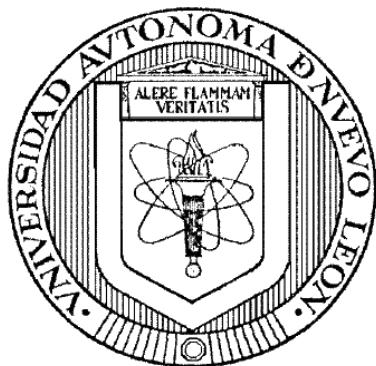


UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

DIVISIÓN DE ESTUDIOS DE LICENCIATURA



DETECCIÓN POR MEDIO DE TÉCNICAS DE VISIÓN COMPUTACIONAL
PARA EL CONTROL DE ROBOTS MÓVILES E-PUCK

POR

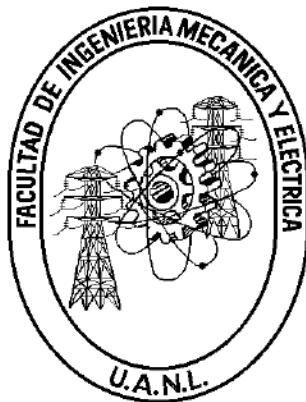
ROLANDO MORALES SUÁREZ

EN OPCIÓN AL GRADO DE
INGENIERO EN MECATRÓNICA

CD. UNIVERSITARIA

AGOSTO DE 2015

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE LICENCIATURA



DETECCIÓN POR MEDIO DE TÉCNICAS DE VISIÓN COMPUTACIONAL
PARA EL CONTROL DE ROBOTS MÓVILES E-PUCK

POR

ROLANDO MORALES SUÁREZ

EN OPCIÓN AL GRADO DE

INGENIERO EN MECATRÓNICA

CD. UNIVERSITARIA

AGOSTO DE 2015

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE LICENCIATURA

Los miembros del Comité de Tesis recomendamos que la Tesis “**Detección por medio de técnicas de visión computacional para el control de robots móviles E-puck**”, realizada por el alumno **Rolando Morales Suárez**, con número de matrícula 1535382, sea aceptada para su defensa como opción al grado de **Ingeniero en Mecatrónica**.

El Comité de Tesis

Dr. Jesús De León Morales
Asesor

Dr. Miguel Francisco Escalante Gutiérrez
Revisor

Dr. Marco Túlio Mata Jiménez
Revisor

Vo. Bo.

Dr. Arnulfo Treviño Cubero
División de Estudios de Licenciatura

Cd. Universitaria, a Agosto de 2015

Agradecimientos

Agradezco al Dr. Jesús de León Morales, por la confianza otorgada para trabajar con el equipo de laboratorio, así como por aceptarme como tesista. También, por la motivación dada a lo largo de la carrera para seguir adelante.

Agradezco al Dr. Miguel Francisco Escalante Gutiérrez y al Dr. Marco Túlio Mata Jiménez, por aceptar ser revisores del presente trabajo de tesis.

Agradezco a la Dra. Satu Elisa Schaeffer, por su ayuda y recomendaciones en el desarrollo del programa de detección.

Agradezco al M.C. Rubén Hernández Alemán, por sus recomendaciones en la elaboración de reportes y en la escritura de la tesis.

Agradezco a los compañeros del laboratorio de Sistemas Eléctricos de Potencia y Protecciones, por su apoyo, su ayuda y por el espacio brindado para trabajar en la tesis.

Por último, aunque no menos importante, agradezco a mis padres, hermanos y amigos por su apoyo. Sin ellos no estaría aquí.

RESUMEN

Rolando Morales Suárez.

Candidato para el grado de Ingeniero en Mecatrónica.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica

Título del estudio:

DETECCIÓN POR MEDIO DE
TÉCNICAS DE VISIÓN COMPUTACIONAL
PARA EL CONTROL DE ROBOTS MÓVILES E-PUCK

Profesor Asesor: Dr. Jesús De León Morales

Los robots son utilizados más a menudo en diversas áreas de aplicación, debido a que ofrecen ventajas tales como de precisión y repetitividad. Sin embargo, existen tareas que no pueden ser realizadas por un solo robot, por lo que se requiere de la cooperación y coordinación de múltiples robots. Tal comportamiento es inducido a través de una ley de control, la cual requiere información de la posición y orientación de cada robot en un plano coordenado.

En el presente trabajo se diseñó un programa de detección de posición y orientación basado en la segmentación por color para tres robots móviles. Esto, con el objetivo de establecer una retroalimentación de la información entre los robots. Asimismo, el programa de detección fue desarrollado en software libre, se asegura su reproducibilidad, y se comprueba su uso en *Windows 7* y *8.1*.

Finalmente, se programó un sencillo controlador *ON/OFF* con banda de histéresis en tres robots E-puck para evaluar el desempeño del programa de detección, así como también su empleo en sistemas con múltiples robots.

Índice general

Resumen	III
Índice General	VIII
Índice de Figuras	xii
Índice de Cuadros	xiii
1. Introducción	1
1.1. Revisión de trabajos previos	3
1.2. Objetivos de la tesis	5
1.3. Organización de la tesis	5
2. Procesamiento Digital de Imágenes (PDI)	7
2.1. Introducción	7
2.2. Etapas en el análisis de imágenes	8
2.3. Representación de una imagen	9
2.4. Vecindad de un píxel	10
2.5. Máscaras	11
2.6. Filtros espaciales	11
2.6.1. Filtro del valor máximo	12
2.6.2. Filtro del valor mínimo	14
2.6.3. Manejo de las fronteras	14
2.7. Imágenes a color	15
2.7.1. Espacio RGB	16

2.7.2. Espacio HSV	17
2.7.2.1. Conversión de RGB a HSV	18
2.8. Imágenes binarias	19
2.8.1. Segmentación por umbrales	20
2.8.2. Operaciones de fuentes múltiples	21
2.8.2.1. Operaciones lógicas (<i>and</i> y <i>or</i>)	21
2.8.3. Operaciones morfológicas	22
2.8.3.1. Erosión	22
2.8.3.2. Dilatación	23
2.8.3.3. Apertura	24
2.8.3.4. Cierre	25
2.8.4. Etiquetado de objetos	26
2.8.5. Características de objetos binarios	29
2.8.5.1. Caja envolvente	29
2.8.5.2. Centroide	30
3. Robótica móvil	31
3.1. Introducción	31
3.2. Clasificación de robots móviles	31
3.3. Ruedas	33
3.4. Postura de un robot móvil	34
3.5. Configuración de robots móviles	35
3.6. Sistemas multi-robot	36
4. Descripción del <i>Hardware</i> y <i>Software</i>	37
4.1. Descripción del <i>Hardware</i>	37
4.1.1. Dispositivo de captura de imágenes	37

4.1.2.	Equipo de cómputo	39
4.1.3.	Robot E-puck	41
4.1.3.1.	Origen del robot	41
4.1.3.2.	Componentes	41
4.1.3.2.1.	Estructura mecánica	42
4.1.3.2.2.	Microcontrolador	42
4.1.3.2.3.	Sensores y actuadores	42
4.1.3.3.	Comunicación	44
4.2.	Descripción del <i>Software</i>	45
4.2.1.	Sistema operativo	45
4.2.2.	Lenguaje de programación para el programa de detección	46
4.2.3.	Librerías para Python	47
4.2.4.	Lenguaje de programación para el robot E-puck	47
5.	Programa de detección	49
5.1.	Introducción	49
5.2.	Estructura del programa	50
5.2.1.	Marcadores	52
5.2.2.	Módulo de detección de robots (<i>Robot_tracker</i>)	53
5.2.2.1.	Inicialización	54
5.2.2.2.	Preprocesamiento	57
5.2.2.3.	Procesamiento	59
5.2.2.4.	Entrega de resultados	63
6.	Configuraciones de la plataforma de experimentación y resultados	66
6.1.	Introducción	66
6.2.	Configuración de la plataforma experimental	67

6.2.1.	Pruebas de tiempo de procesamiento	67
6.2.1.1.	Metodología	69
6.2.1.2.	Resultados	70
6.2.2.	Pruebas de control de postura de robots E-puck	71
6.2.2.1.	Metodología	73
6.2.2.2.	Resultados	74
6.2.3.	Pruebas de calidad y detección con desenfoque	75
6.2.3.1.	Metodología	75
6.2.3.2.	Resultados	75
6.2.4.	Prueba de reproducibilidad	77
6.2.4.1.	Metodología	78
6.2.4.2.	Resultados	78
7.	Conclusiones y trabajos futuros	79
7.1.	Conclusiones	79
7.2.	Trabajos futuros	80
Bibliografía		85
Apéndices		86
Apéndice A. Preparación y uso del programa de detección		87
A.1.	Preparación del programa de detección	87
A.1.1.	Utilizar el Python IDLE(GUI)	88
A.1.2.	Utilizar el <i>Command Prompt</i> de <i>Windows</i>	89
A.2.	Uso del programa desarrollado	92
A.2.1.	Archivo de texto	94

A.2.2. Módulo de detección de robots (<i>Robot_tracker</i>)	95
A.2.3. Módulo de muestra de imagen (<i>show_cam</i>)	97
A.2.4. Módulo de ajuste de umbral (<i>thres_adj</i>)	99
A.2.5. Módulo de muestra de imágenes binarizadas <i>thres_adj</i>	103
A.2.6. Módulo de ajuste de unidades <i>units_selection</i>	104
Apéndice B. Uso y programación del robot E-puck	107
B.1. Uso del robot E-puck	107
B.2. Batería	110
B.3. Emparejamiento	111
B.4. Programación del robot E-puck	113
B.4.1. Requerimientos	113
B.4.2. Creación de un proyecto nuevo	114
B.4.3. Escribir en la memoria del E-puck	116
B.4.4. Ley de control programada	118
B.4.5. Programa en C de la ley de control	122
Apéndice C. Tablas de datos de resultados	127

Índice de figuras

2.1.	Etapas en el análisis de imágenes [45].	9
2.2.	Vecindades de un píxel. a) Vecindad 4 en cruz. b) Vecindad 4 en X. c) Vecindad 8.	11
2.3.	Aplicación del filtro del valor máximo.	13
2.4.	a) Recorrido de la aplicación de una máscara. b) Resultado de la aplicación del filtro del valor máximo.	13
2.5.	Resultado de la aplicación del filtro del valor mínimo.	14
2.6.	Máscara fuera de la frontera.	15
2.7.	Espacio RGB [1].	16
2.8.	Espacio HSV [1].	18
2.9.	Imagen binaria.	20
2.10.	a) Imagen de referencia 1. b) Imagen de referencia 2. c) Resultado de la aplicación de la operación lógica <i>or</i> a las imágenes de referencia. d) Resultado de la operación lógica <i>and</i> a las imágenes de referencia. . .	22
2.11.	a) Imagen original. b) Resultado de la aplicación de la erosión en la imagen a).	23
2.12.	a) Imagen original. b) Resultado de la aplicación de la dilatación en la imagen a).	24
2.13.	Operación de apertura. a) Imagen inicial. b) Imagen erosionada. c) Imagen dilatada.	25
2.14.	Operación de cierre. a) Imagen de entrada. b) Imagen dilatada. c) Imagen erosionada.	25
2.15.	a) Objetos sin etiqueta en una imagen binaria. b) Objetos etiquetados.	26

2.16. Aplicación del algoritmo DFS en una imagen binaria con vecindad	8.	28
2.17. Caja envolvente de un objeto.	.	29
3.1. Robot móvil con rueda fija.	.	34
3.2. Postura de un RMcR con direccionamiento diferencial.	.	34
3.3. Algunos tipos de locomoción de RMcR [50]. a) Configuración Ackerman. b) Configuración delta (Triciclo clásico). c) Configuración síncrona.	.	35
4.1. Cámara web <i>Logitech C210</i> [27].	.	38
4.2. Cámara web <i>Logitech C525</i> [28].	.	39
4.3. Computadora portátil <i>Samsung</i> [47].	.	40
4.4. Robot E-puck.	.	41
4.5. Vista explosionada de la estructura mecánica [35].	.	42
4.6. Esquema de conexiones del E-puck [35].	.	43
4.7. Participación de los SOs en el mercado [36].	.	46
5.1. Esquema de interacción entre usuario y programa.	.	50
5.2. Interacción de los módulos con el archivo de texto.	.	51
5.3. Interfaz gráfica de usuario.	.	52
5.4. Marcadores propuestos.	.	53
5.5. Diagrama de flujo del módulo de detección.	.	55
5.6. Transformación de RGB a HSV. a) Imagen RGB. b) Imagen HSV.	.	58
5.7. a) Imagen binarizada. b) Imagen al aplicarle operaciones morfológicas.	.	59
5.8. Imagen de detección global.	.	60
5.9. Representación del <i>offset</i> establecido para contar los píxeles amarillos.	.	61
5.10. Detección incorrecta.	.	62
5.11. Ángulo θ en un marcador.	.	63
5.12. Resultados mostrados al usuario.	.	65

6.1.	Configuración para la prueba de tiempo 1.	68
6.2.	Configuración para la prueba de tiempo 2.	68
6.3.	a) Posicionamiento de los robots cercanamente uno del otro. b) Posicionamiento de los robots alejadamente uno del otro.	69
6.4.	Aplicación de un control todo o nada con banda de histéresis [56]. . .	71
6.5.	Ejemplo del control implementado.	73
6.6.	Movimiento de un robot E-puck a una posición deseada.	74
6.7.	a) Imagen captada por la cámara C210. b) Imagen captada por la cámara C525.	76
6.8.	imágenes tomadas con la cámara C525. a) Captura desenfocada. b) Captura enfocada.	76
6.9.	Detección de tres robots E-puck con desenfoque.	77
A.1.	GUI de Python.	88
A.2.	Ventana del <i>Command Prompt</i>	89
A.3.	Propiedades del sistema.	90
A.4.	Propiedades avanzadas.	90
A.5.	Selección en variables de usuario.	91
A.6.	Editar variables del sistema.	91
A.7.	Botón para el módulo <i>robot_tracker</i>	95
A.8.	Resultado de una imagen procesada.	97
A.9.	Botón para el módulo <i>show_cam</i>	97
A.10.	Diagrama de flujo del módulo <i>show_cam</i>	98
A.11.	Botón del módulo <i>show_cam</i>	99
A.12.	Diagrama de flujo del módulo <i>thres_adj</i>	99
A.13.	Ventanas emergentes al iniciar el módulo <i>Thres_adj</i>	100

A.14. Botones deslizadores de selección de umbral.	101
A.15.a) Imagen umbralizada. b) Imagen al aplicar operaciones morfológicas.	101
A.16. Ventana <i>save_data</i> .	102
A.17. Módulo <i>thres_adj</i> .	103
A.18. Diagrama de flujo del módulo <i>Show_hsv_binary</i> .	103
A.19. Módulo <i>units_selection</i> .	104
A.20. Diagrama de flujo del módulo <i>Units_selection</i> .	105
A.21. Ventana de entrada de texto.	105
B.1. Interruptor de encendido [17].	108
B.2. Interruptor selector.	108
B.3. Cartón aislante de la batería del E-puck [17].	110
B.4. Ícono de <i>Bluetooth</i> .	111
B.5. Opciones del menú de <i>Bluetooth</i> .	111
B.6. Ventana de dispositivos.	112
B.7. Puertos COM.	112
B.8. Nuevo proyecto de MPLAB v8.	114
B.9. Elección del archivo mcp.	115
B.10. Selección de la herramienta de simulación.	115
B.11. Selección del compilador.	116
B.12. <i>Tiny Bootloader</i> .	117
B.13. Led de transmisión de datos.	118
B.14. Interruptor azul de escritura.	118
B.15. Diagrama de flujo de la ley de control.	119
B.16. Ángulo del comando atan2l.	121

Índice de cuadros

3.1. Clasificación de los robots móviles [43].	32
3.2. Clasificación de robots móviles por su tipo de movimiento [43].	33
4.1. Sensores y actuadores del E-puck [35].	44
4.2. Especificaciones técnicas del protocolo <i>Bluetooth</i> [57].	45
5.1. Configuraciones del puerto serial para un robot.	54
5.2. Valores predeterminados en la clase <i>Default</i>	57
6.1. Resumen de tiempos medidos con ambas cámaras a 60cm con los E-puck cerca uno de otro.	70
C.1. Resultados preliminares 1 (resultados en segundos).	127
C.2. Resultados preliminares 2 (resultados en segundos).	127
C.3. Resultados preliminares 3 (resultados en segundos).	127
C.4. Resultados preliminares 4 (resultados en segundos).	128
C.5. Resultados preliminares 5 (resultados en segundos).	128
C.6. Resultados preliminares 6 (resultados en segundos).	128
C.7. Resultados preliminares 7 (resultados en segundos).	128
C.8. Resultados preliminares 8 (resultados en segundos).	129

Capítulo 1

Introducción

Día a día, el uso de robots en muchas áreas de aplicación va en ascenso. Estos tienden a realizar tareas como la limpieza, hasta tareas muy sofisticadas o riesgosas para el ser humano. Sin embargo, existen tareas que no pueden ser realizadas por un solo robot, por lo que se requiere de un sistema de múltiples robots para lograr un objetivo global. Dicho sistema puede definirse como un conjunto de unidades autónomas que interactúan entre sí en un entorno en común. Esto, para desempeñar tareas tales como exploración, seguridad, misiones de rescate, vigilancia, cartografía, etc. [44]; donde los robots se mueven manteniendo una formación para mejorar la eficiencia y eficacia en sus tareas [46]. Sin embargo, tal comportamiento puede ser inducido a través de una ley de control, la cual requiere información referente a la postura de los robots. El problema de formación consiste en conformar el movimiento de varios robots de modo que, de una manera colaborativa y ordenada, realicen una tarea [12]. Mientras tanto, existen diferentes alternativas para manejar dichas formaciones, entre ellas están: “Estructura virtual, comportamiento grupal y configuración líder seguidor” [44].

Un caso más específico puede verse en [21], donde se presenta un esquema de control y observación basado en modos deslizantes para abordar el problema de control de formación. En dicho esquema, es necesario medir la posición de cada robot en el plano coordenado, para lo cual existen diversas alternativas. Una de ellas es el denominado *Dead Reckoning*, el cual calcula la posición actual del robot, en relación al punto inicial, de acuerdo a los giros de las llantas del mismo (Odometría). Para esto,

el robot necesita sensores en las llantas que registren dichos giros [18]. Este método conlleva a un error absoluto que va incrementándose con el tiempo, por lo que no es muy apropiado para el esquema de control. Otra alternativa son los módulos GPS (Por sus siglas en inglés, “Global Positioning System”) [18], los cuales son sensores fijos al cuerpo del robot. El problema con estos módulos es que no pueden hacer mediciones con gran precisión (“su precisión es de 10-20 *m* para vehículos en movimiento” [18]), aparte de drenar la energía de la batería, por lo que tampoco son una solución muy viable. Otra opción sería montar una o varias cámaras en la estructura de cada robot para el rastreo de *marcadores* (figura o patrón “discriminante” fácil de localizar) o de objetos del entorno, y con ello determinar las posiciones de los robots, tal y como se muestra en [7][19][49][23][9]. El problema es que es demasiado voluminoso, aparte de ser más costoso al requerir más equipo. Una buena alternativa para rastrear los robots, debido a su fácil y económica implementación, además de ser capaz de obtener una buena precisión, es con una cámara posicionada de tal manera que sea posible captar todos los elementos a utilizar. Al tener una cámara como sensor, se necesita de un *Software* capaz de identificar a cada uno de los robots, y poder transmitir la información de su respectiva posición y orientación a estos, las cuales son requerimientos para el algoritmo de control. Para ello, es necesario aplicar técnicas de visión computacional.

Por lo tanto, el problema de detección de postura, consiste en “determinar la localización de un robot con respecto a un sistema de referencia absoluto” [18], ya que esta es un requisito para el esquema de control de formación.

En la siguiente sección se realiza una revisión de los trabajos previos relacionados con los *Software* basados en dichas técnicas.

1.1. Revisión de trabajos previos

Existen varios programas con los que es posible identificar a los robots por medio de técnicas de visión computacional. En [24] se muestra *Predator*, un programa que es capaz de detectar un objeto deseado al seleccionarlo manualmente en la primer *frame* (imagen), para posteriormente rastrearlo en tiempo real o en video mediante una comparación entre la selección inicial y los *frames* consecuentes. Este programa cuenta con capacidad de discriminación entre objetos diferentes y aprendizaje para evitar falsas detecciones, aunque solo se rastrea un objeto a la vez.

Un programa apto para rastrear múltiples objetos a la vez es *Bio-Tracking* [6], el cual tiene la finalidad de detectar el movimiento y orientación de animales. Otro programa para el rastreo de animales es *OpenControl* [2]. En [51], Simkowiak presenta también un programa para el seguimiento de múltiples objetos por medio de la diferencia de píxeles entre el *frame* actual y el anterior, con lo que se determina si existe un objeto en movimiento. Estos programas tienen algo en común: no necesitan marcadores para rastrear los objetos deseados. El problema es que si se quisiera transmitir la información a los objetos detectados, en caso de perder el enfoque o dejar de detectarlo momentáneamente, se perdería el orden de detección, ya que no existe algo que asegure cual objeto es cual.

Para asegurar siempre el rastreo de los objetos deseados generalmente se utilizan marcadores. En [20], se muestra *Reactivision 1.4*, un programa que detecta los marcadores *fiducial* (marcadores propuestos por los autores, formados con patrones de círculos en blanco y negro de múltiples tamaños). Estos marcadores son fácilmente detectables en escala de grises con iluminación difusa. También es posible obtener la información de la posición y orientación de cada uno de los marcadores, conservando siempre la relación entre marcador y objeto. El problema, debido a la naturaleza de

los marcadores, es que puede haber problemas de detección en caso de que exista desenfoque por parte de la cámara. Otro problema se presentaría si los marcadores se encuentran muy alejados de la fuente de grabación y ésta es de baja calidad.

Por otra parte, en [26] se presenta el programa *SwisTrack*, el cual permite la detección de múltiples agentes, tanto con marcadores, como sin ellos. Además, los marcadores pueden ser de color o en blanco y negro. También es posible obtener la posición y orientación de los objetos detectados. Todo esto le da versatilidad para múltiples aplicaciones. A pesar de sus virtudes, existen algunos inconvenientes, como los presentes en el uso de los marcadores, ya que para utilizar marcadores en blanco y negro se requiere de una iluminación difusa, lo más libre de sombras posible, para evitar ruido en la imagen; en el caso de marcadores a color, los umbrales se establecen el espacio de color RGB (por sus siglas en inglés, “Red, Green, Blue”), los cuales pueden ser difíciles de establecer en comparación con otros espacios de color, tal y como se menciona en [8].

La mayoría de los trabajos previamente mencionados, hacen rastreo de objetos sin necesidad de marcadores, lo que puede ser un problema en caso de desenfoque de la cámara. Los otros programas que no tienen este inconveniente, debido a la posibilidad de rastrear marcadores, pueden no ser muy robustos bajo ciertas condiciones de luminosidad. Dicho esto, en la siguiente sección se establecen los objetivos de la presente tesis.

1.2. Objetivos de la tesis

Los objetivos del presente trabajo son:

- Diseñar un programa de detección de marcadores sencillos que sea robusto a ciertas condiciones, tanto de iluminación, como de desenfoque. Dicho programa debe estar basado completamente en *Software* libre y debe de ser reproducible.
- Evaluar el desempeño del software para sistemas de multi-robot utilizando robots E-puck, y transmitiendo la información de la detección a los mismos.

1.3. Organización de la tesis

La tesis cuenta con 7 capítulos. El capítulo 1 muestra una introducción y una visión general de la problemática que se abordará en el presente trabajo. Además, se revisan trabajos previos para la solución de dichos problemas basados en técnicas de visión computacional. También se presentan los objetivos que se buscan en el presente trabajo.

El marco teórico que fue base para el desarrollo de la tesis se presenta en los capítulos 2 y 3. En él capítulo 2 se incluyen las técnicas de visión computacional utilizadas, así como conceptos generales relacionados con el procesamiento de imágenes. Mientras tanto, en el capítulo 3 se presentan los conceptos relacionados con la robótica móvil y se describen las dos diferentes maneras de abordar el problema de sistemas con múltiples robots.

En el capítulo 4, se describe el *Hardware* y el *Software* utilizados en el desarrollo del presente trabajo.

En el capítulo 5, se presenta la solución propuesta. Se muestra la interfaz de usuario utilizada así como la explicación de los detalles y funcionalidades del programa de detección.

Para la evaluación del programa se requiere de configuraciones para la experimentación. Estas se exponen en el capítulo 6, dando a conocer las metodologías para las pruebas, y con ello determinar el desempeño de la solución propuesta. Se presentan también los resultados obtenidos.

Por último, en el capítulo 7 se dan a conocer las conclusiones y trabajos futuros.

Capítulo 2

Procesamiento Digital de Imágenes (PDI)

En el presente capítulo, se definen conceptos generales del PDI. Además, se presentan algoritmos de procesamiento que serán utilizados en capítulos posteriores.

2.1. Introducción

En los últimos años, el área de visión computacional ha crecido vertiginosamente, la cual se ha expandido a muchas otras áreas de investigación gracias al rápido desarrollo tecnológico. Los objetivos de su aplicación pueden ser muchos, tales como rastreo de objetos, reconocimiento de patrones, reconstrucción 3D, etc., pero todos ellos ocupan en mayor o menor medida el PDI.

El PDI se podría definir como el tratamiento que se le da a una imagen digital con el propósito de obtener o resaltar características de interés [45]. El PDI se divide, tal y como se menciona en [11], en tres niveles en relación a los objetivos del tratamiento aplicado:

- **Nivel bajo:** Se refiere al preprocesamiento de una imagen (Aplicación de filtros, umbralización, binarización, etc.).
- **Nivel medio:** Este nivel se refiere a la extracción de características (distancias,

ángulos, cajas envolventes, etc.).

- **Nivel alto:** Se refiere a las relaciones entre características de imágenes (detección de movimiento, reconstrucción 3D, clasificadores, etc.).

En el presente trabajo se utilizaron solamente técnicas correspondientes a los niveles bajo y medio, algunas de las cuales se presentarán más adelante.

2.2. Etapas en el análisis de imágenes

El análisis de una imagen se divide en varias etapas (véase Figura 2.1) [45]. Dicho análisis consiste en tomar una muestra del entorno por medio de un digitalizador. Al tener la imagen digitalizada, se le aplica el preprocesamiento (eliminación de ruido, suavizado, etc.). Una vez preprocesada la imagen, se realiza la segmentación de objetos (umbralización para extraer un objeto del fondo de la imagen) para establecer los límites de cada uno. En la representación se guarda la información relevante de una manera en la que pueda utilizarse con mayor facilidad. La etapa de la descripción se refiere a diferenciar entre objetos de varias clases. La etapa final se compone de tres partes: la detección, donde se indica si un objeto existe en la imagen o no; el reconocimiento, donde se etiqueta al objeto en base a la información entregada en la etapa de descripción; y finalmente, en la interpretación determina el significado de los objetos.

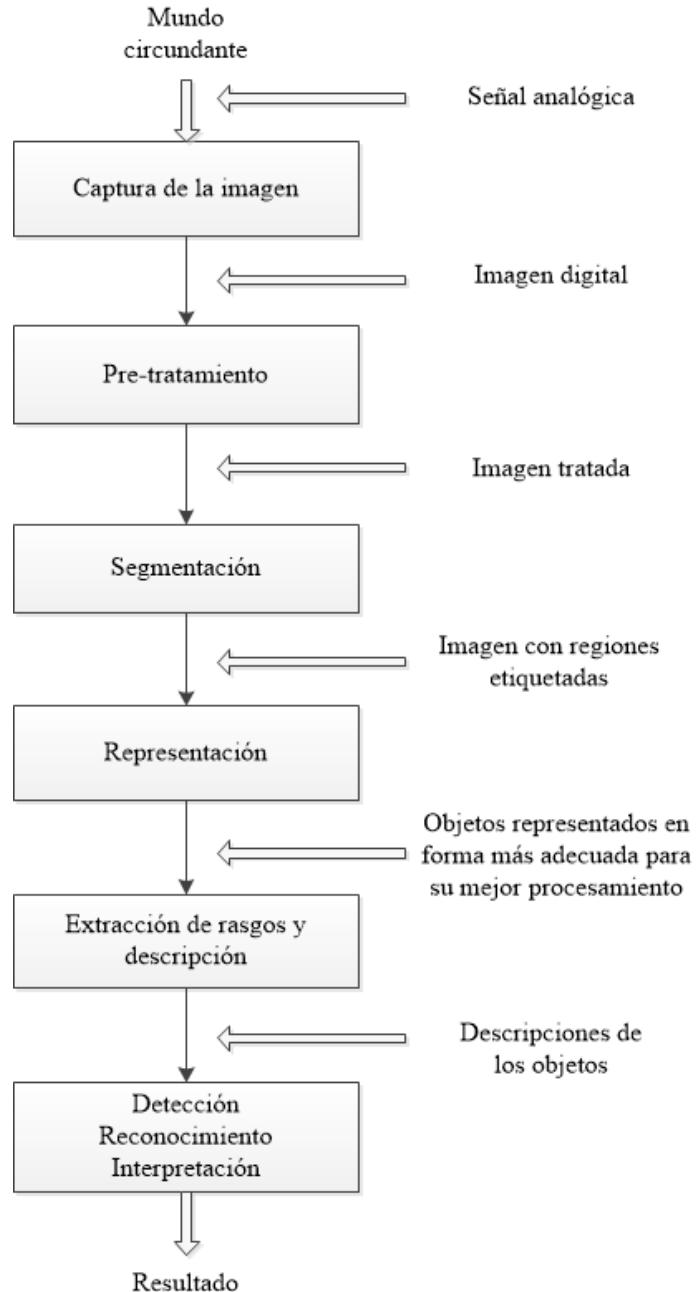


Figura 2.1: Etapas en el análisis de imágenes [45].

2.3. Representación de una imagen

Una captura de una cámara, al ser digitalizada, guarda ordenadamente los valores de la intensidad de luz recibida en pequeños espacios denominados píxeles para

formar una imagen [45]. La cantidad de píxeles de una imagen dependerá de la resolución de la cámara. En el PDI generalmente se trabaja con resoluciones pequeñas, por ejemplo 640x480 píxeles, debido a que el procesamiento suele demandar muchos recursos computacionales.

Una imagen $Img(x, y)$ puede ser representada en forma matricial como

$$Img(x, y) = \begin{bmatrix} Img(1, 1) & Img(1, 2) & \cdots & Img(1, c) \\ Img(2, 1) & Img(2, 2) & \cdots & Img(2, c) \\ \vdots & \vdots & \ddots & \vdots \\ Img(r, 1) & Img(r, 2) & \cdots & Img(r, c) \end{bmatrix}, \quad (2.1)$$

donde r es el número de renglones y c es el número de columnas.

A continuación, en el siguiente apartado se presentará un tipo de relación entre píxeles denominado vecindad.

2.4. Vecindad de un píxel

La vecindad de un píxel se refiere a los píxeles adyacentes al píxel analizado. Existen varios tipos de vecindades, las cuales se pueden observar en la Figura 2.2, donde el píxel analizado p (en el centro), está rodeado por sus vecinos (píxeles oscuros) dependiendo de la vecindad adoptada.

EL uso de la vecindad se basa en la utilidad que se le quiera dar al análisis. Dependiendo de la situación, una vecindad u otra puede ser mejor.

p1	p2	p3
p4	P	p5
p6	p7	p8

a)

p1	p2	p3
p4	P	p5
p6	p7	p8

b)

p1	p2	p3
p4	P	p5
p6	p7	p8

c)

Figura 2.2: Vecindades de un píxel. a) Vecindad 4 en cruz. b) Vecindad 4 en X. c) Vecindad 8.

