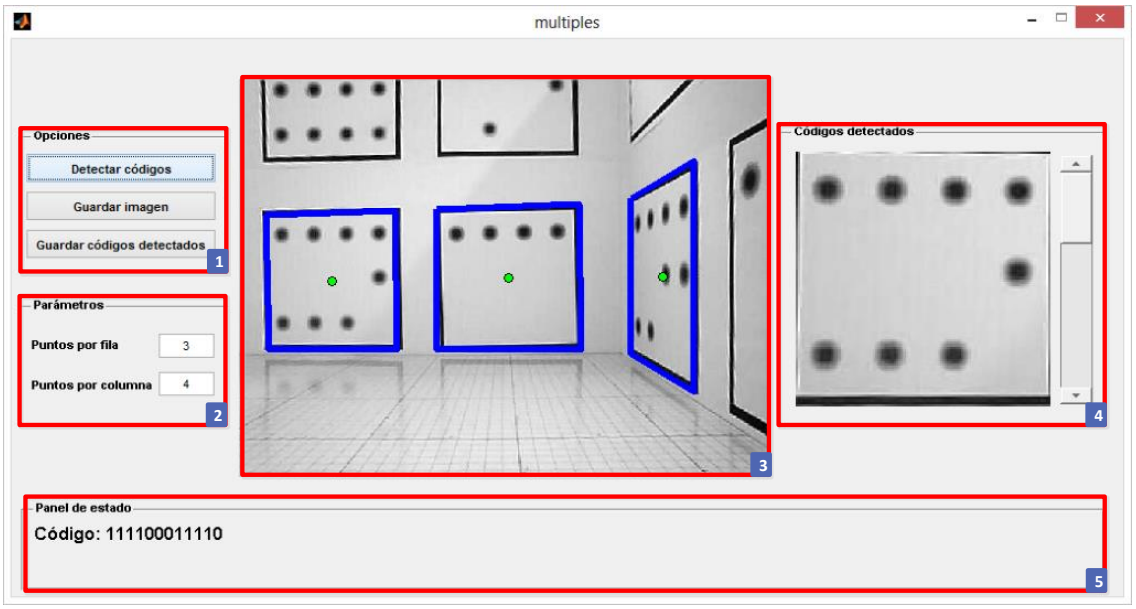


Figura A.5. Pantalla para la detección de balizas en la imagen cargada.

- *Sección 1*. Opciones que disponemos en esta pantalla, detectar códigos, guardar la imagen con los códigos recuadrados o guardar los códigos detectados en formato .txt.

- *Sección 2*. Podemos definir como son los códigos que queremos detectar, número de puntos por fila y columna.

- *Sección 3*. Se representa la imagen cargada y los resultados obtenidos al buscar sus códigos.

- *Sección 4*. Detalle de los códigos obtenidos, podemos verlos deslizando el slide.

- *Sección 6*. En este cuadro se muestran mensajes codificación de cada baliza detectada, errores, etc.

BIBLIOGRAFÍA

[1] R. C. González y R. E. Woods, Tratamiento Digital de Imágenes, Addison-Wesley, 1996.

[2] J. M. S. Prewitt, «Object enhancement and extraction, Picture Processing and Psychopictorics,» *Academic Press,* 1970.

[3] E. Sobel, Camera Models and Machine Perception, Stanford University, 1970.

[4] L. G. Roberts, Machine Perception Of Three-Dimensional Solids, Massachusetts Institued of Technology, 1963.

[5] R. C. González, R. E. Woods y S. L. Eddins, Digital Image Processing Using Matlab, Second Edition, Gatesmark Publishing, 2009.

[6] P. V. C. Hough, «Method and means for recognizing complex patterns». U. S. Patent 3 Patente 069 654, 1962.

[7] D. H. Ballard, «Generalizing the Hough Transform to Detect Arbitrary Shapes,» *Pattern Recognition,* vol. 13, nº 2, pp. 111-122, 1981.

[8] R. O. Duda y P. E. Hart, «Use of the Hough Transformation to Detect Lines and Curves in Pictures,» *Comm. ACM,* vol. 15, p. 11–15, 1972.

[9] M. Pinto, F. Santos, A. P. Moreira y R. Silva, «Robust and Fast Algorithm for Artificial Landmark Detection in an Industrial Environment,» *Journal of Automation and Control Engineering,* vol. 1, nº 2, 2013.

[10] «Surveyor Corporation,» Disponible en: http://www.surveyor.com/. [Último acceso: Mayo 2013].

[11] C. Ruiz Picazo, PFC: Descripción Y Desarrollo De Aplicaciones Sobre El Robot Móvil Surveyor SRV1B, Universidad Miguel Hernández De Elche.