UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SISTEMA DE DETECCIÓN Y UBICACIÓN DE MARCADORES VISUALES PARA EL RASTREO DE ROBOTS MÓVILES

T E S I S

Que para obtener el título de

INGENIERO MECATRÓNICO

P R E S E N T A

EMMANUEL TAPIA BRITO

DIRECTOR DE TESIS:
DR. VÍCTOR JAVIER GONZÁLEZ VILLELA

MÉXICO D.F. **JULIO 2013**

Dedicatoria

A mis padres, por su incondicional apoyo, por ser fuente de inspiración, por su esfuerzo derramado día a día y por aguantarme durante todos estos años.

*

A Getsemaní por llenar de inmenso amor y alegría mi corazón y por haber compartido conmigo una gran etapa de mi vida.

*

A Sofía por brindarme tu apoyo y el amor que me dio ánimos para alcanzar esta meta.

*

A todos mis amigos por haber formado parte de mi vida universitaria.

Agradecimientos

A la Universidad Nacional Autónoma de México por la excelente educación recibida.

*

A los profesores de la Facultad de Ingeniería, en especial a los del Departamento de Ingeniería Mecatrónica, a quienes debo mi formación profesional.

*

Al grupo de trabajo MRG, por su soporte técnico y ayuda en la realización de esta Tesis.

*

Al Dr. Víctor Javier González Villela por la oportunidad, motivación y confianza brindada durante la realización de este proyecto.

*

A Oscar X. Hurtado Reynoso, por su gran ayuda en la etapa de pruebas con el robot.

*

A la DGAPA por el apoyo brindado a través del proyecto PAPIIT 1N115811 con título "Investigación y desarrollo en sistemas mecatrónicos: robótica móvil, robótica paralela, robótica híbrida y teleoperación".

Resumen

Esta tesis presenta los puntos principales del desarrollo de un sistema de visión por computadora con el objetivo de emplearlo como medio de asistencia para el control de robots móviles. El objetivo de este sistema es distinguir marcadores visuales a partir de su color y área, utilizando visión por computadora, una cámara web y algoritmos de reconocimiento e identificación de regiones conectadas de un determinado color. Con el propósito de ubicar, en un sistema de referencia, dentro de un espacio de trabajo, robots móviles. Para ello, se utilizan transformaciones de calibración y corrección de planos, desarrollando software en C++, empleando librerías de OpenCV y Simulink. Una vez desarrollado el software de visión se diseñó una prueba experimental, tomando en consideración los requerimientos correspondientes. La prueba se realizó utilizando un robot móvil dirigido mediante un diferencial eléctrico tipo (2,0). Por último, se documentan los resultados obtenidos para futuras referencias.

Índice general

1. Introducción **15**

1.1. Generalidades 15

1.1.1. Robótica Móvil 16

1.1.2. Visión por computadora 16

1.1.2.1. Visual Servoing 18

1.1.2.2. Precedentes de la visión por computadora. 19

1.1.2.3. Ventajas de la visión por computadora 19

1.1.2.4. Utilidad de la visión por computadora 20

1.1.2.5. Ejemplos de estado del arte y tendencias 21

1.2. Objetivos y descripción de los capítulos 25

1.3. Desarrollo Previo 26

2. Sistemas de detección de marcadores visuales **27**

2.1. Software de visión 27

2.1.1. ReacTIVision 27

2.2. Comparación entre plataformas de desarrollo de sistemas de visión . . 30

2.2.1. OpenCV 31

2.2.2. SimpleCV 31

2.2.3. AForge.NET 32

2.2.4. MatLab® 32

2.2.5. Otras plataformas 32

2.3. Selección de la plataforma de desarrollo 32

3. Arquitectura del sistema de detección y ubicación de marcadores visuales **35**

3.1. Funcionamiento de la visión por computadora 35

3.1.1. Adquisición y digitalización de la imagen 36

3.1.2. Preprocesamiento 36

3.1.3. Segmentación 36

3.1.4. Descripción 37

3.1.5. Reconocimiento y toma de decisiones 37

3.2. Configuración preliminar del sistema de visión 38

3.2.1. Tipo de segmentación a emplear 38

3.2.2. Distribución de los elementos 38

4. Algoritmo de discriminación de colores y áreas 41
4.1. Discriminación de marcadores visuales 41
4.2. Etiquetado de componentes conectados 41
4.3. Vecindad entre pixeles 42
4.4. Algoritmo 43
4.4.1. Two-pass 43
4.4.1.1. Ejemplo del algoritmo 45

5. Calibración de la cámara 51
5.1. Aberraciones 51
5.1.1. Tipos de aberraciones 52
5.1.1.1. Aberración esférica 53
5.1.1.2. Coma 53
5.1.1.3. Astigmatismo 54
5.1.1.4. Curvatura de campo 56
5.1.1.5. Distorsión 56
5.1.1.6. Aberración cromática 58
5.1.2. Discusión 59
5.2. Calibración 60
5.2.1. Tipos de calibración 60
5.2.2. Selección del tipo de calibración 61
5.3. Modelado de la cámara 61

6. Rastreo y mapeo de marcadores visuales 65
6.1. Restricciones de Visión 65
6.2. Eliminación de distorsión 67
6.3. Detección y ordenación de los marcadores visuales 68
6.4. Definición del area de trabajo 71
6.5. Mapeo de los marcadores visuales 72
6.6. Comunicación UDP 74
6.7. Archivo .xml 75

7. Experimento de rastreo sobre un robot móvil 77
7.1. Descripción de los elementos requeridos para el experimento 78
7.1.1. Cámara 78
7.1.2. Robot 78
7.2. Descripción general del sistema 79
7.2.1. Etapas del flujo de información 79
7.2.1.1. Captura de la imagen 79
7.2.1.2. Envío de datos al programa de control 80
7.2.1.3. Programa de control en Simulink 81

7.2.1.4. Transmisión de instrucciones de Simulink al microcontrolador . . . 82
7.2.1.5. Comando del microcontrolador hacia la tarjeta MD-25 83
7.3. Condiciones para las pruebas . . . 84
7.3.1. Entorno de pruebas . . . 84
7.3.2. Operación del sistema . . . 85
7.4. Objetivo de las pruebas . . . 86
7.5. Descripción de las pruebas . . . 86
7.5.1. Prueba de precisión . . . 86
7.5.2. Prueba de lazo abierto . . . 86
7.5.3. Prueba de lazo cerrado . . . 87

8. Resultados **89**
8.1. Resultados de la prueba de precisión . . . 89
8.2. Resultados de la prueba de lazo abierto . . . 90
8.3. Resultados de la prueba de lazo cerrado . . . 93

9. Conclusiones y trabajo a futuro **97**
9.1. Conclusiones . . . 97
9.2. Trabajo a futuro . . . 98

Bibliografía **98**

Índice de figuras

1.1. Relación entre visión por computadora y otros campos. 17
1.2. Robots auxiliares médicos con sistemas de visión 21
1.3. *Curiosity,* un robot de exploración de la NASA en Marte, en 2012 . . 22
1.4. Sensor Kinect y ejemplo de uso 23
1.5. Realidad aumentada mediante Wikitude 24
1.6. Ejemplo de uso de Google Glass 25

2.1. Marcadores visuales *fiducials* 28
2.2. *Fiducials* en elementos físicos para usarse en la Reactable [22] 28
2.3. Componentes de la Reactable 29
2.4. Comparación entre las principales plataformas de desarrollo 33

3.1. Etapas fundamentales del procesamiento digital de imágenes [10] . . . 35
3.2. Disposición de los elementos 39

4.1. Pixeles adyacentes 42
4.2. (a) Conectividad-4, (b) Conectividad-8 42
4.3. Imagen de entrada 45
4.4. Imagen etiquetada por el algoritmo 45
4.5. Imagen de entrada representada como matriz 46
4.6. Etiquetado del primer pixel 46
4.7. Etiquetado de un pixel a la derecha de un pixel de fondo 47
4.8. Etiquetado del primer renglón 47
4.9. Etiquetado del segundo renglón 48
4.10. Etiquetado del pixel (3,4) 48
4.11. Procedimiento de etiquetado de la primer barrida 49
4.12. Reetiquetado de un pixel con etiqueta heredada 49
4.13. Reetiquetado de la imagen hasta la cuarta fila 50
4.14. Reetiquetado de la imagen hasta la séptima fila 50
4.15. Reetiquetado final de la imágen 50

5.1. Aberración esférica. 53
5.2. Efecto de la aberración esférica 53
5.3. Aberración de coma 54
5.4. Efecto de la aberración de coma 54

5.5. Aberración de astigmatismo 55
5.6. Efectos del astigmatismo 55
5.7. Aberración Curvatura del campo 56
5.8. Efectos de la curvatura del campo 56
5.9. Distorsiones de corsé y barril 57
5.10. Aberración de distorsión 57
5.11. Efectos de la distorsión 58
5.12. Aberración cromática 58
5.13. Efectos de la aberración cromática 59
5.14. Modelado de la cámara 62

6.1. Mapa del modelo HSV 66
6.2. Imágenes de captura y segmentación 67
6.3. Imágenes de captura y corrección de la distorsión 69
6.4. Representación del sistema de referencia 72
6.5. Vistas de un marcador visual desde la cámara 73

7.1. Plataforma y Robot acoplado a la Plataforma 77
7.2. Cámara utilizada para el experimento 78
7.3. Robot utilizado para la prueba 79
7.4. Diagrama esquemático del sistema 80
7.5. Protocolo de la cadena de datos enviada 81
7.6. Módulos *XBee* 82
7.7. Tarjeta MD-25 y sus componentes 84
7.8. Banco de pruebas en laboratorio 85
7.9. Esquema de la prueba de lazo abierto 87
7.10. Esquema de la prueba de lazo cerrado 87

8.1. Prueba de Precisión 90
8.2. Desplazamiento de la simulación 91
8.3. Desplazamiento del Robot 92
8.4. Gráfica comparativa entre el desplazamiento simulado y el capturado 93
8.5. Desplazamiento de la simulación con control realimentado 94
8.6. Desplazamiento del robot con control realimentado 95
8.7. Gráfica comparativa entre las posiciones enviadas y las recibidas . . 96

Capítulo 1

Introducción

1.1. Generalidades

Acuñada en 1969 por el ingeniero japonés Tetsuro Mori [16], la palabra *mecatrónica* ha sido definida de varias maneras. Un consenso común es describir a la mecatrónica como una disciplina integradora de las áreas de mecánica, electrónica e informática cuyo objetivo es proporcionar mejores productos, procesos y sistemas. La mecatrónica no es, por tanto, una nueva rama de la ingeniería, sino un concepto recientemente desarrollado que enfatiza la necesidad de integración y de una interacción intensiva entre diferentes áreas de la ingeniería.

Con base en lo anterior, se puede hacer referencia a la definición de mecatrónica propuesta por J.A. Rietdijk:"Mecatrónica es la combinación sinérgica de la ingeniería mecánica de precisión, de la electrónica, del control automático y de los sistemas para el diseño de productos y procesos"[24].

Un sistema mecatrónico típico recoge señales, las procesa y, como salida, genera fuerzas y movimientos. Los sistemas mecánicos son entonces extendidos e integrados con sensores, microprocesadores y controladores. Los robots, las máquinas controladas digitalmente, los vehículos guiados automáticamente y las cámaras electrónicas pueden considerarse como productos mecatrónicos. Al aplicar una filosofía de integración en el diseño de productos y sistemas se obtienen ventajas importantes como son mayor flexibilidad, versatilidad, nivel de "inteligencia" de los productos, seguridad y confiabilidad así como un bajo consumo de energía. [6]

La robótica es la rama de la tecnología que se encarga del diseño, construcción, operación y aplicación de los robots. [7] Un robot es usualmente un dispositivo electromecánico que puede realizar tareas automáticamente. Algunos robots requieren cierto grado de guía, el cuál puede ser mediante control remoto o una interfaz de computadora. Los robots pueden ser autónomos, semiautónomos o remotamente controlados. Una aplicación de la ingeniería mecatrónica es la robótica, y la robótica móvil.

1.1.1. Robótica Móvil

La robótica móvil es un área de la robótica que se encarga de estudiar a aquellos robots que no tienen ningún eslabón anclado a un medio físico no móvil como podría ser la tierra. Los hay acuáticos, voladores y terrestres. Todos ellos, por la manera en que están configurados, y por la intrínseca relación que guardan con las disciplinas de la ingeniería mecánica, electrónica y el control, son considerados productos mecatrónicos de forma inherente [11].

Para los robots móviles (no sólo los terrestres) se tienen tres problemáticas principales a resolver para poder desplazarse: la cinemática del vehículo, que define las capacidades de movilidad propias del mismo; la definición del camino o la trayectoria a seguir y la percepción del entorno para ubicar al robot en su espacio de trabajo. En un caso totalmente manual, los últimos dos rubros son procesados por el usuario y el grado de automatización del vehículo puede ir subiendo, hasta que los tres puntos sean resueltos enteramente por el robot utilizando sensores, procesando e interpretando las señales de éstos para alcanzar el objetivo definido por el usuario.

En el presente proyecto, únicamente se busca una solución para la percepción del entorno y localización de un robot móvil dentro de su espacio de trabajo. Para ello se propone un sistema auxiliar basado en visión por computadora.

1.1.2. Visión por computadora

Los términos *visión por computadora*, *visión artificial* o *visión de máquina*, por lo general se refieren a todo sistema informatizado que mediante una cámara, captura la imagen de un objeto determinado o una escena y procede a la identificación de diferentes parámetros como el color, la textura, la forma, la estructura interna o externa de los objetos, entre otros.

Actualmente, los sistemas basados en visión por computadora se han vuelto muy populares. Esto responde, entre otras cosas, al hecho de que las cámaras web y similares, son cada vez más baratas, pequeñas y con mejores características de captura de imagen. Gracias a ello, se ha desarrollado mucha tecnología que emplea la visión por computadora (o sus algoritmos), como herramienta fundamental. El objetivo de la visión por computadora es percibir "el mundo detrás de la imagen". Un sistema de visión por computadora procesa las imágenes obtenidas de una cámara (o un conjunto de cámaras), y extrae la información significativa de las imágenes adquiridas. Se trata de imitar al sistema de visión humana en la que el cerebro procesa las imágenes procedentes de los ojos.

Los sistemas de visión artificial tienen relación directa con otras ramas de investigación. En la figura 1.1 se muestran algunas de estas ramas y subramas. Más adelante se explica brevemente cómo se llevan a cabo algunas de estas relaciones.

La visión artificial tiene relación con, por ejemplo, las áreas de la inteligencia artificial relacionadas con la planificación autónoma o deliberación de los sistemas robóticos para navegar dentro de un entorno. Una comprensión detallada de estos

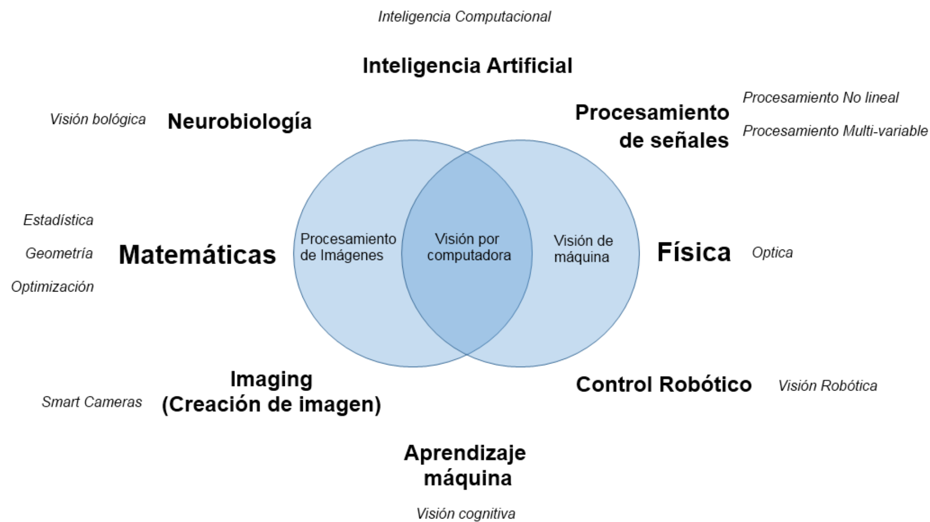


Figura 1.1: Relación entre visión por computadora y otros campos.

entornos es necesario para navegar a través de ellos. Información sobre el ambiente podría ser proporcionada por un sistema de visión por computadora, actuando como un sensor de visión y proporcionar información de alto nivel sobre el mismo ambiente y el robot. La inteligencia artificial y visión por computadora comparten otras áreas como el reconocimiento de patrones y técnicas de aprendizaje. En consecuencia, la visión artificial es en ocasiones visto como una parte del campo de la inteligencia artificial o el campo de ciencias de la computación en general.

La Física es otro campo que está estrechamente relacionado con la visión artificial. La mayoría de los sistemas de visión por computadora se basan en sensores de imagen, que detectan la radiación electromagnética que se encuentra típicamente en la forma de luz visible o infrarroja. Los sensores están diseñados utilizando física del estado sólido. El proceso por el que la luz interactúa con las superficies se explica utilizando la física. La Física además, explica el comportamiento de la óptica que es una parte fundamental de la mayoría de los sistemas de formación de imágenes. Inclusive, sofisticados sensores de imagen llegan a requerir de la mecánica cuántica para proporcionar una comprensión completa del proceso de formación de la imagen. Además, diversos problemas de medición en la física puede ser abordada utilizando la visión por computadora, por ejemplo el movimiento en fluidos.

Un tercer campo que juega un papel importante es la neurobiología, específicamente el estudio del sistema de visión biológica. Durante el último siglo, ha habido un amplio estudio de los ojos, las neuronas, y las estructuras cerebrales dedicadas al procesamiento de los estímulos visuales en humanos y diversos animales. Como resultado, se ha creado a un subcampo dentro de la visión por computadora, donde los sistemas artificiales están diseñados para imitar el procesamiento y el comportamiento de los sistemas biológicos, en los diferentes niveles de complejidad.

Algunas corrientes de investigación de visión por computadora están estrechamente relacionados con el estudio de la visión biológica. De hecho, muchas líneas de investigación en inteligencia artificial están estrechamente vinculados con la investigación de la conciencia humana, y el uso del conocimiento almacenado para interpretar, integrar y utilizar la información visual.

Otro campo relacionado con la visión por computadora es el procesamiento de señales. Sin embargo, debido a la naturaleza específica de las imágenes, hay muchos métodos desarrollados dentro de la visión por computadora que no tienen contraparte en el procesamiento de señales de una sola variable. Junto con la multidimensionalidad de la señal, esto define un subcampo en el procesamiento de la señal como parte de la visión por computadora.

Por último, los campos más estrechamente relacionados con la visión por computadora son el procesamiento de imágenes y análisis de imágenes de visión artificial. Las técnicas básicas que se utilizan y desarrollan en estos campos son más o menos idénticos, algo que puede ser interpretado como que sólo hay un campo con nombres diferentes. Sin embargo, parece ser necesario que los grupos de investigación, publicaciones científicas, congresos y empresas se presenten a sí mismos como pertenecientes específicamente a uno de estos campos.

La visión por computadora es, en cierto modo, lo opuesto de los gráficos por computadora. Si bien los gráficos de computadora producen datos de imágenes a partir de modelos 3D, la visión por ordenador a menudo produce modelos 3D a partir de datos de imágenes. También hay una tendencia hacia una combinación de las dos disciplinas, por ejemplo, como se analiza en la realidad aumentada.

1.1.2.1. Visual Servoing

El término *Visual Servoing* (nombrado en español Control Visual Servo y abreviado VS) se refiere específicamente al control de un robot basado en visión. Es una técnica que utiliza la información de retroalimentación extraída de un sensor de visión para controlar los movimientos de un robot. Una clasificación fundamental de los diferentes enfoques del VS es el diseño del bucle de control. Generalmente se emplean dos esquemas:

El primer esquema se denomina *Direct Visual Servoing* ya que el controlador basado en visión directamente computa la entrada del sistema dinámico. Esto hace que el sistema de VS sea llevado de forma muy rápida.

El segundo esquema de control es el denominado *Indirect Visual Servoing* ya que el control basado en visión computa una referencia de la ley de control que se envía a un controlador de más bajo nivel y este al sistema dinámico. Muchos de los enfoques de la literatura se centran en este esquema y lo denominan "*Dynamic look-and-move*". En este caso, el controlador *servoing* (el de más bajo nivel) debe ser más rápido que el visual servoing.

El control basado en visión puede clasificarse dependiendo del error utilizado en la ley de control, en cuatro grupos: "*position-based*" (basados en posición), "*image-*

based" (basados en la imagen), "*hibrid-based*" (basados hibridamente) y "*motion-based*" (basados en el movimiento).

En un sistema de control "*position-based*", el error se calcula en el espacio Cartesiano 3D (por esta razón este enfoque también se denomina "*3D Visual Servoing*"). En un sistema de control "*image-based*", el error que se calcula es el del espacio de la imagen 2D (por esta razón se llama "*2D Visual Servoing*"). Recientemente han sido probados nuevos enfoques denominados "*Hibrid Visual Servoing*". Por ejemplo se ha propuesto uno llamado $2\frac{1}{2}$ D Visual Servoing ya que el error se expresa en parte en coordenadas Cartesianas 3D y en parte en el espacio de la imagen 2D. Finalmente hay un enfoque llamado "*motion-based*" cuyo error computado es el "optical flow" [1] medido en una imagen y un "optical flow" de referencia que debe obtener el sistema.

1.1.2.2. Precedentes de la visión por computadora.

Los sistemas de visión artificial comenzaron a aparecer iniciado el decenio de 1950, asociados con aplicaciones no robóticas, como lectura de documentos, conteo de especímenes biológicos en la sangre, y reconocimiento aerofotográfico de aplicación militar. En el campo propiamente robótico el desarrolló comenzó a mediados del decenio iniciado en 1960, con el establecimiento de laboratorios de inteligencia artificial en instituciones como el MIT, la U. de Stanford y el Stanford Research Institute. [23]

Las mejoras en los métodos de captura y procesamiento para las imágenes digitales transmitidas continuaron durante los siguientes treinta y cinco años. Sin embargo, fue el advenimiento combinado de las computadoras digitales de gran potencia y del programa espacial lo que puso de manifiesto el potencial de los conceptos del tratamiento digital de imágenes. La tarea de usar técnicas computacionales para digitalizar, enviar y mejorar las imágenes recibidas de una sonda espacial se inició en el Laboratorio de Propulsión Espacial (en Pasadena, California) en 1964 cuando las imágenes de la Luna transmitidas por el Ranger 7 fueron procesadas por una computadora para corregir diversos tipos de distorsión de la imagen inherentes a la cámara de televisión de a bordo. [10]

Para 1980 muchas compañías estaban desarrollando sistemas de visión para robots y para otras aplicaciones industriales. Así, el desarrollo de los sistemas de visión artificial, ha encontrado múltiples aplicaciones hasta la actualidad.

1.1.2.3. Ventajas de la visión por computadora

Los sistemas de visión artificial completan tareas de inspección con un alto nivel de flexibilidad y repetibilidad, nunca se cansan, ni se aburren ni se distraen y pueden ser puestos a trabajar en ambiente donde los inspectores humanos no podrían

[1] El "optical flow" o flujo óptico es el patrón del movimiento aparente de los objetos, superficies y bordes en una escena causado por el movimiento relativo entre un observador (un ojo o una cámara) y la escena. [3]

trabajar por condiciones de seguridad.

Para un inspector humano, los ojos proporcionan información del ambiente que lo rodea, el cerebro interpreta lo que los ojos ven basado en experiencias previas con objetos similares. Basándose en esta interpretación, se toman decisiones y acciones. En forma similar, los sistemas de visión artificial ven al objeto, lo interpretan y toman decisiones.

La implementación de estos sistemas en una empresa mejora la calidad de la producción, refuerza la seguridad del lugar de trabajo, reduce o elimina los cuellos de botella de la inspección y reduce los costos; además de las anteriores, la aplicación de la visión por computadora en la realización de trabajos de inspección tiene como ventaja conseguir velocidades de operación y fiabilidad mejores que en sistemas convencionales. Todos estos elementos llevan a las industrias a posicionarse en un alto nivel competitivo.

1.1.2.4. Utilidad de la visión por computadora

Es preciso reconocer que hoy por hoy la visión por computadora a veces no es la mejor solución a un problema. Existen muchas ocasiones en las que el problema es tan complejo que la solución humana es lo mejor. Por ejemplo la conducción de un vehículo en una carretera con tráfico intenso. Pero a veces, las soluciones humanas tienden a ser inexactas o subjetivas y en ocasiones lentas y presentan una ausencia de rigor así como una pobre percepción. A pesar de ello, la solución humana es menos estructurada que la solución artificial y muchos problemas de visión por computadora requieren un nivel de inteligencia mucho mayor que el que la máquina pueda ofrecer. El sistema de visión humana puede describir automáticamente una textura en detalle, un borde, un color, una representación bidimensional de una tridimensional, ya que puede diferenciar entre imágenes de diferentes personas, firmas, colores, etc., puede vigilar ciertas zonas, diagnosticar enfermedades a partir de radiografías, etc. Sin embargo, aunque algunas de estas tareas pueden llevarse a cabo mediante visión artificial, el software o el hardware necesario no consigue los resultados que serían deseables.