2.5. Máscaras

Una máscara, también llamada *kernel*, es una matriz de coeficientes, que se utiliza para hacer operaciones entre imágenes. Estas matrices pueden ser de cualquier tamaño y se usan en las operaciones con píxeles o en la aplicación de filtros. Algunos ejemplos de máscaras son

$$\begin{bmatrix} 1 & 0 & 1 \\ 2 & 0 & 2 \\ 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix},$$

llamadas Sobel y Prewitt respectivamente, las cuales se usan en la detección de bordes.

Se hablará a continuación de los filtros, los cuales utilizan las máscaras para su funcionamiento.

2.6. Filtros espaciales

Los filtros, llamados en PDI como filtros espaciales, se pueden definir como operaciones realizadas a una imagen de referencia, en la que el valor de los píxeles de la

imagen de salida dependen de los píxeles que abarca una máscara [5].

Existen diferentes tipos de filtros:

- *Filtros lineales* .- Éstos filtros tienen “pesos en una vecindad” [54], que dependen del tamaño de la máscara. Algunos ejemplos son el filtro de la media o promedio, el filtro gaussiano, el filtro prewitt, etc. La aplicación de estos filtros se basa en la *convolución discreta* [5]. Dicho tema no será abordado en el presente trabajo.
- *Filtros no lineales* .- El resultado de la aplicación de éstos filtros depende, generalmente, del orden en el que se encuentren los píxeles al aplicar la máscara. Algunos ejemplos son el filtro de la mediana, el filtro del valor máximo, el filtro del valor mínimo, etc. [5].

En la presente tesis, solo se utilizan los filtros del valor máximo y mínimo, los cuales se explicarán a continuación.

2.6.1. Filtro del valor máximo

El filtro del valor máximo, como su nombre lo indica, toma el valor máximo de la vecindad del píxel analizado. Dicha vecindad está dada por el tamaño de la máscara. La representación matemática de este filtro está dada por (2.2), donde \hat{I} representa a la imagen de salida, (x, y) son las coordenadas del píxel donde se encuentra centrada la máscara e (i, j) son las coordenadas de la máscara [11].

$$\hat{I}(x, y) = \{I(x + i, y + j) | (i, j) \in R\} = \max(R(x, y)). \quad (2.2)$$

Representando una imagen binaria como una matriz de unos y ceros, en la Figura 2.3 se muestra el proceso de aplicación de el filtro del valor máximo, donde la matriz de 3X3 corresponde a la máscara aplicada.

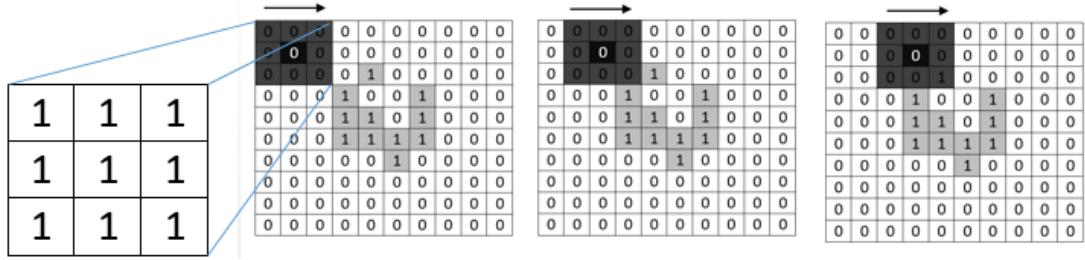


Figura 2.3: Aplicación del filtro del valor máximo.

La máscara recorre a la imagen de entrada píxel por píxel. En cada avance se va tomando el valor máximo de la vecindad dada por la máscara, y dicho valor se escribe en el píxel correspondiente de la imagen de salida [29]. Esto se puede apreciar mejor en la Figura 2.4 , donde la imagen de la izquierda muestra el recorrido de la máscara por la imagen de referencia, y la imagen de la derecha muestra el resultado final.

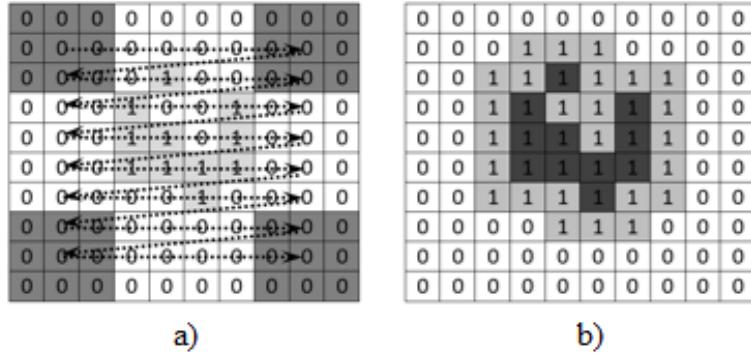


Figura 2.4: a) Recorrido de la aplicación de una máscara. b) Resultado de la aplicación del filtro del valor máximo.

Se logra observar un crecimiento de los bordes de la figura binaria de la imagen de salida. Este filtro, al igual que el filtro del valor mínimo que se verá a continuación, son utilizados en las operaciones morfológicas (sección 2.8.3), las cuales se basan en la forma y geometría de los objetos.

2.6.2. Filtro del valor mínimo

El filtro del valor mínimo es prácticamente igual que el filtro del valor máximo, con la única diferencia de tomar como resultado el valor mínimo que se encuentre en la vecindad de la máscara que se aplique. En (2.3), se muestra la representación matemática segun Cuevas [11].

$$\hat{I}(x, y) = \{I(x + i, y + j) | (i, j) \in R\} = \min(R(x, y)). \quad (2.3)$$

El resultado de la aplicación del filtro del valor mínimo, en la misma imagen de referencia utilizada en la sección anterior, se observa en la Figura 2.5, en la cual se aprecia la diferencia de píxeles con respecto a la imagen de referencia. Este filtro reduce los bordes de objetos binarios, por esa razón el resultado está formado por ceros en su totalidad, ya que el objeto era muy delgado.

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Figura 2.5: Resultado de la aplicación del filtro del valor mínimo.

2.6.3. Manejo de las fronteras

Una *frontera* se define como los límites de una imagen [11]. Un problema común es cuando la máscara está situada en la frontera o cerca de ella, ya que queda fuera

de la imagen (tal y como se muestra en la Figura 2.6). Esto llega a ser un problema si los píxeles de dichas fronteras son de importancia para el procesamiento.

Figura 2.6: Máscara fuera de la frontera.

Para resolver esta problemática existen diversas soluciones [11], las cuales se muestran a continuación:

- Agregar ceros en las fronteras.
 - Replicar los últimos renglones y columnas fuera de las fronteras.
 - Agregar renglones y columnas como si la imagen se espejara.
 - Empezar el recorrido desde el píxel $(1, 1)$ hasta $(x - 1, y - 1)$.
 - Agregar renglones y columnas como si la imagen se repitiera cíclicamente.
 - Realizar el recorrido por toda la imagen pero sin tomar en cuenta los píxeles que quedan fuera.

2.7. Imágenes a color

Para la representación de los colores en una imagen digital se utilizan los *espacios de color*. El espacio más utilizado es el RGB (*Red, Green, Blue*), el cual se puede observar en las pantallas LCD, cámaras digitales, etc. Pero existen muchos más (HSV, YUV, CMY, etc.) [11], de los cuales solo se hablará del RGB y el HSV.

2.7.1. Espacio RGB

El espacio de color RGB está compuesto por los colores primarios rojo, verde y azul. Se puede representar por medio de un cubo unitario, donde un color deseado estaría localizado dentro de su volumen. En la Figura 2.7 se observa dicho cubo.

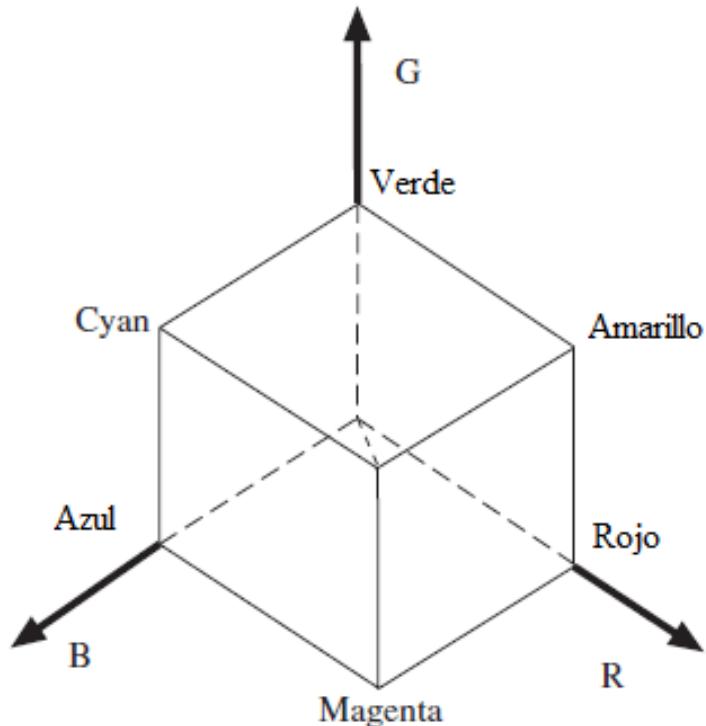


Figura 2.7: Espacio RGB [1].

Una imagen a color en el espacio RGB se representa como una matriz ($r * c * 3$) donde el número de renglones y columnas depende de la resolución de la imagen. La matriz es de *tres dimensiones* porque cada píxel debe de tener el valor que corresponde a la intensidad de color de cada uno de los canales (RGB). Una matriz de tres dimensiones se podría representar como

$$\begin{bmatrix} (0, 255, 180) & (100, 85, 35) & (90, 230, 240) \\ (20, 41, 20) & (200, 30, 63) & (210, 200, 200) \\ (200, 10, 10) & (50, 60, 80) & (255, 255, 255) \end{bmatrix},$$

donde los valores varían entre 0 y 255, los cuales dependen del formato de la imagen (una imagen de 8 bits puede tener valores entre 0 y 255). Cada grupo de tres valores corresponde a un píxel de un color, por lo que la matriz anterior sería una imagen de nueve píxeles, siendo cada uno un color diferente.

2.7.2. Espacio HSV

El espacio de color HSV (por sus siglas en inglés de *Hue, Saturation, Value*) es muy utilizado en el área de PDI debido a que es más intuitivo que el RGB. El espacio HSV es representado como un “cono hexagonal”, tal y como se ve en la Figura 2.8, donde Hue es la tonalidad (dicho de otra manera, el color), la saturación es la cantidad de tonalidad (que tan fuerte es) y el valor (o brillo) es la brillantez de la tonalidad [1].

Las ventaja que tiene el espacio HSV sobre el espacio RGB en la visión computacional, es la fácil selección de umbrales, ya que los colores tienen un canal específico (Hue), no tres como es el caso del espacio RGB. Encontrar umbrales en éste último espacio suele ser una tarea difícil y requiere de un control estricto en las condiciones de iluminación.

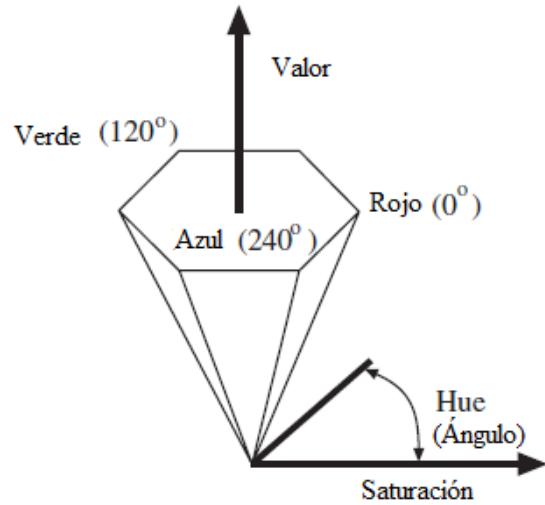


Figura 2.8: Espacio HSV [1].

2.7.2.1. Conversión de RGB a HSV

Las imágenes obtenidas con las cámaras digitales comunes son generalmente en RGB, por lo que tienen que transformarse al espacio de color HSV para obtener sus ventajas de umbralización. El algoritmo de transformación para imágenes de 8 bits (0-255) [11], consiste en normalizar inicialmente los valores de los tres canales RGB

$$R' = \frac{R}{255} \quad G' = \frac{G}{255} \quad B' = \frac{B}{255}. \quad (2.4)$$

Luego, se obtiene C_{max} , C_{min} y Δ como se ve enseguida:

$$C_{max} = \text{Max}(R', B', G') \quad C_{min} = \text{Min}(R', G', B') \quad \Delta = C_{max} - C_{min}. \quad (2.5)$$

Teniendo los valores anteriores, es posible calcular el valor de H de la siguiente manera

$$H = \begin{cases} 60^\circ * (\frac{G' - B'}{\Delta} * mod6) & , C_{max} = R' \\ 60^\circ * (\frac{B' - R'}{\Delta} + 2) & , C_{max} = G' ; \\ 60^\circ * (\frac{R' - G'}{\Delta} + 4) & , C_{max} = B' \end{cases} \quad (2.6)$$

mientras tanto, el valor de la saturación está dado por

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases} . \quad (2.7)$$

Finalmente, el valor de V es calculado como

$$V = C_{max}. \quad (2.8)$$

2.8. Imágenes binarias

Las imágenes binarias son matrices de $(r * c)$ elementos (2 dimensiones). Cada uno de dichos elementos puede tener solamente dos valores que son 1 y 0 (blanco y negro, respectivamente). Debido a la naturaleza de las imágenes binarias, estas son más ligeras que las imágenes RGB, en cuanto al espacio de memoria que ocupan, por ser bidimensionales (2X2). En la Figura 2.9 se puede ver un ejemplo de una imagen binaria.

En una imagen binaria es posible extraer características de interés, tal y como se verá más adelante.

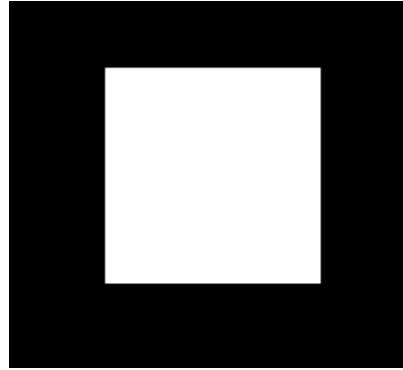


Figura 2.9: Imagen binaria.

2.8.1. Segmentación por umbrales

La segmentación por umbral es “una forma especial de cuantificación”, donde los píxeles de la imagen se asignan a categorías diferentes [11]. Dichas categorías se representan en imágenes binarias donde el píxel será uno o cero dependiendo si se encuentra dentro o fuera del umbral, respectivamente. Se puede representar como

$$I_{bin}(p) = \begin{cases} 0, & \text{si } p < Umbral \\ 1, & \text{si } p \geq Umbral \end{cases} \quad I_{bin}(p) = \begin{cases} 0, & \text{si } p > Umbral \\ 1, & \text{si } p \leq Umbral \end{cases}, \quad (2.9)$$

donde I_{bin} es la imagen binaria y p es un píxel cualquiera.

También es posible que exista más de un umbral, esto es

$$I_{bin}(p) = \begin{cases} 0, & \text{si } p < Umbral_1 \wedge p > Umbral_2 \\ 1, & \text{si } p \geq Umbral_1 \wedge p \leq Umbral_2 \end{cases}. \quad (2.10)$$

La cantidad de umbrales podría ser incluso mayor que 2 (el límite reside en el

espacio de color con el que se trabaje) y dependerá del algoritmo que se utilice, así como del objetivo buscado. Generalmente se utilizan 2 umbrales (valor máximo y mínimo deseado), ya que tener muchos umbrales puede hacer a un algoritmo muy complejo o redundante.

2.8.2. Operaciones de fuentes múltiples

En ocasiones es necesario tener en una sola imagen los resultados de múltiples imágenes, por lo que es común realizar operaciones de fuentes múltiples [11]. Por ejemplo, en programas de televisión es común ver cambios de escena donde una imagen se atenua y otra se esclarece, mostrando en la pantalla el resultado de dos diferentes imágenes. En el presente trabajo solamente se mostrará una clase de operaciones entre múltiples imágenes, las operaciones lógicas (operaciones cuyo resultado es verdadero o falso).

2.8.2.1. Operaciones lógicas (*and* y *or*)

Las operaciones lógicas son exclusivas de imágenes binarias [11]. En (2.11) se representa el resultado I_{binA} del producto lógico (\wedge) y la suma lógica (\vee) entre dos imágenes binarias I_{binB} e I_{binC} , donde p representa cualquier píxel.

$$I_{binA}(p) = I_{binB}(p) \wedge I_{binC}(p) \quad , \quad I_{binA}(p) = I_{binB}(p) \vee I_{binC}(p). \quad (2.11)$$

Por otra parte, en la Figura 2.10, se puede observar el resultado de aplicar las operaciones lógicas a dos imágenes de referencia (las primeras dos). La tercera imagen corresponde al resultado de la operación lógica *or* y la cuarta corresponde a la *and*.

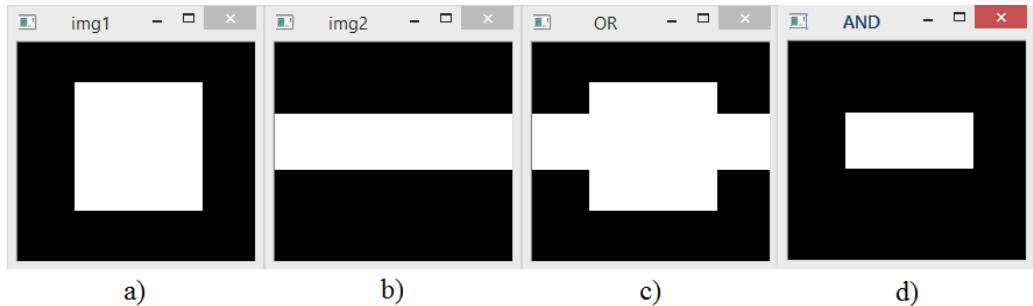


Figura 2.10: a) Imagen de referencia 1. b) Imagen de referencia 2. c) Resultado de la aplicación de la operación lógica *or* a las imágenes de referencia. d) Resultado de la operación lógica *and* a las imágenes de referencia.

2.8.3. Operaciones morfológicas

Las operaciones morfológicas son las que están basadas en la morfología matemática (estudio de las formas [13]), las cuales “realzan la geometría y forma de los objetos” y están fundamentadas con la teoría de conjuntos [39]. Dichas operaciones fueron inicialmente concebidas para trabajar sobre imágenes binarias [11]. En los ejemplos relacionados con las operaciones morfológicas se considerará la aplicación de una máscara que está formada por una matriz de unos de dimensión (3X3) como la utilizada en la sección 2.6.1.

2.8.3.1. Erosión

La operación de erosión (\ominus) es utilizada para reducir artefactos en imágenes binarias [11]. Su representación matemática está dada por (2.12), donde X representa a la imagen de entrada y B es la estructura de referencia (máscara) que es aplicada a X .

$$X \ominus B = \{x | B_x \subseteq X\}. \quad (2.12)$$

Esto significa que, el píxel de la imagen de salida será cero a no ser que todos los píxeles que abarque la máscara sobre la imagen sean iguales a la misma. Un ejemplo de ello se muestra en la Figura 2.11, donde se aplica la operación de erosión. En la imagen de salida se dejaron representados los píxeles iniciales para que la comparación entre entrada y salida sea más fácil.

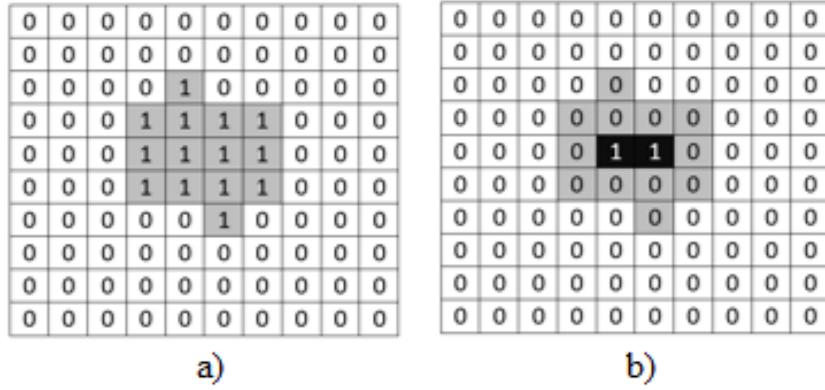


Figura 2.11: a) Imagen original. b) Resultado de la aplicación de la erosión en la imagen a).

2.8.3.2. Dilatación

La operación de dilatación \oplus se utiliza para expandir objetos que se encuentran en imágenes binarias. Su representación matemática está dada por (2.12), donde X representa a la imagen de entrada y B es la estructura de referencia (máscara) que se aplica a X .

$$X \oplus B = \{x | X \cap B_x \neq 0\}. \quad (2.13)$$

Esto quiere decir que, los píxeles del resultado serán uno siempre y cuando por lo menos un píxel en el área cubierta por la máscara sea igual a uno, tal y como se muestra en la Figura 2.12.

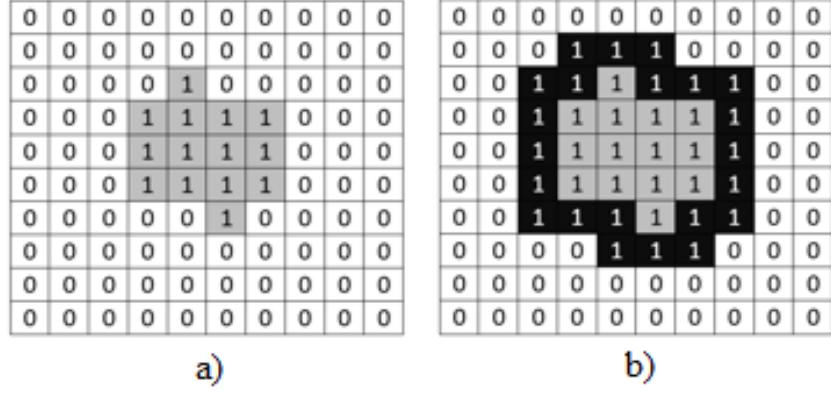


Figura 2.12: a) Imagen original. b) Resultado de la aplicación de la dilatación en la imagen a).

En el PDI es común la combinación de operaciones morfológicas, ya que en algunos casos, esto conlleva a la obtención de mejores resultados. Las combinaciones más utilizadas son las que se describen a continuación.

2.8.3.3. Apertura

La operación de apertura \circ se define como la dilatación de la erosión [11], esto es,

$$X \circ B = (X \ominus B) \oplus B. \quad (2.14)$$

En la Figura 2.13, se describe la operación de apertura y sus usos, donde se observa que el resultado final (2.13.c) son dos objetos separados aproximadamente del mismo tamaño que en la imagen de inicial. Además, los objetos resultantes tienen contornos más finos.

0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0	0 0 0 1 1 0 0 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 1 0 1 1 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 1 1 0 0 0

a)

b)

c)

Figura 2.13: Operación de apertura. a) Imagen inicial. b) Imagen erosionada. c) Imagen dilatada.

2.8.3.4. Cierre

Contrario a la operación anterior, la operación de cierre \bullet se define como la erosión de la dilatación, esto es,

$$X \bullet B = (X \oplus B) \ominus B. \quad (2.15)$$

En la Figura 2.14, se puede apreciar la aplicación de la operación de cierre y sus usos, donde se observa que el resultado final ((2.15).c) es un objeto en el que los hoyos y separaciones fueron llenados para formar el objeto que se desea captar.

0 0 0 0 0 0 0 0 0 0 0 0	0 0 1 1 1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0	0 1 1 1 1 1 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 0 0 0 0	0 1 1 1 1 1 1 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0	0 1 1 1 1 1 1 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0	0 1 1 1 1 1 1 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0	0 1 1 1 1 1 1 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0 0 0	0 1 1 1 1 1 1 1 1 0 0 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 1 0 1 1 1 0 0 0	0 1 1 1 1 1 1 1 1 1 1 0	0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0	0 0 0 1 1 1 1 1 1 1 1 0	0 0 0 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1 1 1 1 0	0 0 0 0 0 0 1 1 1 1 1 0

a)

b)

c)

Figura 2.14: Operación de cierre. a) Imagen de entrada. b) Imagen dilatada. c) Imagen erosionada.

2.8.4. Etiquetado de objetos

El etiquetado de objetos se refiere a decidir qué píxeles corresponden a qué objetos en una imagen binaria, para diferenciar un artefacto de otro [11]. Un ejemplo se muestra en la Figura 2.15.a, donde existen tres objetos diferentes en una imagen binaria de entrada. Después de aplicar un algoritmo de etiquetado, los diferentes objetos se pueden identificar uno de otro, tal y como se observa en la Figura 2.15.b.

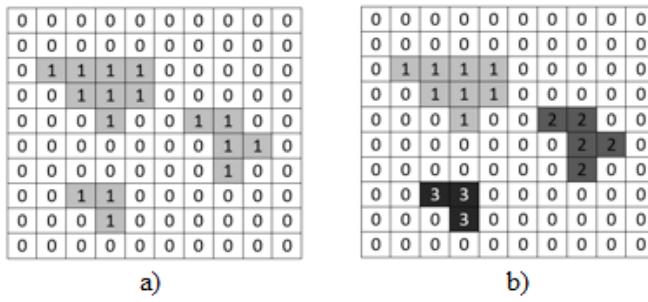


Figura 2.15: a) Objetos sin etiqueta en una imagen binaria. b) Objetos etiquetados.

Existen muchos algoritmos de etiquetado, algunos de ellos con o sin propagación en la búsqueda de objetos [13]. El algoritmo que se utiliza en el presente trabajo es el *Depth First Search* (DFS), el cual tiene muchas aplicaciones, como por ejemplo: encontrar componentes conexos, generar laberintos, solucionar *puzzles*, etc. [59]. Este algoritmo fue seleccionado en el presente trabajo debido a que se puede implementar de manera rápida.

El algoritmo DFS es relativamente sencillo. Lo que hace es recorrer los vértices de un árbol (en este caso los píxeles de una imagen) y decide si el píxel pertenece al objeto actual o no (dependiendo si el píxel se encuentra dentro de la vecindad) [3]. A continuación se explicará un ejemplo en el cual se considera una vecindad de 8.

El proceso de etiquetado se inicia recorriendo la imagen desde el píxel $p(0,0)$, hasta encontrar un $p(x,y) \neq 0$. Una vez que encuentre uno, se inicia la propagación.

Esta consiste en establecer como *padre* al píxel analizado y asignándole la etiqueta del primer objeto. Después se analiza la vecindad de dicho píxel y se le asigna la misma etiqueta a todos los píxeles $p(x, y) \neq 0$ (denominados *hijos*), tal y como se ve en la Figura 2.16.a. Todos las direcciones de los padres que tengan hijos se van guardando en una pila (*stack*) para su posterior uso. Una vez que la vecindad del píxel fue analizada, se procede al análisis de uno de los hijos. La elección del siguiente píxel a analizar dependerá del programador, el cual puede establecer las reglas para el siguiente movimiento (en este ejemplo se usa el último píxel encontrado de izquierda a derecha y de arriba hacia abajo). Al avanzar hacia uno de los hijos, se repite el proceso mencionado anteriormente, donde ahora el hijo analizado pasa a ser un nuevo padre (Figura 2.16.b). En esta imagen se observa que el nuevo padre tiene otra vez 3 hijos, ya que los otros píxeles en su vecindad ya fueron etiquetados. En las Figuras 2.16.c y 2.16.d se sigue recorriendo el objeto con un solo hijo para cada nuevo padre. En la Figura 2.16.e, el nuevo padre no tiene ningún hijo, lo que representa un final de recorrido. Lo que se hace al llegar a uno de estos finales es regresar por el camino tomado, donde se hace uso de la pila anteriormente mencionada. Cada vez que se utiliza la pila, se elimina la dirección del píxel padre, siempre y cuando ya no tenga hijos. El camino de regreso dura hasta que se llega a un píxel que tenía más de un hijo. Al llegar a dicho píxel, se analiza la vecindad de los demás hijos y la propagación continúa (si es que hay nuevos hijos sin etiquetar). El proceso de propagación y regreso se repite hasta que ya no existan píxeles en la pila, lo que significa que ya no hay píxeles sin etiquetar para ese objeto. Al terminar la propagación para un objeto, y estar de regreso al punto de inicio, se sigue analizando la imagen hasta encontrar un nuevo píxel $p(x, y) \neq 0$, para con ello volver a repetir la propagación del nuevo objeto, aunque con diferente etiqueta. El resultado del etiquetado se muestra en la Figura 2.16.f [3].

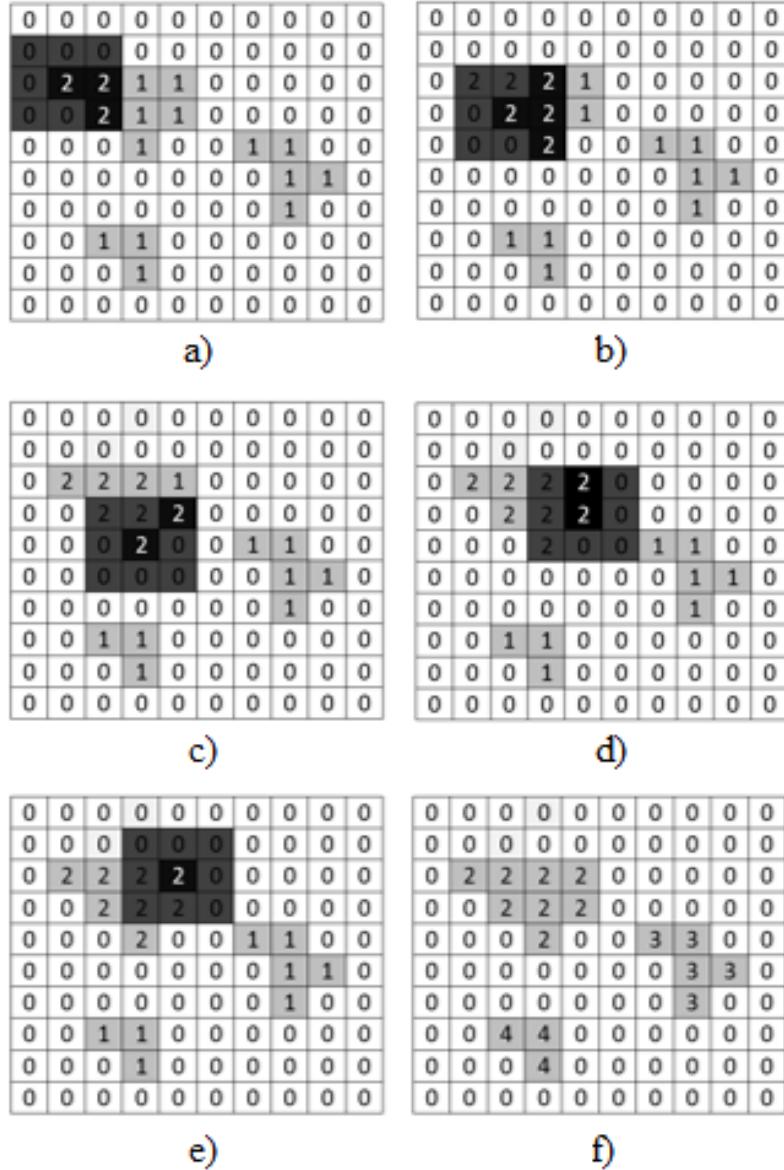


Figura 2.16: Aplicación del algoritmo DFS en una imagen binaria con vecindad 8.