La visión por computadora es un tema rico y gratificante para el estudio y la investigación. Cada vez más, tiene un futuro comercial. En la actualidad hay muchos sistemas de visión en el uso industrial de rutina: cámaras para examinar partes mecánicas, inspeccionar la calidad de la comida, y las imágenes utilizadas en beneficio de la astronomía de las técnicas de visión por computadora. Estudios forenses y los datos biométricos (maneras de reconocer a las personas) mediante la visión por computadora, incluyendo el reconocimiento facial automático y la gente reconoce por la "textura" de sus iris. Estos estudios están desarrollándose en paralelo también por biólogos y psicólogos que investigan sobre el funcionamiento de nuestro sistema humano de la visión, y la forma en que vemos y reconocemos los objetos.

1.1.2.5. Ejemplos de estado del arte y tendencias

En lo que respecta a la ingeniería, dentro de las aplicaciones de los sistemas basados en visión por computadora, hoy en día se pueden mencionar bastantes.

Uno de los campos de aplicación más importantes es la visión de equipo médico o de procesamiento de imágenes médicas. Esta área se caracteriza por la extracción de información de datos de una imagen con el propósito de establecer un diagnóstico médico de un paciente. En general, los datos de la imagen se encuentran en forma de imágenes de microscopía, imágenes de rayos X, imágenes de angiografía, imágenes de ultrasonido e imágenes de tomografía. Un ejemplo de información que puede ser extraída de los datos de imagen tales es la detección de tumores, arteriosclerosis u otros cambios malignos. También pueden ser mediciones de dimensiones de órganos, el flujo sanguíneo, etc. Esta área de aplicación también apoya la investigación médica al proporcionar nueva información, por ejemplo, acerca de la estructura del cerebro, o sobre la calidad de los tratamientos médicos.

Figura 1.2: Robots auxiliares médicos con sistemas de visión

En enero del 2013, la compañía General Electric presentó avances del desarrollo del robot OR'bot (de Operation-Room, ver figura 1.2[2]). Un robot de tipo hibrido diseñado para organizar el funcionamiento de la sala de operaciones, la esterilización y preparación general del instrumental antes de la cirugía. El robot mezcla tecnologías de escaneo de código de barras y la visión por computadora para localizar artículos y darles el tratamiento adecuado para la asistencia directa al equipo médico humano.

Una segunda área de aplicación en la visión por computadora es en la industria, donde la información se extrae para el propósito de soportar un proceso de fabricación. Un ejemplo es el control de calidad donde los detalles o productos finales están siendo automáticamente inspeccionados con el fin de encontrar defectos. Otro ejemplo es la medición de la posición y la orientación de los datos para ser recogido por

[2] http://www.computervisiononline.com/blog/don %E2 %80 %99t-worry-robot-will-clean-scalpel

un brazo de robot. La visión artificial es también muy utilizada en el proceso agrícola para eliminar materia del alimento indeseable de material a granel, un proceso denominado selección óptica.

Las aplicaciones militares son probablemente una de las mayores áreas de aplicación de visión por computadora. Los ejemplos obvios son la detección de los soldados enemigos o vehículos y la orientación de misiles. Modernos conceptos militares, como el de "conciencia del campo de batalla", implican que varios sensores, incluyendo sensores de imagen, ofrecen un amplio conjunto de información sobre una escena de combate que pueden ser utilizados para apoyar las decisiones estratégicas. En este caso, el procesamiento automático de los datos se utiliza para reducir la complejidad y para fusionar la información de múltiples sensores para aumentar la fiabilidad.

Figura 1.3: *Curiosity,* un robot de exploración de la NASA en Marte, en 2012

Una de las áreas de aplicación más recientes son los vehículos autónomos, los cuales incluyen los sumergibles, vehículos terrestres (pequeños robots con ruedas, coches o camiones), vehículos aéreos y vehículos aéreos no tripulados (UAV por sus siglas en inglés). Vehículos totalmente autónomos, típicamente utilizan la visión por computadora para la navegación, es decir, para saber dónde se encuentra, o para producir un mapa de su entorno (SLAM) y para la detección de obstáculos. También se puede utilizar para la detección de ciertos eventos específicos de la tarea, por ejemplo, un UAV en busca de los incendios forestales. Hay muchos ejemplos de vehículos militares autónomos que van desde misiles avanzados, a los vehículos aéreos no tripulados para misiones de reconocimiento o guía de misiles. La exploración espacial está siendo realizada con vehículos autónomos utilizando la visión artificial, por ejemplo, el Mars Rover de la NASA (figura 1.3), el Exploration Rover y el ExoMars de la ESA.

Ejemplos de sistemas de apoyo son los sistemas de detección de obstáculos en los automóviles, y los sistemas de aterrizaje autónomo de las aeronaves. Varios fabricantes de automóviles han demostrado los sistemas de conducción autónoma de vehículos, pero esta tecnología todavía no ha alcanzado un nivel en el que se puede poner en el mercado.

Figura 1.4: Sensor Kinect y ejemplo de uso

En otros ámbitos, la visión por computadora también ha ganado una posición importante. Por ejemplo en los videojuegos podemos mencionar al Kinect. El Kinect es un dispositivo para controlar los juegos creado por Alex Kipman y desarrollado por Microsoft para la videoconsola Xbox 360, y desde junio del 2011 para PC a través de los sistemas operativos de Windows (figura 1.4). Kinect permite a los usuarios controlar e interaccionar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que reconoce gestos, objetos e imágenes. Este fue un dispositivo pionero en su clase, se basa en un sistema de visión estereoscópica mediante una cámara y un sensor de profundidad.

Actualmente, parte del código ha sido liberado para que puedan darse otras aplicaciones a este sensor. Por lo que ha sido utilizado en diversos proyectos de investigación y desarrollo tecnológico.

Por otro lado, la tecnología de los *smartphones*[3] o teléfonos inteligentes ha hecho que una gran cantidad de personas traigan consigo un dispositivo dotado de una cámara y otros sensores que pueden utilizarse para un sin fin de aplicaciones (o apps). Las aplicaciones más sobresalientes son las que emplean la cámara del teléfono para

[3]El término *smartphone* o *teléfono inteligente*,se refiere a todo teléfono móvil construido sobre una plataforma informática móvil, con una mayor capacidad de almacenar datos y realizar actividades semejantes a una minicomputadora y conectividad, que un teléfono móvil convencional.

implementar un software de visión artificial y lograr, por ejemplo, una utilidad de *realidad aumentada*. El término realidad aumentada se emplea para definir una visión directa o indirecta de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta en tiempo real. Un ejemplo muy popular de esto es la app[4] llamada Wikitude.

Wikitude es una app de realidad aumentada móvil, que fue desarrollada por la compañía Wikitude GmbH y publicada desde 2008 como software gratuito. Esta aplicación muestra información sobre el entorno del usuario sobre la vista de la cámara del dispositivo móvil. Wikitude fue la primera aplicación a disposición del público que utiliza un enfoque basado en la ubicación a la realidad aumentada. La información que provee es de carácter diverso, desde información de localización de sitios de interés hasta juegos basados en la visión de la cámara.

Figura 1.5: Realidad aumentada mediante Wikitude

En el futuro, muchas de las aplicaciones de los sistemas de visión serán aplicadas a la realidad aumentada. Próximamente se anunciará un dispositivo que promete revolucionar tanto los sistemas de visión, como las aplicaciones de la realidad aumentada. Este dispositivo son los Google Glasses.

Google Glass, también llamado solamente Glass, es una computadora portátil de realidad aumentada, con una pantalla montada en la cabeza, cuya finalidad será mostrar información disponible para los usuarios de smartphone sin utilizar las manos, permitiendo también el acceso a Internet mediante órdenes de voz. Tendrá un sistema operativo será Android.

En general, las principales tendencias de desarrollo tienen tres vertientes de aplicación.

- Procesos autónomos
- Navegación

[4]El término *app* es una abreviatura de la palabra en inglés *application*. Es decir, una app es una aplicación de software que se instala en dispositivos móviles o tablets para ayudar al usuario en una labor concreta, ya sea de carácter profesional o de ocio y entretenimiento.

- Realidad aumentada

Figura 1.6: Ejemplo de uso de Google Glass

1.2. Objetivos y descripción de los capítulos

El objetivo de este trabajo es desarrollar un sistema capaz de distinguir marcadores visuales a partir de su color y área, utilizando visión por computadora, una videocámara y algoritmos de reconocimiento e identificación de color. Con el propósito de ubicar, en un sistema de referencia, robots móviles. Para ello, se trata de utilizar transformaciones de calibración y corrección de planos desarrollando software en C++ empleando librerías de OpenCV y Simulink.

Los motivos de estos objetivos se irán explicando a lo largo de los siguientes capítulos. En el capítulo 2 se describen los posibles sistemas que se pueden utilizar ya sea para su implementación directa o como plataformas sobre las cuales se puede desarrollar un programa específico de visión. Así mismo se explica cuál de estas opciones se tomó y porqué. En el capítulo 3 se explica a grandes rasgos, cómo funcionan los sistemas de visión por computadora, describiendo los puntos fundamentales del procesamiento digital de imágenes. En el capítulo 4 se muestran los pasos generales para la discriminación de marcadores visuales, también se muestra uno de los algoritmos más utilizados para ello y se explica mediante un ejemplo. El capítulo 5 contiene una descripción de los errores comunes en los sensores ópticos y se explica cómo mediante un proceso de calibración, estos pueden ser corregidos. En el capítulo 6 se explica cómo se lleva a cabo las correcciones de deformación de la cámara, la corrección que se hace para detectar adecuadamente los objetos, cómo se mapean, qué información se extrae de la imagen capturada y qué se hace con dichos datos. En el capítulo 7 se muestra el caso de aplicación de este sistema, las pruebas que se le hacen y cómo se implementa a un sistema de control de un robot móvil. El capítulo 8 contiene los resultados de las pruebas descritas en el capítulo 7. Por último, en el capítulo 9 se dan las conclusiones generales y se hace una proyección hacia los posibles trabajos a futuro.

1.3. Desarrollo Previo

El problema de controlar un robot móvil empleando un sistema de visión por computadora para monitorear su posición, ha sido estudiado anteriormente con diferentes aplicaciones.

El grupo de trabajo MRG (*Mechatronics Research Group*) de la Facultad de Ingeniería de la UNAM, consiste en un conjunto de académicos y estudiantes, dedicados al estudio y desarrollo de tecnología basada en ingeniería mecatrónica. Dentro de este grupo se ha utilizado el auxilio de la visión por computadora para el diseño de un sistema semiautónomo de robots móviles colaborativos, que mediante un software de visión llamado ReacTIVIsion se obtienen valores de posición y ángulo en tiempo real[19]. Otro de los precedentes se encuentra el proyecto realizado por Santiago Blackaller, donde se presenta el diseño de una plataforma de visión por computadora basada en los programas para computadora ReacTIVision y Simulink® (MATLAB) cuyo objetivo es servir de lazo de retroalimentación de la posición, la orientación y la velocidad de desplazamiento de un robot móvil de llantas independientemente actuadas en su tracción (las traseras) y en su orientación (las delanteras). La información obtenida por el sistema de visión es introducida al controlador. [1]Otro antecedente directo y perteneciente al mismo grupo de investigación, es el realizado por Arturo Martínez. Ese trabajo muestra el análisis cinemático de un robot móvil omnidireccional con dos robots diferenciales como elemento de tracción, mediante la implementación de una teoría unificadora. También contempla los resultados de pruebas en lazo abierto y cerrado, de seguimiento de trayectoria y de alcance de objetivo con un modelo funcional del robot que utiliza el software ReacTIVision como sistema de visión para la ubicación del robot. [17]. En el presente trabajo, se plantea seguir con varias de las propuestas de los desarrollos previos en cuanto a su arquitectura y control, que en capítulos posteriores se describen a detalle.

Capítulo 2

Sistemas de detección de marcadores visuales

2.1. Software de visión

Se conoce como *software de visión* a todos aquellos programas informáticos relacionados con realizar las funciones requeridas por la visión artificial, principalmente la captura de imágenes y su procesamiento. En otras palabras, el software de visión es un conjunto de programas, procedimientos, algoritmos y la documentación que tienen que ver con el funcionamiento de un sistema de procesamiento de imágenes.

2.1.1. ReacTIVision

ReacTIVision es una aplicación de visión por computadora, que fue diseñada para interactuar con objetos tangibles, es decir existentes físicamente, en el plano. Su código es abierto y principalmente está diseñada para la construcción de interfaces tangibles bidimensionales con el usuario. Es un software de código libre para el reconocimiento de marcadores visuales espacialmente diseñados (símbolos *fiducial*) como el que se muestra en la 2.1 que pueden ser asociados a un objeto físico como en la figura 2.2, así como de dedos sobre una superficie. Fue diseñado en un principio como el componente sensorial del *Reactable*, expuesto como un instrumento musical electroacústico tangible que cuenta con una serie de estándares para aplicaciones *multi-touch*. [13]

Los marcadores fiducial son reconocidos y localizados por un algoritmo de visión por computadora optimizado para el diseño particular de estos marcadores, mejorando la velocidad promedio y la robustez del proceso de reconocimiento. Estos símbolos marcadores fiducial permiten distinguir un determinado número de identidades únicas, añadiendo la posibilidad del preciso cálculo del ángulo de rotación del marcador en un plano de dos dimensiones. ReacTIVision y sus componentes están estructurados por la combinación de distintas licencias de software libre como son GPL, LGPL, BSD y puede ser obtenido como ejecutable o como código fuente abierto del

sitio SourceForge. [19]

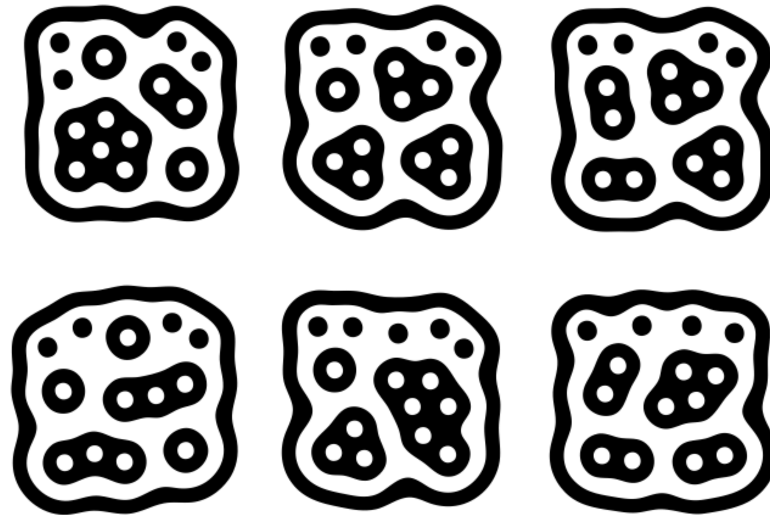


Figura 2.1: Marcadores visuales *fiducials*

Figura 2.2: *Fiducials* en elementos físicos para usarse en la Reactable [22]

El software ReacTIVision envía la posición y orientación de los fiducials de manera codificada mediante el protocolo TUIO a través del puerto UDP 3333, hacia cualquier cliente TUIO disponible. [14] En el cliente TUIO, estos mensajes son decodificados y la información puede ser utilizada por otro software para realizar acciones o tomar decisiones en función de la información obtenida.

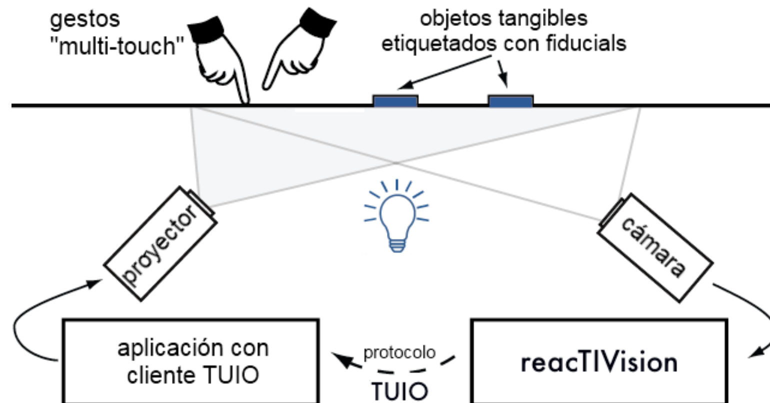


Figura 2.3: Componentes de la Reactable

En los trabajos previos, en los que este proyecto se basa, se utilizó ReacTIVision por su simplicidad, por un lado e aplicación. Esto debido a que dichos trabajos se enfocan principalmente en resolver la cinemática, el control, y otros aspectos de los robots móviles sin darle demasiada prioridad al sistema de visión. En la mayoría de dichos proyectos, se siguió un sólo esquema que constaba de una disposición donde se establece una comunicación entre ReacTIVision y un programa de control del robot en Simulink. En este proyecto se empleará la misma disposición de elementos que se explica en capítulos posteriores.

"*El reto fue mayúsculo y fue el objetivo que requirió de mayor tiempo y dedicación para ser resuelto, aun cuando se utilizaron herramientas preexistentes como el software ReacTIVision y el protocolo TUIO que facilitaron ciertas tareas como el procesamiento de la imagen y la propia obtención de los datos de la misma, pero había que resolver la comunicación del protocolo TUIO y Simulink®.* [1]"

"*El sistema de visión cumple su función, aunque se presentaban errores más grandes cuando se trabajaba en los límites del campo de visión, lo que refleja que la curvatura del lente también puede ser un factor importante para la generación de errores. La resolución de la cámara y la velocidad de captura también son factores importantes que influyen*

principalmente en la velocidad de desplazamiento y giro del robot, así como en el tiempo de reacción ante el cambio de objetivo. Por otro lado, el software utilizado no cumplió en su totalidad lo solicitado al momento de realizar las pruebas. [17]"

Los párrafos citados anteriormente pertenecen a dos proyectos distintos, desarrollados dentro del mismo grupo de trabajo, donde se describe la utilidad de ReaTIVision como sistema de visión auxiliar para llevar a cabo las tareas propuestas en sus respectivos trabajos. No obstante, también se llega a la conclusión de que esta herramienta resulta por un lado difícil de manipular debido a que fue desarrollada para un propósito diferente a la robótica móvil, que además su código no es fácilmente modificable, y por otro lado con errores debidos a la deformación de la imagen, sin mencionar insuficiente en cuanto a su velocidad de operación. Es por ello que se buscó una solución diferente que eliminara o redujera los inconvenientes antedichos.

Utilizar éste software de visión o incluso alguno comercial, pueden resultar útiles para aplicaciones de robótica móvil. Sin embargo, resulta más conveniente desarrollar un sistema ex profeso para las aplicaciones requeridas. Esto debido a que al ser dirigido específicamente para aplicaciones de robótica móvil, los requerimientos de fácil edición, portabilidad, velocidad y precisión, podrían ser satisfechos con mayor facilidad. Es por ello que se analizaron diferentes plataformas de desarrollo para crear al nuevo sistema de visión.

2.2. Comparación entre plataformas de desarrollo de sistemas de visión

Una plataforma de desarrollo o *framework*, es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación. En otras palabras, es un directorio jerárquico que encapsula los recursos compartidos, como librerías dinámicas compartidas, archivos de imágenes, secuencias localizadas, archivos de cabecera y documentación de referencia en un solo paquete. Múltiples aplicaciones pueden utilizar todos estos recursos de forma simultánea. El sistema los carga en la memoria cuando sea necesario y comparte la copia de uno de los recursos entre todas las aplicaciones siempre que sea posible. Una plataforma proporciona librerías de rutinas que pueden ser llamadas por una aplicación para realizar una tarea específica.

Emplear una plataforma útil para desarrollar una aplicación propia de visión resulta ser más conveniente que utilizar un software creado para propósitos especializados. De esta manera, se puede programar de acuerdo a las necesidades y requerimientos específicos del proyecto.

Es por ello que a continuación se describen las características de algunas de las plataformas de desarrollos más relevantes, que se analizaron y en algunos casos se probaron. Así tomando en cuenta sus características, se llegó a la selección de una

para ser utilizada para programar el software de visión requerido por el proyecto.

2.2.1. OpenCV

OpenCV (Open Source Computer Vision Library) es una librería libre con funciones de programación dirigidas principalmente a la visión artificial originalmente desarrollada por Intel y ahora soportada por Itseez y Willow Garage. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libre y gratuitamente para propósitos comerciales y de investigación con las condiciones en ella expresadas. La librería es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica.

Esta plataforma pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado, realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multi núcleo. OpenCV puede además utilizar el sistema de Intel's Integrated Performance Primitives, un conjunto de rutinas de bajo nivel específicas para procesadores Intel, para optimizar las rutinas y acelerar su funcionamiento. [29]

2.2.2. SimpleCV

SimpleCV es una plataforma de código abierto en Python para la creación de aplicaciones de visión artificial. A través de él, se tiene acceso a varias librerías de visión por computadora como con OpenCV, sin necesidad de aprender primero sobre temas necesarios para la programación estructurada así como tampoco de temas específicos de visión artificial como profundidades de bits, formatos de archivos, espacios de color, gestión de buffer, valores propios, etc. En SimpleCV se proporciona un entorno de esqueleto y se puede utilizar un ambiente que resulte más cómodo para el usuario, es decir, Eclipse, TextMate, Xcode, VIM, etc. Las ventajas en SimpleCV es que está construido en la interfaz de comandos. Basta con escribir 'simplecv' desde cualquier lugar en el sistema operativo normal, y automáticamente se inicia el entorno de desarrollo. Desde aquí se puede empezar a programar, o incluso seguir las instrucciones en pantalla que proporcionan ayuda. [20]

Aunque en su aplicación esta plataforma es bastante sencilla, sus mayores inconvenientes se encuentran en su versatilidad y portabilidad.

2.2.3. AForge.NET

AForge.NET es una plataforma de código abierto en C# diseñado para desarrolladores e investigadores en el campo de la visión por computadora e inteligencia artificial (procesamiento de imágenes, redes neuronales, algoritmos genéticos, lógica difusa, aprendizaje automático, robótica, etc.). Está formado por un conjunto de librerías y aplicaciones de ejemplo.

El trabajo sobre la mejora de la plataforma de desarrollo está en progreso constante, lo que significa que el nuevo contenido se está agregando constantemente. [15]

2.2.4. MatLab®

MATLAB es un lenguaje de alto nivel y un entorno interactivo para cálculo numérico, visualización y programación. Usando MATLAB, se puede analizar datos, desarrollar algoritmos, y crear modelos y aplicaciones. El lenguaje, las herramientas y funciones matemáticas incorporadas permiten explorar múltiples enfoques y llegar a una solución más rápida que con las hojas de cálculo o lenguajes de programación tradicionales, tales como C, C++ o Java™.

Se puede utilizar MATLAB para una gama de aplicaciones, incluyendo el procesamiento de imágenes y vídeo, procesamiento de señales y comunicaciones, sistemas de control, finanzas y biología computacional. [18]

2.2.5. Otras plataformas

Por supuesto existen muchas otras y muy diversas plataformas relacionadas con la visión artificial. No obstante, presentan características que los hacen poco prácticos para la aplicación que se propone en este documento.

Si bien, existen plataformas como CellCognition, GemIdent, y ITK-SNAP que son de código abierto, tienen la desventaja de estar diseñadas para un propósito particular como análisis cuantitativo de la microscopía de fluorescencia, identificar regiones de interés y el análisis tridimensional de imágenes médicas, respectivamente. También hay plataformas de propósitos más generales como VXL (Vision 'Something' Library), IVT (Integrating Vision Toolkit) o Ilastik que si bien plantean una buena alternativa, no cuentan con la suficiente documentación para su empleo. Por último, hay herramientas más sofisticadas como LabView, que cuenta con un módulo de visión verdaderamente sencillo de utilizar. En este caso, el inconveniente principal es el costo del software y la demanda de recursos informáticos que requiere.

2.3. Selección de la plataforma de desarrollo

Matlab cuenta con su propio entorno. Si bien esto es bueno, también tiene un lado negativo si no se está familiarizado con la interfaz. OpenCV es todo lo contrario,

porque básicamente basta con instalar las bibliotecas y la configuración de su entorno es responsabilidad del programador.

OpenCV fue hecho exclusivamente para el procesamiento de imágenes. Cada estructura y función de los datos fue diseñada para la captura y el procesamiento de imágenes. Matlab por otro lado, es muy genérico, tiene desde herramientas financieras hasta herramientas de ADN altamente especializadas. Por otro lado, Matlab es simplemente demasiado lento. Matlab en sí está construido sobre Java, y Java se basa en C. Por eso, cuando se ejecuta un programa de Matlab, el sistema está ocupado tratando de interpretar todo ese código Matlab, entonces se convierte en Java y finalmente ejecuta el código. En cambio, utilizando C/C++, como con OpenCV, no se pierde todo ese tiempo. El código se proporciona directamente a la computadora en lenguaje de máquina, y se ejecuta. Así que finalmente se obtiene más procesamiento de imágenes, sin interpretar de más.

Intentando hacer algo de procesamiento de imágenes en tiempo real, tanto con Matlab como con OpenCV, se aprecia que por lo general Matlab tiene velocidades muy bajas, un máximo de cerca de 4-5 fotogramas por segundo en promedio. Con OpenCV, se obtiene procesamiento real en tiempo real en torno a los 30 fotogramas por segundo aproximadamente. [27]

SimpleCV, por otro lado, aunque no es tan rápido como OpenCV es bastante simple de manejar. Sin embargo, una desventaja predominante es su portabilidad nula. Si no se cuenta con SimpleCV instalado en una computadora, no se pueden utilizar los programas creados en esta plataforma.

A continuación, en la figura 2.4 se muestra un cuadro comparativo basado en los análisis realizados en [20] y en [9].

Característica		Plataformas			Plataformas valoradas		
Descrpción	Valoración	OpenCV	MatLab	SimpleCV	OpenCV	MatLab	SimpleCV
Facil manejo	0.5	30	90	80	15	45	40
Velocidad	1	90	20	40	90	20	40
Demanda de recursos (menor)	1	90	40	80	90	40	80
Costo (menor)	1	100	40	100	100	40	100
Ambiente de desarrollo	0.5	60	80	70	30	40	35
Manejo de memoria	0.5	40	90	80	20	45	40
Portabilidad	1	80	30	30	80	30	30
Habilidades de programación requeridas (menor)	0.5	30	60	60	15	30	30
Debugging	0.5	50	90	90	25	45	45
SUMA		570	540	630	465	335	440

Mayor valor
Valor intermedio
Menor valor

Figura 2.4: Comparación entre las principales plataformas de desarrollo

Del lado izquierdo se presenta un listado con las características significativas para distinguir las capacidades de las plataformas evaluadas. Del lado derecho (recuadro azul), se muestran las tres diferentes plataformas de desarrollo y su evaluación en porcentaje de cada categoría. Aquí, sumando los porcentajes de cada una, podría pensarse que SimpleCV es la adecuada para ser empleada. No obstante, hay ca-

racterísticas cuya importancia no es tan grande, es por ello que en la columna de 'Valoración' su factor es de 0.5. Es decir, se tomó la mitad del valor de las características menos relevantes. Del lado extremo derecho (recuadro rosa), se muestran las evaluaciones en porcentaje de cada plataforma tras haber aplicado el factor de valoración. En este caso, la suma de porcentajes muestra que OpenCV es una mejor opción para ser utilizada para desarrollar el programa de visión artificial, esto aunado al hecho de que lleva más tiempo en constante actualización que SimpleCV. Fue por ello que se optó por usar OpenCV.

Capítulo 3

Arquitectura del sistema de detección y ubicación de marcadores visuales

3.1. Funcionamiento de la visión por computadora

El objetivo general de una máquina con visión integrada es derivar una descripción de una escena, analizando una o más imágenes de dicha escena. En algunas situaciones la escena misma es básicamente bidimensional. Por ejemplo si se trabaja con piezas planas sobre una superficie plana. La visión para situaciones bidimensionales es más fácil que para las tridimensionales.

En la figura 3.1 se muestran las etapas fundamentales del procesamiento de imágenes y su interacción con la *base de conocimiento*, este concepto se refiere al conocimiento sobre el dominio del problema. Este conocimiento puede ser tan simple como detallar las regiones de una imagen donde se sabe que se ubica información de interés, limitando así la búsqueda que ha de realizarse para hallar tal información. La base de conocimiento también puede ser muy compleja, como una lista interrelacionada de todos los posibles defectos en un problema de inspección.

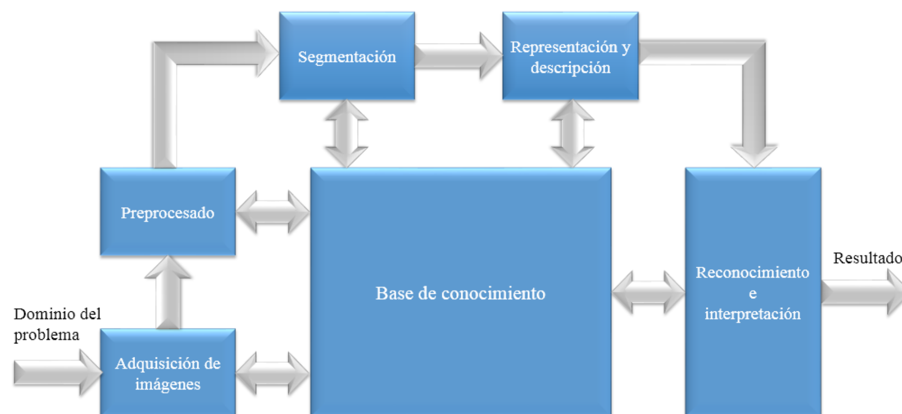


Figura 3.1: Etapas fundamentales del procesamiento digital de imágenes [10]

En esta figura, se observa también que el objetivo global es producir un resultado por medio de un determinado problema por medio de procesamiento de imágenes. En general, la mayoría de los programas de visión por computadora funcionan como se describe a continuación.

3.1.1. Adquisición y digitalización de la imagen

Este es siempre el primer proceso que se lleva a cabo. Para ello se necesitan sensores y la capacidad para digitalizar la señal producida por el sensor. El sensor puede ser una cámara a color o monocromo, y desde una sofisticada video cámara de alta definición hasta una cámara web convencional. Sea cual sea el sensor, debe producir una imagen completa del dominio del problema después de capturar la imagen a inspeccionar se envía esta información a la computadora para ser analizada. Una vez que se ha obtenido la imagen digital, la siguiente etapa trata del preprocesamiento de esa imagen.

3.1.2. Preprocesamiento

En este proceso se modifica la imagen que se adquirió con el fin de mejorarla de acuerdo a los parámetros a analizar, de forma que se aumenten las posibilidades de éxito en los procesos posteriores. El preprocesamiento generalmente se realiza con los siguientes objetivos:

- Eliminación de ruido.
- Acentuar o perfilar las características de una imagen tales como bordes y límites.
- Contrastar la imagen para que sea más útil la visualización gráfica y el análisis de la misma.
- Mejorar la calidad de algunas partes de la imagen.
- Transformar la imagen a otro espacio de representación.

3.1.3. Segmentación

Para que un sistema de visión reconozca partes, perforaciones, etc. en una superficie, y en general objetos, primero debe distinguir los objetos de interés del resto de la escena. En otras palabras, debe ser capaz de "destacar" partes de la imagen que corresponden a esos objetos. Su objetivo es dividir la imagen en las partes que la constituyen o los objetos que la forman. En este proceso se diferencia el objeto y el fondo, extrayendo subconjuntos de una imagen que corresponden a partes relevantes de la escena.

En general, la segmentación autónoma es una de las labores más difíciles del tratamiento digital de imágenes. Por una parte, un procedimiento de segmentación demasiado burdo retrasa la solución satisfactoria de un problema de procesamiento de imágenes. Por otra, un algoritmo de segmentación débil o errática casi siempre garantiza que tarde o temprano habrá un fallo.

A la salida del proceso de segmentación habitualmente se tienen los datos de pixel en bruto, que constituyen bien el contorno de una región o bien todos los puntos de una región determinada. En cada caso es necesario convertir los datos a una forma adecuada para el procesamiento por computadora. La primera decisión que hay que tomar es si los datos se han de representar como un contorno o como una región completa. La representación como un contorno es la adecuada cuando el interés radica en las características de la forma exterior, como esquinas e inflexiones. La representación regional es adecuada cuando el interés se centra en propiedades internas, como la textura o la estructuración. Sin embargo, en algunas aplicaciones ambas representaciones coexisten.

La elección de una representación es sólo una parte de la solución para transformar los datos de pixel en bruto a una forma adecuada para ser posteriormente tratados por computadora. También debe especificarse un método para describir los datos de forma que se resalten los rasgos de interés.

3.1.4. Descripción

La descripción, también denominada *selección de rasgos*, consiste en extraer rasgos con alguna información de interés o que sean fundamentales para diferenciar una clase de objetos de otra. Este proceso etiqueta los objetos teniendo en cuenta información suministrada por la inspección que puede ser:

- Cuantitativa: Realización de medidas (áreas, longitudes, perímetros etc.) y ángulos de orientación.

- Cualitativa: Verificación de la correcta realización de algún trabajo como los movimientos, el traslado, el ensamblado, etc.

3.1.5. Reconocimiento y toma de decisiones

El reconocimiento es el proceso que asigna una etiqueta a un objeto basándose en la información proporcionada por sus descriptores. Posteriormente se hace una interpretación de dicha información, que implica asignar significado a un conjunto de objetos reconocidos. Una vez interpretada la información se procede a tomar las decisiones correspondientes y acciones requeridas para desempeñar la función donde interviene el sistema de visión artificial.

3.2. Configuración preliminar del sistema de visión

3.2.1. Tipo de segmentación a emplear

Una *imagen digital* es una matriz de números que representan valores de iluminación en puntos regularmente espaciados de la imagen de una escena. Los elementos de más bajo nivel de tal imagen se llaman *pixeles* (contracción de "Picture Element"), y sus valores se denominan niveles de colores. La efectividad de una técnica de Segmentación depende de las propiedades de la clase de imágenes a que se aplique. El color en un punto de una imagen puede representarse por tres números que representen, por ejemplo, los niveles de las componentes roja, verde y azul. Entonces una imagen digital a color es una matriz de tripletas de valores. Si esos niveles de los pixeles se grafican como puntos de un espacio cromático, el objeto y el fondo originan cúmulos de puntos. Tales cúmulos son análogos a los picos del histograma y la imagen se puede segmentar en regiones de diferente color separando el espacio cromático de tal manera que se separen los cúmulos. Este ha sido el método clásico utilizado para segmentar las imágenes obtenidas por los sensores remotos multiespectrales en visión robótica.

Si una escena contiene varios objetos sobre un fondo, la segmentación de su imagen da el conjunto de pixeles pertenecientes a todos los objetos; la segmentación no ha distinguido los objetos entre sí. Para trabajar con cada objeto es necesario "etiquetar" o marcar los pixeles de los objetos, de tal manera que los que pertenecen a un mismo objeto tengan la misma marca, y viceversa. Este proceso se denomina marcación de componentes conectados, y asigna una marca distintiva única a cada conjunto de pixeles que están interconectados. El hecho de estar mutuamente conectados es el principio básico del proceso de contar objetos (o sea regiones internamente conectadas) en una imagen. El área de una región se puede medir aproximadamente en función del número de sus pixeles, o sea el número de pixeles que tienen una marca particular.

Se nombrará *marcador visual* a cada elemento formado por una vecindad de pixeles conectados que sea identificado por el software de visión artificial y que provea de información útil para la segmentación de la imagen digital. En el desarrollo de este proyecto, el sistema de visión debe abarcar el area del trabajo y distinguir de esta, la ubicación y orientación del robot móvil. Es por ello que se optó por añadir marcadores visuales al robot de pruebas, esto se describe más detalladamente en el capítulo 6.

3.2.2. Distribución de los elementos

Habiendo decidido que se emplearían marcadores visuales como elementos de identificación para el sistema de visión, resulta necesario determinar también la forma en que todos los elementos deben estar ubicados para que se logre la implementación adecuada de dicho sistema.

En la figura 3.2 se observa que el robot móvil se desplaza sobre una superficie plana, a nivel del piso. Esta condición es una restricción, por lo que no puede ser modificada. Tomando esto como base, se tiene que colocar la cámara de tal forma que se alcance a capturar la totalidad del espacio de trabajo y dentro de éste al robot móvil. Es por ello que se optó por colocar la cámara al centro del espacio de trabajo y con el eje de la lente *normal* al plano que contiene al espacio de trabajo.

En la figura 3.2 se muestra también la colocación de los marcadores visuales, unos sobre el robot móvil y otros sobre el espacio de trabajo. Sobre el robot se colocarán dos marcadores visuales, uno en el frente y otro atrás, para conocer no solo la ubicación sino también la orientación del robot. Por otro lado, se colocarán dos marcadores visuales para el espacio de trabajo que demarcan al eje horizontal, el cual define un plano cartesiano a partir del cual se genera un marco de referencia absoluto. La disposición final de los elementos se describe a detalle en el séptimo capítulo de este texto.

Dado que se identificarán a los marcadores visuales por su color, no es necesario que el fondo del espacio de trabajo sea opaco o muy contrastante. Solo basta con que el color de los marcadores visuales sea suficientemente distinto de todos los demás elementos dentro del espacio de trabajo. Así, los marcadores visuales pueden ser incluso piezas de papel coloreado.

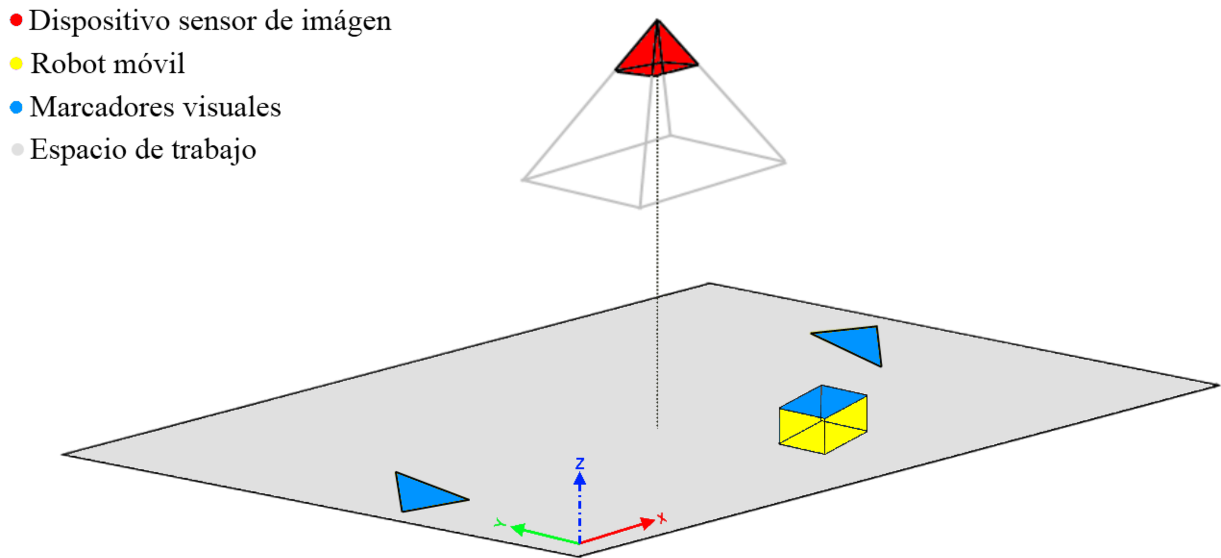


Figura 3.2: Disposición de los elementos

Capítulo 4

Algoritmo de discriminación de colores y áreas

4.1. Discriminación de marcadores visuales

En el campo de la visión artificial, el reconocimiento de marcadores visuales también llamados regiones, se refiere a las técnicas cuyo objetivo es detectar puntos o regiones integradas por pixeles con valores de colores o luminosidad similares en la imagen.

El estudio y desarrollo de estos detectores es importante por varias razones. Los detectores de regiones se usan como paso previo para el reconocimiento de objetos o su seguimiento. Otro uso habitual de estos detectores tiene que ver con el análisis de texturas y su reconocimiento. También, los descriptores de regiones han empezado a usarse para puntos de interés para informar de la presencia de determinados objetos en una imagen.

4.2. Etiquetado de componentes conectados

El etiquetado de componentes conectados (alternativamente llamado análisis de componentes conectados, extracción de blobs, etiquetado de regiones) es una aplicación algorítmica de la teoría de grafos, donde los subconjuntos de componentes conectados están especialmente marcados con base en una determinada heurística. El etiquetado de componentes conectados no debe ser confundido con la segmentación.

El etiquetado de componentes conectados se utiliza en la visión por computadora para detectar regiones unidas normalmente en imágenes digitales binarias, aunque las imágenes en color y datos con mayor dimensionalidad también se pueden procesar. [8] La extracción de blobs o marcadores visuales se realiza generalmente en la imagen binaria resultante de un paso de umbral también llamado *threshold*. Los blobs pueden ser contados, filtrados, y rastreados.

4.3. Vecindad entre pixeles

Para definir de forma adecuada el concepto de vecindad, es necesario revisar el de adyacencia. Dos pixeles son adyacentes si y solo si, tienen en común una de sus fronteras, o al menos una de sus esquinas.

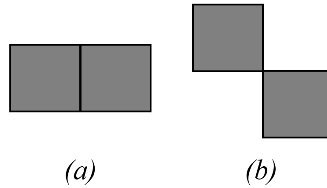


Figura 4.1: Pixeles adyacentes

Dos pixeles son vecinos si cumplen con la definición de adyacencia.

Un pixel P de coordenadas *(x,y)* tiene cuatro vecinos horizontales y cuatro verticales, cuyas coordenadas están dadas por:

$$(x+1, y), (x-1, y), (x, y+1), (x, y-1)$$

Dado que comparten una de sus frontera, se dice que son vecinos directos, como se observan en la figura 4.1*(a)*. Cuando comparten solo una de sus esquinas se les llaman vecinos indirectos, como se observan en la figura 4.1*(b)*. Sus coordenadas son:

$$(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)$$

En la figura 4.2 se pueden observar las vecindades de 4 y de 8. La primera formada por pixeles que son vecinos directos, mientras que la segunda está formada tanto por vecinos directos como por indirectos. A estas vecindades también se les conoce como conectividad-4 y conectividad-8 respectivamente.

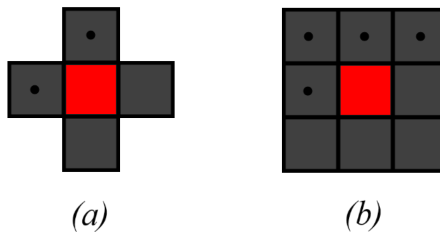


Figura 4.2: (a) Conectividad-4, (b) Conectividad-8

4.4. Algoritmo

Existen diversos tipos de algoritmos relacionados con la identificación de regiones conectadas. Cualquiera de ellos recorre la imagen para etiquetar las regiones, basado en la conectividad entre pixeles y los valores relativos de sus vecinos. Actualmente se emplean mayoritariamente los algoritmos basados en conectividad-4 o conectividad-8. Dichos algoritmos pueden ser generalizados a dimensiones arbitrarias. OpenCV cuenta con librerías de identificación de regiones conectadas. Como es el caso de cvBlob, una librería que emplean un algoritmo basado en conectividad-4 o conectividad-8. [5]

4.4.1. Two-pass

Relativamente simple de implementar y entender, el algoritmo de dos pasos itera a través de datos binarios bidimensionales. El algoritmo realiza dos pasadas sobre la imagen: un paso para registrar las equivalencias y asignar etiquetas temporales y el segundo para reemplazar cada etiqueta temporal por la etiqueta de su clase de equivalencia. [26]

El chequeo de la conectividad se lleva a cabo mediante la comprobación de las etiquetas de los píxeles ubicados al noreste, norte, noroeste y oeste del píxel actual empleando conectividad-8. La conectividad-4 utiliza sólo los vecinos del norte y el oeste del píxel actual. Las siguientes condiciones se comprueban para determinar el valor de la etiqueta que se asigna al pixel que se está analizando (4-conectividad se supone).

Condiciones que se deben comprobar en cada pixel según el algoritmo:

1. ¿El pixel a la izquierda (oeste) tiene el mismo valor?

   a) Sí - El pixel actual está en la misma región. Asignar la misma etiqueta al píxel actual

   b) No - Comprobar el estado siguiente

2. ¿Los píxeles hacia arriba (norte) e izquierda (oeste) del píxel actual tienen el mismo valor, pero no la misma etiqueta?

   a) Sí - Se sabe que los pixeles norte y el oeste pertenecen a la misma región y deben combinar. Asignar al píxel actual la etiqueta menor de los pixeles norte y oeste y grabar su relación de equivalencia

   b) No - Comprobar el estado siguiente

3. ¿El pixel a la izquierda (oeste) tiene un valor diferente y el que está arriba (norte) tiene el mismo valor?

a) Sí - Asignar la etiqueta del pixel norte al píxel actual

b) No - Comprobar el estado siguiente.

4. ¿El pixel de arriba (norte) y el pixel de la izquierda (oeste) tienen diferentes valores?

    a) Sí - Crear un ID de etiqueta nuevo y asignarlo al píxel actual

El algoritmo continúa de esta manera, y crea nuevas etiquetas de la región siempre que sea necesario. La clave de un algoritmo rápido es cómo esta función se realiza. Este algoritmo utiliza la estructura de datos de encontrar uniones que proporciona un rendimiento excelente para hacer el seguimiento de las relaciones de equivalencia. Al encontrar las uniones se almacenan etiquetas que corresponden a la misma región en un conjunto de datos sin procesar.

Una vez que el etiquetado inicial y la equivalencia se han guardado, la segunda pasada simplemente reemplaza la etiqueta de cada píxel con su elemento representante de equivalencia.

Un algoritmo rápido de exploración para la extracción de regiones conectadas se presenta a continuación: [12]

En la primera pasada:

1. Iterar a través de cada elemento de los datos por columna, después por renglón (Raster Scanning).

2. Si el elemento no es el fondo

    a) Obtener los elementos vecinos del elemento actual.

    b) Si no hay vecinos, únicamente etiquetar el elemento actual y continuar.

    c) De lo contrario, buscar al vecino con la menor etiqueta y asignarla al elemento actual.

    d) Guardar la equivalencia entre las etiquetas de los elementos vecinos

En la segunda pasada:

1. Iterar a través de cada elemento de los datos por columna, después por renglón.

2. Si el elemento no es el fondo

    a) Reetiquetar el elemento con la etiqueta más baja equivalente.

Aquí, el fondo es una clasificación, específica para los datos, que se utiliza para distinguir los elementos salientes del primer plano. Si la variable de fondo se omite, entonces el algoritmo de dos pasadas (two-pass) tratará el fondo como otra región.

A continuación se muestra un ejemplo de operación del algoritmo de dos pasadas (o barridas) sobre una imagen binarizada para identificar diferentes regiones conectadas.

4.4.1.1. Ejemplo del algoritmo

Este ejemplo se hará con una imagen binaria (blanco y negro), que regularmente como se utiliza para después manipularla. La imagen es la mostrada en la figura 4.3.

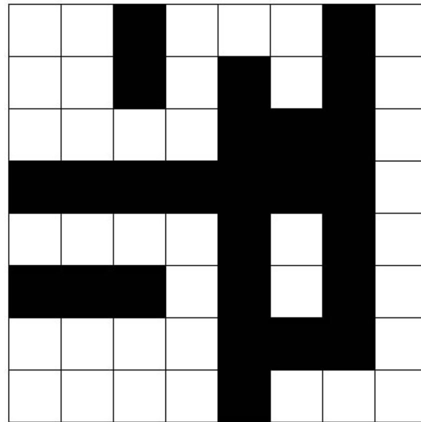


Figura 4.3: Imagen de entrada

Cada uno de los bloques representa un pixel. En el procesamiento, el blanco significa '1' y el negro un '0'. En este caso, tenemos interés en etiquetar los pixeles diferentes a '0'. Idealmente, uno debería poder aplicar el algoritmo y obtener una imagen como esta:

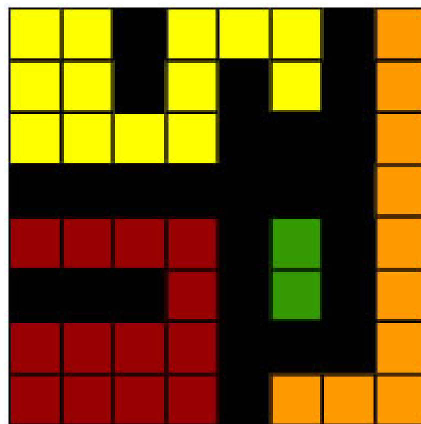


Figura 4.4: Imagen etiquetada por el algoritmo

Las etiquetas que se obtienen en la matriz son 1, 2, 3, 4, etc. En la imagen 4.4 los colores son representaciones visuales de las etiquetas enumeradas.

Primera Barrida A continuación se muestra la imagen de entrada, representada como una matriz. Se empieza siempre de la esquina superior izquierda.

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

Figura 4.5: Imagen de entrada representada como matriz

Revisando del primer pixel, su pixel superior y el de la izquierda no existen. Así que se crea una nueva etiqueta. Se etiqueta al primer pixel de arriba y la izquierda con la etiqueta 1 (en amarillo).