El algoritmo DFS recorre por lo menos una vez todos los píxeles de una imagen.

El problema con este algoritmo es que tiene un alto coste computacional, por lo que se recomienda utilizarlo en imágenes pequeñas o en áreas reducidas de las mismas.

2.8.5. Características de objetos binarios

Las características de objetos binarios son “determinadas medidas cuantitativas, las cuales pueden ser obtenidas por el cálculo directo de los píxeles que lo componen” [11]. Existen muchos tipos de características, de las cuales solo se explicarán dos: la caja envolvente (también conocida como *bounding box*) y el centroide.

2.8.5.1. Caja envolvente

La caja envolvente de un objeto se define como el área rectangular mínima que lo contiene [11]. En la Figura 2.17 se muestra la caja envolvente (línea punteada) de un objeto (letra V). La caja envolvente esta dada por

$$CE(O) = \{(x, y) \in Img : y_{min} < y < y_{max} \quad si \quad x = x_{min} \quad o \quad x = x_{max}, \quad (2.16)$$
$$x_{min} < x < x_{max} \quad si \quad y = y_{min} \quad o \quad y = y_{max}\}.$$

donde CE representa a la Caja Envolvente, O es el objeto, (x, y) son los pares de coordenadas pertenecientes a la Imagen Img ; $x_{min}, x_{max}, y_{min}, y_{max}$ son las coordenadas de los límites de la caja.



Figura 2.17: Caja envolvente de un objeto.

2.8.5.2. Centroide

El centroide (o centro de masa) $\bar{C} = (\bar{x}, \bar{y})$ de un objeto, es el punto de concentración de su masa y se representa como “el punto medio aritmético en las coordenadas (\bar{x}, \bar{y}) ” [11], dado por,

$$\bar{x} = \frac{1}{Area(O)} \sum_{(x,y) \in O} x \quad , \quad \bar{y} = \frac{1}{Area(O)} \sum_{(x,y) \in O} y. \quad (2.17)$$

Con todo lo mencionado hasta ahora, se puede representar un robot como un objeto de una imagen binaria, donde su centroide equivale a la posición del mismo dentro de un plano coordenado. Por lo tanto, es posible utilizar una cámara como sensor de posición de múltiples robots, con la finalidad de retroalimentar ésta información a los mismos.

En la siguiente sección se introducen los conceptos relacionados con robótica.

Capítulo 3

Robótica móvil

En este capítulo, se presentan los conceptos relacionados con la robótica móvil, tales como clasificaciones de los robots y sus tipos de locomoción.

3.1. Introducción

Con los avances de la tecnología, el uso de robots se ha ido propagando cada vez más debido a su versatilidad. Estos realizan tareas como el ensamblado de automóviles, manipulación de objetos en altas temperaturas, exploración de minas e incluso limpieza de hogares. Algunos de los robots, como los utilizados generalmente en la industria automotriz (manipuladores), se mantienen fijos en un punto, por lo que su área de trabajo se reduce al alcance del mismo. Debido a esto, se diseñaron robots que tuvieran movilidad en su entorno para que fuera posible trabajar en un mayor espacio. A esto se le conoce como *robótica móvil* [43].

3.2. Clasificación de robots móviles

Los robots móviles, dependiendo del tipo de locomoción, se clasifican tal y como se observa en el Cuadro 3.1.

Los robots de interés en el presente trabajo son los Robots Móviles con Ruedas

Robótica móvil	
Clasificación	Descripción
<i>Robots terrestres</i>	Son los que se desplazan por el suelo, generalmente mediante ruedas o patas; tienen aplicaciones en rastreo y traslado de objetos, evasión de obstáculos, limpiez del área del hogar, etc.
<i>Robots aéreos</i>	Son los vehículos aéreos no tripulados (UAV's) como por ejemplo pequeños helicópteros, aviones o drones; se utilizan para reconocimiento de terrenos y superficies, etc.
<i>Robots subacuáticos</i>	Son submarinos equipados con sensores especiales para navegar dentro del agua; tienen aplicaciones de búsqueda, reparación, etc.

Cuadro 3.1: Clasificación de los robots móviles [43].

(RMcR). Esto, debido a que el tamaño de algunos de ellos es adecuado para pruebas de experimentación (un tamaño pequeño es ideal para hacer pruebas en muy diversos lugares al no requerir una gran área para la movilización de las unidades a utilizar), así como el relativamente bajo precio de cada robot (4.1.3). Sin embargo, pertenecen a una clase de sistemas mecánicos caracterizados por restricciones cinemáticas no holonómicas [43]. Por lo tanto, la dirección y la orientación no pueden ser estabilizadas simultáneamente [25]. Cabe aclarar que un sistema no holonómico es aquel que no puede cambiar inmediatamente de dirección, ya que necesita de rotación previa [15]. Dicho sistema corresponde a una clasificación de los robots móviles por su tipo de movimiento, la cual se observa en el Cuadro 3.2.

Robots móviles	
Tipo	Descripción
<i>Holonómico</i>	Capaces de cambiar de dirección inmediatamente. Por ejemplo, robots móviles de ruedas omnidireccionales.
<i>No holonómicos</i>	Necesitan una serie de movimientos para cambiar de dirección. Pueden utilizar ruedas fijas, de centro orientable, etc.

Cuadro 3.2: Clasificación de robots móviles por su tipo de movimiento [43].

3.3. Ruedas

Las ruedas permiten a los RMcR realizar su movimiento, aunque también son las responsables de imponer restricciones de movilidad [43]. Las ruedas utilizadas pueden ser:

- Ruedas fijas.
- Ruedas de centro orientable.
- Ruedas de centro orientable desplazado.
- Ruedas omnidireccionales.

Las ruedas de interés son las *fijas*, las cuales se encuentran ancladas sobre un eje, por lo que el único movimiento posible es el de rotación. En la Figura 3.1, se ve un ejemplo de una rueda fija de un robot móvil.



Figura 3.1: Robot móvil con rueda fija.

3.4. Postura de un robot móvil

En los algoritmos de control basados en el modelo cinemático (movimiento del robot sin tomar en cuenta las fuerzas que lo producen) de los RMcR, es necesario conocer la *postura* de los mismos. La postura de un robot móvil se puede definir como la ubicación y orientación del marco de referencia local (no inercial y unido al robot) con respecto al marco de referencia global (marco inercial) [43]. Dicha postura está compuesta por tres variables escalares (x, y, θ), donde y y x definen la posición del robot y el ángulo θ describe su orientación, todas las variables con respecto al marco de referencia global. En la Figura 3.2, se muestra un ejemplo de la postura de un RMcR.

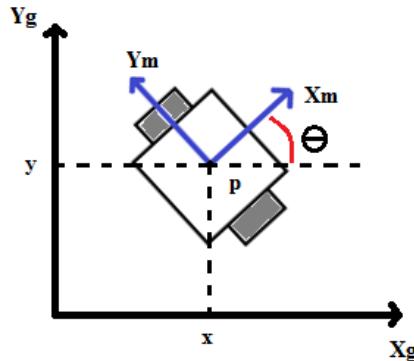


Figura 3.2: Postura de un RMcR con direccionamiento diferencial.

3.5. Configuración de robots móviles

Los robots móviles pueden tener muchos tipos de configuraciones con respecto a los sistemas de locomoción [43], algunos de los más comunes son:

- *Configuración Ackerman.*- Cuenta con cuatro ruedas que le proporcionan buena estabilidad al robot. Solo dos de ellas son motrices (Figura 3.3.a).
- *Configuración delta.*- Cuenta con tres ruedas en la que una sirve de direccionamiento, como un triciclo (Figura 3.3.b).
- *Direccionamiento diferencial.*- Cuenta con dos ruedas, como en la Figura 3.2, que le permiten girar sobre su propio eje y avanzar en línea recta. En ocasiones se le añade una o dos ruedas para mantener el equilibrio.
- *Configuración síncrona.*- En esta configuración, las ruedas del robot siempre apuntan a la misma dirección (Figura 3.3.c).

La configuración de interés es la de direccionamiento diferencial, ya que su sencillez facilita la implementación, lo cual los hace más económicos de adquirir en comparación a otras alternativas.

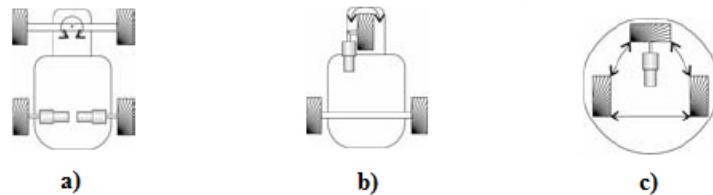


Figura 3.3: Algunos tipos de locomoción de RMcR [50]. a) Configuración Ackerman. b) Configuración delta (Triciclo clásico). c) Configuración síncrona.

3.6. Sistemas multi-robot

Apesar de las ventajas que traen consigo los RMcR, referente a su espacio de trabajo, existen tareas que son muy complicadas de realizar para un solo robot. Un ejemplo de ello sería el transporte de un cristal de grandes dimensiones, donde se requerirían diferentes puntos de apoyo para evitar fracturas por pandeos. Este tipo de problemáticas queda resuelta al utilizar más de un robot en una sola tarea. Entonces, se puede definir a un *Sistema multi-robot* (SM) como una agrupación de robots que trabajan en conjunto para llegar a una meta en común [53].

Capítulo 4

Descripción del *Hardware* y *Software*

En el presente capítulo, se describe el equipo seleccionado y el *Software* utilizado para el desarrollo del programa de detección, así como la justificación de dichas preferencias.

4.1. Descripción del *Hardware*

El *Hardware*, utilizado para el programa de detección, consiste en un dispositivo de captura de imágenes, un equipo de cómputo (para el procesamiento de imágenes y transmisión de datos) y en un grupo de RMcR E-puck.

4.1.1. Dispositivo de captura de imágenes

Los dispositivos de captura de imágenes (DCI) proporcionan la entrada de información de un sistema de PDI. Estos dispositivos, como sugiere su nombre, captan las intensidades de luz del entorno para su digitalización (formación de una imagen). Entonces, esta imagen es recibida por el equipo, al que se encuentra conectado [11]. El proceso de digitalización no se considera relevante para la comprensión del presente trabajo, por lo que es omitido.

En la presente tesis se utilizan cámaras web USB como DCI por su bajo costo y su alta disponibilidad en el mercado. Las cámaras web utilizadas son:

Cámara web *Logitech C210*.- Es una cámara web de gama baja (ver Figura 4.1) [27]. Sus especificaciones son las siguientes:

- Captura de video (640x480 píxeles).
- Videoconferencias (640x480 píxeles).
- Fotos: Hasta 1.3 Megapíxeles
(Mejora por *Software*).
- Micrófono integrado.
- Certificación USB 2.0.
- Clip universal para monitores.



Figura 4.1: Cámara web *Logitech C210* [27].

Cámara web *Logitech C525*.- Es una cámara web de gama media (ver Figura 4.2) [28]. Sus especificaciones son mostradas a continuación:

- Captura de video (1280x720 píxeles).
- Videoconferencias (1280x720 píxeles).
- Autoenfoque.
- Fotos: Hasta 8 Megapíxeles (Mejora por *Software*).
- Micrófono integrado.
- Certificación USB 2.0.
- Clip universal para monitores.



Figura 4.2: Cámara web *Logitech C525* [28].

4.1.2. Equipo de cómputo

El Equipo de Cómputo (EC) es el que se encarga de realizar los cálculos para el procesamiento de la imagen [45]. Normalmente, este tipo de procesamientos suele ser una gran carga para los EC, aunque con los rápidos avances tecnológicos de hoy en día, las computadoras portátiles tienen los requerimientos suficientes para

poder realizar dichos procesamientos sin muchos problemas. Dicho esto, se aclara que no se eligió un equipo de cómputo que cumpliera con ciertas características para la implementación del programa de detección.

El EC disponible es una computadora portátil cuyas especificaciones se muestran a continuación:

Computadora portátil marca *Samsung*, modelo NP550P5C-A01UB.-

Es una computadora que salió al mercado en el 2013 (ver Figura ??pcimg))[47].

Contiene las siguientes especificaciones:

- Procesador Intel Core i5-3210M (Doble núcleo a 2.5Ghz).
- 6Gb de memoria RAM.
- 750Gb de capacidad de disco duro.
- Chip gráfico intel HD4000.
- Bluetooth 4.0 incluído.



Figura 4.3: Computadora portátil *Samsung* [47].

4.1.3. Robot E-puck

Para demostrar que el programa de detección se puede aplicar para el control de múltiples robots simultáneamente, se hace uso de los robots E-puck, los cuales son RMcR con direccionamiento diferencial (Sección 3.5).

4.1.3.1. Origen del robot

Los orígenes del robot E-puck se remontan al año 2004, cuando el Dr. Francesco Mondada y Michael Bonani, del Instituto Federal Suizo de Tecnología en Lausanne (EPFL), comenzaron a diseñar un robot con propósitos de enseñanza a nivel universidad. Su motivación fue el hecho de que los robots existentes eran demasiado caros, por lo que quisieron diseñar un robot que fuera versátil, pequeño y económico. El resultado fue el robot E-puck (Figura 4.4), cuyo precio ronda los 450 euros. Aunque la finalidad era un robot con propósitos de enseñanza, el robot ha adquirido tal popularidad que es utilizado en áreas de investigación [35].



Figura 4.4: Robot E-puck.

4.1.3.2. Componentes

Aquí se muestran los componentes del robot E-puck, desde la estructura mecánica, hasta los componentes electrónicos:

4.1.3.2.1. Estructura mecánica La parte mecánica del robot E-puck consta de un chasis de plástico transparente que contiene todos los elementos, tanto los motores y llantas, como las placas electrónicas. Dicho chasis permite que el robot sea ligero. En la Figura 4.5, se muestra una vista explosionada de la estructura mecánica.

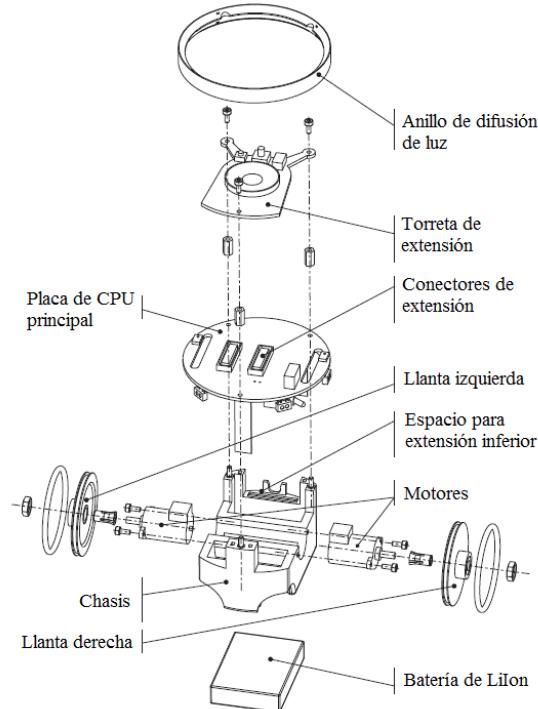


Figura 4.5: Vista explosionada de la estructura mecánica [35].

4.1.3.2.2. Microcontrolador El cerebro del E-puck se compone de un microcontrolador dsPIC30F6014a [31], el cual es de 16 bits. Sus características lo hacen adecuado para su aplicación en entornos universitarios, tal y como se menciona en [35]. En la Figura 4.6 se muestra el esquema de las conexiones electrónicas, donde se puede observar la gran cantidad de componentes disponibles. También, se muestra la gran capacidad que tiene el microcontrolador para administrar una gran cantidad de entradas y salidas.

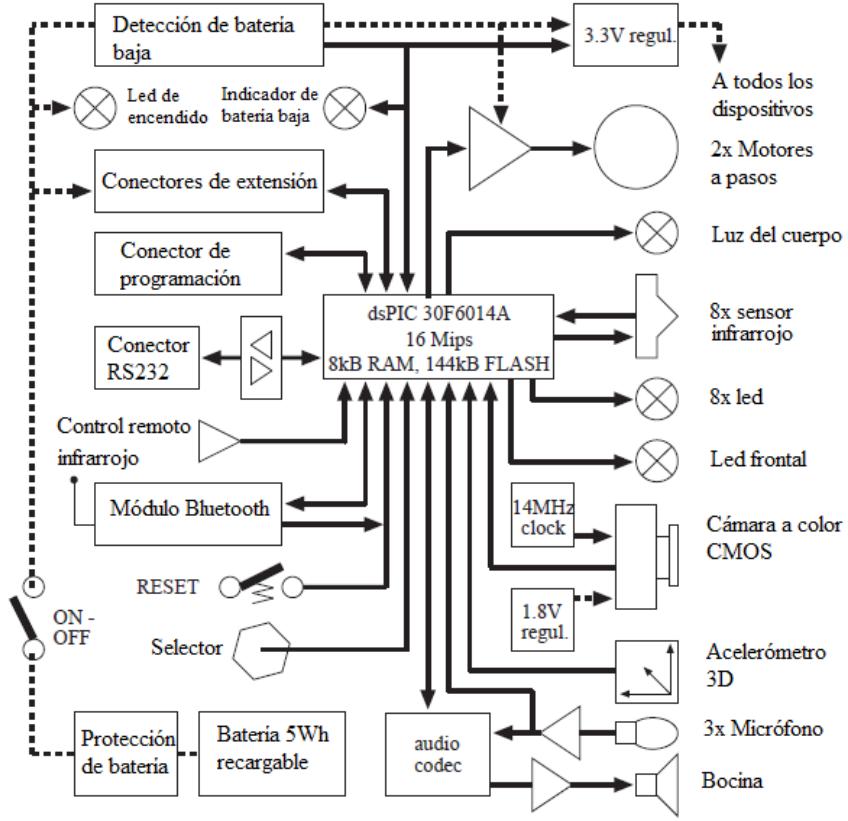


Figura 4.6: Esquema de conexiones del E-puck [35].

4.1.3.2.3. Sensores y actuadores El E-puck, aunque cuente con dimensiones pequeñas, tiene una gran cantidad de sensores y actuadores, lo que lo hace muy versátil. Con los dispositivos de sensado es posible detectar objetos del entorno para evitar colisiones, conocer la inclinación del robot o si está cayendo, conocer la fuente de algún sonido y tomar imágenes a color (de muy baja resolución debido a las limitaciones de ancho de banda [35]). Con los actuadores es posible hacer aplicaciones más visuales y llamativas, reproducir sonidos y, quizás lo más importante, mover al robot. Todos los sensores y actuadores disponibles en el E-puck proporcionan una amplia gama de aplicaciones posibles. Una de ellas, por ejemplo, se trata de un sistema de rehabilitación con ayuda de un robot E-puck [57]. En el Cuadro 4.1, se engloban los componentes incluídos.

E-puck	
Sensores	Actuadores
<ul style="list-style-type: none"> ● 8x Sensores infrarrojos ● Acelerómetro 3D ● 3x micrófonos ● Cámara a color CMOS 	<ul style="list-style-type: none"> ● Dos motores a pasos ● Una bocina ● 8x leds indicadores ● Leds verdes en el cuerpo ● Un led frontal

Cuadro 4.1: Sensores y actuadores del E-puck [35].

Apesar de la gran cantidad de componentes incluídos en el robot, es imposible utilizarlo en cualquier tipo de experimentación. Debido a esto, el E-puck es capaz de expandirse con ayuda de aditamentos extras, lo que permite adaptar al robot a muchas otras necesidades, ya que se puede dotar al robot con nuevas y/o mejores capacidades a las de la configuración original [35]. Las adaptaciones tienen un costo extra, por lo que se tiene que planear muy bien previamente la elección de tales aditamentos.

4.1.3.3. Comunicación

La carencia de algún método de comunicación en el robot E-puck lo volvería incapaz para ser utilizado en un sistema multirobot que requiera información proveniente del exterior (una computadora por ejemplo). Afortunadamente, este robot cuenta con un módulo *Bluetooth* para el envío y recepción de información.

El *Bluetooth* es un estándar de comunicación inalámbrica creado para eliminar los cables que transmitían información por medio de la comunicación serial. Debido a eso, el uso de módulos *Bluetooth* suele ser muy similar al uso de la comunicación serial. La primera versión del *Bluetooth* fue creada en 1994, y en la actualidad se encuentra en su versión 4.0 [57]. En el Cuadro 4.2 se presentan algunos datos técnicos del protocolo.

Estándar	Bluetooth
IEEE	802.15.1
Frecuencia de banda	2.4 Ghz
Máxima velocidad de señal	1 Mb/s
Rango nominal	10 m
Potencia TX nominal	0 - 10 dBm
Número de canales RF	79

Cuadro 4.2: Especificaciones técnicas del protocolo *Bluetooth* [57].

4.2. Descripción del *Software*

En la parte del software, se engloban todos los programas utilizados, tanto para el desarrollo del programa de detección, como para la programación del robot E-puck. La información acerca de los *links* de descarga e instalación de los programas utilizados, se encuentra en el Apéndice B.

4.2.1. Sistema operativo

El sistema operativo (SO), es el programa (o programas) que se encarga de la administración de recursos de un sistema informático [55]. En la actualidad, existe un gran número de SOs. Algunos de ellos requieren de una licencia de paga para su uso (como *Windows*), en cambio otros son de libre distribución (como *Ubuntu*). Cada uno ofrece ventajas y desventajas, donde el usuario final es el encargado de elegir la mejor oferta.

En la Figura 4.7, se muestra una estadística actualizada (junio 2015) de la participación de los SOs en el mercado, proveniente de [36]. En ella se observa que el SO *Windows 7* es el más utilizado a nivel global, seguido de *Windows 8.1*.

La selección del SO se hizo en base a la disponibilidad por parte del autor, por

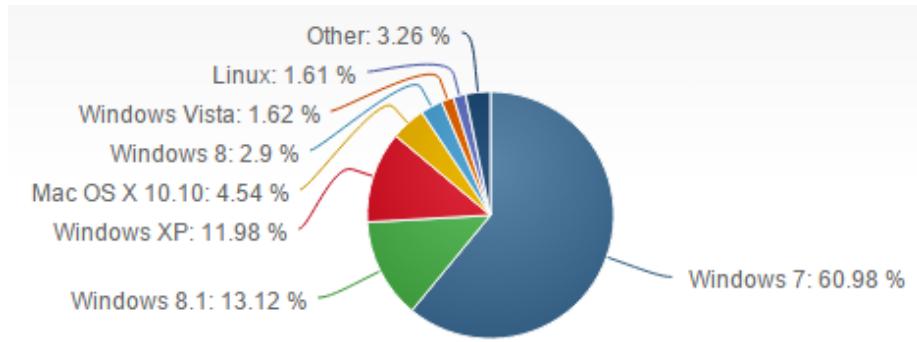


Figura 4.7: Participación de los SOs en el mercado [36].

lo que el programa de detección es desarrollado en *Windows 8.1* y se comprueba su reproducibilidad en el SO *Windows 7*.

4.2.2. Lenguaje de programación para el programa de detección

Para el desarrollo del *Software*, existían varias alternativas de lenguajes de alto nivel y compatibles con las librerías de OpenCV (las cuales se describen más adelante): C++, Python y Java. Las tres opciones son capaces de proveer herramientas para el desarrollo de programas profesionales, además de ser de libre distribución. El lenguaje de programación seleccionado fue el Python, ya que ofrece facilidad de uso (se considera entre la comunidad de programadores como uno de los lenguajes más sencillos de aprender).

Este lenguaje de programación fue creado en su mayoría por Guido Van Rossum, apareciendo por primera vez en 1991. Su filosofía se basa en hacer el código más entendible y fácil de leer, lo cual se logra con la reducción de signos de puntuación para delimitar comandos, usando en su lugar tabulaciones [42]. Con el tiempo fueron apareciendo más versiones de Python: Python 2.x y Python 3.x. La versión 2.x (actualmente la 2.7), es la que tiene mayor tiempo en desarrollo, y por ende, mayor

número de librerías, por lo que fue la opción a elegir.

4.2.3. Librerías para Python

El uso de *librerías* permiten implementar código optimizado [42]. Las librerías utilizadas en el presente proyecto, se muestran a continuación:

- *Pyserial 2.7*. Contiene comandos para accesar de manera sencilla al puerto serial, el cual se utiliza para comunicar la computadora con el robot E-puck (con ayuda de los módulos *Bluetooth*) [40].
- *NumPy 1.9*. Es un paquete que permite trabajar con arreglos matriciales, el cual es de mucha utilidad con el álgebra lineal. Debido a eso, se utiliza mucho en PDI (Es un requerimiento para OpenCV) [48].
- *OpenCV 2.4*. La librería de Visión Computacional de Código Abierto (por sus siglas en inglés de Open Source Computer Vision), es ampliamente usada en el área de PDI. Contiene una gran cantidad de algoritmos optimizados para un rápido procesamiento y una mejor implementación [38].

4.2.4. Lenguaje de programación para el robot E-puck

Los robots E-puck cuentan con una amplia variedad de lenguajes de programación para su uso [33], los cuales se enlistan enseguida:

- *Aseba*. El Aseba es una arquitectura basada en eventos para el control de robots móviles, aunque no es muy conocido [30].

- *Matlab*. Matlab es un programa muy utilizado en áreas de investigación, ya que sus kits de herramientas son aplicables en muchos segmentos. En el caso de los E-puck, matlab realiza los cálculos del algoritmo de control y le transmite la información necesaria al robot. Requiere licencia para su uso [33].
- *Python*. Ofrece un lenguaje amigable y de libre distribución. Para el control de los E-puck, la computadora se hace cargo de la carga computacional y de dar las órdenes [41].
- *Ensamblador*. Es un lenguaje de bajo nivel, que permite tener un completo dominio sobre las direcciones de memoria del microcontrolador del E-puck, para un programa con control de tiempos más eficiente. Debido a que es de bajo nivel, inclusive los programas sencillos suelen ser muy largos y difíciles de leer. Con él es posible darle autonomía a los robots (el robot hace los cálculos) [17].
- *Lenguaje C*. Lenguaje de medio nivel, que lo hace más sencillo de entender en comparación con el lenguaje ensamblador. Con él es posible otorgarle autonomía a los robots [33].

Como se puede ver en las descripciones, el lenguaje C es el más apto para la programación de los robots E-puck (de acuerdo a los objetivos de la tesis). Este lenguaje, aparte de ser relativamente sencillo de interpretar, permite que los robots cuenten con la autonomía necesaria para que tomen sus propias decisiones. Los programas utilizados para la programación del microcontrolador del E-puck se describen en el Apéndice B.

Una vez descrito el *Hardware* y el entorno de programación a utilizar, se procede a explicar la solución propuesta en el siguiente capítulo.

Capítulo 5

Programa de detección

En el presente capítulo, se explica la estructura que compone al programa de detección. También, se muestra la interfaz gráfica de usuario (GUI, por sus siglas en inglés de *Graphics User Interface*) implementada para acceder a cada una de las funcionalidades del programa. Por último, se describe cada uno de los módulos incluidos, explicando los algoritmos utilizados en cada uno de ellos.

5.1. Introducción

El programa de detección del presente trabajo de tesis debe ser capaz de detectar múltiples robots teniendo un cierto grado de robustez (bajo ciertas condiciones de iluminación y de desenfoque), el cual tiene que ser de bajo costo y reproducible. Para ello, se hace uso del lenguaje Python en su versión 2.7, debido a que es compatible con las librerías de OpenCV. Con esto, el programa de detección está basado completamente en *Software libre*.

5.2. Estructura del programa

El programa de detección está conformado por cinco módulos principales. Para acceder a cada uno de estos módulos, el usuario interactúa con una GUI muy sencilla. En la Figura 5.1 se muestra un esquema de interacción entre el programa y el usuario.

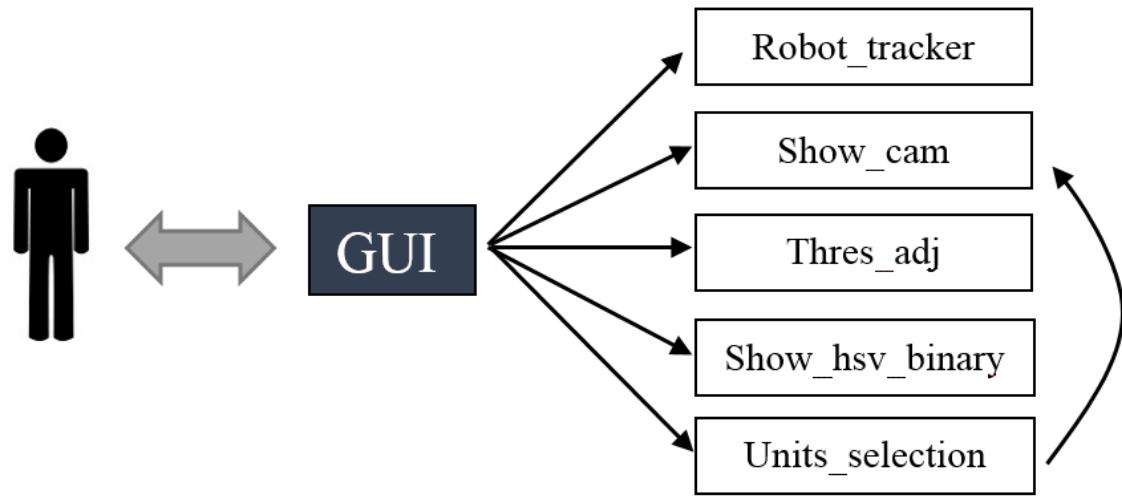


Figura 5.1: Esquema de interacción entre usuario y programa.

El módulo principal (`Robot_tracker`) es el encargado de realizar la detección de los marcadores, calcular la posición, la orientación y transmitir la información a los robots. Los demás módulos se crearon para facilitar otras tareas secundarias, pero que pueden ser útiles para llevar a cabo la tarea principal, como por ejemplo: Mostrar la imagen de la cámara, ajustar los valores de los umbrales, ver el resultado de la umbralización o calibrar las unidades de medición (parcialmente implementado). Cabe recalcar que el módulo `Units_selection` llama a otro de los módulos principales, por lo que existe interacción entre módulos. Más adelante se verá ésto a detalle.

Por otra parte, se tiene que aclarar que varios de los módulos del programa utilizan un archivo de texto para facilitar al usuario la puesta en marcha de un experimento. Dicho archivo contiene las configuraciones de los umbrales para la detección

de los marcadores, así como información para la calibración del programa. El archivo de texto no es necesario para que el programa funcione, aunque se recomienda su presencia para que todos los módulos trabajen correctamente. En la Figura 5.2, se logra ver la interacción de los módulos con el archivo de texto, donde es posible notar que algunos módulos pueden leer y escribir sobre dicho archivo, mientras que el módulo *show_hsv_binary* solamente puede leer.