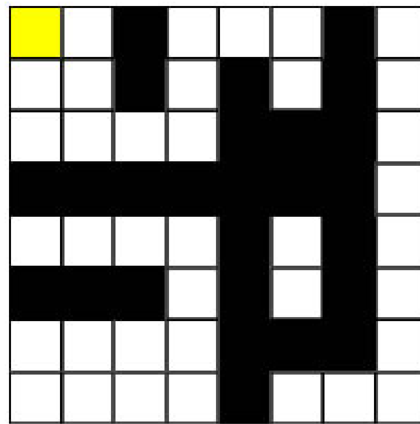


Figura 4.6: Etiquetado del primer pixel

Pasando al segundo pixel de la fila 1, columna 2 o el pixel (1,2). Este si tiene un pixel a su izquierda, entonces se copia la etiqueta de este. El siguiente pixel (1,3) es un pixel de valor '0', no interesa. Simplemente se le asigna la etiqueta 0.

Sigue el pixel (1,4). No hay un pixel encima, pero el pixel a la izquierda es de fondo, tiene un valor '0', es por ello que se le asigna una nueva etiqueta. Se marca la etiqueta del pixel (1,4) como un 2. (Con un color verde olivo).

Los dos siguientes pixeles, (1,5) y (1,6), tienen pixeles a su izquierda. Se les otorga una etiqueta 2. El siguiente (1,7) es fondo, se queda con una etiqueta 0. Para (1,8) se va crear una nueva etiqueta. El pixel de arriba no existe, pero el de la izquierda es un pixel de valor 0.

Terminando la columna 1, tenemos un resultado similar a este:

Ahora el pixel (2,1) no tiene nada a la izquierda, pero si tiene un pixel etiquetado encima de él. Como la etiqueta es 1, se hereda esa etiqueta. De manera parecida está el pixel (2,2). De hecho, es lo mismo para toda la columna. Todos los pixeles en la columna 2, tienen un pixel etiquetado encima de ellos. Por tanto el resultado va a

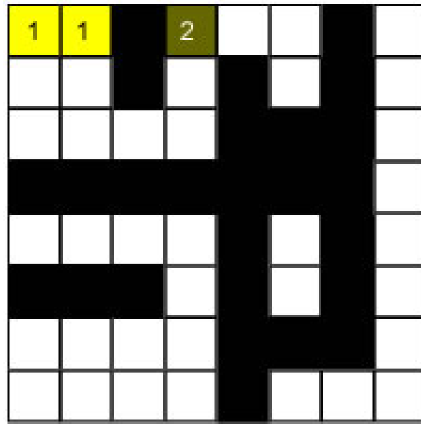


Figura 4.7: Etiquetado de un pixel a la derecha de un pixel de fondo

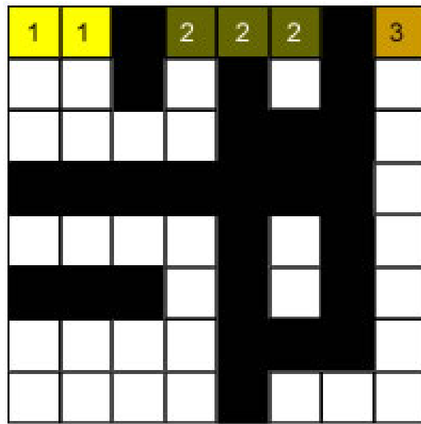


Figura 4.8: Etiquetado del primer renglón

ser:

En la tercera columna, los pixeles (3,1), (3,2) y (3,3) son sencillos de evaluar, se etiquetan como 1, pero el pixel (3,4) tiene un problema. Tiene un pixel arriba y uno a la izquierda y ambos tienen una etiqueta diferente. Es ahí donde se encuentra con que los elementos de la etiqueta 1 y 2 están en realidad conectados.

En este caso lo que se hace es tomar la etiqueta con un valor menor (en este caso 1) y etiquetar el pixel en cuestión (3,4) de esa manera. Es importante también inmediatamente guardar esa información sobre la etiqueta 2, definiendo a las etiquetas 2 como hijos de la etiqueta 1.

Procediendo con el análisis de la imagen, ya no debería existir ninguna otra dificultad que no podamos manejar. Los siguientes pasos serían:

Segunda Barrida De principio, se tienen muchas etiquetas para regiones que en realidad están conectadas. Para corregirlo, se vuelve a recorrer la imagen pixel por pixel. Se revisan las estructuras de herencia para las etiquetas. Al revisar se nota

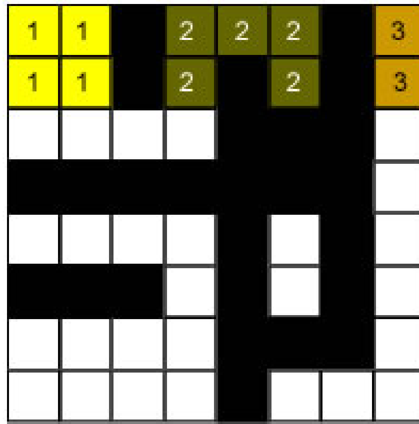


Figura 4.9: Etiquetado del segundo renglón

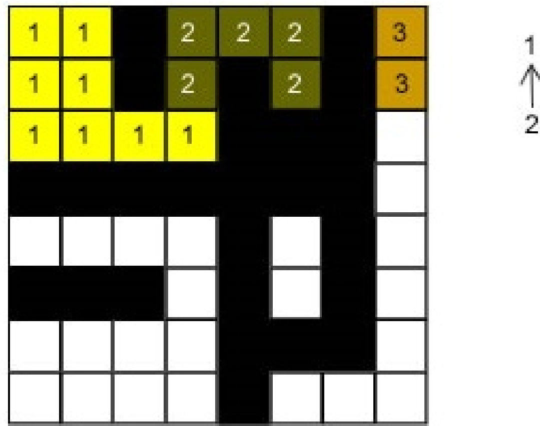


Figura 4.10: Etiquetado del pixel (3,4)

que 1 no es hijo de ninguna otra etiqueta. Es una raíz en realidad, así que continúa. Igual para los pixeles de fondo o con etiqueta 0.

Ahora el pixel (1,4), se revisa la estructura para la etiqueta 2, es un heredero de 1. Entonces se convierte la etiqueta 2 a 1 y se nota que en realidad 1 es una raíz.

De igual manera se continúa con toda la fila siguiendo este criterio. Para la cuarta fila, la matriz se debe ver así:

Ya se ve progreso. La región en la izquierda superior ya está correcta. Hasta la fila 5, se mantiene sin cambios. 4 es una raíz y también 5 y 3 lo son. La sexta fila permanece sin cambios.

Para la séptima fila ya tenemos un cambio interesante:

Y terminando la octava fila ya hay un resultado final:

Este es el resultado que se estaba buscando, donde todas las regiones conectadas tienen un solo valor de etiqueta. Demostrando el funcionamiento del algoritmo.

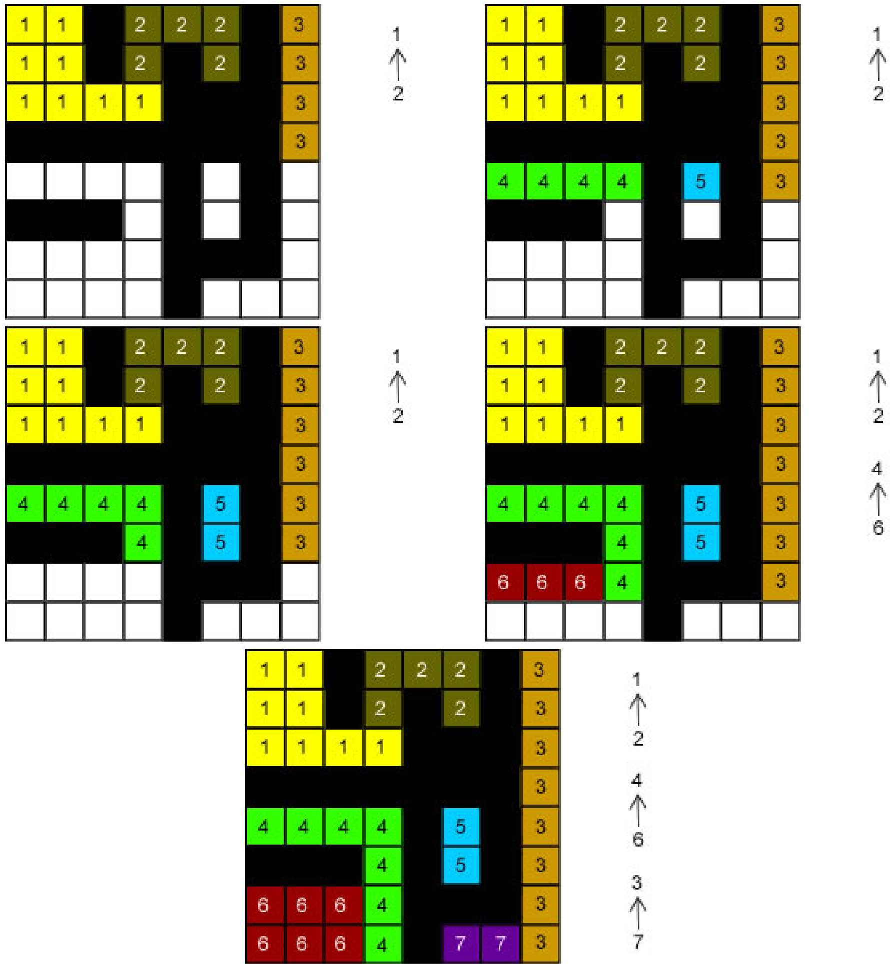


Figura 4.11: Procedimiento de etiquetado de la primer barrida

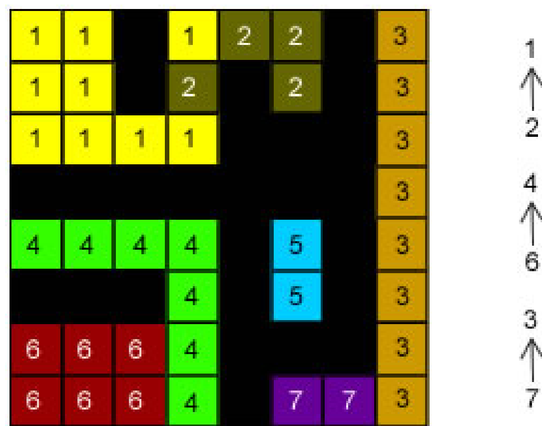


Figura 4.12: Reetiquetado de un pixel con etiqueta heredada

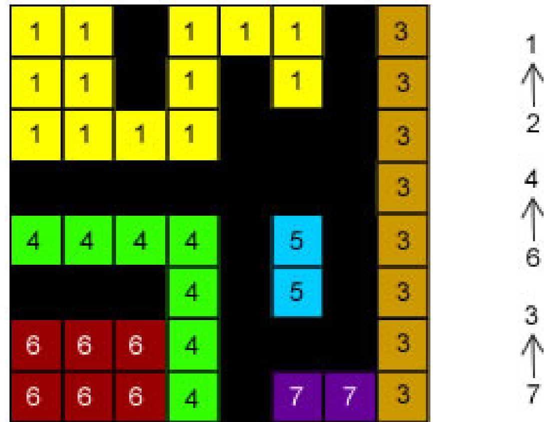


Figura 4.13: Reetiquetado de la imagen hasta la cuarta fila

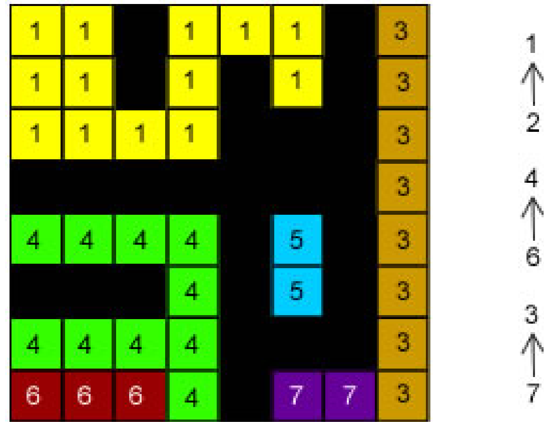


Figura 4.14: Reetiquetado de la imagen hasta la séptima fila

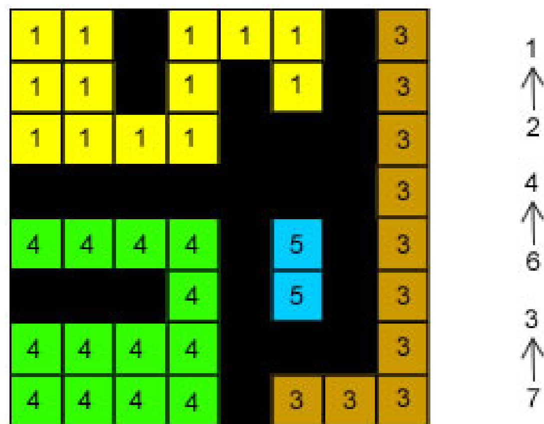


Figura 4.15: Reetiquetado final de la imágen

Capítulo 5

Calibración de la cámara

5.1. Aberraciones

Las cámaras se han utilizado ya, por un largo tiempo. No obstante, con la introducción de las cámaras digitales y cámaras web a finales del siglo XX, la presencia de estas se hizo común en la vida cotidiana. El bajo costo de dichas cámaras conlleva varias desventajas, las principales son aberraciones significativas para la imagen. Se denominan aberraciones a los defectos de un sistema óptico. Las aberraciones producen defectos en las imágenes que empobrecen su calidad.

Las aberraciones son las diferencias que existen entre un resultado teórico ideal y el real, mismas que deterioran la capacidad de una lente para producir una copia clara y exacta del objeto. En este apartado, se describirán los tipos básicos de aberraciones ópticas con la finalidad de definir cuáles son las que más afectan al desarrollo de este proyecto, para posteriormente encontrar alguna solución a dichas aberraciones.

Esto se debe a que las ecuaciones de la óptica geométrica para los elementos ópticos son obtenidas del cálculo de la refracción sobre superficies esféricas con la simplificación $sen(x) = x$, y considerando que los medios no son dispersivos. Cuando se ignoran los términos sucesivos en la serie, de manera que se comporta de la forma $sen(x) = x$, se obtiene la óptica de primer orden, en la que todas las lentes son perfectas. Cuando se incluye el término de x al cubo, entonces se ha procedido a la óptica de tercer orden, en los que las aberraciones derivadas de la naturaleza de las lentes reales se hacen evidentes. El hecho de que en la realidad estas condiciones no se cumplan exactamente produce dichas diferencias entre el resultado predicho por la teoría y lo observado.

Los efectos de las aberraciones ópticas pueden ser poca nitidez o distorsiones geométricas. Si bien las aberraciones son características de los elementos ópticos convencionales, y no pueden evitarse, si pueden anularse en los instrumentos ópticos.

5.1.1. Tipos de aberraciones

Las aberraciones se clasifican fundamentalmente por su origen. A las originadas por la geometría del elemento, se les llaman aberraciones geométricas, y a las originadas por la variación en el índice de refracción, se les llaman aberraciones cromáticas. Las aberraciones geométricas también se clasifican según su orden. La serie de Taylor la función seno es,

$$senx = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \cdots$$

Si para aproximar la función seno se utiliza el primer término de la serie, a la teoría desarrollada se le llama teoría de primer orden. Si se utilizan los dos primeros términos, se le llama teoría de tercer orden, y así sucesivamente. A las diferencias entre las teorías de primer orden y tercer orden, se les llama aberraciones de primer orden o aberraciones de Seidel, por ser Ludwig von Seidel quien primero estudió en detalle estos cinco tipos básicos de aberraciones debidas a la geometría de lentes o espejos, descritas en un documento de 1857.[25] Las aberraciones de Seidel son:

- Aberración esférica.- Es la diferencia de predominio entre el punto focal generado por la parte central de la lente con el punto focal generado por el periférico de la misma.

- Coma.- Es la modificación de la imagen que se produce por la diferencia de ángulo de incidencia de un rayo respecto al eje óptico. La luz procedente de un punto fuera del eje óptico al atravesar la lente forma una imagen borrosa en forma de gota.

- Astigmatismo.- Se presenta cuando los rayos de luz provenientes de un objeto que se encuentra fuera del eje óptico de la lente, al atravesarlo forman una imagen asimétrica. Impide que un punto objeto se enfoque en un punto imagen.

- Curvatura del campo.- Es producido por falta de correspondencia entre el plano objeto al plano imagen, es decir, que la imagen creada por un plano objeto no es plana, sino curva.

- Distorsión.- Es la modificación de la forma de la imagen. Si los bordes de la imagen creada se curvan hacia afuera, se habla de distorsión de barril, y si se curvan hacia adentro es una distorsión de corsé o almohada.

Las anteriores son aberraciones que pueden ser observadas usando luz monocromática, por ello son llamadas aberraciones monocromáticas, sin embargo también existen las aberraciones cromáticas. A continuación se explican todas ellas con más detalle.

5.1.1.1. Aberración esférica

En la figura 5.1 se ilustra el efecto de la aberración esférica

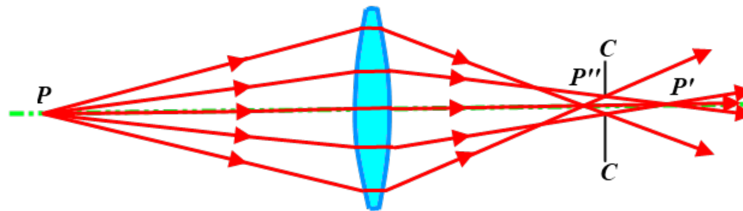


Figura 5.1: Aberración esférica.

Los rayos paraxiales procedentes de un punto P sobre el eje de la lente, forman imagen en el punto P'. Los rayos que inciden sobre la lente en su periferia, resultan difractados más fuertemente, formando la imagen en P''. Los rayos que inciden sobre la lente en zonas intermedias, forman imagen entre P' y P''. No existe entonces un plano en el que converjan todos los rayos procedentes de P, que forme una imagen nítida. El plano C-C, indicado en la figura, representa el plano para el cual la imagen de P es lo más pequeña posible, teniendo la mayor nitidez que se puede conseguir. Entonces, si tanto el objeto como la imagen son reales, la aberración esférica nunca puede eliminarse por completo. En la siguiente imagen se muestran los efectos de la aberración esférica, incrementados de manera gradual de izquierda a derecha, en una fotografía.

Figura 5.2: Efecto de la aberración esférica

5.1.1.2. Coma

La aberración de coma afecta a rayos procedentes de puntos no situados sobre el eje de la lente, a diferencia de la aberración esférica. Es debida la incapacidad de la lente de hacer que los rayos centrales, y los que atraviesan la periferia de la lente, converjan a un único punto (como en la aberración esférica). En la aberración esférica la imagen de un punto es un círculo, y en la de coma, una figura con forma de cometa. De ahí el nombre que se le ha dado a esta aberración.

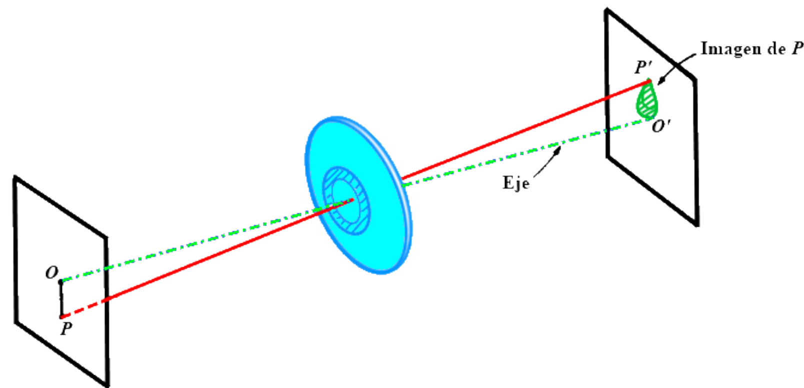


Figura 5.3: Aberración de coma

En la figura 5.3 se muestra el efecto de la aberración de coma. OO' es el eje de la lente, y P es un objeto puntual bajo este. El punto P' está dado por los rayos provenientes de P que atraviesan en centro de la lente. Los rayos proveniente de P que atraviesan la corona marcada en la lente, forman imagen en el círculo que se ve bajo P'. Los rayos que atraviesan la parte interior de la corona dan como imagen círculos de menor diámetro, más cercanos a P', y los que atraviesan el exterior de la corona dan como imagen círculos de mayor diámetro, más alejados de P'. Todos los rayos que atraviesan la lente dan como resultado la imagen de cometa que se ve en la figura 5.3. En la siguiente imagen se muestran los efectos de la aberración de coma, incrementados de manera gradual de izquierda a derecha, en una fotografía.

Figura 5.4: Efecto de la aberración de coma

5.1.1.3. Astigmatismo

Cuando un punto objeto está situado a una distancia apreciable del eje óptico, el cono de rayos incidente sobre la lente será asimétrico, originando con ello la tercera aberración primaria conocida como astigmatismo. El efecto que causa esta aberración está representado en la figura 5.5.

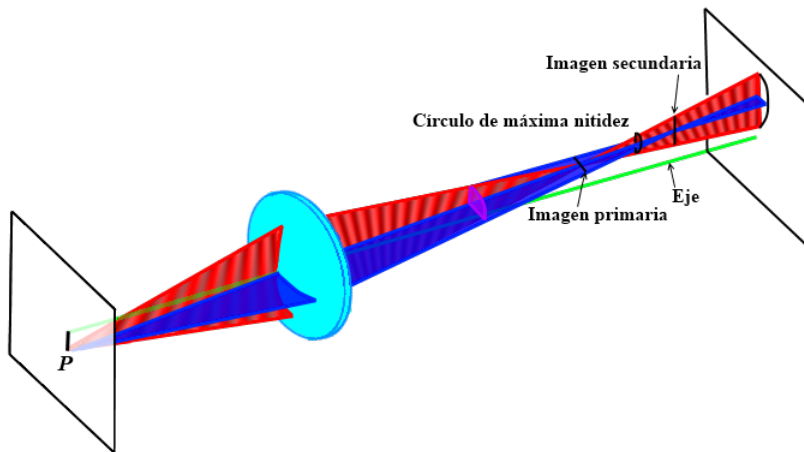


Figura 5.5: Aberración de astigmatismo

En la figura se muestran sombreadas dos secciones del cono de rayos procedentes del punto P y refractados por la lente. Una de las secciones corresponde al plano meridional o plano tangencial (que se define como el que contiene al rayo principal y al eje óptico), y la otra al plano sagital (que es el que contiene al rayo principal y es perpendicular al plano meridional). Esta aberración surge del hecho de que los rayos contenidos en el plano sagital se inclinan más con respecto a la lente que los rayos contenidos en el plano meridional, por consiguiente tienen distancia focal más larga. Se puede ver en la figura 5.5 que el cono, después de atravesar la lente, toma sección elíptica. Al llegar al plano focal de los rayos meridionales, la sección del cono degenera en una recta, llamada imagen primaria. Al llegar al plano focal de los rayos sagitales, la imagen toma forma de recta (llamada imagen secundaria), perpendicular a la de la imagen primaria. Entre la imagen primaria y la imagen secundaria se encuentra el círculo de máxima nitidez, que es donde la imagen del punto P resulta más clara. En la siguiente imagen se muestran los efectos del astigmatismo en una fotografía. El primer cuadro muestra la imagen generada en el foco tangencial (imagen primaria), la del centro es la imagen en el círculo de máxima nitidez y a la derecha la imagen en el foco sagital (imagen secundaria).

Figura 5.6: Efectos del astigmatismo

5.1.1.4. Curvatura de campo

Considando las imágenes de todos los puntos de un objeto plano. El lugar de la imagen primaria es una superficie curvada, al igual que el de las imágenes secundarias. La superficie correspondiente a los círculos de máxima nitidez se encontrará entre estas dos, y será en general una superficie curva. Esta aberración se llama curvatura de campo. La figura 5.7 ilustra este efecto.

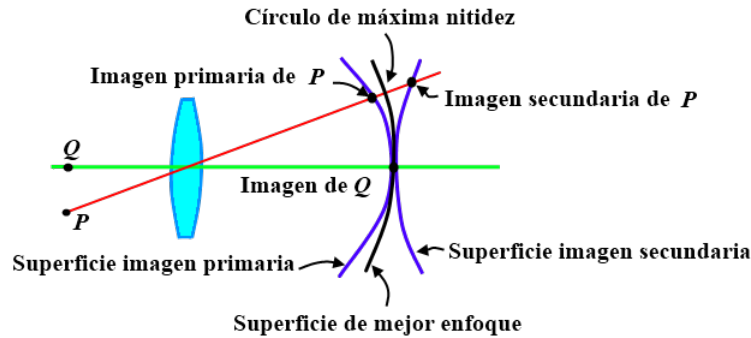


Figura 5.7: Aberración Curvatura del campo

Para eliminar la curvatura de campo se necesita que ambas superficies (la de las imágenes primarias y secundarias) sean iguales y opuestas. Para eliminar el astigmatismo, ambas superficies tienen que ser iguales. Para distancias angulares pequeñas resulta más perjudicial la aberración de coma que la de astigmatismo, mientras que para distancias angulares grandes sucede al revés. Por eso en los anteojos astronómicos se corrige principalmente la coma mientras que en los objetivos de cámaras fotográficas, el astigmatismo. En la siguiente imagen se muestran los efectos de la curvatura de campo, incrementados de manera gradual de izquierda a derecha, en una fotografía.