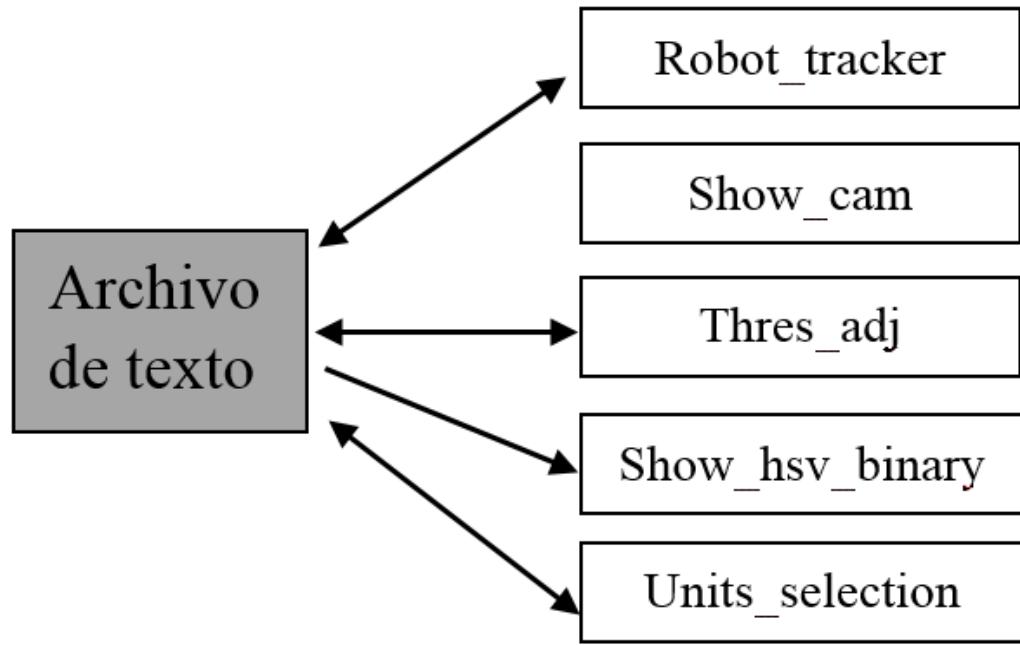


Figura 5.2: Interacción de los módulos con el archivo de texto.

En la Figura 5.3, se muestra la GUI implementada, la cual consta de una ventana que contiene cinco botones que el usuario puede pulsar para la inicialización de alguno de los módulos principales. Al hacer clic en alguno de los botones, este manda llamar a uno de los módulos principales, apareciendo así las ventanas correspondientes de dicho módulo. Cada módulo principal cuenta con diferentes funcionalidades, las cuales se explicarán más adelante.

La información para acceder a la interfaz, así como el correcto uso de la misma, se encuentra en los apéndices.

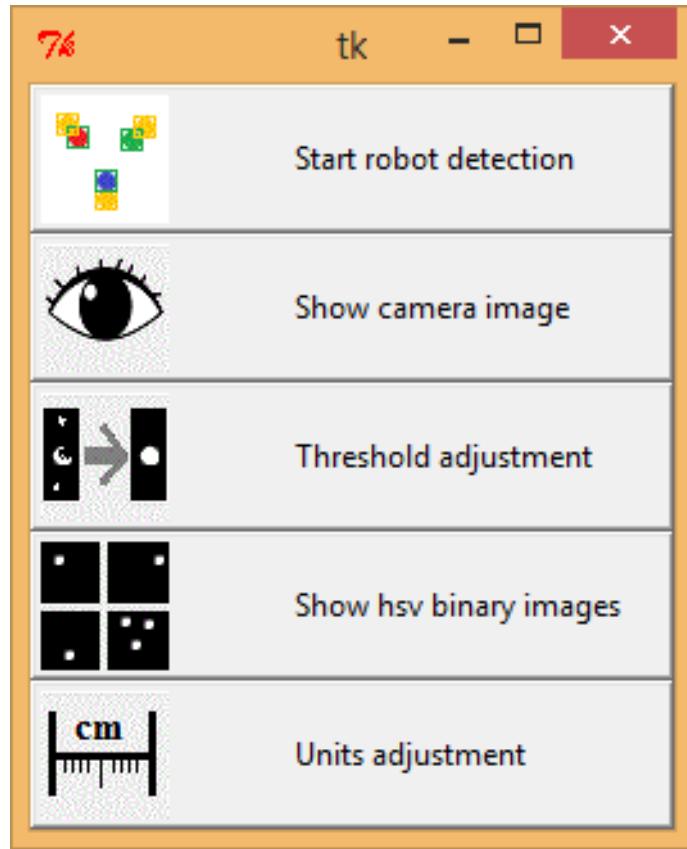


Figura 5.3: Interfaz gráfica de usuario.

Antes de comenzar la explicación de los algoritmos, se tiene que conocer qué es lo que se quiere detectar, por lo que a continuación se propone un marcador para la detección de los robots E-puck.

5.2.1. Marcadores

Un *marcador* es una figura que se utiliza generalmente en PDI para el rastreo de objetos. En la presente tesis se proponen marcadores circulares para la fácil detección de su centro de masa. Cada marcador se compone de dos círculos de color dentro de otro blanco. La medida de los círculos pequeños está dada por $d = \frac{D}{3}$, donde $D = 7\text{cm}$ (diámetro del E-puck), por lo que $d = \frac{7}{3} \approx 2,33\text{cm}$. Uno de los círculos pequeños tiene que ser concéntrico al círculo envolvente y debe de ser de un color

diferente al de los demás marcadores. El otro círculo pequeño es tangencial a las dos circunferencias restantes y su color tiene que ser igual al círculo correspondiente de los demás marcadores (en este trabajo se usa un color amarillo). Los marcadores mencionados se muestran en la Figura 5.4, donde los círculos centrales son de los color rojo, verde y azul, respectivamente.

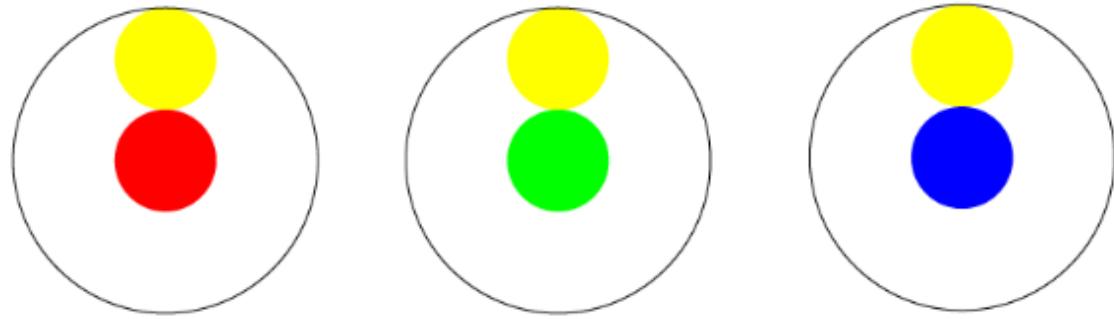


Figura 5.4: Marcadores propuestos.

Ahora, teniendo en cuenta los marcadores a detectar y sabiendo como puede un usuario acceder a cada uno de los módulos principales, se explicará cómo funciona cada uno de ellos, comenzando con el algoritmo más importante del programa creado, el módulo *Robot_tracker*.

5.2.2. Módulo de detección de robots (*Robot_tracker*)

El módulo de detección es el más importante de todos, ya que es el que realiza el objetivo del presente trabajo de tesis. En la Figura 5.5, se muestra el diagrama de flujo general de dicho módulo, el cual se compone de cuatro partes principales:

- Inicialización.
- Preprocesamiento.
- Procesamiento.
- Entrega de resultados.

Cada una de estas partes será explicada a detalle a continuación.

5.2.2.1. Inicialización

La inicialización, como se observa en la Figura 5.5, se refiere a los ajustes necesarios para que el programa inicie sus operaciones correctamente. Estas incluyen la configuración del puerto serial, la lectura del archivo de texto y la inicialización de variables globales.

En la *configuración del puerto serial*, se establecen los valores de los puertos COM correspondiente a cada robot E-puck, así como su velocidad de transmisión de datos y el tiempo de espera de respuesta (*timeout*). Para establecer las configuraciones se tiene que importar la librería serial (*Pyserial*). Los valores de la configuración de un puerto serial, para uno de los robots E-puck utilizados, se muestra en el Cuadro 5.1.

Puerto serial	
Configuración	valor
Puerto COM	9
Velocidad (baudios)	115200
<i>timeout</i>	0

Cuadro 5.1: Configuraciones del puerto serial para un robot.

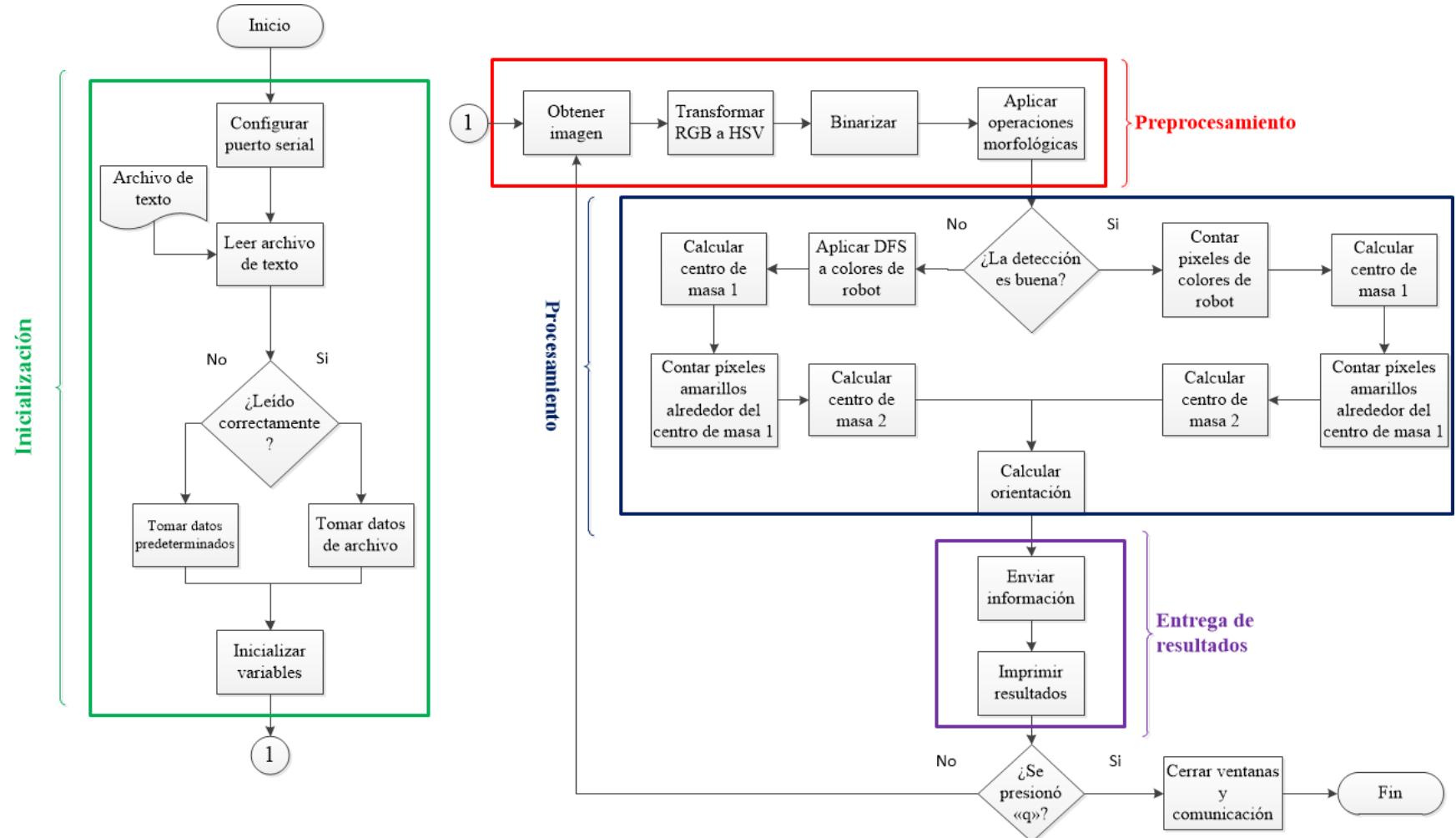


Figura 5.5: Diagrama de flujo del módulo de detección.

La razón de la selección de dichos valores se encuentra en los apéndices. Hay que tener en cuenta de que dichos valores se escriben en el código (aún no existe modularidad en esta parte).

En caso de que no se encuentre alguno de los robots, el programa arrojará un error debido a una conexión fallida. Dicho error no permitirá que el módulo se ejecute, por lo que se tiene que asegurar que todos los robots inicializados sean encontrados por el *bluetooth*. En caso de que no se ocupe alguno de ellos, es necesario comentar las líneas correspondientes a la comunicación del robot (los comentarios en python inician con el símbolo `#`), las cuales son:

- `ser = serial.Serial(9, 15200, timeout = 0)`. El cual se encuentra en la inicialización del programa, como ya se mencionó.
- `ser.write(string)`. Este comando se encuentra en la parte de entrega de resultados.
- `ser.close()`. Se encuentra al final del módulo (en el cierre).

Hay que notar que, estos comandos sirven para poder establecer comunicación entre la computadora y los robots E-puck. Más adelante se explicará la función de cada comando mencionado.

Después de la inicialización del puerto serial, se busca el archivo de texto, cuyo nombre se puede modificar por código al inicio del *script* de python (en la clase *default*). La clase *default* contiene todos los valores predeterminados en caso de que no se encuentre el archivo de texto buscado (o este se encuentre corrompido), a lo cual se mostrará un mensaje de error, pero sin afectar la ejecución del programa. La información que contiene la clase *Default* se encuentra en el Cuadro 5.2, donde se aclara que los valores de múltiplo y píxeles por centímetro están parcialmente

implementadas (Algunos ajustes son necesarios para su correcta implementación, aunque ello no afecta el funcionamiento del programa). La idea del uso de *múltiplo* es de escalar la imagen al valor escrito, para disminuir el tiempo de procesamiento, aunque se necesitan algunos ajustes para su correcto funcionamiento. En cuanto a *píxeles por centímetro*, lo que se busca es entregar el resultado en centímetros (actualmente se encuentra en píxeles). El uso de dichos valores se propone como trabajo futuro.

Clase <i>Default</i>
<ul style="list-style-type: none"> • Valores de umbral • Colores utilizados • Múltiplo • Nombre de archivo .txt • Coordenadas de puntos deseados • Píxeles por centímetro

Cuadro 5.2: Valores predeterminados en la clase *Default*

Una vez que se tienen los valores para la ejecución del programa (ya sea por haberlos tomado del archivo de texto o por la clase *Default*), se inicializan las variables globales.

Antes de entrar en el ciclo infinito, el cual abarca desde el preprocesamiento hasta la entrega de datos (Figura 5.5), se elige la cámara a utilizar y se crea la ventana en la cual se mostrarán los resultados de la detección. La ventana es creada desde el principio para poder hacer uso de eventos, como se explicará más adelante.

Finalmente, se procede a iniciar el ciclo infinito, donde se inicia la captura de imágenes y su preprocesamiento, los cuales se explican en la siguiente sección.

5.2.2.2. Preprocesamiento

El primer paso del preprocesamiento es obtener una captura de la cámara (con el comando *cv2.VideoCapture*), para así poder comenzar a trabajar sobre ella. De dicha captura, se extrae la información referente a su tamaño (atributos *read()* y *shape*)

para establecer las fronteras en operaciones posteriores. Lo que sigue es *transformar la imagen* del espacio de color RGB al espacio HSV (con el comando `cv2.cvtColor` y el argumento `cv2.COLOR_BGR2HSV`). Para ello, se hace uso de las librerías de OpenCV. En la Figura 5.6, se muestra la transformación de una captura de un espacio de color a otro. En esa imagen, es posible diferenciar cual es el fondo y cuales son los marcadores, gracias al contraste que se obtiene.

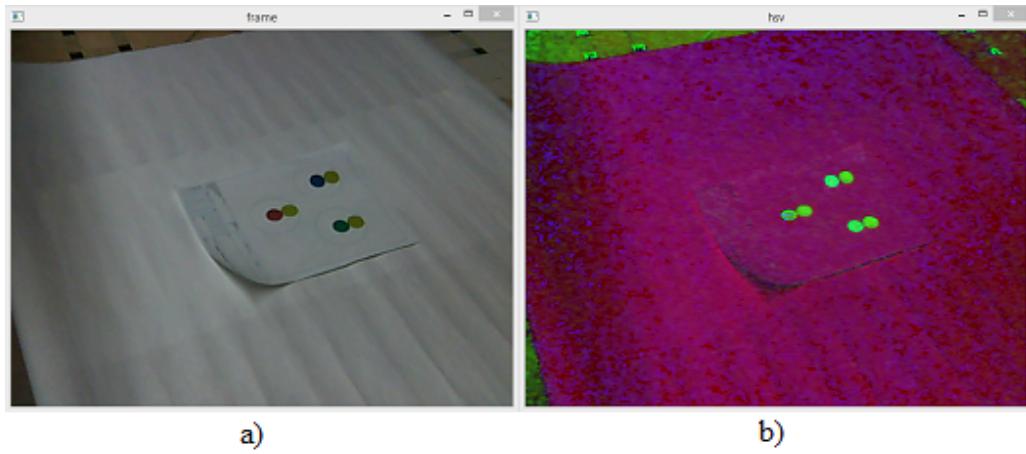


Figura 5.6: Transformación de RGB a HSV. a) Imagen RGB. b) Imagen HSV.

A partir de la imagen en espacio HSV, se realiza una binarización (con el comando `cv2.inRange`) aplicando los valores obtenidos en la lectura del archivo de texto. La imagen binarizada obtenida no siempre aísla los colores exactamente como se desea. Por ejemplo, en la Figura 5.7.a, se muestra el resultado de aislar el color rojo, donde se logran apreciar manchas blancas en otros lugares aparte del círculo, por lo que es necesario aplicar las *operaciones morfológicas*.

Se utilizan las operaciones morfológicas de apertura (comando `cv2.morphologyEx` y el argumento `cv2.MORPH_OPEN`) y después dilatación (utilizando el comando `cv2.dilate`) para limpiar la imagen. En la Figura 5.7.b, se aprecia el resultado de la aplicación de dichas operaciones. También, si se pone atención al contorno del objeto detectado, se notará que se ve un poco más fino.

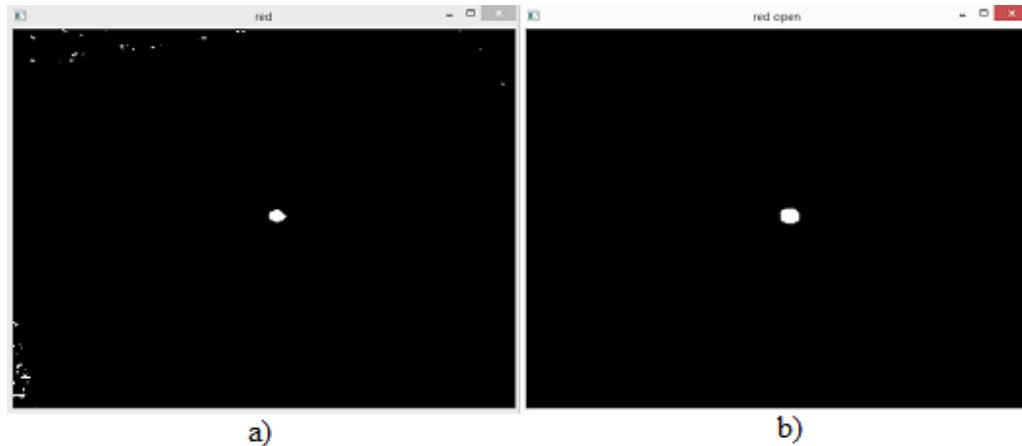


Figura 5.7: a) Imagen binarizada. b) Imagen al aplicarle operaciones morfológicas.

Obviamente, la aplicación de la binarización y las operaciones morfológicas se realiza a cada uno de los colores a detectar (en este caso rojo, azul, verde y amarillo). Además de eso, se hace uso de la operación lógica *or* (con el comando *cv2.bitwise_or*) aplicada a todos los colores mencionados anteriormente para obtener una imagen que englobe todas las detecciones (imagen global). Dicha imagen es utilizada más adelante.

Con esto se termina el preprocesamiento de la imagen, cuyos resultados sirven de base para extraer la información de interés al procesar la imagen, lo cual se verá en la siguiente sección.

5.2.2.3. Procesamiento

El procesamiento de la imagen inicia con la captura binarizada con todas las detecciones, la cual se observa en la Figura 5.8. Tal imagen es utilizada para conocer si la detección es correcta o no tomando en cuenta que, si la caja envolvente de la detección global es menor a un cuarto del tamaño total de dicha captura, se considera como correcta, y para el caso contrario, se considera como incorrecta. Si el usuario requiere cambiar este criterio de decisión, lo puede hacer por código modificando la

regla de los *if*, justo al inicio del procesamiento (donde se compara la caja envolvente con *height/4* y *width/4*).

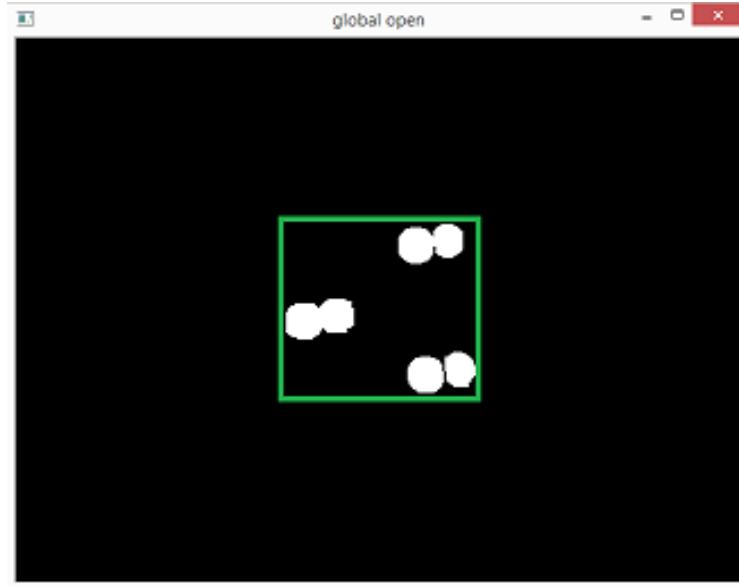


Figura 5.8: Imagen de detección global.

Si la detección es tomada como correcta, se procede con lo siguiente:

Detección correcta. Se cuentan los píxeles de cada uno de los colores que identifican a cada robot. En este caso son tres imágenes diferentes (rojo, verde y azul), tomando en cuenta los límites de la imagen global para no tener que escanear la captura completa (lo que requeriría más tiempo). Al contar los píxeles, también se extrae la información para la caja envolvente y la información necesaria para calcular el centro de masa (el cual se denominará por CM1). Toda esta información se guarda en una lista de clases, donde cada elemento de la lista contiene todos los valores guardados de cada uno de los colores. Ya con los píxeles contados, y con la información extraída, se calcula el CM1. Después, se procede a contar los píxeles amarillos dentro de un *offset* establecido por código (su valor predeterminado es de $(1,6)(d_{detectado}) : d = \text{diámetro}$). Una representación de ello, puede verse en la Figura 5.9.

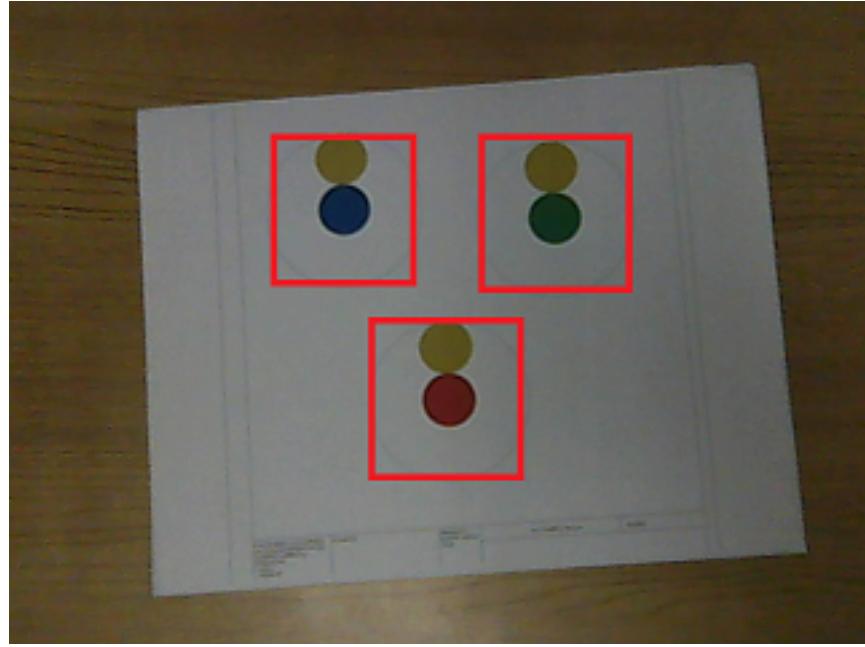


Figura 5.9: Representación del *offset* establecido para contar los píxeles amarillos.

Al contar los píxeles amarillos, la información de la caja envolvente y la necesaria para su centro de masa (que se denominará como CM2) se guarda en una lista, al igual que con los otros colores. Con la información obtenida se calcula el CM2.

Ahora se verá el otro caso, cuando la detección no es correcta:

Detección incorrecta. Si, por ejemplo, se supone que existe un error de detección solamente en la imagen binaria para detectar el color rojo, como se muestra en la Figura 5.10, el programa aplicará el algoritmo DFS para etiquetar todos los objetos detectados. Después de haber diferenciado los objetos, se procede a eliminar los objetos que estén fuera de un rango de píxeles equivalente al tamaño aproximado que debe de tener el círculo. Dicho rango puede modificarse a criterio del usuario en el código. En caso de que más de un objeto esté dentro del rango establecido, se toma el primer objeto detectado dentro de dicho rango.

Ya con la información de los círculos, se calcula el CM1. Para sacar la información de los círculos amarillos, se procede igual que como se mencionó anteriormente,

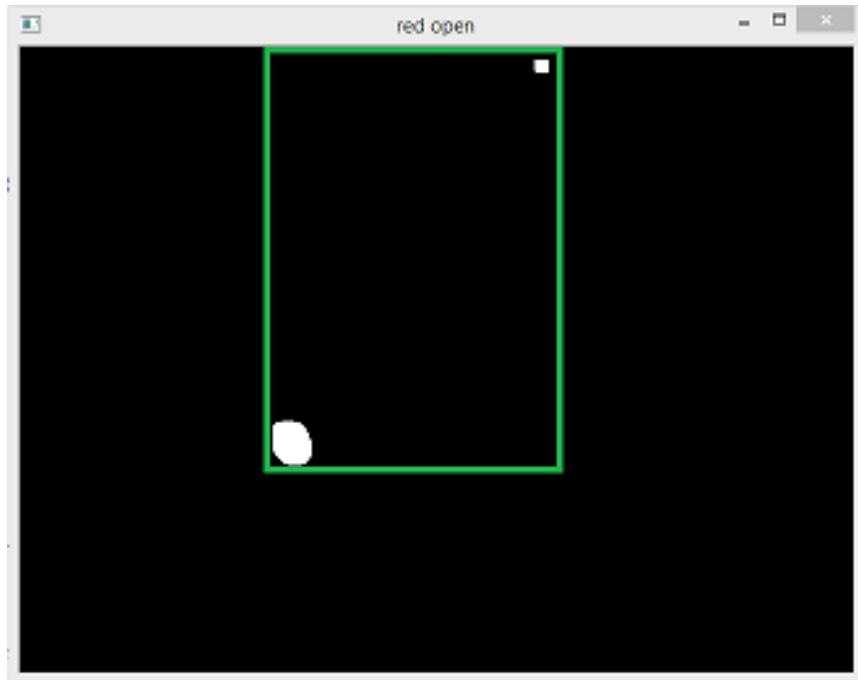


Figura 5.10: Detección incorrecta.

contando los píxeles dentro de un *offset* establecido en el código. Con esta información se calcula entonces el CM2.

Para decidir la correspondencia entre CM1 y CM2, se calcula la distancia euclíadiana entre cada uno de los centros de masa, donde la menor distancia indica la relación.

La ventaja de tener círculos para detectar, es que la caja envolvente siempre será un cuadrado (si la cámara se posiciona de manera ortogonal a los marcadores), por lo que si el marcador gira, el centro de masa se conservará igual. En cuanto a los círculos amarillos, se decidió ponerlos de esa manera para calcular la orientación del robot, con ayuda de sus centros de masa. Al tener los CM1 y CM2 se calcula la orientación como

$$\theta = \arctan \frac{\Delta y}{\Delta x}, \quad (5.1)$$

donde sustituyendo Δy y Δx por la diferencia de las coordenadas “y” y “x” respectivamente de ambos centros de masa, resulta

$$\theta = \arctan \frac{CM2_y - CM1_y}{CM2_x - CM1_x}, \quad (5.2)$$

con lo cual se obtiene la orientación de cada uno de los robots. En la Figura 5.11, se observa el ángulo calculado con respecto a los centros de masa.

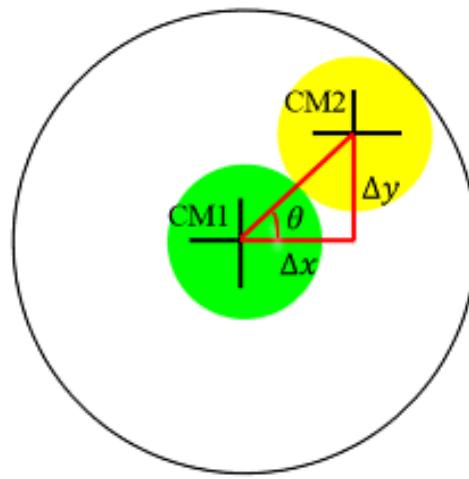


Figura 5.11: Ángulo θ en un marcador.

Al tener el ángulo calculado, termina la parte del procesamiento, ya que hasta este punto se tiene la descripción de los centros de masa (posición) y el ángulo (orientación) de los marcadores. Lo que hace falta es entregar la información obtenida, lo cual se explica en la siguiente sección.

5.2.2.4. Entrega de resultados

La entrega de resultados consiste en enviar la información obtenida en el procesamiento, y en mostrarla al usuario. Antes de mostrar dicha información, se establece un manejo de evento, con el cual se detecta si el usuario hace doble clic en la pantalla

que mostrará las capturas. Esto sirve para que el usuario elija un punto deseado (p_d) a donde quiera que se muevan los robots.

Antes de enviar la información a los robots, el ángulo detectado (radianes) se transforma a grados en el rango de $[0, 360)$ para su fácil manejo y entendimiento.

Finalmente, se crea un *string* con el siguiente formato:

$$\text{string} = p_x \ p_y \ \theta \ p_{dx} \ p_{dy},$$

donde p_y y p_x son las coordenadas del CM1 (que denota la posición del robot), θ es la orientación y p_{dx} y p_{dy} son las coordenadas del punto deseado que se obtiene al darle clic en la pantalla. El espacio en blanco entre los valores es necesario para diferenciar a los caracteres cuando el robot los reciba. El string es entonces enviado al robot correspondiente con el siguiente atributo de la clase *Serial*:

```
write(string)
```

Todos los resultados se imprimen también en una pantalla para la visualización del usuario, tal y como se ve en la Figura 5.12. En dicha Figura se observa un *límite de “seguridad”*, el cual está puesto para que el usuario sepa hasta donde es segura la detección de los robots. Si un robot sale de este límite, podría ser que el marcador de orientación quede total o parcialmente fuera de la imagen, por lo que el cálculo de la detección no sería correcto. Aún así, el programa no se detiene si uno de los marcadores ya no es detectado, solamente deja de enviar la información al robot.