Figura 5.8: Efectos de la curvatura del campo

5.1.1.5. Distorsión

Las aberraciones descritas anteriormente provocan falta de nitidez en la imagen, y se deben a la imposibilidad de las lentes de formar imágenes puntuales de objetos.

La distorsión es una aberración que no provoca falta de nitidez en la imagen, sino variación del aumento con la distancia el eje. En la figura 5.9 se observan imágenes distorsionadas.

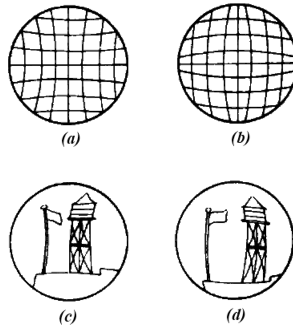


Figura 5.9: Distorsiones de corsé y barril

La distorsión de las figuras *(a)* y *(c)* se denominan de corsé, y se dan cuando el aumento crece con la distancia al eje. La distorsión de las figuras *(b)* y *(d)* se denominan de barril, y se dan cuando el aumento disminuye con la distancia al eje. En un instrumento destinado a uso visual una distorsión moderada no molesta. Para un instrumento destinado para mediciones sobre la imagen obtenida, la distorsión debe ser eliminada. Una lente delgada está libre de distorsión para todas las distancias objeto si no hay diafragmas que limiten el cono de rayos que incide sobre ella. Si existen diafragmas sobre el eje habrá, en general, distorsión. En la figura 5.10 se ilustran tres casos de lentes delgadas con diafragmas.

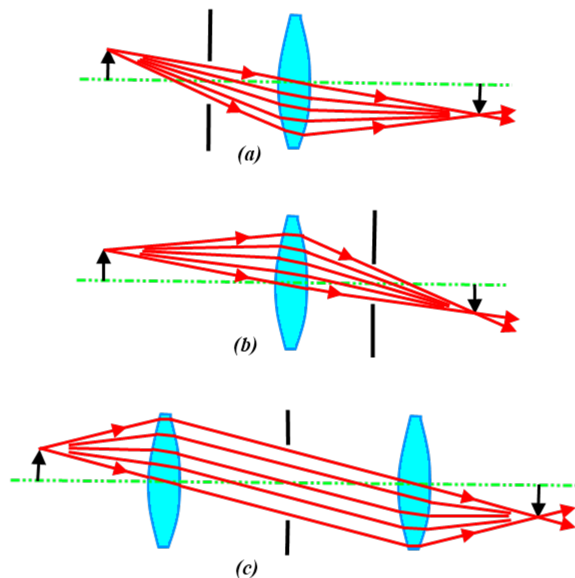


Figura 5.10: Aberración de distorsión

En el caso de la figura (a) en que el diafragma se coloca entre el objeto y la lente, se presentará distorsión tipo barril. En el caso (b) en que el diafragma se coloca entre la lente y la imagen, se tendrá distorsión tipo corsé. En el caso (c) el diafragma se coloca en medio de dos lentes iguales. En este caso la distorsión originada en la primera lente es opuesta a la de la segunda, por lo que ambas se compensan, y el sistema completo está libre de distorsión. En la siguiente imagen se muestran los efectos de la aberración de distorsión en una fotografía. A la izquierda se muestra la imagen original, al centro la imagen con distorsión de barril y a la derecha la misma imagen con distorsión de corsé (ambas afectando solo horizontalmente).

Figura 5.11: Efectos de la distorsión

5.1.1.6. Aberración cromática

La distancia focal de una lente depende del índice de refracción del material que la forma, y puesto que este varía con la longitud de onda de la luz transmitida, la distancia focal es distinta para los diferentes colores. En consecuencia, una lente no forma simplemente una imagen de un objeto, sino una serie de imágenes a distintas distancias de la lente, una para cada color presente en la luz incidente. Además, como el aumento depende de la distancia focal, estas imágenes tienen tamaños diferentes. Esto se ilustra en la siguiente figura.

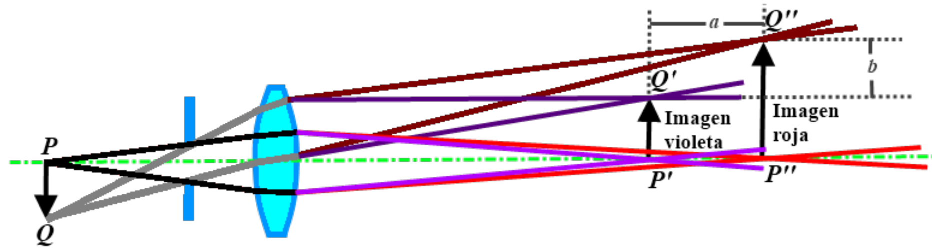


Figura 5.12: Aberración cromática

La variación de la distancia imagen con el índice de refracción se denomina aberración cromática longitudinal o axial, y la variación de tamaño de la imagen es la aberración cromática lateral. En la figura 5.12 se ha representado (exageradamente) las imágenes correspondientes a los colores violeta y rojo, que son los extremos del espectro visible. El violeta tiene una longitud de onda menor que el rojo, por lo

que el índice de refracción resulta mayor y la distancia focal menor. La aberración cromática longitudinal se mide por la distancia a entre las imágenes, y la aberración cromática lateral por la distancia b. No existe un plano en el cual resulten enfocadas todas las imágenes. En la siguiente imagen se muestran los efectos de la aberración de coma, incrementados de manera gradual de izquierda a derecha, en una fotografía. En la siguiente imagen se muestran los efectos simulados de la aberración cromática, incrementados de manera gradual de izquierda a derecha, en una fotografía.

Figura 5.13: Efectos de la aberración cromática

5.1.2. Discusión

Las primeras cuatro aberraciones descritas se derivan de las características físicas de las lentes que se emplean en un sistema óptico, el resultado de dichas deformaciones es una pérdida en la calidad o nitidez de la imagen de salida. La aberración de coma, como la esférica, puede corregirse mediante una elección adecuada de los radios de curvatura de las superficies de la lente y es posible eliminaras totalmente en una lente delgada para un par dado de puntos objeto e imagen. Desgraciadamente los radios de curvatura para eliminar la aberración de coma no son los mismos para lograr la aberración de esfericidad mínima.

De manera distinta, la aberración de distorsión afecta a la forma de la imagen, no a la nitidez. En un sistema que será empleado para determinar la posición de ciertos objetos en un área, resulta una aberración altamente perjudicial. Afortunadamente, tal distorsión es constante y un proceso de calibración y remapeo posterior puede corregirla.

En el caso de las aberraciones cromáticas, aunque no puedan minimizarse o corregirse en el sistema óptico, podrán ser compensadas o incluso despreciadas durante la etapa de preprocesamiento de la imagen digital. Es por ello que estas aberraciones no son en realidad significativas.

En general, en un sistema óptico resulta imposible eliminar simultáneamente las aberraciones comentadas, ni siquiera minimizarlas. No obstante, sí pueden utilizarse sistemas ópticos compuestos por varios elementos de manera que se cancelen las aberraciones más significativas. Cuanto mayor sea el número de elementos ópticos del sistema, mayor será el grado de corrección que podrá obtenerse. Cuando se proyecta una lente, o en este caso, cuando se selecciona un sistema de captura de imágenes, es necesario determinar las aberraciones más perjudiciales. Así, por ejemplo, para

un instrumento astronómico (que cubre un pequeño ángulo) resultan cruciales las aberraciones esférica, de coma y cromática axial, mientras que en el caso de detección de marcadores visuales en un espacio de trabajo amplio, son más importantes las aberraciones como el astigmatismo, la curvatura de campo y sobre todo la distorsión.

5.2. Calibración

En lo relacionado con sistemas de visión el término calibración se refiere al proceso de encontrar los factores internos de la cámara que afectan su proceso de captura de imagen. Estos factores son: el centro de la imagen, la distancia focal y los parámetros de distorsión de la lente.

La calibración es un proceso necesario por diversos motivos. Por un lado, facilita el uso de cámaras web, las cuales resultan convenientes debido a su bajo precio. Por otro lado, una calibración precisa es requerida para aplicaciones que requieran interpretación de imágenes en tres dimensiones, reconstrucción de modelos del mundo real, interacción de un robot con el mundo, etc.

5.2.1. Tipos de calibración

La calibración de las cámaras web es un paso que resulta necesario con el fin de extraer información de imágenes en dos dimensiones. De acuerdo a la dimensión del objeto auxiliar para la calibración de una cámara, se pueden clasificar las técnicas de calibración de la siguiente forma:

- Calibración basada en objetos 3D. La calibración de la cámara es realizada mediante la observación de un objeto de calibración cuya geometría en el espacio de tres dimensiones es conocida con muy buena precisión. El objeto de calibración usualmente consiste en dos o tres planos ortogonales entre sí. Este enfoque requiere un sofisticado sistema de calibración y una configuración complicada.

- Calibración basada en planos 2D. Las técnicas en esta categoría requieren la observación de la cámara a un patrón plano, mostrado en diferentes orientaciones.

- Calibración basada en líneas 1D. Los objetos de calibración empleados en esta categoría están compuestos por una serie de puntos colineales. Así, la cámara puede ser calibrada observando una línea moviéndose alrededor de un punto fijo.

- Auto calibración. Las técnicas en ésta categoría no necesitan de ningún objeto de calibración, y sólo puntos de correspondencia en la captura de imagen son requeridos. No obstante, un gran número de parámetros debe ser estimado, resultando en un problema matemáticamente más complexo. [28]

5.2.2. Selección del tipo de calibración

Para este proyecto resulta más conveniente emplear la calibración basada en planos 2D, ya que al no estar trabajando con un sistema estereoscópico o con sensado de profundidad, se estaría trabajando en realidad con imágenes bidimensionales.

Otros métodos como la calibración basada en líneas o la autocalibración, requieren de algoritmos más complejos de calibración y un conocimiento avanzado de programación. Sin embargo los resultados no serían distintos, la diferencia radica en la simplicidad para emplearlos.

La calibración basada en planos 2D interpreta información de varias imágenes tomadas con la cámara fija de un patrón dentro de un plano, como una cuadrícula o un patrón de círculos. En esas imágenes, el plano con el patrón, debe encontrarse en distintas posiciones. De ahí, un programa de calibración debe encontrar ese patrón e interpretar la posición del plano para obtener los datos de calibración requeridos.

5.3. Modelado de la cámara

El modelado de la cámara significa analizar como una escena tridimensional en el mundo real se proyecta en una imagen capturada en un plano bidimensional.

Para este análisis se considera a la cámara como una *cámara estenopeica*, donde se describe la apertura de la cámara como un punto y no se utilizan lentes para enfocar la luz. El modelo no incluye, por ejemplo, las distorsiones geométricas o imágenes borrosas de objetos desenfocados causada por las lentes y aberturas de tamaño finito. Esto significa que el modelo de cámara oscura o estenopeica sólo se puede utilizar como una aproximación de primer orden del mapeo de una escena 3D a una imagen 2D.

El proceso de calibración de la cámara, implica que deben conocerse los parámetros intrínsecos que generan las distorsiones en la imagen, para posteriormente compensarlos en la imagen capturada para obtener como resultado una imagen corregida útil. Estos parámetros intrínsecos, dependen de la perspectiva de la cámara.

En la figura 5.14 se muestran los sistemas coordenados relativos, correspondientes a la cámara, al espacio de trabajo en el mundo real o escena, y al plano de la imagen generada.

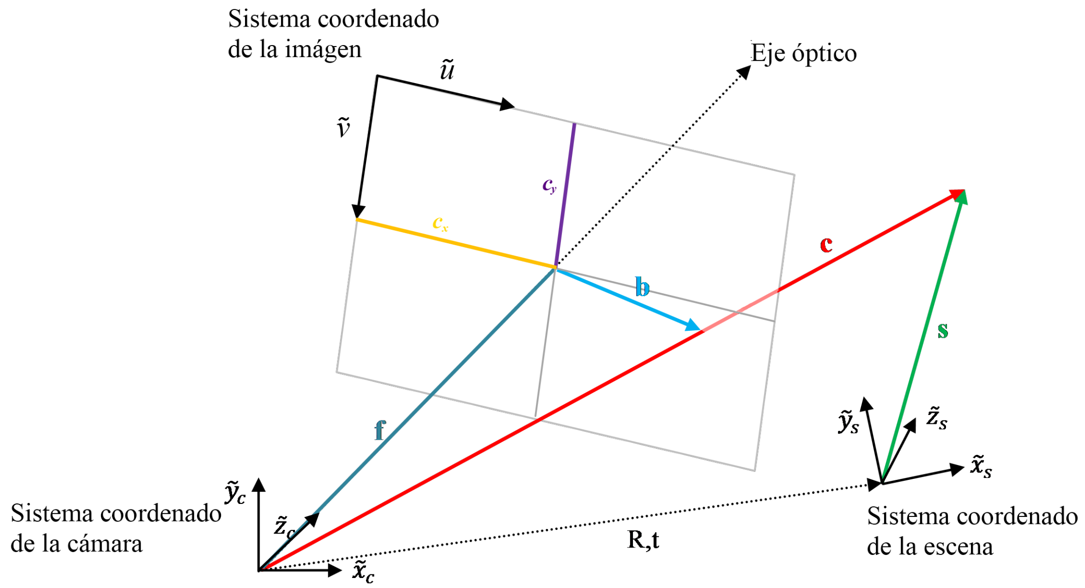


Figura 5.14: Modelado de la cámara

Haciendo una interpretación de los parámetros intrínsecos de la cámara, se observa que la distancia focal de la cámara al centro de la imagen está dada por $f(f_x,f_y)$. Por otra parte, podría pensarse que el eje óptico es colineal al centro de la cámara, sin embargo eso requiere una impecable precisión en el diseño y construcción de la cámara. De hecho, el centro del dispositivo que captura la imagen generalmente no está en el eje óptico. Es por ello que se introducen dos nuevos parámetros c_x y c_y. Para modelar un posible desplazamiento (alejado del eje óptico) del centro de coordenadas de la imagen generada. El resultado es un modelo relativamente simple, en el cual un punto de coordenadas (x_s, y_s, z_s) en la escena, es proyectado en la imagen en algún pixel ubicado el sistema coordenado de la imagen capturada (u, v) [2]. La transformación de perspectiva, en coordenadas homogéneas se calcula de la siguiente forma:

Sistema de coordenadas de la imagen — Sistema de coordenadas de la cámara — Sistema coordenado de la escena

$$b = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix} \begin{pmatrix} x_s \\ y_s \\ z_s \\ 1 \end{pmatrix}$$

Parámetros intrínsecos de la cámara — Parámetros extrínsecos de la cámara

Aquí se aprecia que la matriz de parámetros intrínsecos no depende de la escena vista y, una vez estimada, puede ser reutilizada, siempre y cuando la longitud focal sea

fija (sin zoom). La segunda matriz, de rotación y traslación $[R|t]$, se denomina matriz de parámetros extrínsecos. Se utiliza para describir el movimiento de la cámara en torno a una escena estática, o viceversa, el movimiento de un objeto en frente de la cámara fija. Es decir, que traduce las coordenadas de un punto (x_s, y_s, z_s) a algún sistema de coordenadas fijado con respecto a la cámara. La transformación anterior es equivalente a lo siguiente (con $z \neq 0$).

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [R|t] \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix}$$

$$x' = \frac{x}{z}$$

$$y' = \frac{y}{z}$$

$$u = f_x \cdot x' + c_x$$

$$v = f_x \cdot y' + c_y$$

Sin embargo, como se mencionó al principio de este capítulo, las lentes reales suelen tener algo de distorsión, en su mayoría distorsión radial y también una leve distorsión tangencial. Así, el modelo anterior de amplía de la siguiente forma:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [R|t] \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix}$$

$$x' = \frac{x}{z}$$

$$y' = \frac{y}{z}$$

$$x" = x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x'y' + p_2(r^2 + 2x'^2)$$

$$y" = y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y'^2) + 2p_2 x'y'$$

$$\text{donde } r^2 = x'^2 + y'^2$$

$$u = f_x \cdot x" + c_x$$

$$v = f_x \cdot y" + c_y$$

Los coeficientes de distorsión no dependen de la escena vista, y siguen siendo los mismos independientemente de la resolución de captura de la imagen. Es decir, si, por ejemplo, una cámara ha sido calibrada para una resolución de 320 x 240, absolutamente los mismos coeficientes de distorsión se pueden utilizar para imágenes de resolución 640 x 480 para la misma cámara, mientras que f_x, f_y, c_x y c_y necesitan ser escalados apropiadamente [21]. En el entorno de OpenCV se emplean las ecuaciones anteriores, adaptadas para corregir el plano de la imagen capturada pixel por pixel. Para la distorsión radial se utilizan las siguientes ecuaciones:

$$x_{corregida} = x(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

$$y_{corregida} = y(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

Por lo que para cada viejo pixel con coordenadas (u,v) en la imagen de entrada, le corresponde uno de coordenadas $(x_{corregida}, y_{corregida})$ en la imagen corregida de salida. Hay que recordar que la presencia de la distorsión radial se manifiesta en forma de barril.

La distorsión tangencial ocurre porque las lentes que captan la imagen no son perfectamente paralelas al plano de la imagen. Para corregir esta aberración se emplean las ecuaciones:

$$x_{corregida} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$

$$y_{corregida} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

Por lo tanto la corrección de la imagen depende de cinco parámetros de distorsión $(k_1, k_2, p_1, p_2, k_3)$. k_1, k_2 y k_3 son coeficientes de distorsión radial, en tanto p_1 y p_2 son coeficientes de distorsión tangencial. Los coeficientes de orden mayor no son considerados en OpenCV. Estos parámetros son los que se requieren para poder calibrar la cámara y corregir las distorsiones. En OpenCV existen funciones que facilitan la tarea de obtener estos parámetros. Estas funciones se describirán con detalle en el siguiente capítulo.

Capítulo 6

Rastreo y mapeo de marcadores visuales

A partir de la investigación antes descrita y respetando todas las decisiones tomadas a lo largo de ella, fue que se desarrolló un programa como parte central del sistema de visión auxiliar para el control de un robot móvil, como se planteó desde un inicio. El programa fue desarrollado en el lenguaje C++ dentro del entorno de desarrollo integrado Microsoft Visual Studio 2010© empleando la plataforma de desarrollo y conjunto de librerías de OpenCV, integrando también la librería cvBlob. A continuación se describen algunas de fragmentos del código utilizados dentro del programa, clasificados por su función. El programa final completo se anexa en el Apéndice II.

6.1. Restricciones de Visión

Cómo se menciona en el capítulo 3, después de la captura de la imagen suele hacerse un ajuste de la misma, con el fin de adecuar la imagen capturada para posteriormente aplicar las funciones necesarias para la identificación de las áreas conectadas que en este caso son los marcadores visuales.

En el siguiente fragmento de código fuente, se muestran las funciones provistas por las librerías de OpenCV para almacenar temporalmente la captura de la imagen y después hacer la segmentación o *thresholding.*

Dicha segmentación se hace considerando los colores de pixel utilizando el modelo de colores HSV. El modelo HSV (*Hue, Saturation, Value* – Matiz, Saturación, Valor), en ocasiones también llamado HSB (*Hue, Saturation, Brightness* – Matiz, Saturación, Brillo), define un modelo de color en términos de sus componentes en coordenadas cónicas.

Las dos primeras líneas del siguiente fragmento de código, son funciones para generar cuadros de imagen, estos cuadros son interpretados como arreglos de pixeles. La segmentación se hace empleando la función *cvInRangeS*, cuyos parámetros son:

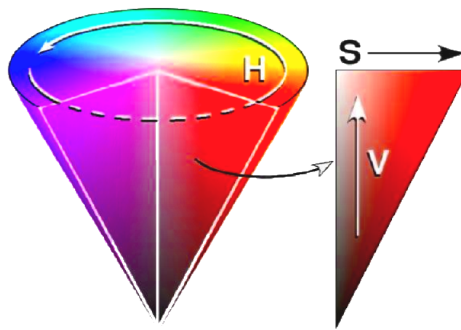


Figura 6.1: Mapa del modelo HSV

el cuadro de imagen de entrada nombrado “hsvframe” expresado como un arreglo, el valor mínimo de segmentación, el valor máximo de segmentación, ambos expresados como una tercia de valores escalares que en este caso representan los valores HSV del rango de colores admitidos para los marcadores visuales, y el cuadro de imagen de salida que en este caso lleva el nombre de “threshy”.

La función *cvInRangeS* toma la imagen de entrada y compara los valores de cada pixel con los valores de ventana del rango admisible de colores. Si el valor del pixel analizado se encuentra dentro de dicho rango, entonces se le asigna el valor del color blanco en la imagen de salida, de lo contrario se le asigna el valor del color negro. De esta forma, se binariza la imagen y se discrimina el color de los marcadores visuales del resto de la escena. Aquí es importante recordar que por esta razón, los marcadores visuales deben ser de un color contrastante o suficientemente diferente del resto de la escena capturada.

Algoritmo 6.1 Restricciones de Visión

```
// Crea una variable de imagen en espacio de color HSV
IplImage *hsvframe = cvCreateImage(cvSize(w,h),8,3);
// Crea una variable de imagen llamada "threshy"
IplImage *threshy = cvCreateImage(cvSize(w,h),8,1);
.
.
.
// Segmentación del marco para el color de los marcadores visuales
cvInRangeS(hsvframe, cvScalar(MinHue, MinSat, MinVol), cvScalar(MaxHue,
    MaxSat, MaxVol), threshy);
```

Hasta este punto se tienen dos mapas de imagen, por un lado la captura de la imagen y por otro la captura binarizada después de la segmentación, semejantes a lo que se observa en la figura 6.3

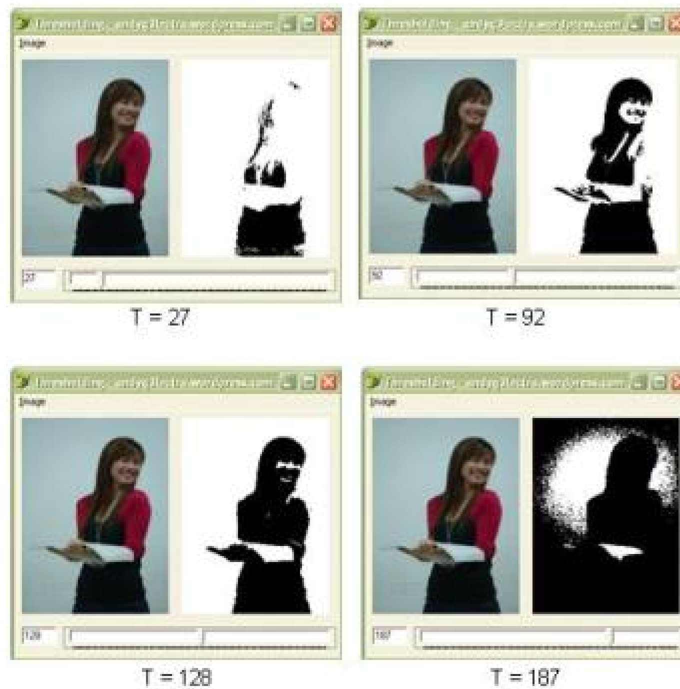


Figura 6.2: Imágenes de captura y segmentación

6.2. Eliminación de distorsión

Después de la segmentación, se tiene una imagen binarizada de tal forma que podrían ya detectarse las regiones de color conectadas y ubicarlas. Sin embargo, como se estableció en el capítulo 5 de este texto, un sistema óptico, como lo sería el dispositivo de adquisición de la imagen, trae consigo aberraciones que pueden alterar significativamente la imagen. Particularmente, como se pretende utilizar el sistema para determinar la posición de un objeto, la aberración que más afecta es la distorsión. Si la imagen está distorsionada, la posición determinada mediante el software no será veraz.

Es por ello que se utiliza una función que emplea los parámetros intrínsecos de la cámara y los coeficientes de distorsión para aplicar una trasformación a la imagen, de tal forma que resulte una imagen corregida es decir, sin distorsión radial ni tangencial. Tanto los coeficientes de distorsión como los parámetros intrínsecos de la cámara, se obtienen de un proceso de calibración. El proceso que se implementó, y el código fuente del programa desarrollado para la calibración están descritos en el Apéndice III.