En la imagen que se muestra al usuario, el punto deseado al hacer doble clic, se imprime como un círculo de color celeste. Con esto, el usuario puede conocer visualmente el punto final del recorrido. El punto deseado se puede modificar en cualquier momento (no afecta la ejecución del programa).

Después de la visualización de los resultados, se vuelve a repetir el ciclo, y el

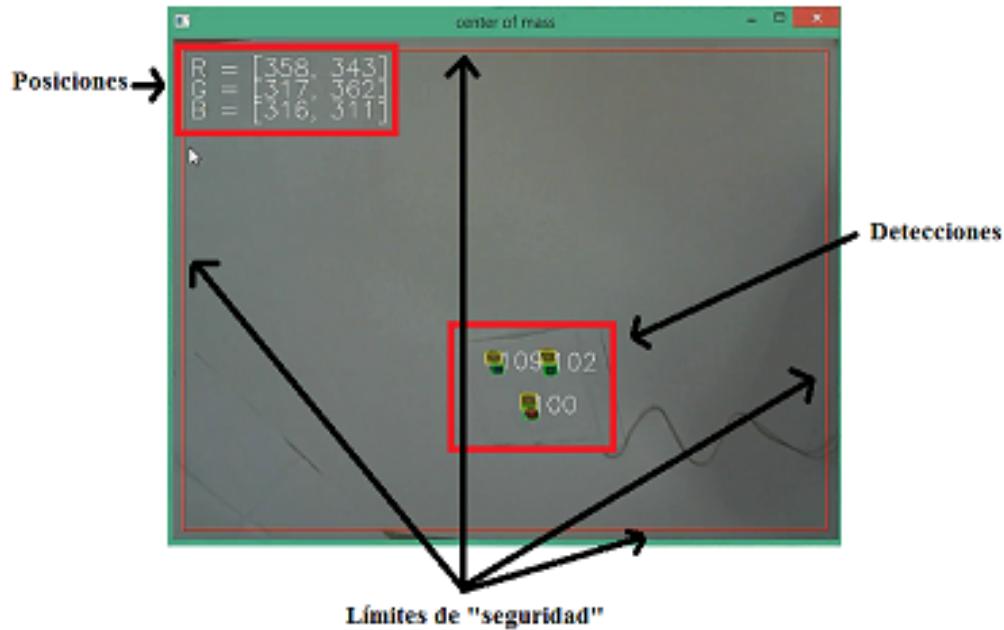


Figura 5.12: Resultados mostrados al usuario.

programa toma una nueva captura para iniciar de nueva cuenta el preprocesamiento. El ciclo se repetirá hasta que el usuario presione la tecla “q” (*quit*). Si el usuario presiona dicha tecla, el programa se detendrá, la ventana de visualización se cerrará, y la comunicación entre la computadora y los robots se cortará gracias al siguiente atributo de la clase *Serial*:

```
close()
```

Con eso termina la explicación del funcionamiento del programa más importante. En el siguiente capítulo, se abordarán los temas de la configuración utilizada para realizar los experimentos, así como también los resultados obtenidos en los mismos.

Capítulo 6

Configuraciones de la plataforma de experimentación y resultados

En el presente capítulo se explicarán las configuraciones utilizadas para la plataforma experimental, así como la metodología de los experimentos. Al final de cada sección, se presentan los resultados obtenidos de las pruebas realizadas.

6.1. Introducción

Para realizar experimentos, y con ello poder conocer el rendimiento del programa desarrollado, se necesita una plataforma experimental, de manera que sea posible probar el *Software* bajo diferentes circunstancias. Además, la realización de diferentes pruebas ayuda a conocer el grado de robustez del programa. Con todo esto, es posible percatarse de las fortalezas y debilidades con respecto a otras soluciones, y con ello poder optimizar el programa en versiones futuras.

A continuación, se presentará la configuración utilizada para la experimentación.

6.2. Configuración de la plataforma experimental

La configuración de las plataformas de experimentación dependen da la prueba a realizar, para averiguar el rendimiento del programa con respecto a diferentes factores. Las pruebas son:

- Pruebas de tiempo de procesamiento.
- Pruebas de control de postura de robots E-puck.
- Pruebas de calidad y detección con desenfoque.
- Prueba de reproducibilidad.

Se iniciará con mostrar la configuración de las pruebas para conocer el tiempo de procesamiento de las imágenes.

6.2.1. Pruebas de tiempo de procesamiento

Para estas pruebas, se utilizaron dos diferentes configuraciones, debido a que el objetivo es comprobar la detección con respecto a la distancia. Es necesario aclarar que se utilizan ambas cámaras mencionadas en el apartado 4.1.1 para realizar una comparación entre las imágenes obtenidas con diferentes dispositivos, y con ello percibir si existe alguna diferencia en la detección. Para realizar estas pruebas, es necesario acondicionar a los robots E-puck, a los cuales se les asignó un marcador a cada uno. Dichos marcadores fueron impresos en papel normal blanco sin ninguna característica especial. Para estas pruebas no es necesario programar un *firmware* en los robots, ya que solo se busca conocer el tiempo de procesamiento de la detección, no la respuesta de los E-puck. Dichas configuraciones de las plataformas se muestran a continuación:

Prueba de tiempo de procesamiento 1. En esta prueba se posiciona la cámara sobre una mesa de 60cm, a un ángulo de 30 grados, tal y como se muestra en la Figura .

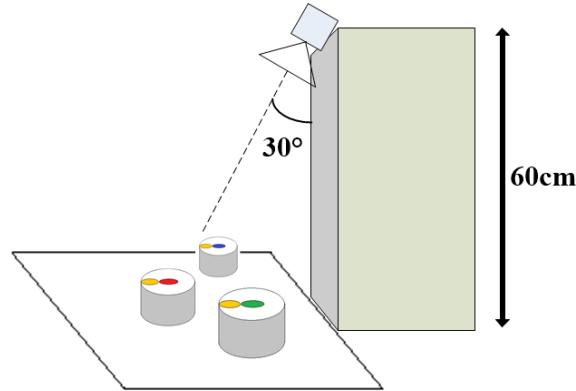


Figura 6.1: Configuración para la prueba de tiempo 1.

Prueba de tiempo de procesamiento 2. En esta prueba, una persona se encarga de sostener la cámara de manera orthogonal al plano horizontal de movimiento de los robots. Esto se representa en la Figura 6.2.

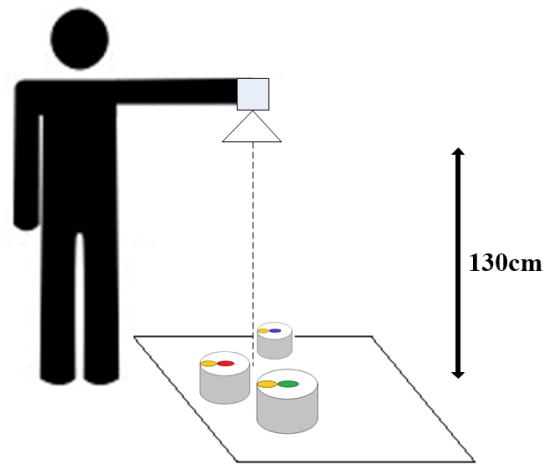


Figura 6.2: Configuración para la prueba de tiempo 2.

Estas son las dos configuraciones utilizadas para las pruebas de tiempo. Ahora, se explicará la metodología utilizada en dichas pruebas.

6.2.1.1. Metodología

Para realizar las pruebas experimentales de tiempo de procesamiento, se posiciona la cámara a dos distancias diferentes con respecto a los robots (60cm y 130cm). En cada una de las configuraciones, se acomodan los robots cercanamente uno del otro (Figura 6.3.a), así como también alejadamente (Figura 6.3.b). Esto con la finalidad de probar el rendimiento del algoritmo implementado.

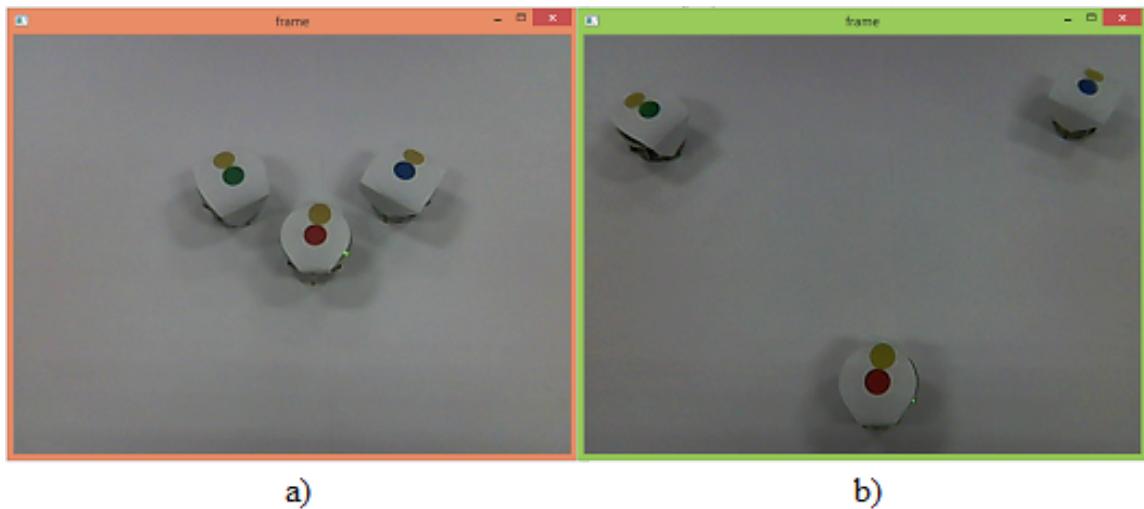


Figura 6.3: a) Posicionamiento de los robots cercanamente uno del otro. b) Posicionamiento de los robots alejadamente uno del otro.

En cada uno de los posicionamientos, se inicia el programa de detección hasta tomar veinte capturas. Una vez terminado, se muestra la información de la duración máxima, mínima y promedio del procesamiento total de las veinte capturas. El proceso de captura y medición de tiempo se repite las veces que se considere necesarias (En el presente trabajo se hicieron cinco repeticiones). Para éstas pruebas se utilizó la computadora portátil mencionada en la sección 4.1.2. En caso de que el usuario quiera tomar mediciones de tiempo propias, solo es necesario quitar los comentarios de las líneas de código que muestran tal información. Dichas líneas de código se encuentran en el módulo de rastreo de robots, después de enviar la información del *string* a cada uno de los robots.

6.2.1.2. Resultados

Los resúmenes de los resultados de los tiempos medidos, utilizando ambas cámaras, se encuentran en el Cuadro 6.1.

60cm_cerca				
Cámara	<i>t</i> máx. medido	<i>t</i> mín. medido	\bar{t}	fps
C210	0.510s	0.192s	0.461s	2.169
C525	0.559s	0.418s	0.440s	2.271
60cm_lejos				
Cámara	<i>t</i> máx. medido	<i>t</i> mín. medido	\bar{t}	fps
C210	0.637s	0.404s	0.458s	2.181
C525	0.510s	0.398s	0.419s	2.389
130cm_cerca				
Cámara	<i>t</i> máx. medido	<i>t</i> mín. medido	\bar{t}	fps
C210	0.437s	0.312s	0.340s	2.938
C525	0.180s	0.127s	0.146s	6.748
130cm_lejos				
Cámara	<i>t</i> máx. medido	<i>t</i> mín. medido	\bar{t}	fps
C210	0.413s	0.292	0.320s	3.127
C525	0.352s	0.314s	0.337s	2.970

Cuadro 6.1: Resumen de tiempos medidos con ambas cámaras a 60cm con los E-puck cerca uno de otro.

En el resumen de resultados se aprecia que, en las pruebas con una distancia de 130cm, el programa de detección se desempeña de una mejor manera que con las pruebas realizadas a una distancia de 60cm (desde un 24 % hasta un 197 %). Esto se debe a que el algoritmo cuenta los píxeles pertenecientes al círculo detectado, por lo que, mientras más grande sea el marcador (más cerca de la cámara), más tardará la detección. También se observa que el mejor desempeño es cuando los robots están posicionados cercanamente uno de otro a la distancia de 130cm con la cámara C525, ya que el algoritmo está desarrollado para trabajar preferentemente de esa manera.

6.2.2. Pruebas de control de postura de robots E-puck

Para las pruebas de control de robots, se utilizó la configuración de la Figura 6.1. En éstas pruebas, es necesario comprobar la transmisión de información y la respuesta de los E-puck. El controlador implementado está basado en el denominado todo o nada (*ON/OFF*) con banda de histéresis (diferencia entre los tiempos de apagado y encendido [14]). Este tipo de controlador es de fácil implementación, aunque se requiere que la velocidad de reacción del proceso donde se aplica sea lento [4]. Este tipo de control actúa de la siguiente manera: Para alguna función $g(t)$ variable en el tiempo, la respuesta $f(t)$ está dada por

$$f(t) = \begin{cases} 1(ON) & , \text{si } g(t) < sp + \epsilon \\ 0(OFF) & , \text{si } g(t) > sp - \epsilon \end{cases}, \quad (6.1)$$

donde sp (*setpoint*) es el valor deseado, y ϵ denota la banda de histéresis (diferencial). En la Figura 6.4, se puede apreciar la aplicación de un control todo o nada con banda de histéresis.

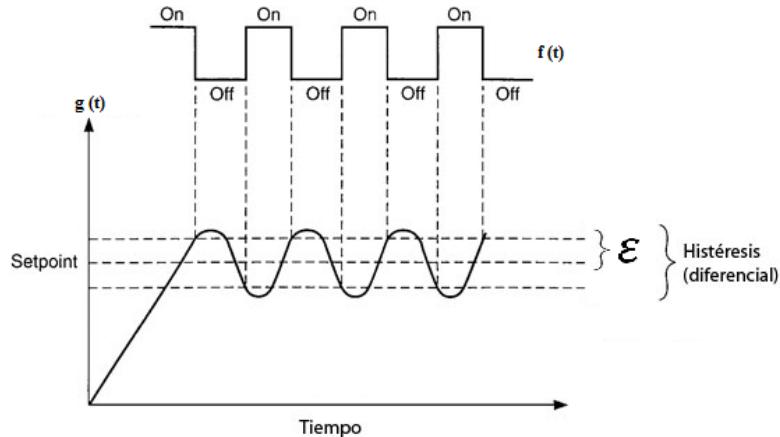


Figura 6.4: Aplicación de un control todo o nada con banda de histéresis [56].

Debido a que en la prueba a realizar, los robots pueden girar tanto a favor de

las manecillas del reloj (*clockwise*, *CW*) como en contra (*counterclockwise*, *CCW*), el control aplicado se describe por la función de giro (f_g), la cual está dada por

$$f_g(t) = \begin{cases} 0,05 * V_{max}(CW) & , \text{si } (\tilde{\theta} < -15 \wedge \tilde{\theta} > -180) \vee \tilde{\theta} \geq 180 \\ 0,05 * V_{max}(CCW) & , \text{si } (\tilde{\theta} > 15 \wedge \tilde{\theta} < 180) \vee \tilde{\theta} \leq -180 \\ 0(OFF) & , \text{si } |\tilde{\theta}| \leq 15 \end{cases}, \quad (6.2)$$

donde V_{max} es la velocidad máxima del robot E-puck, la banda de histéresis $\epsilon = 15$ grados y $\tilde{\theta}$ es el error de orientación, el cual se calcula por

$$\tilde{\theta} = \theta_d - \theta, \quad (6.3)$$

donde θ_d es la orientación deseada (*set point*) y θ es la orientación medida. En la Figura 6.5, se observa un ejemplo del control implementado.

Cuando el error de orientación se encuentra dentro de la banda de histéresis, entra en acción el controlador de posición (función de avance, f_a), el cual se define por

$$f_a(t) = \begin{cases} 1(ON) & , \text{si } |\tilde{p}| > 10 \\ 0(OFF) & , \text{si } |\tilde{p}| \leq 10 \end{cases}, \quad (6.4)$$

donde $ON = 0,1 * V_{max}$, la banda de histéresis $\epsilon = 10$ unidades y \tilde{p} , que denota el error de posición, está dado por

$$\tilde{p} = p_d - p, \quad (6.5)$$

donde p_d es la posición deseada, y p es la posición medida.

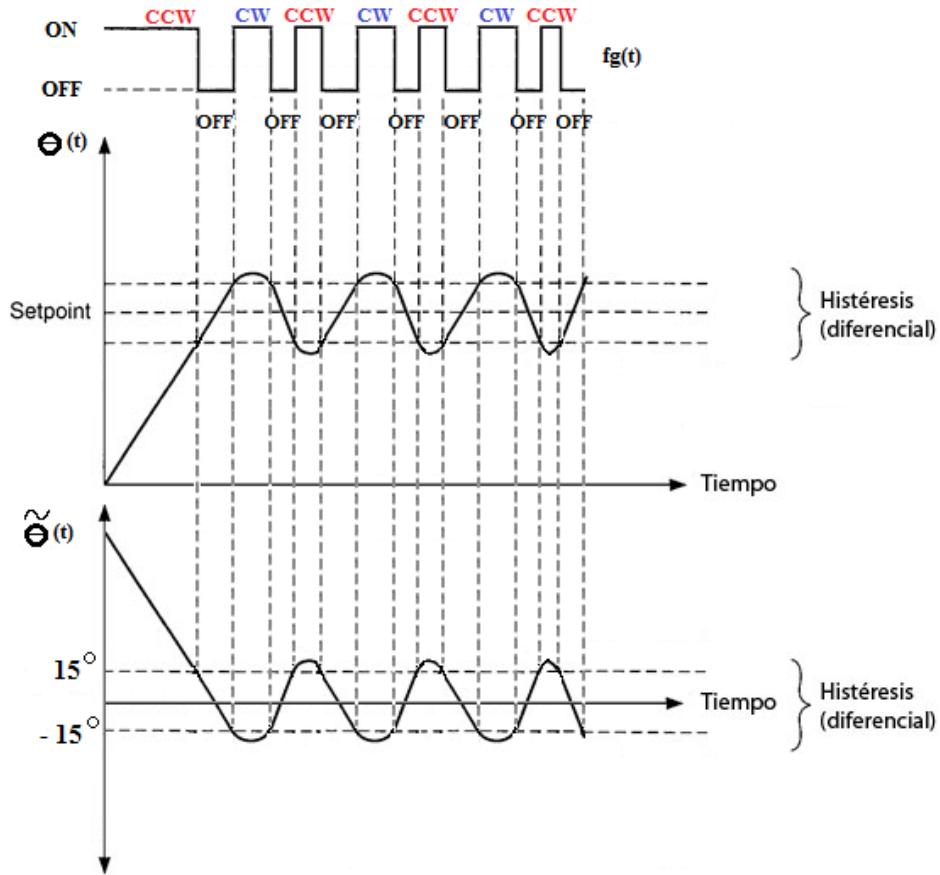


Figura 6.5: Ejemplo del control implementado.

Esta sencilla ley de control se programa en lenguaje C a la memoria de los robots, dándoles a los mismos autonomía propia para que decidan su movimiento. La información referente al uso del robot E-puck se encuentra en los apéndices, al igual que el diagrama de flujo y el código de la básica ley de control implementada.

6.2.2.1. Metodología

Para hacer las pruebas, se posicionó a un robot en algún lugar visible para la cámara. Como ya se mencionó, se utiliza la configuración que se ve en la Figura 6.1. Después de iniciar la ejecución del programa de detección, el usuario hace doble clic

en el lugar deseado donde quiere que se posicione el robot. En seguida, el programa mandará dicha información al E-puck, el cual al recibirla, girará a favor o en contra de las manecillas del reloj, dependiendo del error de orientación. Cuando el error se encuentra dentro de la banda de histéresis del control de orientación, el robot deja de girar, dando paso así al movimiento en línea recta hacia la posición deseada. Como el E-puck no llega exactamente al *setpoint*, el error de orientación aumentará cuando el robot avance en línea recta, por lo que llegará a un punto donde $\tilde{\theta}$ se encuentre fuera de la banda de histéresis, con lo cual el robot detendrá su movimiento en línea recta para volver a girar y reducir nuevamente el error a un valor aceptable. Esto se repite hasta que el robot se encuentra dentro de la banda de histéresis de la posición. El usuario puede en cualquier momento cambiar el punto deseado al hacer doble clic en cualquier otro lugar. La prueba termina hasta que se cierre la conexión con el robot (cuando se cierre el programa de detección).

6.2.2.2. Resultados

La prueba muestra que el robot es capaz de recibir la información generada en la computadora, ya que el E-puck se mueve desde su posición inicial hasta el punto deseado. En la Figura 6.6, se muestra una secuencia de imágenes que representa dicho movimiento.

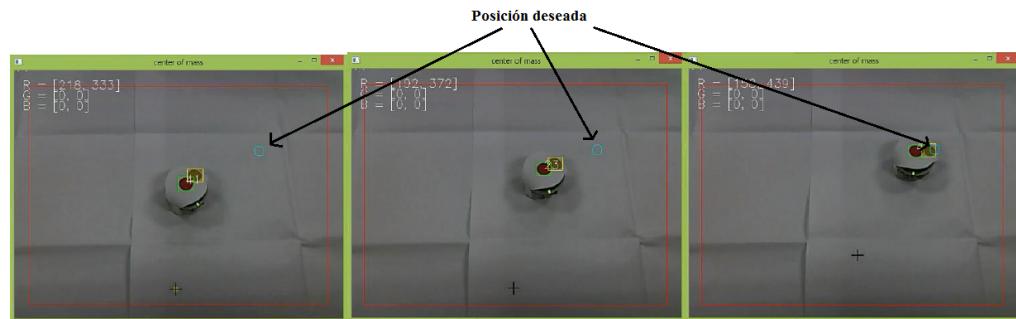


Figura 6.6: Movimiento de un robot E-puck a una posición deseada.

También es posible enviar la información simultáneamente a tres robots E-puck, el retardo de tiempo en la transmisión de datos es menor a 1ms (aproximadamente 250 μ s para cada robot), por lo que se considera un tiempo despreciable comparado con lo que tarda el procesamiento de la imagen.

6.2.3. Pruebas de calidad y detección con desenfoque

La configuración de la plataforma para las pruebas de calidad se observa en la Figura 6.2. En las pruebas de detección con desenfoque se utiliza la configuración de la Figura 6.1. Para las pruebas de calidad, se utilizan las dos cámaras mencionadas en la sección del *Hardware* (4.1.1) con el fin de conocer si la calidad de la imagen facilita la detección. En las pruebas de detección con desenfoque, se utiliza solamente la cámara *Logitech C525*, ya que esta es la única cámara disponible con autoenfoque.

6.2.3.1. Metodología

Para realizar las pruebas de *calidad*, se posiciona primero la cámara C210, tal y como se ve en la Figura 6.2. Después, se inicia el programa de detección y se prueba su desempeño. Al terminar, se repite el proceso con la cámara C525.

Para realizar las pruebas de *detección con desenfoque*, se busca que ocurra un desenfoque y se posiciona la cámara como se ve en la Figura 6.1. Se observa el funcionamiento del programa ante el desenfoque presentado.

6.2.3.2. Resultados

Los resultados, en cuanto a calidad de imagen se refiere, se pueden ver en la Figura 6.7, donde se observa que la imagen de la cámara C525 (Figura 6.7.a) tiene una

mejor calidad que la imagen de la cámara C210 (Figura 6.7.b), donde ambas imágenes tienen el mismo nivel de acercamiento. La calidad de imagen afecta la detección de objetos, siendo mejor la detección con la primer cámara. Esto se verifica en las pruebas donde, en ocasiones, la cámara C210 deja de detectar momentáneamente los marcadores debido a la calidad de la imagen, lo cual se podría resolver con una mejor selección de umbrales, o con una mejor cámara.

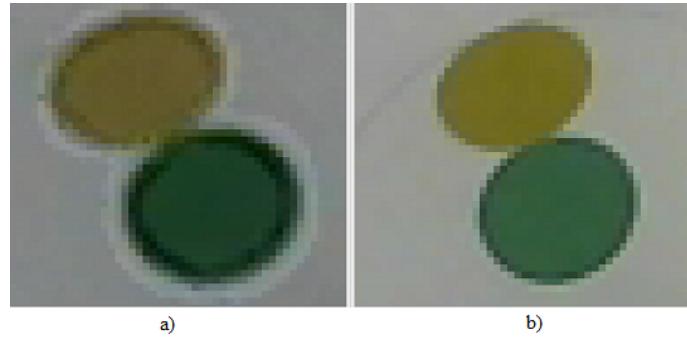


Figura 6.7: a) Imagen captada por la cámara C210. b) Imagen captada por la cámara C525.

En cuanto al autoenfoque con la cámara C525, en la Figura 6.8 se aprecia la diferencia entre una captura desenfocada (Figura 6.8.a) y una enfocada (Figura 6.8.b). Este tipo de desenfoque puede afectar en la detección de marcadores con figuras muy pequeñas, ya que no se podrían reconocer o porían confundirse unos marcadores con otros.

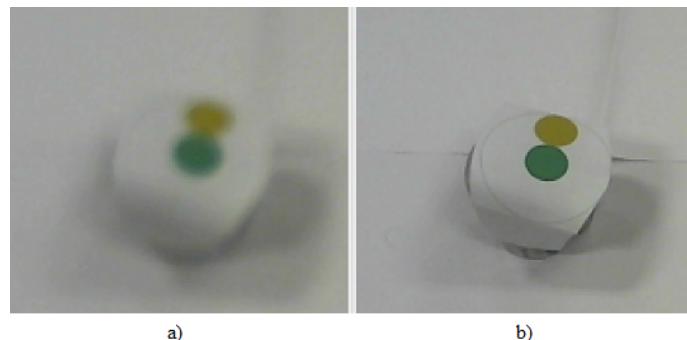


Figura 6.8: imágenes tomadas con la cámara C525. a) Captura desenfocada. b) Captura enfocada.

Se hicieron pruebas para obtener un desenfoque al estar ejecutando el programa de detección. En la Figura 6.9, se observa una de las pruebas, donde se aprecia que existe una detección de los tres robots apesar del desenfoque presente, por lo que se puede decir que el programa es robusto en este aspecto, esto gracias a la segmentación por color.

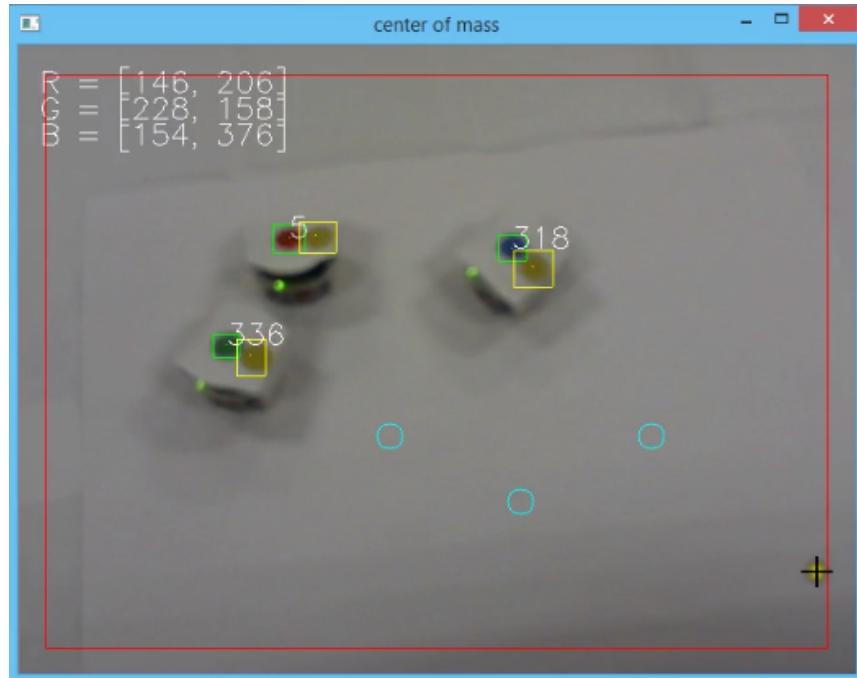


Figura 6.9: Detección de tres robots E-puck con desenfoque.

A continuación se presentan las generalidades para la última prueba restante, la prueba de reproducibilidad.

6.2.4. Prueba de reproducibilidad

En las pruebas de reproducibilidad, se utiliza otra computadora portátil para probar que el programa puede ejecutarse sin problemas en un equipo diferente.

6.2.4.1. Metodología

En otra computadora portátil se instalan las librerías necesarias para la ejecución del programa. Al tener las librerías instaladas, se ejecuta el programa. Para establecer la comunicación con los robots E-puck, es necesario emparejarlos con la nueva computadora.

6.2.4.2. Resultados

Como resultado, al instalar las librerías y ejecutar el programa de detección en otra computadora (con el SO *Windows 7*), y dado que no se presentan problemas al utilizar el *Software* de rastreo de robots, se demuestra que el programa es reproducible y se puede utilizar en la mayoría de las computadoras actuales (Figura 4.7).

Con todo lo anteriormente mencionado, se terminan de exponer los resultados del presente trabajo, por lo que en el siguiente capítulo se presentan las conclusiones.

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones

En el presente trabajo de tesis, se presentó el diseño de un programa de detección para el control de postura de robots móviles.

Los objetivos que se presentados en el capítulo 1 se cumplieron casi en su totalidad, concluyendo lo siguiente:

- El programa desarrollado es capaz de detectar los marcadores propuestos aún con desenfoque de la cámara.
- La detección de los marcadores no necesita de condiciones estrictas de iluminación (no le afectan las sombras), ya que está basado en el espacio de color HSV.
- El desarrollo del programa está desarrollado completamente con *Software libre*.
- El programa de detección es reproducible.
- Se puede aplicar para el control de múltiples robots, con la posibilidad de utilizar hasta tres simultáneamente.

La desventaja que hay con el programa desarrollado es su tiempo de procesamiento, siendo más lento que otras soluciones existentes. Otro problema que presenta el

programa es que no tiene modularidad, por lo que solo es posible detectar hasta tres robots.

7.2. Trabajos futuros

Para resolver los problemas encontrados, se propone lo siguiente como trabajo futuro:

- Incrementar la velocidad de detección reduciendo la imagen a un múltiplo dado con respecto al tamaño original. Esto se encuentra parcialmente implementado.
- Agregar modularidad al programa de detección, para que sea capaz de detectar un mayor número de robots.
- Mostrar al usuario una medida de la posición en centímetros para una mejor interpretación de los resultados.

Bibliografía

- [1] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 1 edition, 2002.
- [2] Paulo Aguiar, Luís Mendonça, and Vasco Galhardo. Opencontrol: A free open-source software for video tracking and automated control of behavioral mazes. *Journal of Neuroscience Methods*, (166):66–72, 2007.
- [3] Algorithms, Data Structures with implementations in Java, and C++. Depth-first search (dfs) for undirected graphs. http://www.algolist.net/Algorithms/Graph/Undirected/Depth-first_search.
- [4] Luis Almache and Luis Toapanta. Control de temperatura on-off con histéresis mediante la utilización de un cautín. <http://es.scribd.com/doc/97395385/Informe-de-Control-on-Off-Con-Histeresis#scribd>, 2012.
- [5] Visión Artificial. Filtrado espacial. http://www.varpa.org/~mgpenedo/cursos/Ip/Tema4/nodo4_2.html.
- [6] Tucker Balch, Aaron Bobick, Jim Rehg, Irfan Essa, Brian Hrolenok, Andy Quitmeyer, Stephen Motter, David Stolarsky, Blacki Li Rudi Migliozi, Ilho Song, Vikram Parthasarathy, Matt Flagg, and Adam Feldman. Bio-tracking. <http://www.bio-tracking.org>, 2012.
- [7] Patrick Benavidez and Mo Jamshidi. Mobile robot navigation and target tracking system. In *Proc. of the 2011 6th International Conference on System of Systems Engineering*, pages 299–304, 2011.
- [8] James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2061–2066, 2000.
- [9] Frédéric Chenavier and James L. Crowley. Position estimation for a mobile robot using vision and odometry. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, volume 3, pages 2588–2593, 1992.
- [10] C. Chiculita. Tiny pic bootloader. <http://www/etc.ugal.ro/cchiculita/software/picbootloader.htm>, 2014.
- [11] Erik Cuevas, Daniel Zaldívar, and Marco Pérez. *Procesamiento digital de imágenes con Matlab y Simulink*. Alfaomega,Ra-ma, 1 edition, 2010.

- [12] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowsky, J. Spletzer, and Taylor C. A vision-based formation control framework. In *IEEE Transactions on Robotics and Automation*, volume 18, pages 813–825, 2002.
- [13] E. R. Davies. *Computer and Machine Vision - Theory, Algorithms, Practicalities*. Academic Press, 2012.
- [14] Sistemas de control automático. Guía 5: El controlador on-off (si-no o todo y nada). <http://www.udb.edu.sv/udb/archivo/guia/electronica-ingenieria/sistemas-de-control-automatico/2013/i/guia-5.pdf>, 2015.
- [15] electric Bricks. Sistemas holonómicos. <http://blog.electricbricks.com/2010/07/holonomic-robot/>, 2010.
- [16] Fernando Fernández, Germán Gutiérrez, and José M. Molina. Cooperación en sistemas distribuidos de robots reactivos minimizando la cantidad de información comunicada. In *Simposio Español de Informática Distribuida (SEID)*, pages 61–68, 2000.
- [17] GCtronic. E-puck. <http://www.gctrionic.com/doc/index.php/E-Puck>, 2015.
- [18] Javier González Jiménez and Anibal Ollero Baturone. Estimación de la posición de un robot móvil. *ResearchGate*, 2015.
- [19] Nestor Andrés González Vargas. Sistema de visión por computadora para la medición de distancia e inclinación de obstáculos para robots móviles. *IEEE Colombian Workshop on Robotics and Automation*, 2005.
- [20] Music Technology Group and Pompeu Fabra University. reactivision 1.4. <http://reactivision.sourceforge.net/>, 2008.
- [21] Rubén Hernández, Óscar Salas, and Jesús De León Morales. Formation maneuvers via adaptive super twisting approach. In *PHYSCON*, 2013.
- [22] MPLAB X IDE. Mplab x. <http://www.microchip.com/pagehandler/en-us/family/mplabx/>, 2014.
- [23] Andrew J Davidson and David W Murray. Mobile robot localisation using active vision. In *Proc 5th European Conference on Computer Vision*, volume 2, pages 809–825, 1998.
- [24] Zdenek Kalal. Tracking-learning-detection (tld). <http://personal.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html>, 2013.
- [25] Jonathan Lawton, Randal Beard, and Brett Young. A decentralized approach to formation maneuvers. *IEEE Transactions on Robotics and Automation*, (6):933–941, 2003.

- [26] Thomas Lochmatter, Pierre Roduit, Chris Cianci, Nikolaus Correll, Jacques Jacot, and Alcherio Martinoli. Swistrack - a flexible open source tracking software for multi-agent systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.
- [27] Logitech. Webcam c210. http://support.logitech.com/es_es/product/webcam-c210, 2015.
- [28] Logitech. Webcam c525. <http://support.logitech.com/product/hd-webcam-c525>, 2015.
- [29] Jorge M. Runc. Filtros espaciales, clase 2. http://www2.fisica.unlp.edu.ar/materias/procesamiento_de_imagenes/Clase2_Imagenes_2011.pdf, 2011.
- [30] Stéphane Magnenat, Philippe Retornaz, Florian Vaussard, and Francesco Mondada. Aseba: An event-based architecture for distributed control of mobile robots. <http://mobots.epfl.ch/aseba.php>, 2010.
- [31] Microchip. dsPIC30F6014a. <http://www.microchip.com/wwwproducts/Devices.aspx?product=dsPIC30F6014A>, 2011.
- [32] Microchip. Downloads archive. <http://www.microchip.com/pagehandler/en-us/devtools/dev-tools-parts.html>, 2014.
- [33] Francesco Mondada. Software available for the e-puck platform. http://www.e-puck.org/index.php?option=com_content&view=article&id=18&Itemid=24, 2010.
- [34] Francesco Mondada and Michael Bonani. Tutorial for programming the e-puck robot using the bootloader via bluetooth. http://www.e-puck.org/index.php?option=com_phocadownload&view=category&id=5:tutorials&Itemid=38, 2006.
- [35] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptozz, Magnenat Stéphane, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59–65, 2009.
- [36] Netmarketshare. Desktop operating system market share. <http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0>, 2015.
- [37] Aníbal Ollero Baturone. *Robótica; Manipuladores y Robots Móviles*. Marcombo, 2001.
- [38] Opencv.org. Opencv downloads. <http://opencv.org/downloads.html>, 2014.

- [39] Carlos Platero. Procesamiento morfológico. http://www.elai.upm.es/webantigua/spain/Asignaturas/MIP_VisionArtificial/ApuntesVA/cap6VAProcMorf.pdf, 2009.
- [40] Pyserial. Pyserial. <http://pyserial.sourceforge.net/>, 2013.
- [41] Python.org. Python 2.7.5 release. <https://www.python.org/download/releases/2.7.5/>, 2013.
- [42] Python.org. The python tutorial. <https://docs.python.org/2/tutorial/>, 2015.
- [43] Fernando Reyes Cortés. *Robótica - Control de Robots Manipuladores*. Alfaomega, 2011.
- [44] Adriana Riveros and Leonardo Enrique Solaque. Formación de robots móviles mediante el uso de controladores. *Ing. USBMed*, (2):62–65, 2013.
- [45] Roberto Rodríguez Morales and Juan Humberto Sossa Azuela. *Procesamiento y Análisis Digital de Imágenes*. Alfaomega,Ra-ma, 1 edition, 2012.
- [46] C. Sagües, A. R. Mosteo, D. Tardioli, Murillo A. C., J. L. Villarroel, and L. Montano. Sistema multi-robot para localización e identificación de vehículos. *Revista Iberoamericana de Automática e Informática industrial*, (9):69–80, 2012.
- [47] Samsung. Np550p5c-a01ub-specs. <http://www.samsung.com/us/computer/pcs/NP550P5C-A01UB-specs>, 2013.
- [48] Scipy.org. Obtaining numpy and scipy libraries. <http://www.scipy.org/scipylib/download.html>, 2015.
- [49] Stephen Se, David G. Lowe, and J. Little James. Vision-based global localization and mapping for mobile robots. In *IEEE Transactions on Robotics*, volume 21, pages 364–375, 2005.
- [50] R. Silva Ortigoza and J. R. García Sánchez. Una panorámica de los robots móviles. *Telematique*, (3), 2007.
- [51] Derek Simkowiak. Motion tracking with python. <http://derek.simkowiak.net/motion-tracking-with-python/>, 2011.
- [52] Sjriek. Programming. <http://sjriek.nl/projects/e-puck/programming/>, 2014.
- [53] Ondřej Staněk. Centralized multirobot system. Master’s thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2012.
- [54] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

- [55] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1 edition, 1992.
- [56] Oficios Técnicos. Modos de control de controladores industriales. http://www.tecnoficio.com/electricidad/instrumentacion_industrial4.php, 2015.
- [57] Sergi Torrellas. Cognitive stimulation through task-oriented telerobotics. Master's thesis, Universitat Oberta de Catalunya, 2013.
- [58] Hardvard University. About epucks. <http://isites.harvard.edu/icb/icb.do?keyword=k104755&pageid=icb.page690475>, 2014.
- [59] Wikipedia. Depth-first search. https://en.wikipedia.org/wiki/Depth-first_search, 2015.

Apéndices

Apéndice A

Preparación y uso del programa de detección

Para utilizar el *Software* de detección de robots, es necesario instalar y configurar algunos programas antes de comenzar. Se aclara que las instrucciones son para utilizarse con el SO *Windows 8.1*, por lo que puede ver algunas diferencias con otros SOs de *Windows*, aunque deben de ser mínimas. Se iniciará con el procedimiento de la preparación de los programas necesarios para iniciar a utilizar el programa desarrollado.

A.1. Preparación del programa de detección

Para poder utilizar el programa de detección, es necesario instalar los siguientes programas y librerías (se mostrarán las versiones utilizadas. Todas las versiones son de 32 bits):

- Python 2.7.5 [41].
- OpenCV 2.4.10 [38].
- Numpy 1.9.0 [48].
- PySerial 2.7 [40].

Las demás librerías utilizadas vienen incluídas en el instalador de Python (están de manera predeterminada en la instalación). La instalación de todos los programas y librerías es sencilla, ya que es igual que con cualquier programa de *Windows*, no

hay necesidad de utilizar comandos para la instalación. Para conocer más sobre la programación en Python, existen muy buenos tutoriales en internet, uno de ellos puede encontrarse en [42].

A.1.1. Utilizar el Python IDLE(GUI)

El Python IDLE (por sus siglas en inglés de *Integrated Development Environment*) es la interfaz gráfica que viene predeterminada en la instalación de Python. Para abrirla, solamente se necesita hacer doble clic en el ícono del IDLE. Una vez abierta, aparece una ventana como la mostrada en la Figura A.1.

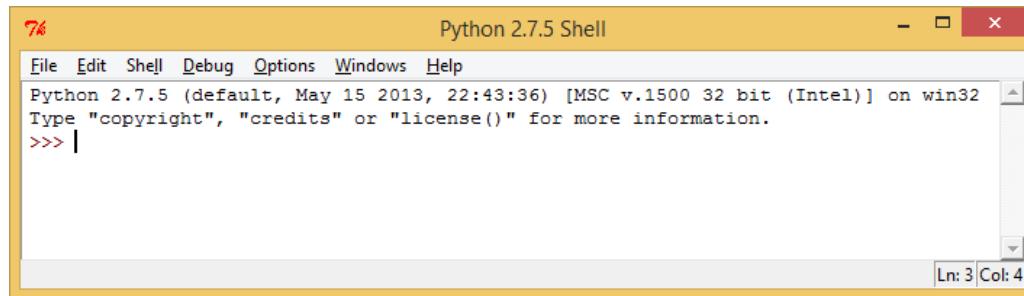


Figura A.1: GUI de Python.

Se puede observar que una nueva línea de comando inicia con los símbolos >>>. Para la ejecución de una línea de comando, se presiona la tecla *Enter*.

Para corroborar que las librerías se encuentran correctamente instaladas, es necesario escribir en el Python IDLE y los siguientes comandos:

```
import cv2  
import numpy  
import serial
```

Al escribir el comando *import* y el nombre de la librería a cargar, generalmente el IDLE se pone en estado de espera, y después de algunos segundos, debe de aparecer

una línea nueva para volver a escribir comandos. Si no aparece ningún mensaje de error, significa que la librería se importó con éxito.

Para ejecutar el Programa de detección, es necesario abrirlo y presionar la tecla F5 (Correr programa, *run*). Con ésto, debe de aparecer la GUI del programa de detección.

A.1.2. Utilizar el *Command Prompt* de *Windows*

Es posible utilizar el *Command Prompt* (CP) de *Windows* para la ejecución de los programas de Python, aunque para ello se tiene que agregar este último al *path* del *SO*. En la Figura A.2, se muestra la pantalla perteneciente al CP.

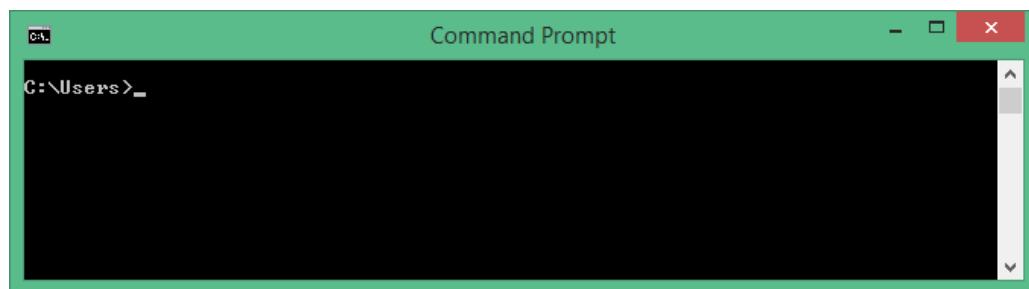


Figura A.2: Ventana del *Command Prompt*.

Para agregar Python al *path* de *Windows*, es necesario dirigirse a las propiedades del sistema, y hacer clic en *Configuraciones avanzadas* (Figura A.3)

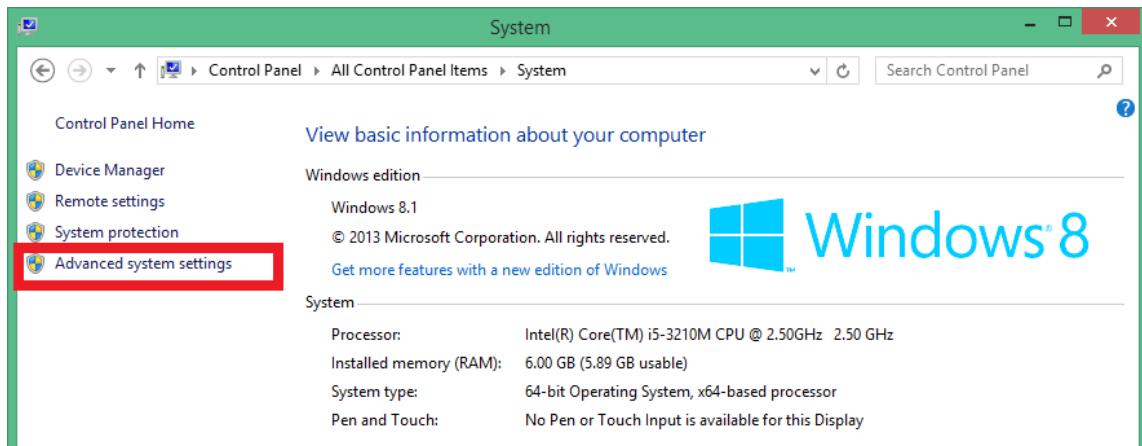


Figura A.3: Propiedades del sistema.

Después aparecerá una ventana como la mostrada en la Figura A.4, donde se tiene que hacer clic en la opción de *Variables del sistema*.

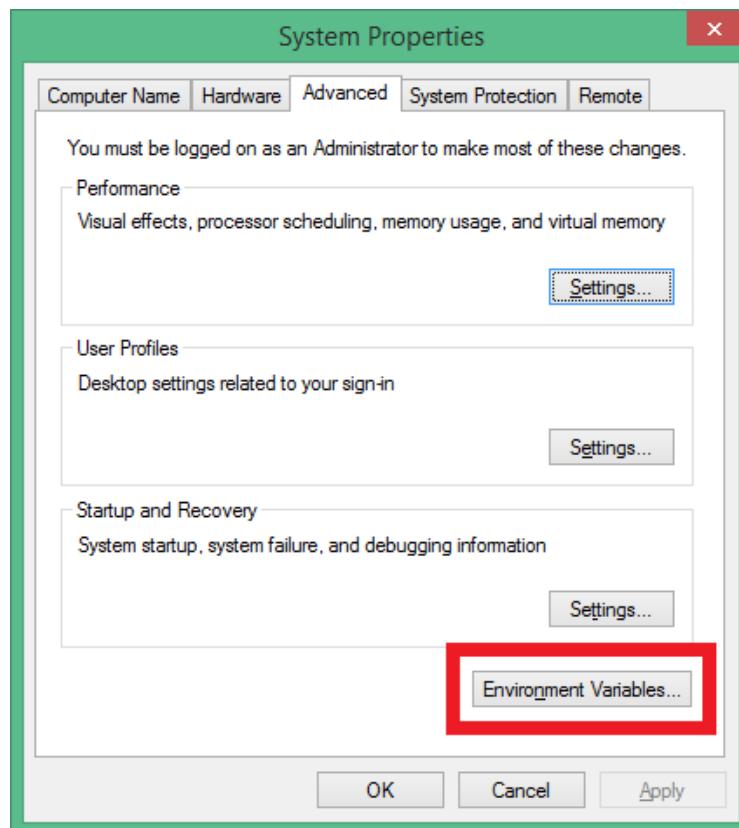


Figura A.4: Propiedades avanzadas.

Al seleccionar dicha opción, deberá emerger una ventana como la mostrada en la Figura A.5, donde se tiene que asegurar que *path* esté seleccionado en el apartado de *Variables del sistema*. Una vez seleccionado, se hace clic en editar.

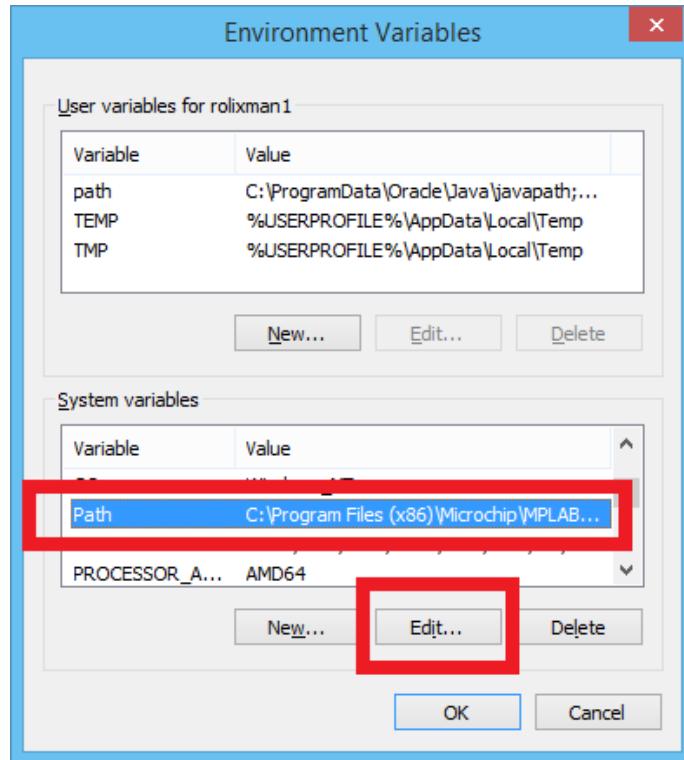


Figura A.5: Selección en variables de usuario.

Deberá de aparecer otra ventana como la que se muestra en la Figura A.6, y en el apartado de *valor de la variable* se escribirá: ;C:Python27 , donde el signo de puntuación (;) sirve para separar los directorios de otras variables y *Python27* es el directorio de instalación predeterminado en la instalación de la versión 2.7 de Python.

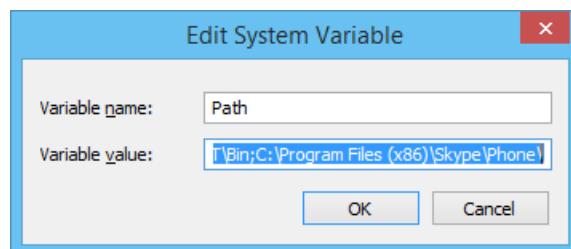


Figura A.6: Editar variables del sistema.

Con esto es posible utilizar python en el CP (funciona como el IDLE de Python). Ahora, para corroborar que las librerías se instalaron correctamente, en la ventana del CP se escribe el comando:

```
python
```

con esto, se abre Python desde el CP. Después se utilizan los mismos comandos mencionados en la sección anterior para la importación de librerías. Al igual que como se dijo anteriormente, si no aparece ningún mensaje de error, significa que las librerías fueron importadas con éxito.

Para salir de Python desde el CP, es necesario escribir el comando:

```
quit()
```

Para ejecutar el programa de detección, es necesario dirigirse al directorio que contiene el *script* (el archivo del programa). Para moverse de directorios en el CP, se utilizan los comandos *cd* para entrar a una carpeta y *cd..* para regresar una carpeta. Cuando ya se llegue a la carpeta deseada, se debe escribir el comando *python* y el nombre del script separados por un espacio, como se ve a continuación:

```
python robot_tracker.py
```

con esto, se ejecutará el script y emergirá la GUI del programa de detección.

La decisión de utilizar cualquiera de los dos métodos mencionados dependerá del usuario. En el presente trabajo se utiliza el segundo método.

A.2. Uso del programa desarrollado

Lo primero que se tiene que hacer es dirigirse al siguiente enlace: <https://github.com/>. Al estar en la página principal, se tiene que escribir en el busca-

dor el nombre del repositorio que contiene los archivos pertenecientes al programa de detección: TESIS_RMS_FIME. Una vez que se haya encontrado el repositorio, hacer clic en la liga que aparece para ver los archivos. En la carpeta hay algunas notas que pueden ser de ayuda. Al terminar de leer dichas notas, hacer clic en el botón *Download ZIP*, el cual se encuentra en la parte derecha de la ventana, lo cual descargará un archivo ZIP que contiene todo lo que se visualiza en la página.

Para correr el programa hay tres maneras, puede hacer doble clic sobre el archivo *robot_tracker.py*; puede abrir el archivo desde el IDLE de python, hacer clic en *run* y después en *run module* (o presionar directamente F5), ó puede acceder desde el *command prompt* de *Windows*, entrando a la carpeta del programa y escribiendo: *python robot_tracker.py* (note el espacio entre python y el nombre del programa). Las dos últimas opciones son las mejores, porque de esa manera es posible visualizar los errores y mensajes que se presenten.

En ocasiones, al correr el programa puede salir un error conteniendo las palabras *show_cam*, el cual se debe generalmente a que el número de la cámara es incorrecto, por lo que se tendría que cambiar en el código, guardar los cambios y volver a iniciar el programa hasta que funcione. Dicha configuración se encuentra en la clase *default>>cam_num*. Esto se debe a que a veces se tienen varias cámaras conectadas, por lo que se le asigna un número entero positivo aleatoriamente (empezando de 0 y en orden creciente). Por ejemplo, si se tienen 2 cámaras conectadas, una será la cámara 0 y la otra la 1.

Si no aparece ningún error, el programa desarrollado inicia con la GUI mostrada en la Figura 5.3 del capítulo 5. Esta cuenta con cinco botones, donde se observa que cada uno de ellos cuenta con una imagen descriptiva de la función que realiza, lo que le da una mejor comprensión a la interfaz.

Antes de dar las instrucciones para el uso de cada uno de los módulos principales, se explicará el formato del archivo de texto utilizado.

A.2.1. Archivo de texto

Como se mencionó en el capítulo 5, el programa desarrollado hace uso de un archivo de texto, el cual debe de estar en la misma carpeta donde se encuentra el *script* de python. Este archivo contiene los umbrales de cada uno de los colores a detectar en el siguiente formato:

blue	blue
low	up
h=104	h=130
s=78	s=255
v=0	v=124

donde primeramente se encuentra el nombre del color a detectar, seguido de la descripción del umbral (*low* si es el límite bajo o *up* si es el límite que se encuentra arriba). Después, se encuentran los valores de cada uno de los canales de la imagen hsv correspondientes a cada límite. Se recomienda modificar dichos valores directamente con el programa, para evitar que se corrompa el archivo. Si el archivo llega a corromperse, es posible que algunos módulos no trabajen correctamente, por lo que es necesario tener una copia de seguridad del archivo en otro lugar diferente a la carpeta del programa.

Al final del archivo de texto se encuentran las siguientes líneas:

(pixels, cms, pixels/cm)

units

10

2.33

4.29184549356

donde los valores corresponden a lo que viene en la primera línea (pixels $\bar{1}0$, cms $\bar{2}.33$ y pixels/cm $\bar{4}.292$). Los valores pueden no ser iguales a los aquí mostrados, ya que dependerán de la calibración que se haga.

A.2.2. Módulo de detección de robots (*Robot_tracker*)

Para utilizar éste módulo, el usuario debe de hacer un clic al primer botón de la interfaz, como se ve en la Figura A.7.

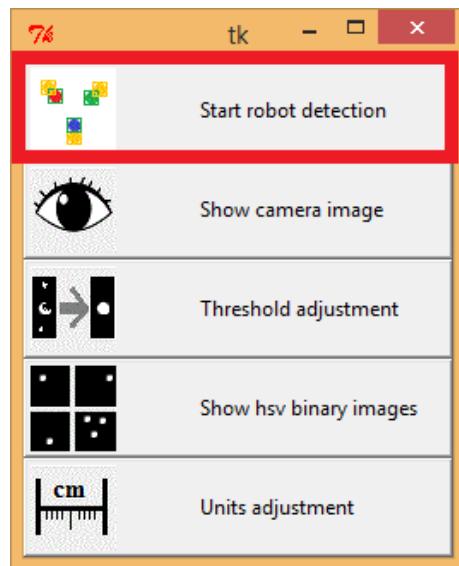


Figura A.7: Botón para el módulo *robot_tracker*.

Después, una nueva ventana emergirá. Esta ventana no contendrá nada hasta que el programa haya procesado la primera captura. Para esto, antes se tiene que establecer la comunicación con los robots E-puck. Para establecer la comunicación, los robots que se vayan a utilizar tienen que estar encendidos y emparejados con la computadora (el proceso para usar los robots se ve en el siguiente apéndice). Si el

programa no logra completar la conexión con todos los robots a utilizar (máximo tres) se mostrará un error de conexión fallida. Por esa razón, si alguno de los robots no será utilizado, es necesario comentar las tres líneas siguientes (como ya se ha mencionado, un comentario en python se precede por el signo #):

```
ser = serial.Serial(9, 115200, timeout = 0)  
ser.write(string)  
ser.close()
```

donde la *primer línea* se encuentra en la inicialización del módulo *Robot_tracker*, el num. 9 = Puerto COM del robot, el num. 115200 = Velocidad de transmisión de datos (baudios) y timeout = tiempo de espera de respuesta en la comunicación, la cual es igual a cero, lo que significa que esperará indefinidamente (si se elige un número mayor a 0, el programa esperará el número escrito en segundos en la transmisión de datos y, en caso de que la comunicación tarde esa cantidad de tiempo, arrojará un error de comunicación). La *segunda línea* es la que envía la información al robot y se encuentra después de la parte del procesamiento de la imagen. La *tercer línea* es la que cierra la comunicación y se encuentra al final del módulo. En caso de que no se comenten las dos primeras líneas de uno de los robots, el módulo de detección no se ejecutará.

Una vez que aparece una imagen procesada con sus resultados, como en la Figura A.8, el usuario puede dar doble clic en cualquier lugar de la pantalla, con lo que aparecerá un círculo en ese lugar. Entonces, la información será enviada a los robots. Para que los robots se dirijan a dicho punto, éstos tienen que tener el *firmware* que se muestra en el siguiente apéndice, o cualquier otro que se a capaz de interpretar la información recibida.

Para salir del módulo, solo es necesario cerciorarse de que la ventada que muestra las capturas procesadas esté seleccionada, para después presionar la tecla “q” (*quit*)

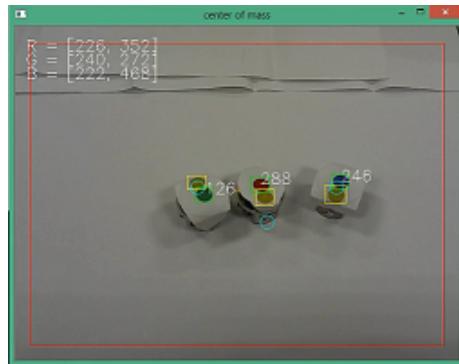


Figura A.8: Resultado de una imagen procesada.

con lo que se cerrará y estará listo para cualquier otro módulo regresando a la GUI principal.

A.2.3. Módulo de muestra de imagen (*show_cam*)

El módulo de muestra de imagen se accede por medio del segundo botón de la GUI, como se muestra en la Figura A.9.

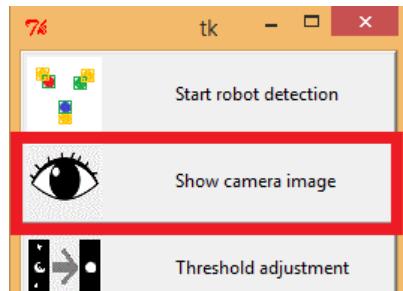


Figura A.9: Botón para el módulo *show_cam*.

El diagrama de flujo perteneciente a este módulo, se muestra en la Figura A.10. Este módulo es el más sencillo de todos, ya que su única función es mostrar las capturas, en tiempo real, de la cámara utilizada. Esto es útil en ocasiones, como por ejemplo, cuando el usuario quiere posicionar la cámara en el lugar correcto para iniciar las pruebas, puede hacer uso de este módulo sin necesidad de abrir otro *software*.

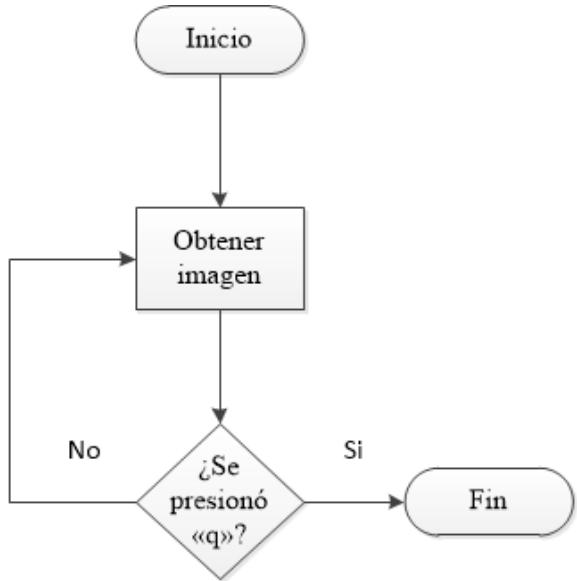


Figura A.10: Diagrama de flujo del módulo *show_cam*.

Este módulo muestra una nueva ventana, donde solamente se observa la captura a color en tiempo real de la cámara con una resolución de 640x480 píxeles. Para salir del módulo, es necesario cerciorarse de que la ventana que muestra las capturas esté seleccionada, para después presionar la tecla “q”. En caso de tener varias cámaras conectadas, en la clase *default* al inicio del *script* de python, se encuentra la opción *cam_num*, con la cual se puede seleccionar la cámara deseada. No hay forma de saber cual cámara es cual, por lo que tienen que hacerse varias corridas para comprobarlo (con corrida quiere decir volver a ejecutar el programa con los cambios realizados).

Este es el único módulo que no tiene interacción con el archivo de texto, y esto se debe a que es el único que no procesa la imagen, por lo que no ocupa los valores de los umbrales.

Ahora se explicará el tercer módulo principal, el módulo de ajuste de umbrales.

A.2.4. Módulo de ajuste de umbral (*thres_adj*)

Este módulo se selecciona con el botón que se muestra en la Figura A.11.

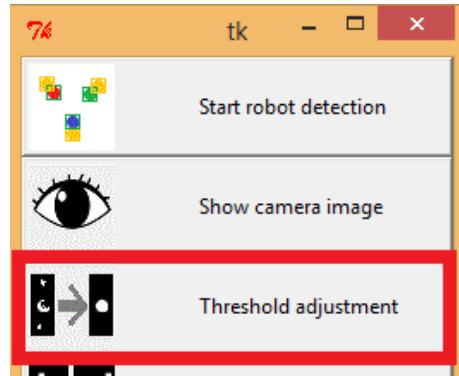


Figura A.11: Botón del módulo *show_cam*.

Su diagrama de flujo correspondiente se muestra en la Figura A.12.