Dentro de las librerías de OpenCV, el método *cvUndistort2* transforma la imagen para compensar la distorsión de lente radial y tangencial. Para cada píxel en la imagen de salida de la función, calcula las coordenadas de la ubicación correspondiente en la imagen de entrada utilizando las fórmulas descritas en el capítulo 5. Entonces, el valor de píxel se calcula utilizando la interpolación bilineal. Si la resolución de las imágenes

es diferente de lo que se utilizó en la etapa de calibración, f_x, f_y, c_x, c_y necesitan ser ajustados de forma apropiada, mientras que los coeficientes de distorsión siguen siendo los mismos. En el siguiente fragmento de código fuente se muestra cómo fue utilizado el método *cvUndistort2*. Primero se crea una variable para la imagen de origen, después se crea una variable para la imagen destino y por último se invoca al método *cvUndostort2*, cuyos parámetros son las dos variables antedichas, la matriz de valores intrínsecos de la cámara y el vector de coeficientes de distorsión (estos últimos obtenidos del programa de calibración).

Algoritmo 6.2 Eliminación de distorsión

```
// Crea una variable de imagen llamada "source" que obtiene el cuadro
    actual de captura
IplImage *source=cvQueryFrame(capture);
//Si falla la obtención rompe el ciclo
if (!source)
        break;
// Crea una variable de imagen llamada "fram"
IplImage *fram = cvCreateImage(cvSize(source->width,source->height),
    IPL_DEPTH_8U, 3);
// Función para eliminar la distorsión
cvUndistort2(source, fram, intrinsic, distortion);
```

Después de esta serie de instrucciones, se genera una nueva variable de imagen, que es con la que se trabaja en lo posterior. En esta nueva imagen se han corregido las distorsiones (después del proceso de calibración correspondiente). En la figura 6.3 se muestra un ejemplo de la comparación entre los cuadros de captura original y después de la corrección.

6.3. Detección y ordenación de los marcadores visuales

Aquí se describe a detalle el diagrama de flujo que aparece en el Apéndice I sobre la discriminación de los marcadores visuales, después se muestra cómo es que se implementó en el código explicando ciertos snippets.[1]También se describe el método desarrollado para numerar los marcadores visuales por área.

Una vez corregido el mapa de imagen, se procede a detectar las regiones conectadas (o *blobs*), que en esta aplicación son los marcadores visuales que indicarán tanto

[1]En programación, snippet es una pequeña porción de código o texto de programación. Son utilizadas generalmente para minimizar la repetición de códigos, hacer más claros los algoritmos o permitir que una aplicación genere el código automáticamente.

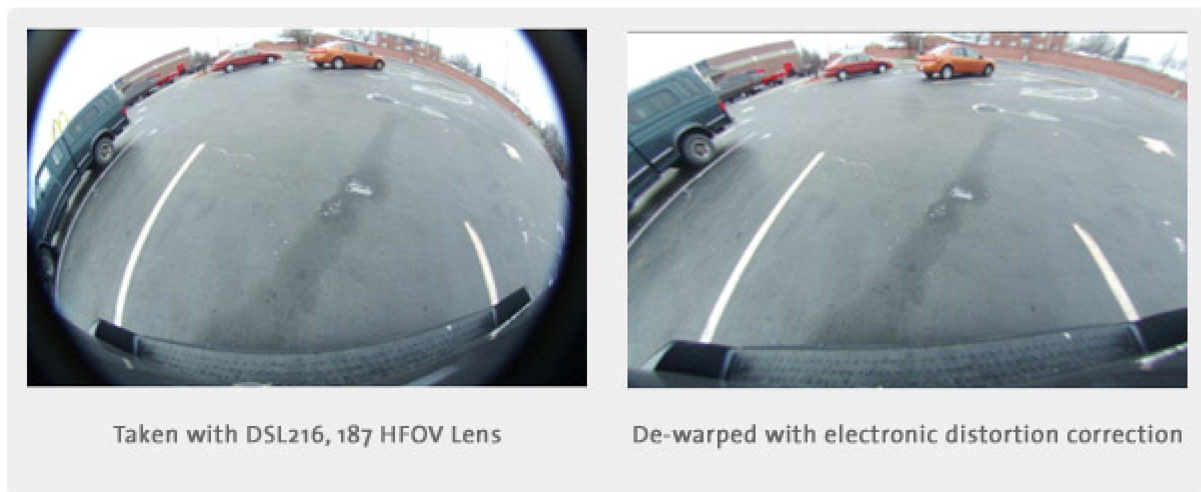


Figura 6.3: Imágenes de captura y corrección de la distorsión

la referencia en el espacio de trabajo, como a los objetos a ubicar en dicho espacio.

Se utilizó la librería cvBlob[4], que provee de funciones relacionadas con la detección de las regiones conectadas. En el Apéndice I se encuentra el diagrama de flujo donde se muestra a grandes rasgos cómo funciona la función *cvBlobs*, que distingue las regiones conectadas y las etiqueta de acuerdo al orden en que aparecen en la imagen de captura. En otras palabras, los marcadores visuales que aparecen más cerca del origen del mapa la imagen (esquina superior izquierda), reciben la primera etiqueta y a medida que se alejan reciben etiquetas de un número mayor. En este caso, como las figuras estarán en movimiento, este criterio no resulta útil, es por ello que se debió buscar la forma de reetiquetar a los marcadores visuales. La forma más simple resultó el identificarlos por su área, siendo el marcador visual más pequeño el que identifica con la etiqueta de menor número e ir incrementando a medida que su área sea mayor.

Esta forma de etiquetarlo y en general el programa desarrollado es tan versátil que podrían identificarse hasta 100 marcadores visuales (con las condiciones de procesamiento y captura adecuadas). Para evitar confusiones entre marcadores visuales, se multiplica su área por un factor para compararlos. No obstante, se encontró recomendable que las áreas entre marcadores visuales fueran al menos un 10 % distintas entre sí.

Para determinar la posición, la librería cvblob dispone de una función que facilita encontrar el centro geométrico de las regiones conectadas. A continuación se muestran las partes más importantes para la ordenación de marcadores visuales, descrita anteriormente.

Algoritmo 6.3 Interpretación de los Blobs

```
//Ciclo de interpretación de los blobs
for (CvBlobs::const_iterator it=blobs.begin(); it!=blobs.end(); ++it)
{
        double moment10 = it->second->m10;
        double moment01 = it->second->m01;
        double area = it->second->area;
        .
        .
        .
        blobarea[i]=area*1.3;
        i++;
        .
        .
        .
        if (savedblobsflag==2)
        {
                //Variable para mantener posición
                double x1;
                double y1;
                //Calcula la posición actual
                x1 = moment10/area;
                y1 = moment01/area;
                .
                .
                .
                //Ciclo de ordenación de los marcadores visuales a
                    partir de su area
                while (k<=n)
                {
                        if(area <= blobarea[k] && area >= blobarea[k
                            -1])
                        {
                                a1[k]= x;
                                a2[k]= y;
                        }
                        k++;
                }
                k=0;
        }
}
```

6.4. Definición del area de trabajo

Para conocer la posición de los marcadores visuales, se necesita un sistema al cual estén referidas dichas posiciones. Una opción podría ser tomar alguna de las esquinas de la imagen como referencia, sin embargo, cualquiera de esos puntos depende de la posición de la cámara.

Considerando un escenario donde los marcadores visuales estén estáticos, se detectaría su ubicación, pero si por alguna razón la cámara llegase a moverse, los valores de las ubicaciones cambiarían debido al movimiento de la cámara. Por esta razón se planteó un sistema de referencia a partir de dos marcadores visuales (los dos más pequeños). Estos, definirían el eje horizontal a partir del cual se puede generar un eje perpendicular y así, tener un sistema de referencia donde se mantenga la posición de los marcadores visuales sin importar que la cámara se mueva (siempre y cuando permanezca a la misma distancia de la escena).

En el siguiente cuadro, se muestra la parte del código donde se etiquetan a los dos menores marcadores visuales como “X” y “-X” y al resto se les asignan números.

Algoritmo 6.4 Definición de eje X y orden del resto de los blobs

```
//Cuando los Blobs ya están ordenados
//Etiquetado de la parte negativa del eje horizontal
stringstream labelx;
labelx << "-X";
cvPutText(frame,labelx.str().c_str(), cvPoint(a1[1]*w/screenx,a2[1]*h/
    screeny), &font, cvScalar(0,0,255));
//Etiquetado de la parte positiva del eje horizontal
stringstream labelxx;
labelxx << "X";
cvPutText(frame,labelxx.str().c_str(), cvPoint(a1[2]*w/screenx,a2[2]*h/
    screeny), &font,cvScalar(0,0,255));
//Ciclo para mostra las etiquetas del resto en pantalla
for (int j=3 ; j<=n ; j++)
{
        stringstream label;
        label << j;
        cvPutText(frame,label.str().c_str(),
                cvPoint(a1[j]*w/screenx,a2[j]*h/screeny),
                &font,   cvScalar(0,255,0));
}
```

En la figura 6.4, se muestra una captura de pantalla donde se muestra cómo se detectan los marcadores visuales que definen al eje X y cómo se representa en

pantalla.

Figura 6.4: Representación del sistema de referencia

6.5. Mapeo de los marcadores visuales

Después de tener los marcadores visuales identificados y ubicados dentro del espacio de trabajo, hay que definir su función. Para este caso de aplicación, se requiere saber la posición de un robot móvil, lo que implica que sobre este debe haber marcadores visuales.

Sin embargo, surge el problema de que los marcadores visuales que están sobre el robot, se encuentran en un plano elevado del suelo. Al encontrarse en otro plano, la distancia medida desde el centro de visión de la cámara y el centro del objeto es errónea.

Cuando la cámara detecta la ubicación de un marcador visual, si no conoce su altura, entonces se está interpretando la distancia desde la referencia hasta el centroide del marcador visual como si este estuviera a la misma altura que los marcadores de referencia. Ya que esto no es verdad, se tiene que hacer una corrección de la distancia medida. En la parte superior derecha de la figura 6.5, se observa una representación de un marcador visual desde el punto de vista de la cámara. Desde esa perspectiva, no es posible apreciar que el marcador visual está a una altura diferente del nivel del piso, no obstante, la cámara no lo percibe. Si el sistema fuera un sistema estereoscópico, se detectaría la profundidad y por lo tanto la altura. Al tratarse de un sistema con una sola cámara, se debe corregir la medición, para que sea útil.

La forma de corregir esa distancia medida es a partir de dos alturas conocidas, por una parte la altura a la que estaría el marcador visual (en este caso al altura del robot), y por el otro, la altura a la que está la lente de la cámara, ambas con

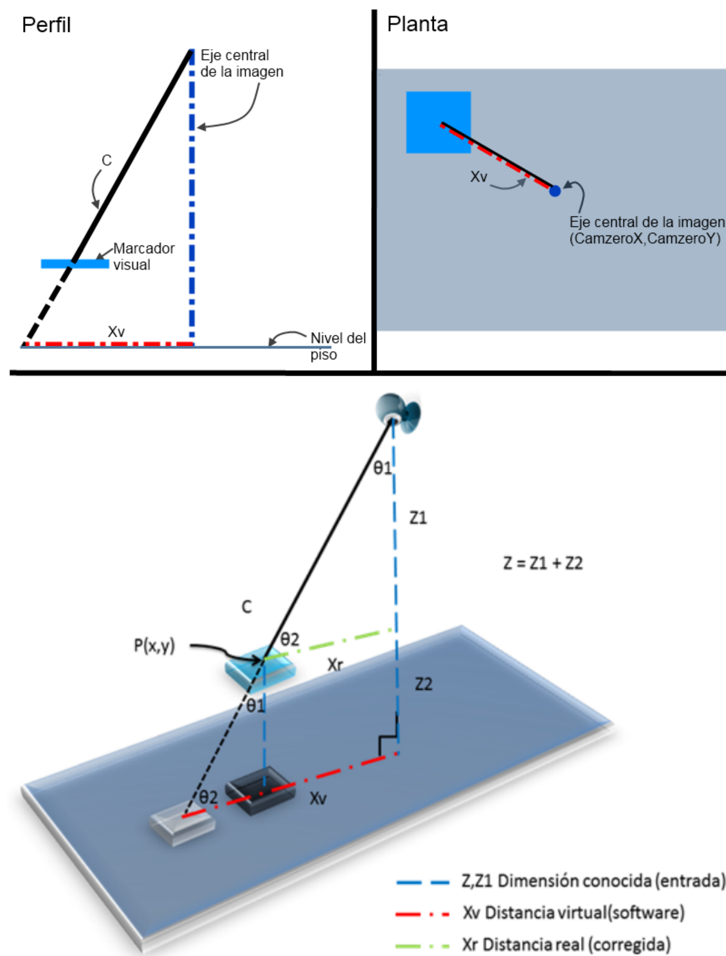


Figura 6.5: Vistas de un marcador visual desde la cámara

respecto del nivel del suelo. Luego, se obtienen la orientación del marcador visual y se hace una traslación en el plano, proporcional al triangulo semejante resultado de las alturas del robot de la cámara.

En el siguiente cuadro de código se muestran las principales operaciones realizadas para corregir la posición del objeto (robot) y trasladarlo sobre el eje definido a partir de su centro virtual y el centro de imagen de la cámara.

Esta corrección permite también obtener las distancias y ubicaciones de los objetos en unidades de medida reales. Tras la corrección de la imagen debida a la calibración y ésta corrección de posición, se puede decir que la localización es verosímil.

Algoritmo 6.5 Corrección de la posición de los marcadores visuales

```
//Calcula la distancia ente la cámara y el objeto
Z1= Z-Z2;

//Calcula la distancia entre el centro real de la camara y el centro del objeto
Xv= sqrt((pow(Xref-CamzeroX,2))+(pow((Yref-CamzeroY),2)));

//Calcula el angulo ente el centro de la camara y el centro del objeto
blobangle=atan((CamzeroY-Yref)/(CamzeroX-Xref));

//Calcula la hipotenusa formada entre el centro virtual del objeto y la cámara
C=sqrt((pow(Z,2))+(pow(Xv,2)));

//Calcula el ángulo entre la hipotenusa y la distancia virtual desde el centro de
//la cámara hasta el centro del objeto

theta1= asin(Xv/C);
Xr= (tan(theta1))*(Z1);

//Definición del centro del objeto corregido
CorrectionX= CamzeroX + (Xr*(cos(blobangle)));
CorrectionY= CamzeroY + (Xr*(sin(blobangle)));
.
.
.
//Definición del centro corregido y escalado del objeto
float correctiondistance= sqrt((pow((CorrectionX-OriginXX),2))+
        +(pow((CorrectionY-OriginXY),2)));
float PositionX=abs(correctiondistance*(cos(correctionangle)))*scale;
float PositionY=abs(correctiondistance*(sin(correctionangle)))*scale;
.
.
.
```

6.6. Comunicación UDP

El protocolo UDP (por sus siglas en inglés User Datagram Protocol), permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. Tampoco tiene confirmación ni control de flujo y tampoco se sabe si ha llegado correctamente, ya que no hay confirmación de entrega o recepción. Su uso principal es para protocolos en los que el intercambio de paquetes de la conexión y desconexión son mayores o no son rentables con respecto a la información transmitida, así como para la transmisión de audio y vídeo en tiempo real, donde no es posible realizar retransmisiones por los estrictos requisitos de retardo que se tiene en estos casos.

Dentro del programa aquí descrito, servirá para transmitir la información de la

posición capturada hacia el programa que controla al robot móvil. Naturalmente, en dicho programa de control, debe existir un *socket* o receptáculo para dicha información bajo las condiciones del protocolo UDP.

Algoritmo 6.6 Comunicación UDP

```
//Estructura para enviar datos via UDP
WSAStartup(MAKEWORD(2,2), &wsaData);
SendSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
RecvAddr.sin_family = AF_INET;
RecvAddr.sin_port = htons(Puerto);
RecvAddr.sin_addr.s_addr = inet_addr(ip);
sendto(SendSocket,SendBuf,strlen(SendBuf)+1,0,(SOCKADDR *) &RecvAddr,
    sizeof(RecvAddr));
```

El programa descrito en este apartado, requiere indicaciones sobre la dirección IP y el puerto del destino para la recepción de datos. Los datos se envían en una variable de tipo cadena con decodificación ASCII. En su recepción se deben asignar los datos recibidos a variables. El formato de la cadena es *;X;XXX;XXX;XXX;XX* donde entre cada signo de punto y coma se envían (de izquierda a derecha) el identificador de la figura, la posición del centoride en el eje X, posición del centroide en Y, el ángulo de orientación de la figura y dos valores que indican el cuadrante. Esta cadena debe ser separada nuevamente por el receptor para poder interpretar los datos.

6.7. Archivo .xml

Adicionalmente, para guardar la información de la calibración y ajustes, se genera un archivo con extensión *.xml*. XML significa, por sus siglas en inglés eXtensible Markup Language ('lenguaje de marcas extensible'), es un lenguaje de marcas. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones se deben comunicar entre sí o integrar información. XML no sólo tiene gran aplicación para Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier otra aplicación que lo requiera.

Para este programa se empleó la librería *xml.h* y adicionalmente se incluyó un programa auxiliar para poder utilizar de forma simple las funciones de escritura y lectura de los archivos con extensión *.xml* generados.

Capítulo 7

Experimento de rastreo sobre un robot móvil

Las pruebas del sistema de visión se realizaron sobre un robot móvil (2,0) cuya función consiste en desplazarse sobre un plano, con una trayectoria recta hasta encontrarse con una plataforma deslizante pasiva. La plataforma consta de cuatro ruedas, las dos traseras acopladas al mismo eje y dos ruedas locas al frente; también cuenta con un par de ganchos para anclarse al robot. Una vez, que el robot se sujeta con la plataforma, éste cambia su configuración a (1,1) y tiene una nueva trayectoria. Todo lo referente al robot, fue desarrollado fuera de este trabajo escrito por Oscar Xavier Hurtado Reynoso, sólo se describe para una mejor explicación de las pruebas.

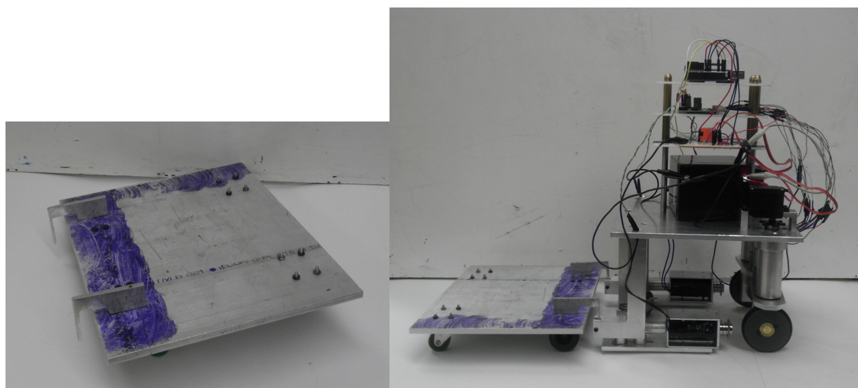


Figura 7.1: Plataforma y Robot acoplado a la Plataforma

A continuación se muestran los componentes de las pruebas y su funcionamiento.

7.1. Descripción de los elementos requeridos para el experimento

7.1.1. Cámara

La cámara utilizada para la prueba experimental, es una Microsoft LifeCam Studio (figura 7.2) conectada a la computadora a través de un cable USB. Es un dispositivo de captura de video que se clasifica como una Webcam HD (Cámara Web de alta definición) de gama alta. La cámara brinda una resolución máxima para captura de video de 1280 x 720 pixeles. La pérdida de la linealidad asociada a la curvatura de la lente de la cámara es corregible mediante el programa de calibración desarrollado para este trabajo. Resulta fundamental para la calidad de la imagen adquirida que la cámara se encuentre muy bien enfocada sobre la superficie de trabajo, a fin de evitar una imagen borrosa o desvanecida y que la superficie de trabajo este muy bien iluminada. La velocidad promedio de captura de la cámara, en condiciones suficientes de procesamiento, es de 30 fps (cuadros por segundo).

Figura 7.2: Cámara utilizada para el experimento

7.1.2. Robot

El robot utilizado para la prueba experimental es el mostrado en la figura 7.3. Se trata de un robot por diferencial eléctrico tipo (2,0). Esto quiere decir que tiene dos llantas fijas sobre el mismo eje y no se direccionan las llantas. Su direccionamiento está basado en la velocidad diferencial entre la llanta derecha e izquierda.

Por motivos de estabilidad y soporte, el robot cuenta con una rueda loca o castora, no direccionable. Cuenta con pisos de acrílico para la colocación de los circuitos necesarios para controlar las velocidades de las llantas y manejar la comunicación con la computadora. Los motores son de corriente directa, modelo EMG-30, que funcionan con un voltaje nominal de 12[V]. Cuentan con encoders de cuadratura internos de 360 pulsos por vuelta, acoplados al eje y una caja reductora con una

relación 30:1, también con un capacitor entre las líneas de alimentación del motor, como supresor de ruido. La batería utilizada es de 12[V] y 4.0[Ah] recargable, lo que proporciona una autonomía al robot de aproximadamente 30 minutos en operación continua.

Adicionalmente, por motivos de su aplicación, este robot cuenta con un par de solenides montados en la base del robot, que sirven para sujetar a la plataforma pasiva al momento de transportarla. Estos solenoides están controlados directamente por el microcontrolador.

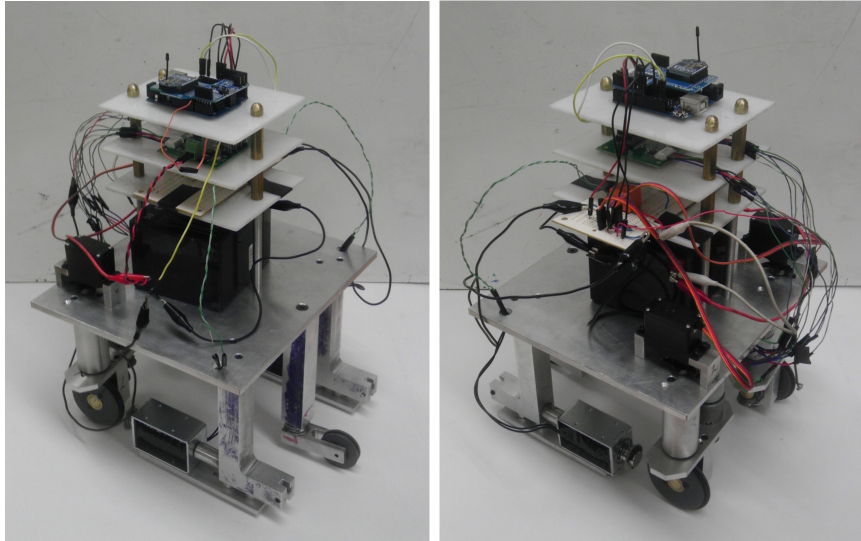


Figura 7.3: Robot utilizado para la prueba

Los planos del robot están disponibles en el Apéndice X.

7.2. Descripción general del sistema

El sistema que se creó para las pruebas, es semejante al de los desarrollos previos. Consta de diferentes etapas de flujo de información. Estas se muestran de forma esquemática en la figura 7.4 Y a continuación se explican a detalle.

7.2.1. Etapas del flujo de información

7.2.1.1. Captura de la imagen

La primera etapa de flujo de información inicia en el entorno físico. Es decir, se inicia cuando la cámara captura las imágenes del espacio de trabajo. En éste, deben estar los marcadores visuales que definen el eje horizontal de referencia y también los marcadores visuales que indican la posición del robot, montados sobre el último piso del mismo. Todos deben ser visibles desde el primer instante de captura y deben ser de tamaños diferentes y colores contrastantes con el resto de la escena.

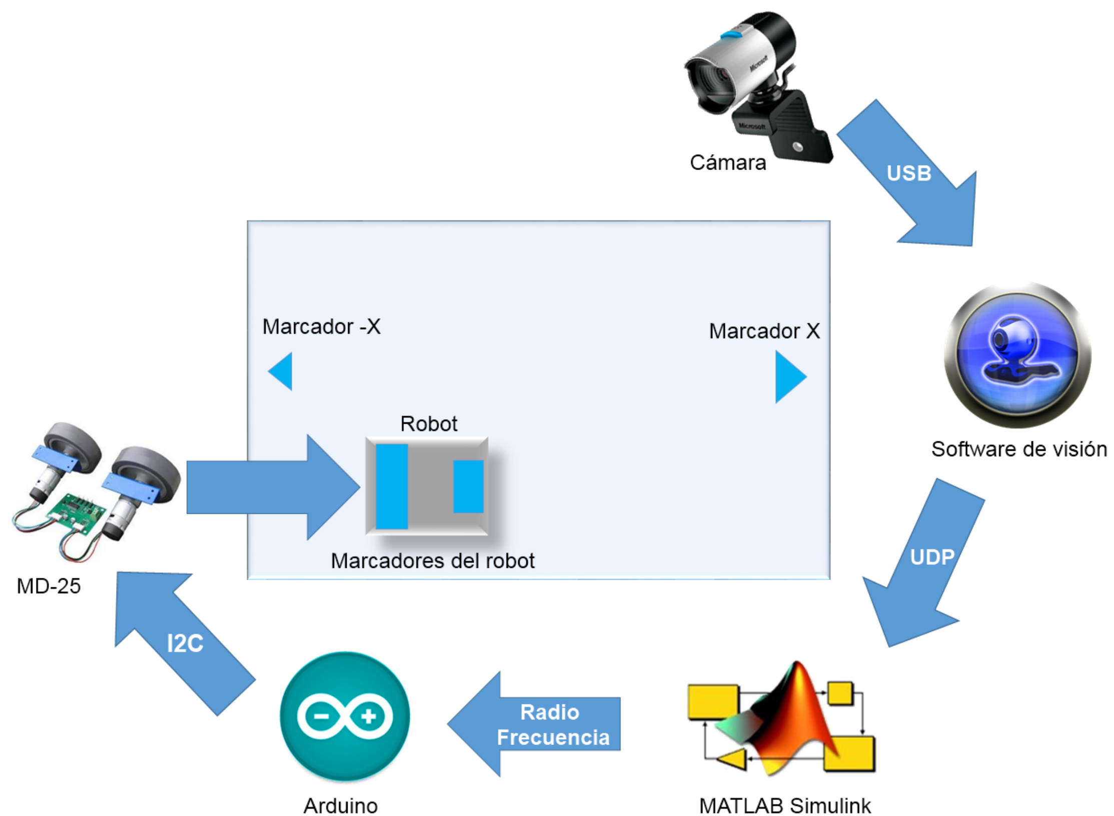


Figura 7.4: Diagrama esquemático del sistema

Al ser detectados por la cámara, ésta envía la información de cada cuadro capturado en forma de mapa de bits al programa desarrollado para este proyecto. El programa interpreta las imágenes, siguiendo los procedimientos de segmentación y detección de regiones conectadas descritos en capítulos anteriores. Dentro de la misma aplicación, se genera un sistema coordenado que funge como referencia para la posición y orientación del robot. La posición se escala a partir de parámetros reales conocidos, como la altura a la que está la cámara y la altura del robot con respecto del suelo. Con dicha información se obtiene una correspondencia entre pixeles y centímetros, y es posible saber a qué distancia está el robot del centro del sistema coordenado. Esos datos de posición y orientación, se obtienen una vez por cada cuadro y se almacena en una variable de cadena como texto. Dicho texto es después desplegado en pantalla y simultáneamente enviado mediante el protocolo UDP por el puerto 1000.

7.2.1.2. Envío de datos al programa de control

Como se mencionó, el software de visión envía una cadena de datos, integrada por las coordenadas de posición y el ángulo de orientación del robot. Dicha cadena

es recibida por un módulo del programa, desarrollado en MatLab Simulink. Primero se encuentra un bloque receptor de datos a través de UDP con una dirección IP dada (0.0.0.0. o *localhost* por default) y un puerto (en este caso es el puerto 1000). En la siguiente imagen, se muestra cómo está configurado el subprotocolo desarrollado para el envío de los datos, las "X" son caracteres numéricos. El valor de identificación, es el número de la pieza mayor de cada pareja de identificadores visuales. Las posiciones están enviadas en centímetros y el ángulo en grados. Los últimos dos valores, indican el número de cuadrante en binario.

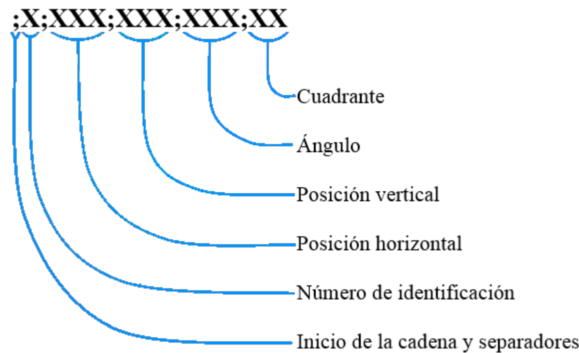


Figura 7.5: Protocolo de la cadena de datos enviada

La siguiente parte del programa cuenta con un bloque decodificador de código ASCII (acrónimo inglés de American Standard Code for Information Interchange o Código Estándar Estadounidense para el Intercambio de Información), que reconvierte los datos enviados a caracteres numéricos. Los datos recibidos y decodificados, son ordenados e interpretados. Durante la recepción, el programa cuenta el número de datos recibidos hasta que son 17, entonces se corta la cadena y se espera a la siguiente.

A la salida de este bloque decodificador, se obtienen los datos de posición y orientación en tres variables. En los Apéndices IV a VI se muestra el código fuente y los diagramas de los bloques empleados.

7.2.1.3. Programa de control en Simulink

El modelado del control del robot móvil, se basa en la teoría de campos potenciales. La cuál se consiste en la regulación de la velocidad lineal y angular del robot móvil en su totalidad. Ésta se basa en el concepto de que existen ciertos radios de acoplamiento o 'docking' y preacoplamiento o 'predocking', en los cuales se disminuye la velocidad inversamente proporcional a la distancia, es decir, entre más cerca esté menor velocidad. Para lograr el acoplamiento, primero se tiene que llegar a un punto de predocking, donde el robot se alinea con la plataforma pasiva. Dentro de ese radio, se reduce la velocidad y se continúa acercando. Cuándo el robot se ha acercado lo suficiente se llega al radio de 'docking'. Una vez ahí, se concluye la tarea de

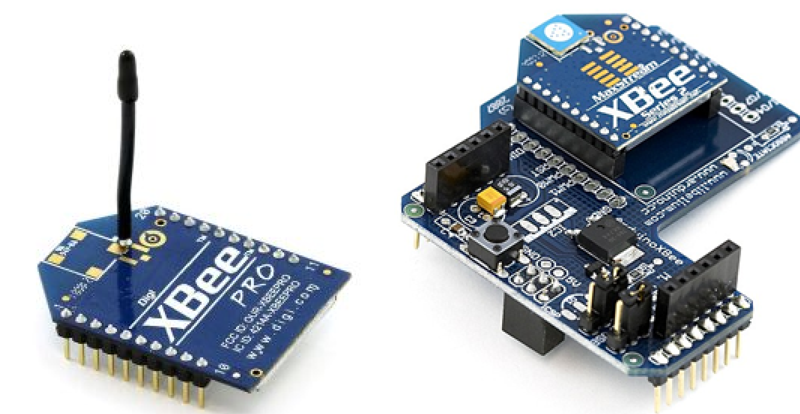
acoplamiento entre el robot y la plataforma. El control se basa en modificaciones del modelo desarrollado por el Dr. Víctor Javier González Villela para robots móviles.

En la primera fase de la tarea el robot actúa con una configuración (2,0) donde sólo tiene dos ruedas de tracción con diferencial para las vuelta en tanto que para la segunda parte de la tarea, después del acoplamiento, el robot actúa con una configuración (1,1) donde se direccionan las ruedas frontales mediante servomotores pero se pierden grados de maniobrabilidad debido a las ruedas extra de la plataforma de carga.

7.2.1.4. Transmisión de instrucciones de Simulink al microcontrolador

La comunicación entre el programa de control ejecutado en una computadora y la electrónica del robot, se realiza de forma inalámbrica mediante módulos XBee. Los módulos XBee son módulos de radiofrecuencia que trabajan en la banda de los 2.4[GHz] con un protocolo de comunicación 802.15.4 (Zigbee) estandarizado por la IEEE. Son fabricados por la empresa Digi®. Normalmente son utilizados en automatización de casas, sistemas de seguridad, monitoreo de sistemas remotos, aparatos electrodomésticos, etc. Cuentan con un alcance en interiores de hasta 30[m] y en exteriores de hasta 100[m].

Para este experimento se utilizan dos módulos XBee. Uno como emisor conectado a través de una interfaz USB a la computadora, que lo detecta y configura como un puerto serial. El otro módulo sirve como receptor, se encuentra acoplado al microcontrolador mediante una placa de circuito impreso (*shield*), la cual está diseñada para conectar el XBee con el microcontrolador. Ambas mostradas en la figura 7.6.



(a) Emisor de radiofrecuencia *XBee.*

(b) Receptor *XBee* insertado sobre el *shield.*

Figura 7.6: Módulos *XBee*

Desde el programa de control, empleando el bloque “Serial Send” se envían todos los datos para el XBee. Se envían los datos de posición en grados para los servomotores que direccionan los ejes de las ruedas de tracción mediante datos enteros de

8 bits. Después, de la misma forma se envían los valores de velocidad. En este caso después de enviar los valores de velocidad, se envía un valor de 0 o 1 que funciona como un booleano para indicar la activación de los solenoides.

Al recibirse, el microcontrolador interpreta los datos de acuerdo al orden en que arriban, para de esta forma controlar los diversos actuadores del robot móvil.

7.2.1.5. Comando del microcontrolador hacia la tarjeta MD-25

El microcontrolador descrito anteriormente, interpreta dichas instrucciones recibidas para convertirlas en una señal para controlar los motores del robot. Para ello, se utilizó la tarjera MD25.

La tarjeta MD25 es un controlador de doble *puente H* diseñado para manejar dos motores EMG30, a través de comunicación serial o I2C con un microcontrolador. Para este caso se decidió utilizar la I2C para dejar la serial libre para la comunicación entre el microcontrolador y la computadora. Esta tarjeta facilita el control de los motores, gracias a los comandos preestablecidos que sólo requieren la activación de distintos registros.

La figura7.7 muestra la tarjeta con sus componentes. Esta tarjeta es capaz de leer los encoders de los motores para conocer la distancia recorrida y la dirección. También puede controlar dos motores de forma independiente o combinada y permite establecer diferentes rectas de aceleración, y regular la potencia de cada motor, con capacidad de obtener la corriente requerida por cada motor. Para este proyecto, los motores se manejan en modo independiente con datos de entrada de velocidad que van de 0 a 255, siendo 0 la velocidad máxima en un sentido y 255 la velocidad máxima en sentido contrario.

Únicamente requiere de una alimentación de 12[V], además un regulador de 5[V] montado sobre la tarjeta es capaz de alimentar la circuitería externa, como el microcontrolador, a una corriente continua de 300[mA].

A la tarjeta MD25 se le envían datos de velocidad para las llantas mediante una función lineal que transforma la relación de [rad/s] a valores de entre 0 y 255 . El microcontrolador está programado mediante la interfaz de Arduino en su lenguaje nativo. En este caso, se emplea la librería "Wire". Esta librería permite comunicarse con dispositivos I2C y TWI. En las placas Arduino UNO con el diseño R3, la SDA (línea de datos) y SCL (línea de reloj) se encuentran en los pines A4 y A5 respectivamente. Dentro de programación se hace un llamado a dicha librería se inicia la comunicación, se selecciona un registro de dirección predeterminado por el fabricante de la tarjeta MD25 y se establece el valor de velocidad a otro registro también preestablecido por el fabricante.

Más detalles de esta tarjeta se encuentran en el Apéndice IX.

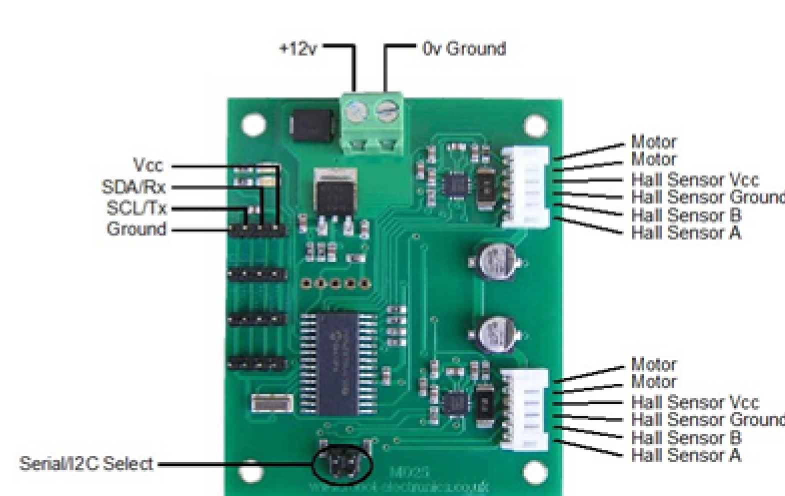


Figura 7.7: Tarjeta MD-25 y sus componentes

7.3. Condiciones para las pruebas

7.3.1. Entorno de pruebas

La prueba fue ejecutada a nivel laboratorio. Como se aprecia en la figura 7.8, ahí se colocó la cámara descrita al inicio de éste capítulo en el techo del laboratorio, siguiendo el esquema explicado al final del capítulo tercero de éste texto. Con la cámara ubicada aproximadamente a 2.38[m] sobre el nivel del piso, se genera un espacio de trabajo aproximadamente de 1.62[m] en el eje horizontal y 1.3[m] para el eje Y. Con estas constantes de proporción se hace el escalamiento de los datos de visión, como se describen en el capítulo anterior.

Los marcadores visuales son cuatro recortes de papel cartulina color azul. Se eligió ese color simplemente porque es suficientemente contrastante con el fondo. No obstante, de haber elegido otro para la prueba, éste pudo haberse calibrado dentro del menú de ajustes del programa desarrollado. Cómo se ha explicado, su forma no es importante, sin embargo su área sí lo es. Las dos piezas que definen la referencia dentro del espacio de trabajo son de 15[cm2] y 27.5[cm2]. Las dos piezas que van montadas sobre el robot, que definen su orientación y posición son de 44.5[cm2] y 100[cm2]. Y las dos piezas que van sobre la plataforma pasiva son de 135[cm2] y 190[cm2]. Se eligieron esas áreas ya que dentro del área de visión de la cámara se apreciaban claramente.

Los marcadores visuales puestos sobre el robot y la plataforma, están colocados de tal forma que el centro entre los centroides de cada figura, se encuentra comprendido al centro del eje de las ruedas.

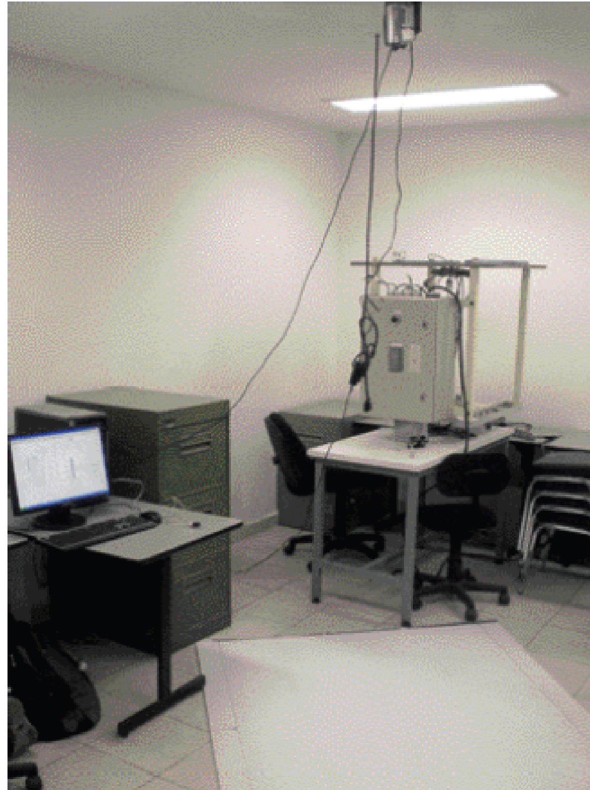


Figura 7.8: Banco de pruebas en laboratorio

7.3.2. Operación del sistema

En este apartado se describen los pasos que deben llevarse a cabo para poner el sistema de pruebas en funcionamiento.

Primero, la cámara debe estar en su posición inicial, una donde se aprecie la totalidad del área de trabajo. La cámara debe estar perpendicular al suelo y ubicada en un punto central del espacio de trabajo.

Después deben colocarse todos los marcadores visuales que serán detectados por el sistema de visión, tanto los que definen al eje horizontal cómo los que vayan sobre el robot u otras referencias. También, debe colocarse al robot en su posición de inicio, previo a su operación.

Deben conocerse la altura del robot, la altura a la que está la cámara y la distancia entre los marcadores visuales que definen al eje horizontal de referencia. Dichos valores deben ser introducidos al programa a través del archivo “Settings.xml” empleando el programa “Configuration File.exe”.

Tanto la cámara como el emisor XBee de radio frecuencia deben estar conectados, reconocidos por la computadora y por los programas.

Por último se ejecuta el sistema de visión desarrollado en este proyecto y posteriormente el programa de control de Simulink.

7.4. Objetivo de las pruebas

Cómo se ha dicho a lo largo del presente escrito, la finalidad del sistema de visión desarrollado es la de auxiliar al control de robots móviles. Los requerimientos para poder ser empleado para el control son dos. Por un lado se requiere una medición precisa de la ubicación del robot para que ésta le sirva al programa de control para conocer en qué etapa de las acciones se encuentra el robot. Por el otro, se requiere una velocidad suficiente, tal que la información de la posición obtenida pueda ser utilizada por el programa de control para mantener el curso de las acciones del robot o incluso para dirigirlas o cambiarlas.

Por lo tanto, las pruebas buscan medir tanto la precisión como la velocidad .

7.5. Descripción de las pruebas

7.5.1. Prueba de precisión

Esta prueba consiste en comprobar mediante un cuadro de captura estático, la capacidad del programa desarrollado de determinar la posición y orientación de una pareja de marcadores visuales. De esta manera puede comprobarse la precisión del sistema de visión. En esta no se pone a prueba su utilidad con los robots móviles.

7.5.2. Prueba de lazo abierto

Un sistema de lazo abierto es aquel en que sólo actúa el proceso sobre la señal de entrada y da como resultado una señal de salida independiente a la señal de entrada, pero basada en la primera. Esto significa que no hay retroalimentación hacia el controlador para que éste pueda ajustar la acción de control. Es decir, la señal de salida no se convierte en señal de entrada para el controlador.

Una prueba de lazo abierto significa en este caso, que el control del robot se hará únicamente mediante el envío de instrucciones. Es sistema de visión, será ejecutado en paralelo y utilizado solamente para monitorear las acciones del robot. Al final, se compararán las medidas reales de desplazamiento, con las obtenidas mediante el software de visión. Estos resultados dirán cuánto error existe entre la media real comparada con la medida capturada. Esta será la forma de determinar la precisión del programa de visión. Lo anterior se muestra en la figura 7.9

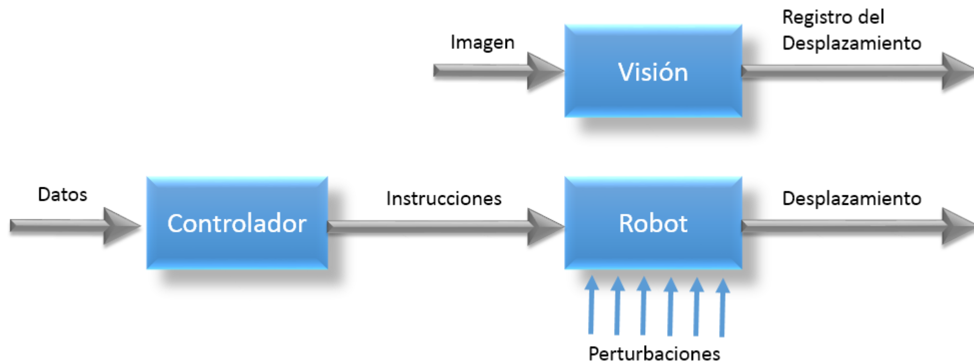


Figura 7.9: Esquema de la prueba de lazo abierto

7.5.3. Prueba de lazo cerrado

En los sistemas de lazo cerrado, la acción de control está en función de la señal de salida. Los sistemas de circuito cerrado usan la retroalimentación desde un resultado final para ajustar la acción de control en consecuencia.

En la prueba de lazo cerrado, el control del robot se retroalimentará con la información enviada por el sistema de visión a manera de un punto de ajuste. En este caso, el control del robot depende directamente del sistema de visión, por lo que deben ser ejecutados bajo los mismos parámetros de velocidad de muestreo. De esta forma, se puede medir si la velocidad del robot es suficiente para cumplir con los objetivos e incluso si es mayor de lo necesario. Lo anterior se ejemplifica en la figura 7.10

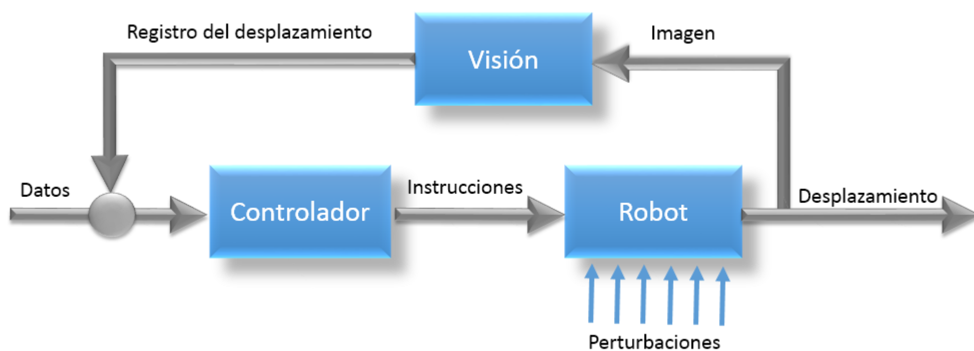


Figura 7.10: Esquema de la prueba de lazo cerrado

Capítulo 8

Resultados

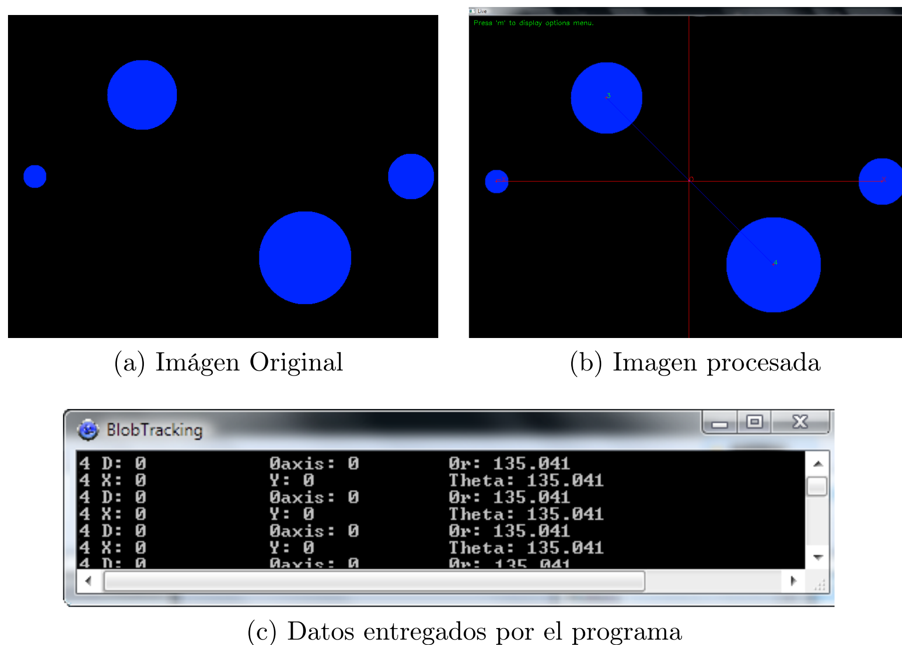
Cómo se menciona en el capítulo anterior, se realizaron diversas pruebas del sistema de visión como auxiliar para un robot móvil siguiendo dos distintos esquemas, uno de lazo abierto y otro de lazo cerrado. La trayectoria programada en ambos casos, consiste en tres etapas. La primera etapa es una línea recta desde la posición inicial del robot, hasta el punto de *predocking*. La segunda etapa es una línea recta a una velocidad menor desde la posición de *predocking* hasta la posición de *docking* o acoplamiento con la plataforma móvil. La última etapa de la trayectoria, consiste en un segmento curvilíneo hacia la derecha del robot. Se definió una trayectoria diferente debido a que de manera paralela, estas pruebas sirven para comprobar el correcto funcionamiento del control. Se optó por una curva muy abierta, ya que las restricciones mecánicas del robot al momento que se realizaron las pruebas solo permitían curvas de esa naturaleza.

8.1. Resultados de la prueba de precisión

En la prueba de presición se utilizó una imágen estática generada artificialmente emulando una captura de la cámara. Las variaciones de iluminación alteran los bordes de los marcadores visuales, es por ello que se decidió generar una imagen en vez de utilizar una captura real. En la figura 8.1 se muestra la imágen que se utilizó y cómo fue interpretada por el sistema de visión.

La posición no presenta error alguno, esto se debe a que en la imagen las figuras son nítidas en cada pixel y están colocadas con presición. Por lo que después de hacer el barrido pixel por pixel, el programa identifica los centroides de las figuras exactamente.

El ángulo por otro lado, no se mide, sino que se calcula a partir de los datos de los centroides. Es por ello que se encuentra un error en el valor arrojado por el programa. En esta prueba, el programa calcula un ángulo de 135.041º (figura 8.1c).