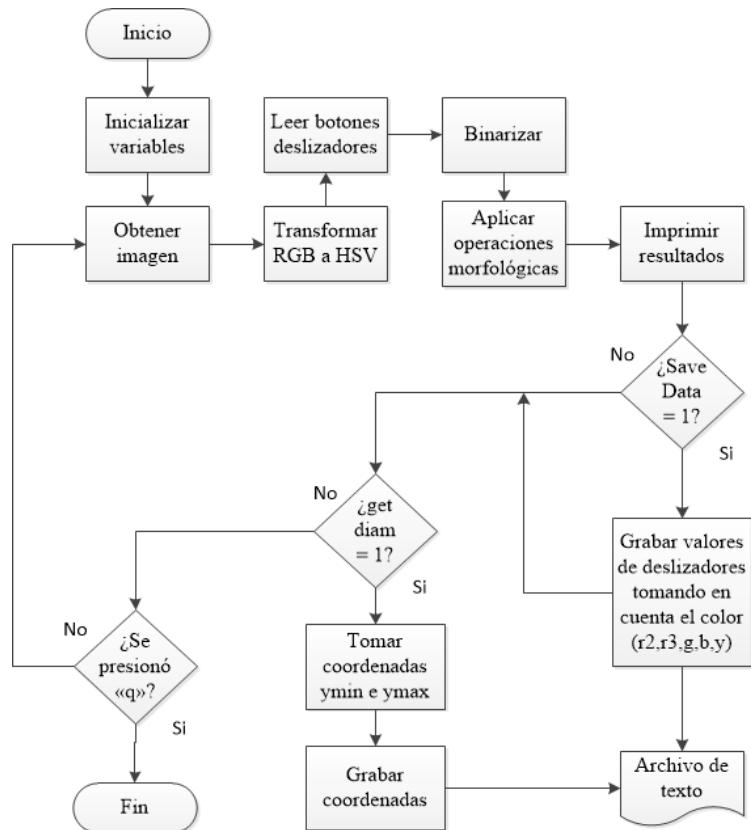


Figura A.12: Diagrama de flujo del módulo *thres_adj*.

Al ejecutar el módulo, aparecen ventanas como se muestra en la Figura A.13.

En ella se observan dos ventanas que muestran la imagen captada con los umbrales deseados. La ventana titulada como *mask*, corresponde a la imagen binarizada con respecto a los umbrales deseados, mientras que la imagen titulada como *open* muestra a la imagen binarizada después de aplicar operaciones morfológicas de apertura. Al iniciar, las ventanas muestran una imagen completamente blanca, ya que el umbral inicial abarca todo el rango posible, por lo que no filtra nada.

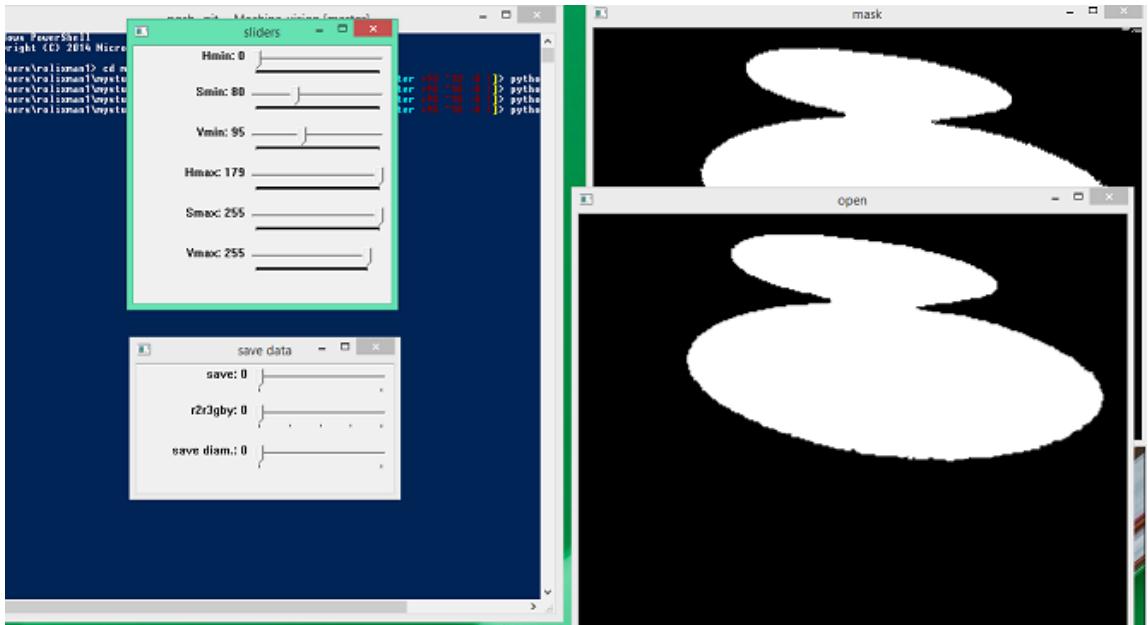


Figura A.13: Ventanas emergentes al iniciar el módulo *Thres_adj*.

En la Figura A.14, se observa una pequeña ventana en la cual el usuario puede establecer los umbrales que desee. Con ella, es posible cambiar cada uno de los tres canales en el espacio de color HSV.

Al cambiar los umbrales, se puede ver en las ventanas como cambia la imagen en tiempo real debido a la filtración. En la Figura A.15, se muestra un ejemplo de una captura con umbrales ajustados.

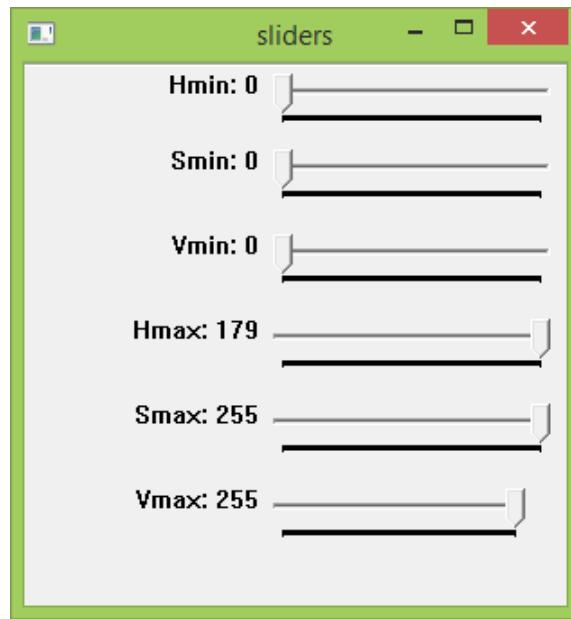


Figura A.14: Botones deslizadores de selección de umbral.

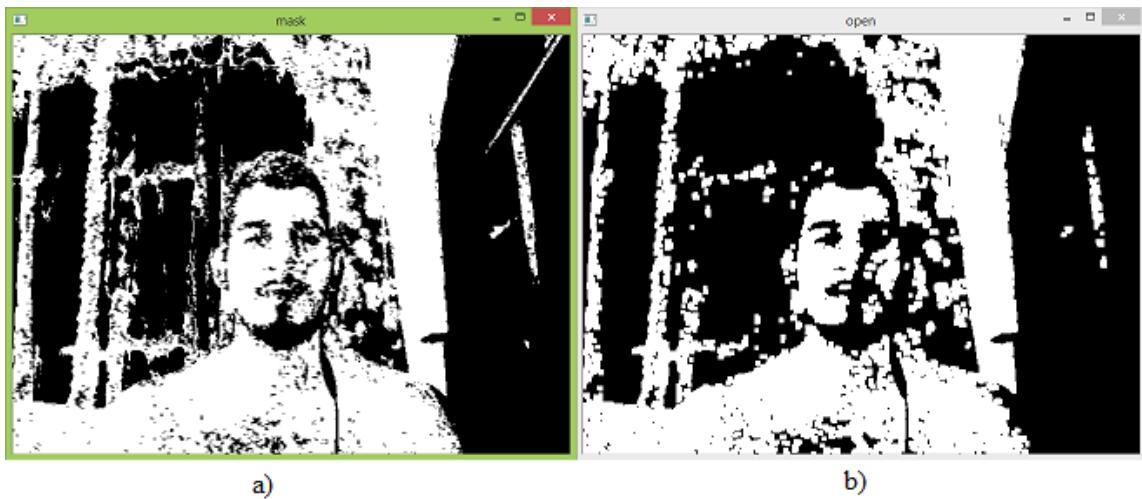


Figura A.15: a) Imagen umbralizada. b) Imagen al aplicar operaciones morfológicas.

Al tener los umbrales ajustados, es posible guardarlos en el archivo de texto para su posterior uso. Para esto, se utiliza la ventana con botones deslizadores que se muestra en la Figura A.16, titulada *save_data*. Esta ventana permite la interacción con el archivo de texto.

El primer botón deslizador permite guardar el umbral seleccionado en el archivo de texto. Para utilizarlo, solo es necesario moverlo hacia el número uno. Con esto,

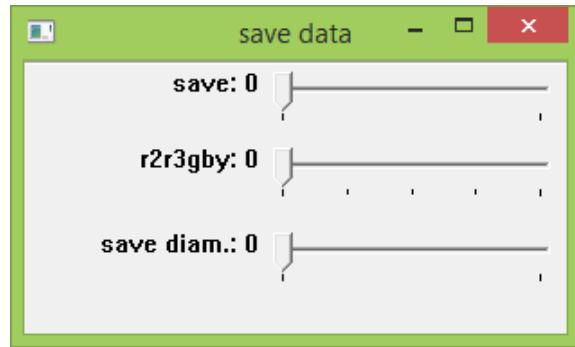


Figura A.16: Ventana *save_data*.

se guarda el valor de los umbrales en dicho archivo. Este deslizador regresará automáticamente a su posición inicial (cero) si el archivo de texto se encuentra en buen estado en la carpeta del proyecto. En caso de que el archivo se encuentre corrupto o fuera de la carpeta del proyecto, el programa arrojará un error y el usuario tiene que regresar manualmente el botón deslizador a cero.

Para seleccionar a qué color le corresponde el umbral seleccionado, se utiliza el segundo botón deslizador titulado *r2r3gby* (red2,red3,green,blue,yellow) que corresponde a los colores a detectar. Se puede observar que el rojo contiene dos umbrales, ya que de esta manera es posible detectarlo de manera correcta (se determinó a prueba y error). Para guardar el umbral en el color deseado, simplemente se desliza el botón al color objetivo. Este deslizador no regresa a su posición inicial.

El propósito del tercer botón deslizador se explicará más adelante, en el módulo de ajuste de unidades. Al igual que todos los módulos, para regresar a la GUI, es necesario presionar “q” mientras una de las ventanas que muestran imágenes está seleccionada.

Con esto finaliza el módulo de ajuste de umbrales. A continuación se explicará el funcionamiento del módulo de muestra de imágenes binarizadas.

A.2.5. Módulo de muestra de imágenes binarizadas *thres_adj*

Este módulo se ejecuta al presionar el cuarto botón de la GUI, el cual se muestra en la Figura A.17.

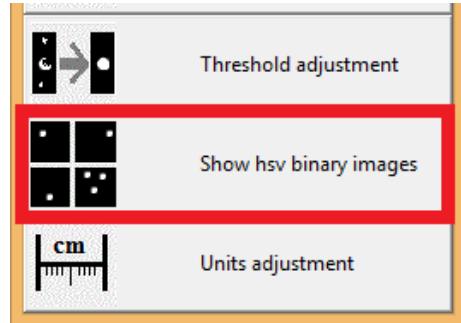


Figura A.17: Módulo *thres_adj*.

El diagrama de flujo correspondiente al módulo se muestra en la Figura A.18.

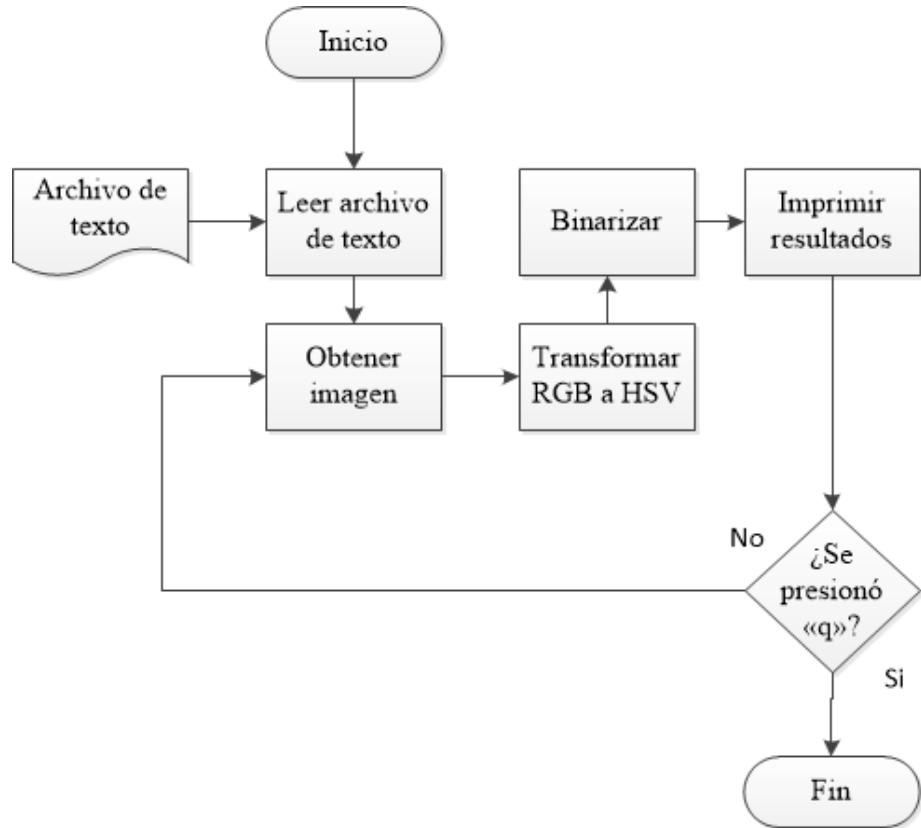


Figura A.18: Diagrama de flujo del módulo *Show_hsv_binary*.

Al presionarlo, deben emerger cuatro ventanas mostrando los umbrales actuales

del archivo de texto correspondientes a los colores establecidos. En caso de no encontrar el archivo de texto, se tomarán los valores predeterminados en la clase *default*. Este módulo sirve para corroborar la correcta selección de umbrales para asegurar la detección de los marcadores. En caso de que no se visualicen los marcadores de manera correcta, se debe de utilizar el módulo de ajuste de umbrales para obtener los umbrales correctos.

Para salir de este módulo, se presiona “q” mientras una de las ventanas que muestra capturas está seleccionada.

A.2.6. Módulo de ajuste de unidades *units_selection*

Para ejecutar este módulo se presiona el botón mostrado en la Figura A.19.

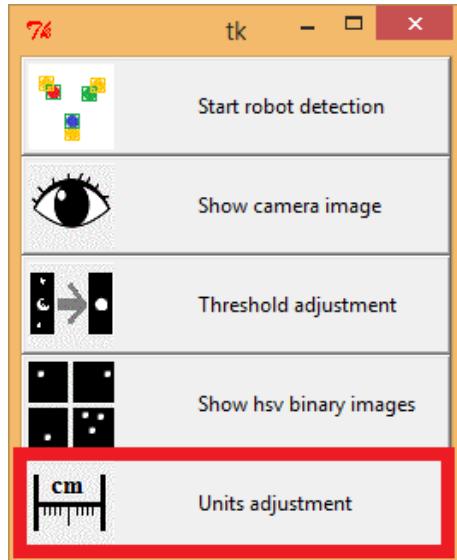


Figura A.19: Módulo *units_selection*.

El diagrama de flujo se muestra en la Figura A.20.

Este módulo ejecuta también al módulo *thres_adj*, ya que se complementan el uno con el otro. Al ejecutarse, aparecen las ventanas para el ajuste de umbrales, así como también la ventana de la Figura A.21.

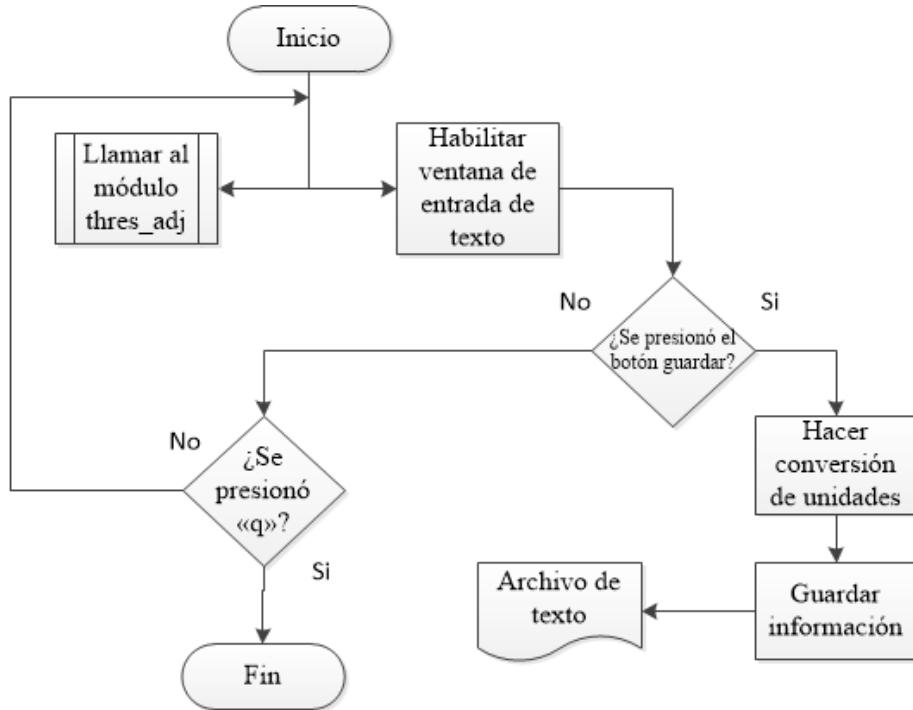


Figura A.20: Diagrama de flujo del módulo *Units_selection*.

Para poder ajustar las unidades se tiene que poner un objeto (de preferencia los marcadores o un objeto circular) de dimensiones conocidas a la vista de la cámara. Después se debe de ajustar el umbral hasta que el objeto sea lo único visible en la imagen binarizada (la que se titula *open*). Al tener esto, el usuario puede utilizar el tercer botón deslizador que se ve en la Figura A.16, etiquetado como *save_diam*. Lo que hace es calcular el diámetro del círculo tomando la diferencia entre la coordenada y_{min} y la coordenada y_{max} del objeto. Al ser un círculo, la orientación del objeto no importa, por lo que la caja envolvente siempre será igual. En caso de no contar con un círculo para la calibración, se puede posicionar algún objeto lo más vertical posible.

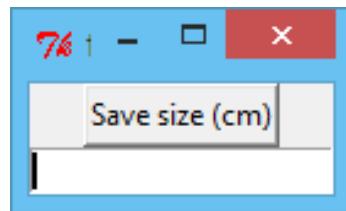


Figura A.21: Ventana de entrada de texto.

Para que el ajuste de unidades esté completo, el usuario tiene que ingresar la

medida del objeto en cms en el cuadro de texto de la Figura A.21. Una vez ingresado, el usuario tiene que dar clic en el botón *save size (cms)* para que el programa guarde automáticamente la información en el archivo de texto y calcule, con la información del diámetro, la cantidad equivalente de píxeles por centímetro.

Para cerrar este módulo es necesario presionar “q” mientras una de las ventanas que muestran imágenes está seleccionada, y después, se tiene que cerrar la ventana restante.

Apéndice B

Uso y programación del robot

E-puck

En el presente apéndice se muestra todo lo relacionado con el uso y la programación de los robots E-puck. Se presentarán las instrucciones que permitirán utilizarlos junto con el programa de detección.

B.1. Uso del robot E-puck

Los robots E-puck utilizados son fabricados por la compañía GCtronic [17]. Estos robots vienen de fábrica con un *firmware* demo con el cual se pueden ver inmediatamente algunas de sus funcionalidades. Para empezar a utilizar al robot con su configuración de fábrica, primeramente se tienen que encender con el switch que se encuentra en uno de los costados del robot como se aprecia en la Figura B.1. Al encender el robot, debe encender un led verde en la parte superior del robot. En ocasiones no se enciende al primer intento, por lo que se tiene que volver a intentar hasta que se encienda. En algunas ocasiones se debe a la batería, hay que asegurarse de que haga buen contacto con las láminas conductoras.

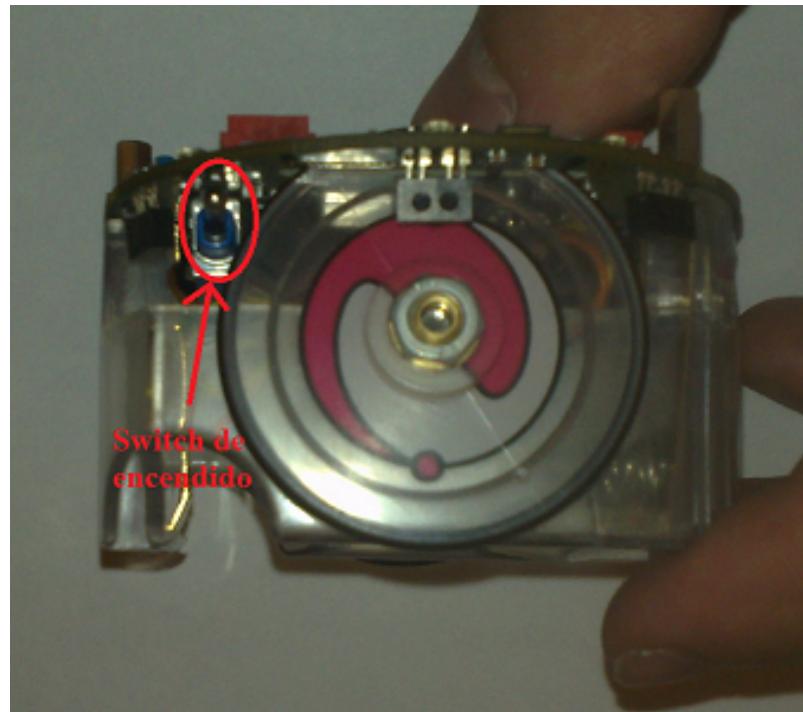


Figura B.1: Interruptor de encendido [17].

Una vez encendido el robot, se pueden elegir las funcionalidades por medio del switch selector blanco que se encuentra en la parte superior del robot, tal y como se muestra en la Figura B.2.

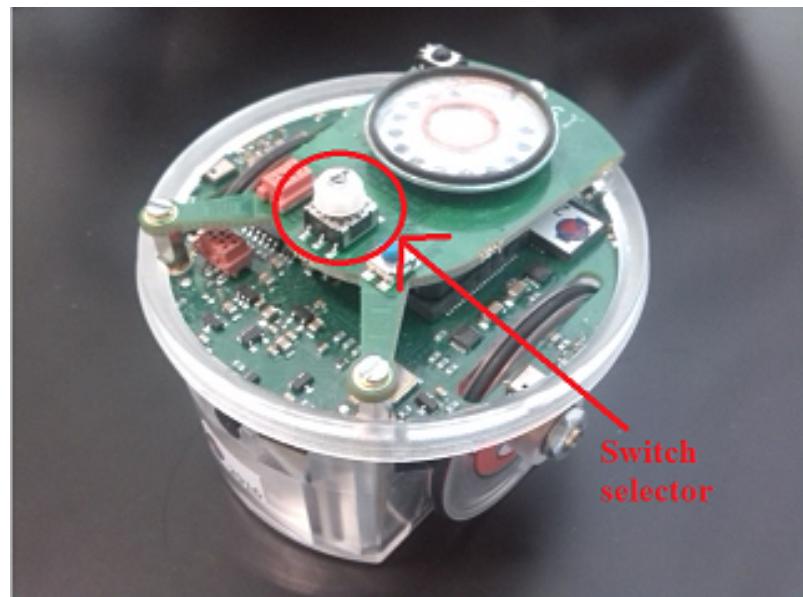


Figura B.2: Interruptor selector.

El switch selector cuenta con 16 posiciones y cada una de ellas con diferente funcionalidad. Tales funcionalidades son las siguientes:

- Posición 0. Detección de colisiones.
- Posición 1. Detección fuente de sonidos.
- Posición 2. Seguimiento de la pared.
- Posición 3. Protocolo sercom avanzado.
- Posición 4. Movimiento en trayectoria cuadrada.
- Posición 5. Sensor “display de retroalimentación”.
- Posición 6. Detección de fuente de luz con la cámara.
- Posición 7. Actúa como traductor RS232 a I2C.
- Posición 8. Muestra la dirección del suelo.
- Posición 9. Muestra la velocidad de rotación de los ejes del giroscopio.
- Posición 10. Ésta posición se usa para trabajar con la extensión “gumstix”.
- Posición 11. Configuración Bluetooth.
- Posición 12. Prueba global.
- Posición 13. Transmisor de UART1 a UART2.
- Posición 14. Seguir lo detectado por los 2 sensores de proximidad frontales.
- Posición 15. Comportamiento de limpiador automático.

B.2. Batería

Después de utilizar al robot por un tiempo, es posible que un led naranja (situado a un lado del led de encendido) se encienda. Esto significa que las baterías de los robots necesitan recargarse. Para esto se requiere de una cargador especial. En él se coloca una batería y se deja cargando por aproximadamente cuatro horas. Cuando el led rojo del cargador se apaga, significa que la batería se encuentra completamente cargada.

Las baterías que tienen los robots E-puck son los modelos anteriores a los del 2012, por lo que el aislante que tienen en la parte superior (el cual se ve en la Figura B.3) es muy frágil. Con el uso, este cartón se va desgastando, con lo que podría crear un cortocircuito, por lo que se recomienda empujar completamente los resortes al insertar la batería en el robot. Una solución dada en la página de GCtronic [17], dice que poniendo cinta de aislar sobre el cartón, y a su alrededor, puede ayudar a reforzarlo.



Figura B.3: Cartón aislante de la batería del E-puck [17].

B.3. Emparejamiento

Para poder comunicar al robot con la computadora es necesario emparejarlo. Para esto, se tiene que encender primeramente el robot. Después, se hace clic derecho en el ícono de *Bluetooth* que se encuentra en la barra de inicio, como se muestra en la Figura B.4.

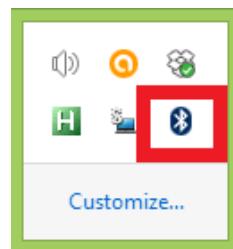


Figura B.4: Ícono de *Bluetooth*.

Aparecerá un menú, donde se tiene que hacer clic en la opción que dice “Mostrar dispositivos *Bluetooth*”, como se observa en la Figura B.5.

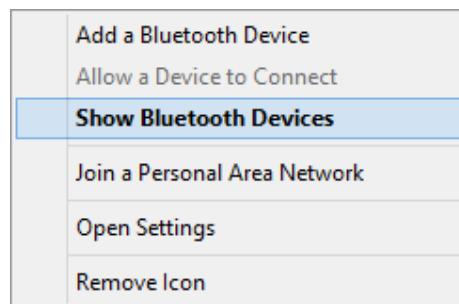


Figura B.5: Opciones del menú de *Bluetooth*.

Después, aparecerá una pantalla donde se muestran los dispositivos detectados por el *Bluetooth*. En esa pantalla, debe de aparecer el nombre E-puck seguido del número del robot. Al dar clic en emparejar, aparecerá otra pantalla para ingresar el *código de emparejamiento* de cuatro dígitos, el cual es el mismo del nombre del E-puck. Por ejemplo, si el nombre del robot aparece como *e_puck_2916*, el código es 2916. Al ingresar el código, se tiene que emparejar el robot con la computadora, con

lo que aparecerá una ventana como la que se ve en la Figura B.6, donde se observa que el robot ya se encuentra emparejado.

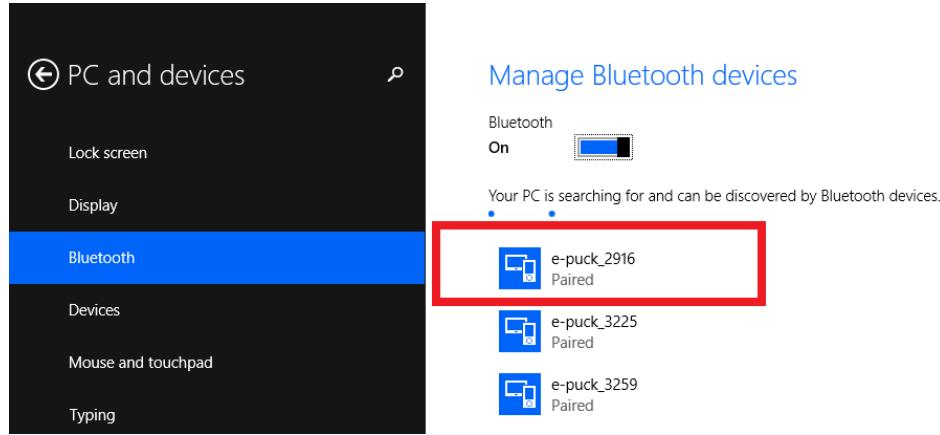


Figura B.6: Ventana de dispositivos.

Una vez emparejado, se hace clic derecho en el ícono de *Bluetooth* mostrado en la Figura B.4, donde se le da clic en la opción *abrir configuraciones*. Aparecerá una ventana, en la cual, al dar clic en la pestaña *Puertos COM* saldrá una lista de los puertos COM correspondientes de cada uno de los robots, tal y como se ve en la Figura B.7. El puerto COM a elegir para el programa de python es el que dice *outgoing* en *dirección*. Si no aparece, elegir el número inmediato anterior al que aparece en dirección *ingoing*.

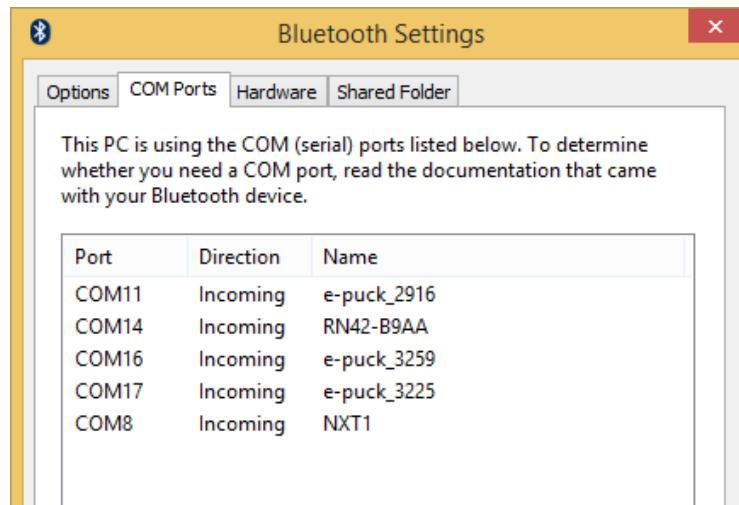


Figura B.7: Puertos COM.

B.4. Programación del robot E-puck

El robot se puede programar con distintos lenguajes como por ejemplo: ensamblador, C, python, Matlab, etc. Aquí se presenta solamente el lenguaje C, ya que además de ser relativamente sencillo, no requieren licencia para su uso, además de que con él es posible darle una mayor autonomía a los robots (que sean capaces de tomar sus propias decisiones), a diferencia de otros lenguajes (la computadora toma las decisiones).

B.4.1. Requerimientos

Para programar el robot con un nuevo *firmware* en lenguaje C es necesario instalar lo siguiente:

- MPLAB IDE [22].
- Compilador MPLAB C para PIC24 MCUs y dsPIC DSCs (compilador C30) [32].
- Tiny bootloader [10].

En el MPLAB IDE se pueden escribir programas propios. El compilador es necesario si se quiere programar en lenguaje C. El bootloader se utiliza para pasar el archivo hex generado en la compilación al pic del E-puck.

B.4.2. Creación de un proyecto nuevo

Existen varias páginas y tutoriales de universidades para crear un proyecto nuevo en MPLAB [34][58][52]. Se probaron varias maneras, pero ninguna funcionó (nunca se pudo compilar exitosamente un programa y la razón no está del todo clara, quizás por alguna configuración errónea) por lo que se optó por bajar los archivos demo de la página de GCtronic (Basicdemos.zip en el apartado de *software*). Estas demos son muy útiles para entender y empezar a programar los robots en lenguaje C.

Los pasos que se siguen para la compilación exitosa de un programa son los siguientes:

- Se comienza copiando uno de los proyectos demo para asegurar que se tengan todas las configuraciones necesarias.

- Después se abre el MPLAB IDE y se da clic en crear un proyecto nuevo.

La versión utilizada es la X y los proyectos de la página de GCtronic fueron hechos en la versión 8, por lo que en la ventana emergente al crear un proyecto nuevo, se tiene que elegir la opción de “Proyecto existente de MPLAB v8” en el apartado de “proyectos”. En el apartado de categorías se elige “Microchip embebido”. Estas configuraciones se ven en la Figura B.8.

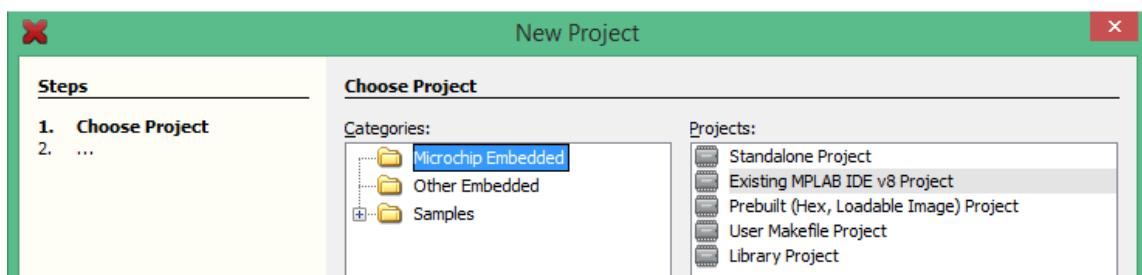


Figura B.8: Nuevo proyecto de MPLAB v8.

- Después, en la nueva ventana (Figura B.9), se selecciona el archivo .mcp que viene en las carpetas de los proyectos que se bajaron.

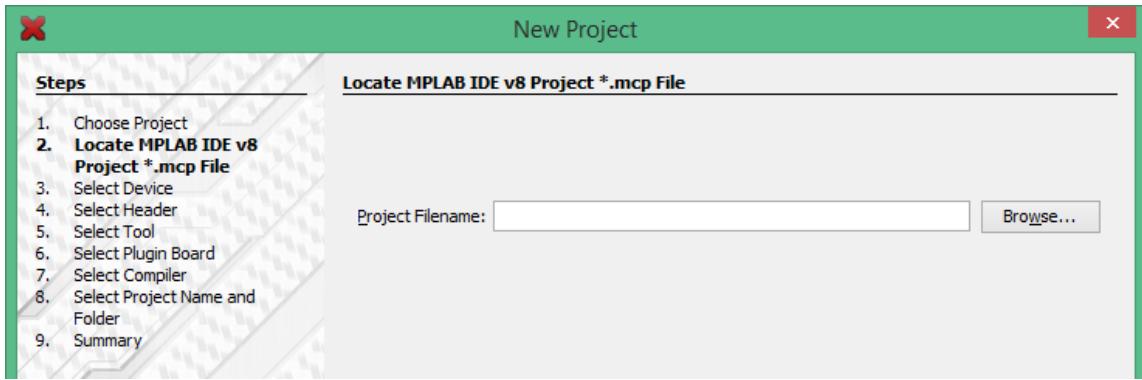


Figura B.9: Elección del archivo mcp.

- A continuación, en la ventana siguiente, se elige el microcontrolador del epuck: dsPIC30F6014A.
- Después, en la siguiente ventana (Figura B.10), en “Herramientas de hardware” se selecciona “Simulador”.

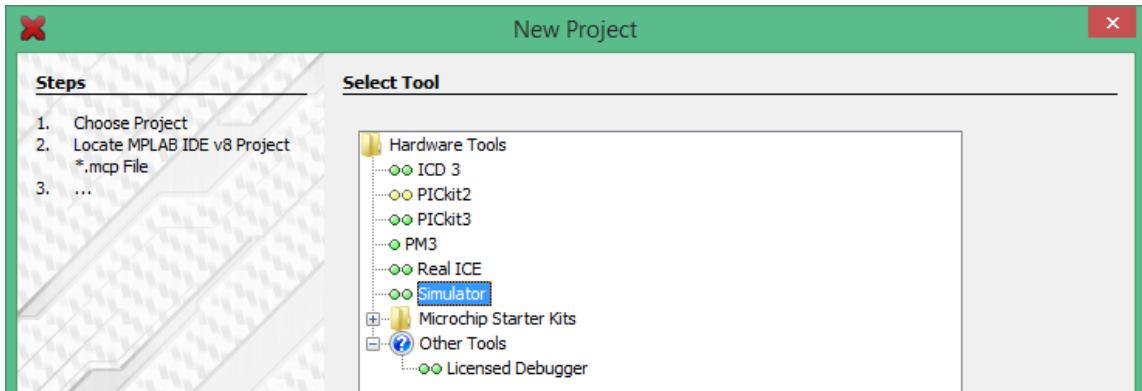


Figura B.10: Selección de la herramienta de simulación.

- También se tiene que elegir el compilador a utilizar. Si se instaló correctamente el mencionado previamente (C30) debería de aparecer. No confundir con el ASM30. Dicha selección se hace como se muestra en la Figura B.11.
- Por último se elige el nombre del proyecto y la dirección. Se recomienda guardar el proyecto en un directorio cercano al directorio raíz, ya que si el nombre se la dirección del directorio es demasiado largo, puede salir un error de compilación a la hora de hacer un programa.

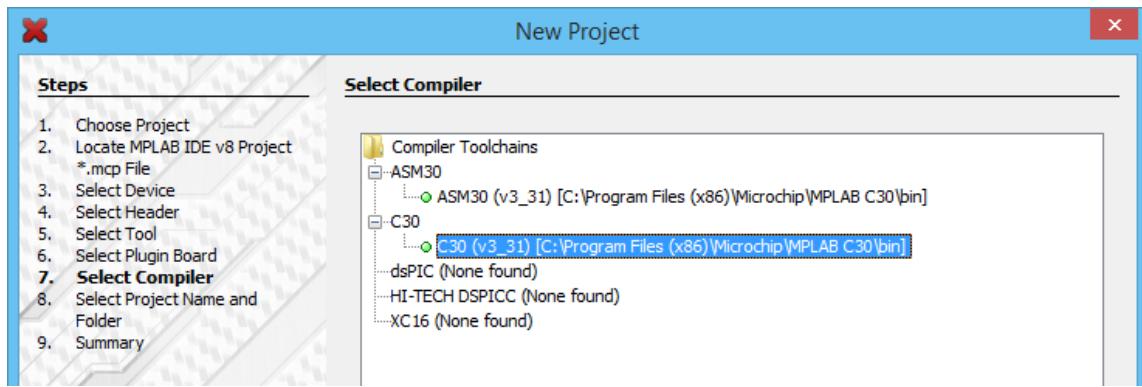


Figura B.11: Selección del compilador.

Una vez que se tiene un nuevo proyecto creado se agregan los *headers* y los *source files* necesarios para el proyecto (si es que no se encuentran actualmente). Para agregar archivos se le hace clic derecho a la carpeta (en el árbol de archivos del lado izquierdo de la pantalla), por ejemplo *Header files*, y se elige la opción agregar archivo existente para añadir el *header* al proyecto. Después se puede iniciar a escribir el programa.

Ya que se termina el programa se procede a compilar. La compilación se hace haciendo clic derecho en el nombre del proyecto de lado izquierdo (en el “árbol” del proyecto, las letras en negritas), y haciendo click en *Build*. Si la compilación tuvo éxito el archivo hex se guarda en la siguiente dirección:

... \Project_folder\X folder\dist\default\production\file_name.X.production

Este archivo es el que se utiliza en el *tiny bootloader* para escribir en la memoria del E-puck.

B.4.3. Escribir en la memoria del E-puck

En caso de que se quiera sobreescribir en la memoria del E-puck, simplemente se siguen los siguientes pasos:

- Asegurarse de que el robot se encuentre emparejado con la computadora y de tener disponible un archivo hex para escribir.
- Se abre el *tiny bootloader* (Figura B.12) y se conecta al puerto COM correspondiente (donde se encuentre conectado el robot en modo outgoing. Para ello se debe checar en ajustes de bluetooth/Puertos COM, como ya se mencionó anteriormente)

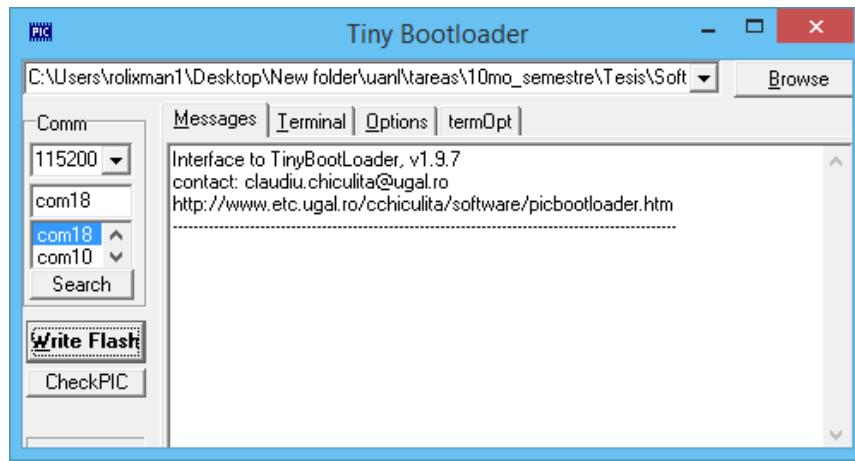


Figura B.12: *Tiny Bootloader*.

- Al elegir el archivo hex a subir, y después de haber seleccionado correctamente el puerto COM, se presiona el botón de *Write flash* para subir el programa al robot. En caso de que la computadora encuentre al robot se debe encender un led naranja que se observa en la Figura B.13 (por default dura 5 segundos encendido).
- Una vez que se prenda el led naranja en la parte de atrás, se debe de presionar el botón azul en la parte superior (el cual se muestra en la Figura B.14) y esperar a que termine de escribir en la memoria del robot. No es necesario dejar presionado el botón.
- Cualquier mensaje de éxito o fracaso en la escritura aparecerá en la pantalla del *tiny bootloader*.

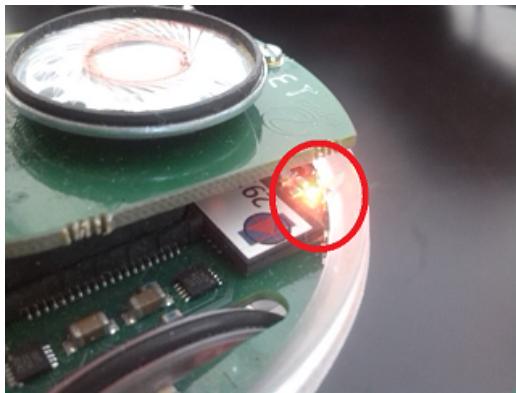


Figura B.13: Led de transmisión de datos.

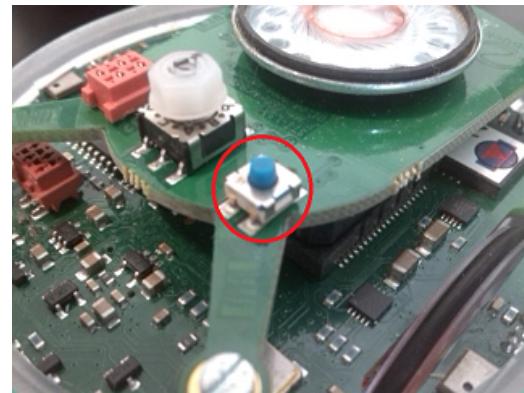


Figura B.14: Interruptor azul de escritura.

B.4.4. Ley de control programada

Para probar el funcionamiento del programa de detección del *script* de python, se programaron los epuck en lenguaje C. Considerando que “X1” y “Y1” son las coordenadas de la posición actual del robot y “X2” y “Y2” son las coordenadas de la posición deseada, en la Figura B.15 se muestra el diagrama de flujo escrito en la memoria del robot.

Básicamente el robot inspecciona constantemente el buffer del puerto serial (Comunicación bluetooth). Si existe un dato en él, va a llegar la siguiente cadena de caracteres desde la computadora: “X1 Y1 orientación X2 Y2” en formato de char.

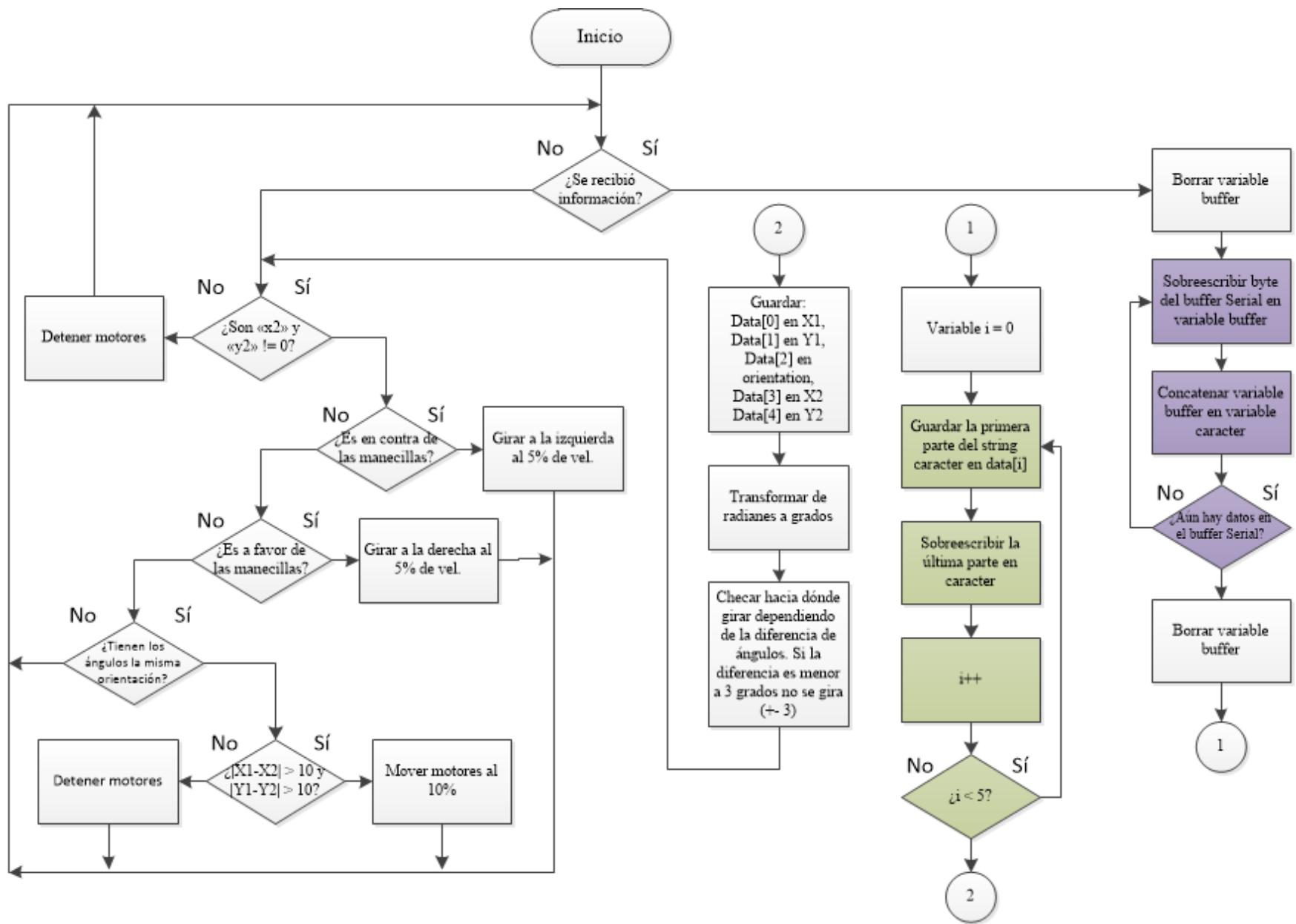


Figura B.15: Diagrama de flujo de la ley de control.

Cada char (con tamaño de un byte, así se transmite la información por el puerto serial y el *Bluetooth*) se guarda en una variable buffer y luego es concatenado en el string caracter. Cuando ya no haya datos en el buffer del puerto serial se termina la concatenación y el robot debe de tener en el string caracter el mismo formato enviado por la computadora: X1 Y1 orientación X2 Y2. La velocidad de transmisión y recepción de datos del e-puck (115200 baudios) debe de ser suficiente para que no se escriban más datos en una cadena de caracteres.

Después de obtener el string completo, este se va “cortando” en partes por medio del comando `strtol` (string to long) el cual “corta” el string hasta llegar a un espacio en blanco. La parte sobrante la guarda en este caso en la variable `end`, la cual se vuelve a sobreescibir en la variable carácter para volver a repetir el proceso cuatro veces más (ya que son 5 datos en total). Cuando ya se tienen los valores numéricos se guardan en las variables correspondientes:

X1=`data[0]`

Y1=`data[1]`

orientation=`data[2]`

X2=`data[3]`

Y2=`data[4]`

Al tener los datos en las variables se calcula, por medio de una función, el ángulo de la posición final con respecto a la posición inicial por medio de la pendiente.

La función `atan2l` entrega el ángulo en radianes como se muestra en la Figura B.16, empezando desde π en la horizontal de lado izquierdo y decrementando a favor de las manecillas del reloj hasta la horizontal de lado izquierdo (cero). De ahí empieza a seguir disminuyendo hasta llegar a $-\pi$.

Cuando se tiene el ángulo en radianes se transforma a grados (0-360) para que

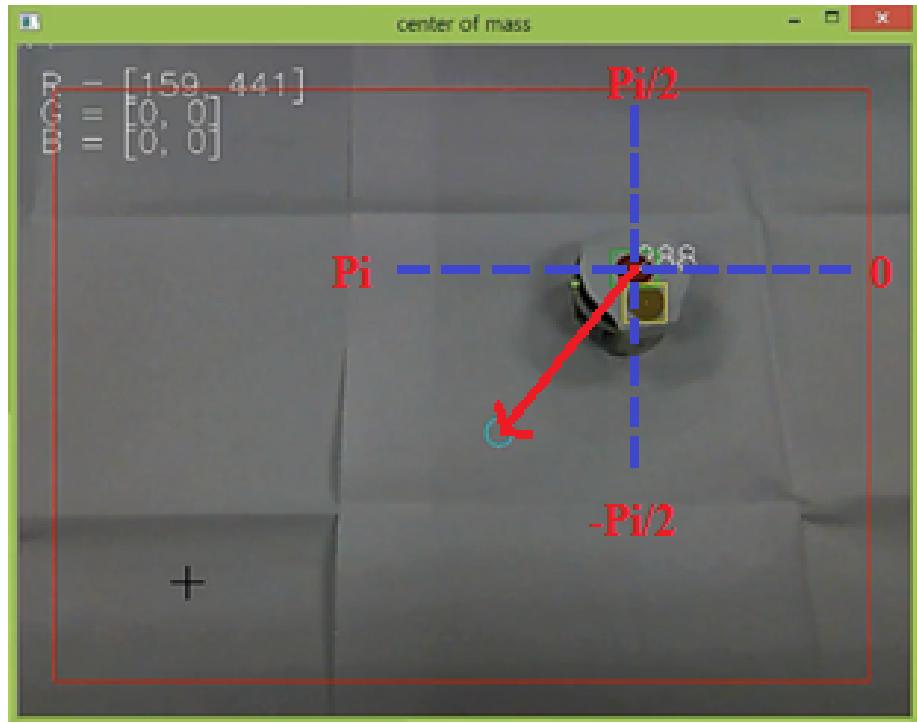


Figura B.16: Ángulo del comando atan2l.

tenga el mismo formato que el ángulo enviado por la computadora de la orientación del robot. La dirección de la rotación va a depender de la diferencia entre los ángulos (orientación del robot y el calculado de la posición final). Se toma un margen de error de ± 3 grados. En el programa en C que se muestra posteriormente se pueden ver las reglas que se tomaron para la dirección de la rotación. Una vez que se tiene la orientación (en el programa se denota como: 2 = en contra de las manecillas, 1 = a favor de las manecillas y 0 = se mantiene la dirección) el robot va a girar de acuerdo a la orientación dada al 5 % de su velocidad durante un corto periodo de tiempo. El tiempo de la variable `DELAY2`, la cual se puede observar en el programa en C, es el tiempo aproximado de $\frac{1}{4}$ de vuelta, por lo que el tiempo de giro (`DELAY2/180`) debe de dar como resultado un giro menor a un grado.

En la práctica es difícil de determinar cuánto gira exactamente. Por ejemplo, cuando se trató de hacer un giro de 90 grados por pasos (`DELAY2/90 * 90` pasos) se obtenían resultados diferentes a un giro continuo (`DELAY2 * 1` paso).

Cuando los ángulos coincidan (orientación = 0) se inspecciona el error de distancia existente entre la posición final y la actual, y si este error es mayor a 10 unidades (las mediciones se encuentran por lo pronto en píxeles) el robot se moverá hacia el punto final al 10% de la velocidad por un periodo corto de tiempo (la velocidad se establece por el usuario y varía de 1000 a -1000, siendo el signo la dirección de giro), el cual fue determinado a prueba y error y está sujeto a cambios.

B.4.5. Programa en C de la ley de control

```
*****  
** Programa que recibe información de una PC vía bluetooth en el  
siguiente **/  
** formato: X1 Y1 orientación X2 Y2. De acuerdo con ésta info. el **/  
** robot se mueve al punto (X2,Y2) desde su posición actual (X1,Y1)  
**/  
*****  
  
#include "p30f6014A.h"  
#include "e_epuck_ports.h"  
#include "e_init_port.h"  
#include "e_led.h"  
#include "e_uart_char.h"  
#include "e_motors.h"  
#include <string.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>  
#define DELAY2 5000000 // ~90 deg  
#define PI acos(-1) // PI  
  
//*****Declaración de variables*****  
char buffer[100], ending[100];  
char caracter[1000];  
int i;  
long data[5];  
char *end;  
long integer, position_x1=0, position_y1=0, angle1=0, timer;  
long position_x2=0, position_y2=0;  
long double angle_rad = 0;  
long double angle_deg = 0;  
int angle1_int = 0, angle2_int = 0, orientation = 0;
```

```

/*Declaración de funciones*/
int rad2deg(long posx1,long posy1,long posx2, long posy2);

int main() {
    //Inicialización
    e_init_port();           // Configuración de los puertos
    e_init_uart1();          // Inicializar UART a 115200 baud
        e_led_clear();
    e_set_body_led(0);
    e_set_front_led(0);

    while (1) {

        // ¿Existe información en el buffer?
        if(e_ischar_uart1()){
            sprintf(buffer, "\n",strlen(buffer)); // Reset de
                buffer con un salto de línea
            do{
                e_getchar_uart1(buffer); // Obtiene el primer
                    carácter y lo guarda en buffer
                strcat(caracter,buffer); // Concatena los
                    caracteres del Buffer en la variable caracter
                }while(e_ischar_uart1() != 0); // Se repite
                    hasta que ya no haya información en el
                    buffer del micro

                // Reset del Buffer
                sprintf(buffer, "\n",strlen(buffer)); // Agrega un
                    salto de línea al buffer vacío

                // Separación de la información. Se realiza 5 veces ya
                    que se reciben 5 datos.
                for(i=0;i<5;i++){
                    data[i] = strtol(caracter, &end, 10); // Convierte
                        de string a long int en base 10 hasta un espacio
                        en blanco
                    sprintf(caracter, "%s", end); // Guarda el
                        resto en la variable caracter
                }

                // Reset de caracter
                sprintf(caracter, "\n",strlen(caracter));

                // Haciendo eco de la información recibida. Se imprime
                    dato por dato.
                for(i=0;i<5;i++){
                    sprintf(caracter, "%lu",data[i]);
                    e_send_uart1_char(caracter, strlen(caracter));

```

```

        while(e_uart1_sending());
        sprintf(caracter, "\n",strlen(caracter));
        e_send_uart1_char(caracter, strlen(caracter));
        while(e_uart1_sending());
    }

    // Guardando la información en las variables
position_x1 = data[0];
position_y1 = data[1];
angle1 = data[2];
position_x2 = data[3];
position_y2 = data[4];

    // Obteniendo ángulos
angle2_int =
    rad2deg(position_x1,position_y1,position_x2,position_y2);
angle1_int = (int)angle1; // Convierte a int

    // Elijiendo la orientación de giro de acuerdo a la
    // diferencia entre ángulos
if (angle2_int > angle1_int) {
    if ((angle2_int - angle1_int) > 180) orientation =
        1; //Clockwise
    else orientation = 2; //Counterclockwise
}
if (angle2_int < angle1_int){
    if ((angle1_int - angle2_int) > 180) orientation =
        2; //Counterclockwise
    else orientation = 1; //Clockwise
}
if (abs(angle1_int - angle2_int) < 15) orientation = 0;
}

// Checando valores de posición iniciales. Solo se moverá
// si los valores son diferentes de 0
if ((position_x2 != 0) && (position_y2 != 0)){

    // Checando orientación y estableciendo la velocidad y
    // sentido de giro de los motores
if (orientation == 2){
    // Counterclockwise
    e_set_speed_left(-50);
    e_set_speed_right(50);
    for(timer=0;timer<(DELAY2/180);timer++); // Delay
} else if (orientation == 1){
    //Clockwise
    e_set_speed_left(50);
    e_set_speed_right(-50);
}
}

```

```

        for(timer=0;timer<(DELAY2/180);timer++); // Delay
    } else{
        // Detener giro
        e_set_speed_left(0);
        e_set_speed_right(0);
    }

    // Si los ángulos coinciden
    if (orientation == 0){
        // Checando si el error de posición es mayor a 10
        unidades y estableciendo la velocidad de los
        motores
        if ((abs(position_x1 - position_x2) > 10) &&
            (abs(position_y1 - position_y2) > 10)){
            e_set_speed_left(100);
            e_set_speed_right(100);
            for(timer=0;timer<(DELAY2/4);timer++); // Delay
        }else{
            // Stopping motors
            e_set_speed_left(0);
            e_set_speed_right(0);
        }
    }
}else{
    // Deteniendo motores si el error de posición es
    aceptable
    e_set_speed_left(0);
    e_set_speed_right(0);
}
}

// Función que, dados 2 pares de coordenadas (X,Y), calcula
// el ángulo entre ellos y lo regresa en grados.
int rad2deg(long posx1,long posy1,long posx2, long posy2){

    /*Declaración de variables locales*/
    long double rads = 0, deg_double = 0;
    long double y = posy2 - posy1;
    long double x = posx2 - posx1;
    int deg;

    // Casos especiales (division por 0)
    if ((x == 0)&&(posy1 < posy2)){
        rads = PI / 2; // Caso especial 1: 90 grados}
    else if((x == 0)&&(posy1 > posy2)){
        rads = -PI / 2; // Caso especial 2: 270 grados}
    else{
        rads = atan2l(y,x); // Default}
}

```

```
// Conversión del ángulo
deg_double = rads / ((2*PI) / 360.0);

// Si es negativo, agregar 360
if (posy1 > posy2){
    deg_double += 360;}
deg = (int)deg_double;
return deg;
}
```

Apéndice C

Tablas de datos de resultados

c210_60cm_cerca_20frames\corrida			
Corrida	t_{max}	t_{min}	\bar{t}
1	0.510278237	0.192242623	0.455646103
2	0.500357777	0.446169857	0.465958413
3	0.525636168	0.441836712	0.468886909
4	0.493950223	0.42013527	0.462919875
5	0.495509417	0.186328116	0.451973767

Cuadro C.1: Resultados preliminares 1 (resultados en segundos).

c210_60cm_lejos_20frames\corrida			
Corrida	t_{max}	t_{min}	\bar{t}
1	0.495278288	0.431908452	0.455989921
2	0.519267204	0.439444142	0.460756073
3	0.636826192	0.445317186	0.476771407
4	0.483672602	0.429939139	0.452953702
5	0.497697131	0.403805203	0.445710761

Cuadro C.2: Resultados preliminares 2 (resultados en segundos).

c210_130cm_cerca_20frames\corrida			
Corrida	t_{max}	t_{min}	\bar{t}
1	0.437115616	0.312755786	0.339355098
2	0.407390773	0.311724124	0.339252568
3	0.362070714	0.323369631	0.336084877
4	0.394446759	0.324545389	0.343401283
5	0.392355519	0.328394109	0.343846585

Cuadro C.3: Resultados preliminares 3 (resultados en segundos).

c210_130cm_lejos_20frames\corrida			
Corrida	t_{max}	t_{min}	\bar{t}
1	0.358197362	0.301232206	0.320037584
2	0.351284857	0.291623749	0.317787325
3	0.329690152	0.302566429	0.317500857
4	0.413369734	0.30673372	0.323989531
5	0.376307077	0.298878227	0.319896711

Cuadro C.4: Resultados preliminares 4 (resultados en segundos).

c525_60cm_cerca_20frames\corrida			
Corrida	t_{max}	t_{min}	\bar{t}
1	0.463087392	0.43043547	0.438349238
2	0.444762149	0.418188536	0.433297275
3	0.558758557	0.434994407	0.448503146
4	0.47647601	0.426506697	0.439945009
5	0.458970596	0.43519064	0.441317453

Cuadro C.5: Resultados preliminares 5 (resultados en segundos).

c525_60cm_lejos_20frames\corrida			
Corrida	t_{max}	t_{min}	\bar{t}
1	0.450223842	0.410068661	0.417929615
2	0.447561143	0.397693642	0.414590342
3	0.43623585	0.40783702	0.418660194
4	0.509623852	0.407934315	0.423693334
5	0.434961975	0.406690409	0.41831344

Cuadro C.6: Resultados preliminares 6 (resultados en segundos).

c525_130cm_cerca_20frames\corrida			
Corrida	t_{max}	t_{min}	\bar{t}
1	0.18013527	0.144134243	0.148827054
2	0.224700364	0.14255986	0.16761911
3	0.162036229	0.126826705	0.132078617
4	0.15894658	0.141464977	0.146367589
5	0.162808436	0.141589367	0.146083625

Cuadro C.7: Resultados preliminares 7 (resultados en segundos).

c525_130cm_lejos_20frames\corrida			
Corrida	t_{max}	t_{min}	\bar{t}
1	0.362111356	0.314863858	0.335229373
2	0.352375635	0.328506594	0.335639144
3	0.362809257	0.327051265	0.336801314
4	0.363470211	0.329798943	0.338359501
5	0.356175912	0.326375943	0.337072756

Cuadro C.8: Resultados preliminares 8 (resultados en segundos).