(a) Imágen Original

(b) Imagen procesada

(c) Datos entregados por el programa

Figura 8.1: Prueba de Precisión

Dado que las figuras están colocadas con precisión tal que se sabe que el ángulo correcto es de 135º, es posible calcular el error de la siguiente manera.

$$\%error = \frac{\Theta_{ideal} - \Theta_{calculada}}{\Theta_{ideal}} \times 100$$

$$\%error = \frac{135 - 135.041}{135} \times 100 = 0.3\overline{037}\,\%$$

Este error se debe a la forma en que internamente se calcula el ángulo y al truncamiento de valores que implica. Sin embargo, cómo se menciona en el Capítulo VII, el protocolo de comunicación únicamente envía los datos enteros del ángulo, por lo que el error aquí descrito es intrascendente en la aplicación.

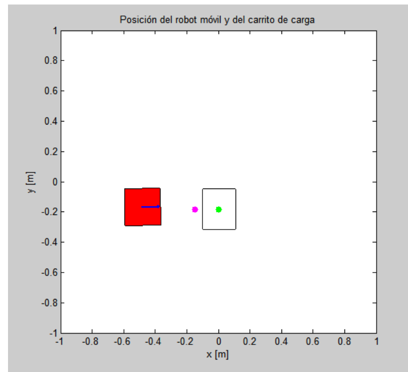
8.2. Resultados de la prueba de lazo abierto

En las pruebas de lazo abierto la información provista por el sistema de visión es independiente al control del robot. A continuación se muestran las imágenes y gráficas resultantes en las pruebas de lazo abierto

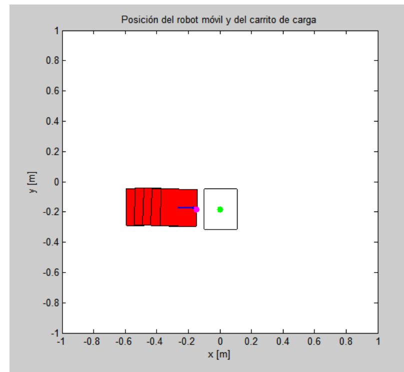
La figura 8.5 muestra las distintas etapas de la simulación, dichos datos fueron enviados directamente al robot. La estela del rastro del robot está tomada cada 2 segundos de ejecución de la simulación.

En la imagen 8.3 se observan fotografías editadas con la estela que refleja el movimiento seguido por el robot, que fue capturado por la cámara en intervalos de 2 segundos.

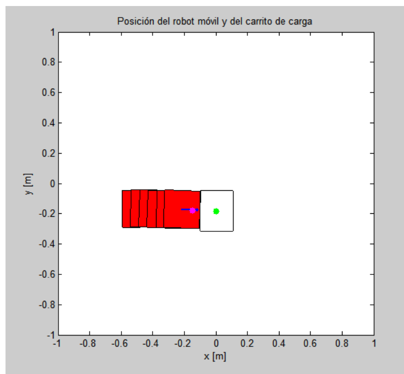
Desplazamiento de la simulación en lazo abierto



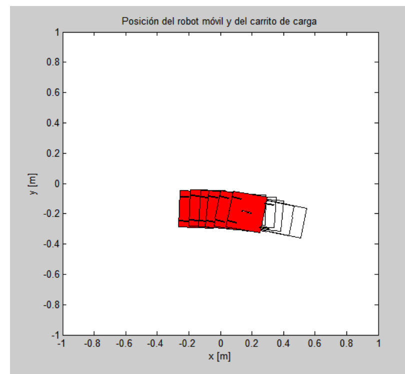
(a) Posiciones iniciales de la simulación



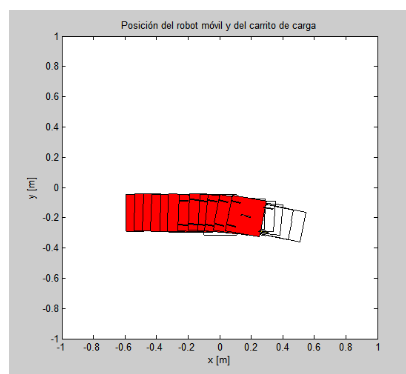
(b) Desplazamiento de la simulación hasta el punto de predocking



(c) Desplazamiento de la simulación hasta el punto de docking



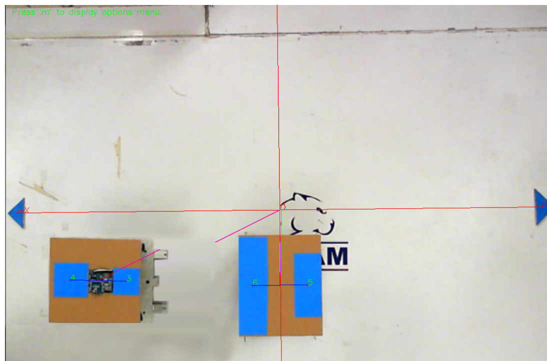
(d) Desplazamiento de la simulación desde el punto de docking hasta el punto final



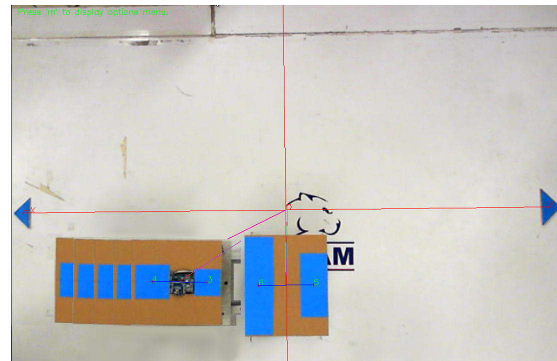
(e) Desplazamiento de la simulación completa con estela

Figura 8.2: Desplazamiento de la simulación

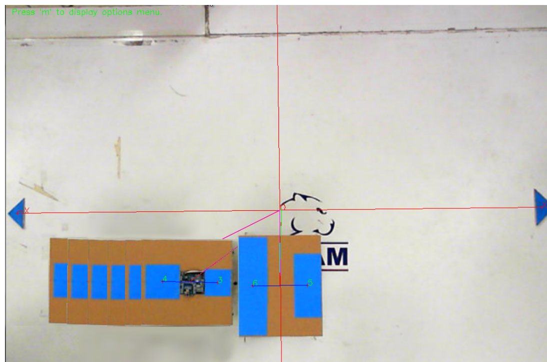
Desplazamiento del robot en lazo abierto



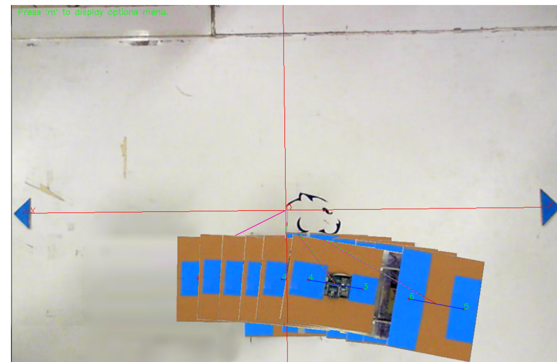
(a) Posción inicial del robot en captura



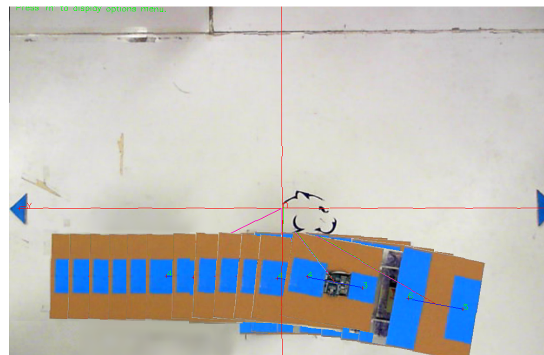
(b) Desplazamiento del robot en captura hasta el punto de predocking



(c) Desplazamiento del robot en captura hasta el punto de docking



(d) Desplazamiento del robot en captura hasta el punto final



(e) Desplazamiento del robot en captura completa

Figura 8.3: Desplazamiento del Robot

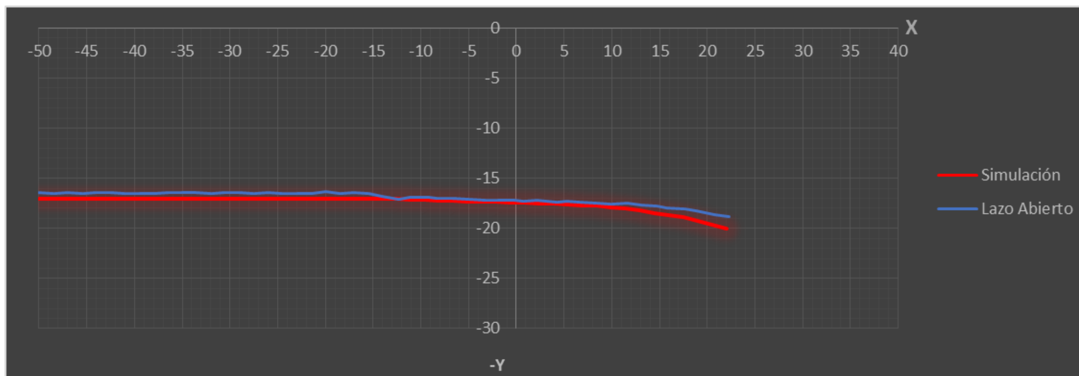


Figura 8.4: Gráfica comparativa entre el desplazamiento simulado y el capturado

Gráfica comparativa del desplazamiento

Aquí se aprecia que la captura es semejante a la simulación, sin embargo con errores que pudieran ser significativos en una aplicación que requiera un error menos a 2[cm] en el desplazamiento. No obstante, aunque un porcentaje del error pudiera deberse al sistema de visión, otro seguramente se debe al sistema de control. Como se desconoce la proporción, el cálculo del error no determinaría ni la veracidad del sistema de visión ni del control. Sin embargo, esta prueba sirve para demostrar la utilidad del sistema de visión aquí propuesto con robots móviles.

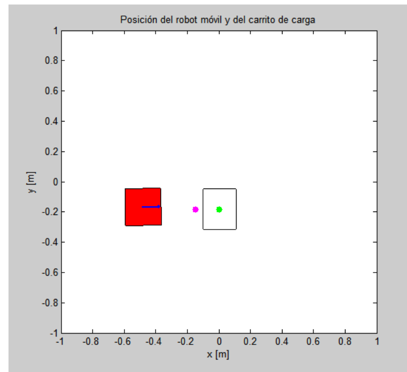
8.3. Resultados de la prueba de lazo cerrado

En las pruebas de lazo cerrado se emplea la información provista por el sistema de visión para realimentar el control del robot. A continuación se muestran las imágenes y gráficas resultantes en las pruebas de lazo cerrado.

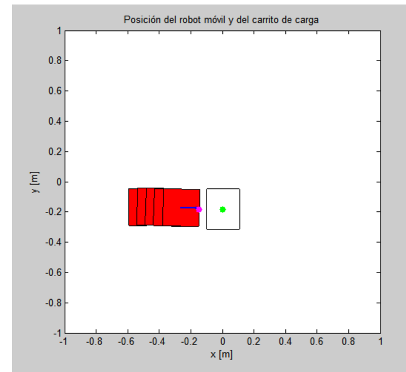
La figura 8.5 muestra las distintas etapas de la simulación, dichos datos fueron enviados al robot después de la realimentación provista por el sistema de visión. La estela del rastro del robot está tomada cada 2 segundos de ejecución de la simulación.

En la imagen 8.3 se observan fotografías editadas con la estela que refleja el movimiento seguido por el robot, que fue capturado por la cámara en intervalos de 2 segundos.

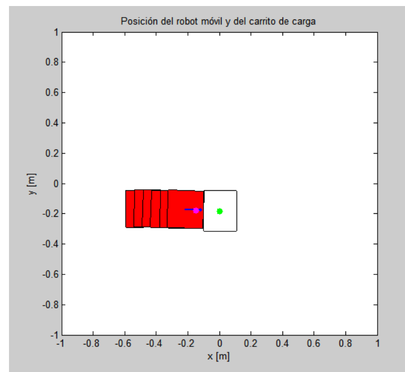
Desplazamiento de la simulación con control realimentado



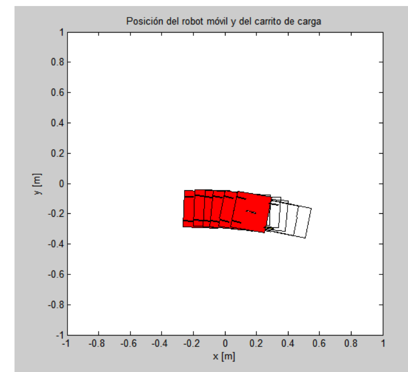
(a) Posiciones iniciales de la simulación



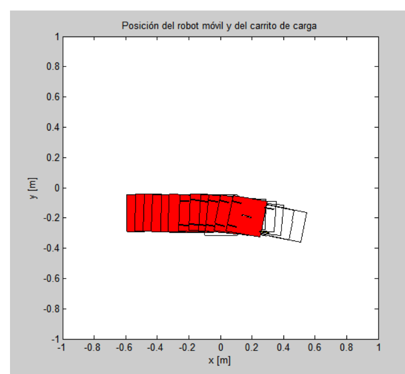
(b) Desplazamiento de la simulación hasta el punto de predocking



(c) Desplazamiento de la simulación hasta el punto de docking



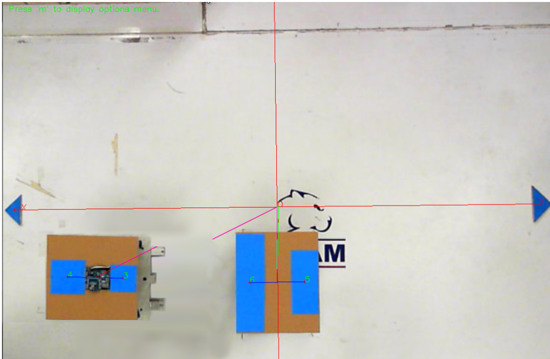
(d) Desplazamiento de la simulación desde el punto de docking hasta el punto final



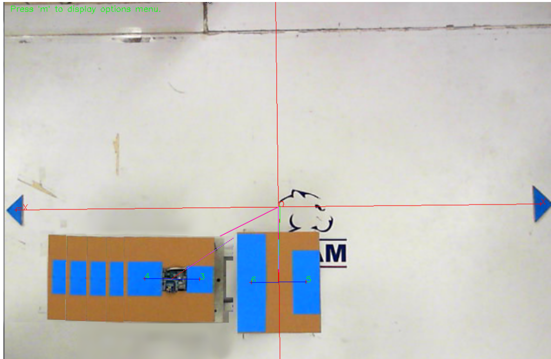
(e) Desplazamiento de la simulación completa con estela

Figura 8.5: Desplazamiento de la simulación con control realimentado

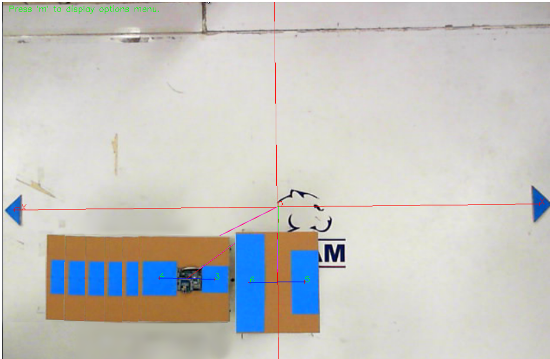
Desplazamiento del robot con control realimentado



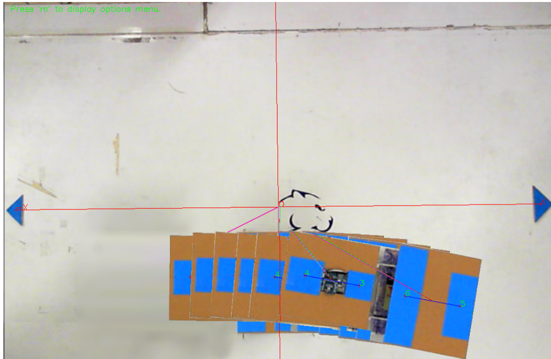
(a) Posción inicial del robot en captura



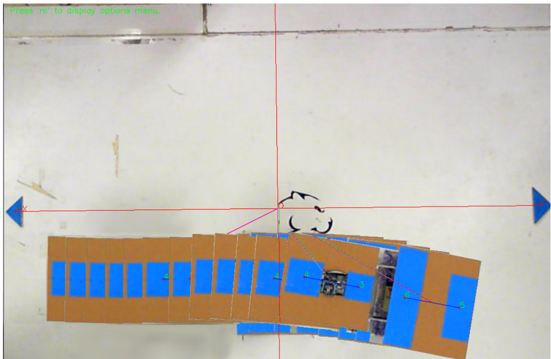
(b) Desplazamiento del robot en captura hasta el punto de predocking



(c) Desplazamiento del robot en captura hasta el punto de docking



(d) Desplazamiento del robot en captura hasta el punto final



(e) Desplazamiento del robot en captura completa

Figura 8.6: Desplazamiento del robot con control realimentado

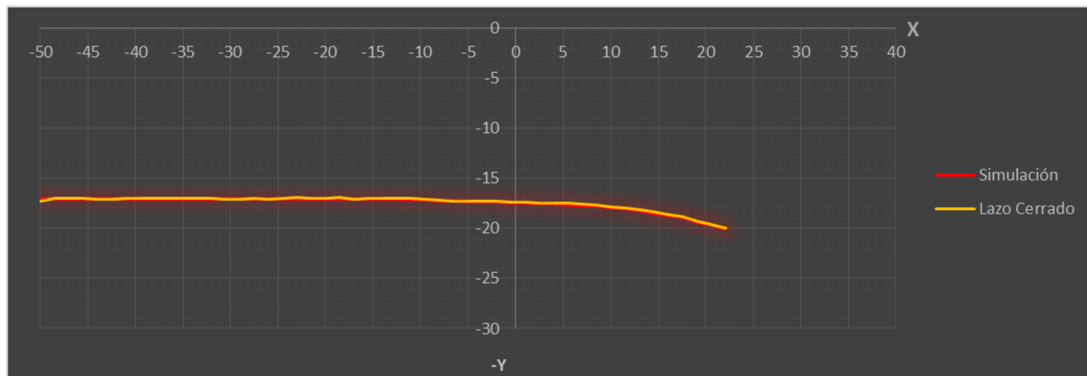


Figura 8.7: Gráfica comparativa entre las posiciones enviadas y las recibidas

Gráfica comparativa del desplazamiento

Aquí se observa que la simulación es mucho más similar a la captura que en la prueba anterior. Reflejando así un comportamiento adecuado en el sistema de control.

Una vez más esta prueba describe la utilidad del sistema de visión para el control realimentado de un robot móvil. La simulación estuvo regulada por el bloque "Real-Time Pacer" con velocidad igual a 1. Dentro de la simulación, el tiempo de muestreo en los bloques de control era de 0.5[s] y así fue posible mantener una comunicación estable entre todos los componentes de la prueba. Este valor es menor al documentado en los trabajos previos desarrollados dentro del grupo de trabajo MRG.

No obstante, al igual que en la prueba anterior, los errores del control y los del sistema de visión están combinados, por lo que el cálculo del error no es concluyente. Esto se debe a que el sistema de control fue desarrollado paralelamente al sistema de visión y al momento de las pruebas el control seguía en una etapa temprana de su desarrollo.

Capítulo 9

Conclusiones y trabajo a futuro

En el presente documento se muestra el desarrollo de un sistema de visión para sensar parámetros de un robot móvil, compatible con el software Simulink® y se comprueba físicamente.

9.1. Conclusiones

En conclusión, los objetivos fueron alcanzados satisfactoriamente. El primero de ellos era desarrollar un software de visión por computadora con la capacidad de identificar marcadores visuales a través de sus características apreciables mediante una webcam convencional, como son su color y su área. Se logró hacer un programa en lenguaje C++ y emplear las librerías de visión OpenCV para ello. Dentro de los objetivos planteados, también se establecía la necesidad de que dicho sistema de visión permitiera obtener las coordenadas y orientación de un 'objeto' que se desplazara dentro del área enfocada por la cámara y posteriormente introdujera las lecturas de los datos a Simulink®.

El programa está diseñado tomando en cuenta las múltiples aplicaciones que se le ha dado a otros sistemas de visión dentro del grupo de trabajo MRG. Soporta cualquier tipo de webcam o videocámara conectada a una computadora. No requiere instalación, está basado en software libre y el código está disponible en los apéndices de este escrito. Mediante un programa de calibración, evita las distorsiones provocadas por la propia cámara. Corrige los defectos en la detección de la posición de los objetos con altura, debida a la perspectiva de la cámara. Los marcadores visuales, requeridos son muy sencillos de elaborar. El programa permite ajustar los rangos de tamaño y color de los marcadores visuales y las propiedades del espacio de trabajo. El sistema de visión puede detectar varios marcadores simultáneamente, hasta 50 parejas que pueden servir para identificar robots, obstáculos o cualquier objeto según la aplicación lo requiera. La velocidad del sistema está condicionada a la velocidad de refrescamiento de la cámara. Transmite los datos directamente al programa de control en Simulink®, evitando pasos intermedios que entorpecerían la

comunicación.

El segundo objetivo era utilizar el sistema de visión, cambiar el simple objeto por un robot móvil desplazándose sobre la superficie e introducir un control. Este objetivo se alcanzó utilizando un robot construido por Oscar Xavier Hurtado Reynoso y un control basado en el diseñado por el Dr. Víctor Javier González Villela. Adicionalmente, se logró que los datos provenientes del sistema de visión sirvieran para cerrar el lazo de control con la finalidad de seguir una trayectoria programada. Si bien lo anterior no era parte de los objetivos, sí representa una ventaja significativa y una utilidad mayor para darle otras aplicaciones. Una vez que el sistema de visión identificaba los marcadores correctamente, y el robot móvil por su parte era capaz de ser comandado a través de instrucciones comunicadas inalámbricamente desde una computadora, el paso final fue ensamblar todo el sistema en uno sólo, que contuviera el control, la codificación de las instrucciones hacia el robot, la comunicación inalámbrica y el sistema de visión. Todo funcionó exitosamente. El tiempo de muestreo empleado en las simulaciones fue de 0.5[s], mucho menor al reportado por proyectos similares previos.

9.2. Trabajo a futuro

Como consecuencia, durante el desarrollo de este proyecto surgieron ideas para incorporar mejoras a futuro. Una de esas mejoras implicaría la utilización de una computadora con mayor capacidad de procesamiento dado que el equipo utilizado resulta fundamental para la velocidad de procesamiento de datos. También podrían realizarse más pruebas del sistema de visión con otras trayectorias y otros robots móviles para comprobar su funcionalidad. Otra sería mejorar la interfaz gráfica del programa de configuración. En cuanto a la cámara, si se utiliza una de mayor velocidad de captura la velocidad de transmisión de los datos se incrementa. Al respecto de sus funciones, el sistema de visión aquí presentado, podría complementarse aplicando algoritmos para estimar o medir la velocidad de los objetos. Pensando más a futuro, podría plantearse la posibilidad de utilizar este programa con una cámara a bordo del robot como sensor de obstáculos. En general se pueden encontrar diversas aplicaciones, particularmente en el campo de la robótica móvil, para el que fue diseñado.

Bibliografía

[1] S. Blackaller. *Aplicación de software de visión y simulación en el seguimiento de trayectorias en robótica móvil*. Facultad de Inegniería, UNAM, June 2012.

[2] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Incorporated, 2008.

[3] Andrew Burton and John Radford. *Thinking in perspective: critical essays in the study of thought processes*. Methuen, 1978.

[4] C. Carnero Liñán. cvblob. Google Project Hosting, May 2012. http://goo.gl/OuuBD.

[5] Cristóbal Carnero Liñán. Cvblob faq, December 2012. http://goo.gl/r8CQi.

[6] CINVESTAV. ¿qué es mecatrónica?, February 2005. http://goo.gl/6tVFO.

[7] Oxford Dictionaries. 'robotics', April 2010. http://goo.gl/T9nLZ.

[8] M.B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM (JACM)*, 39(2):253–280, 1992.

[9] Fixational. Opencv vs matlab. Blog, March 2012. http://goo.gl/KSB4A.

[10] R.C. González, R.E. Woods, F.D. Rodríguez, and L. Rosso. *Tratamiento digital de imágenes*. Addison-Wesley Wilniington^ eDelaware Delaware, 1996.

[11] B Gopalakrishnan, S Tirunellayi, and R Todkar. Design and development of an autonomous mobile smart vehicle: a mechatronics application. *Mechatronics*, 14(5):491–514, 2004.

[12] L. He, Y. Chao, and K. Suzuki. A run-based two-scan labeling algorithm. *Image Processing, IEEE Transactions on*, 17(5):749–756, 2008.

[13] S. Jorda, M. Kaltenbrunner, G. Geiger, and R. Bencina. The reactable*. In *Proceedings of the international computer music conference (ICMC 2005), Barcelona, Spain*, pages 579–582, 2005.

[14] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. Tuio - a protocol for table based tangible user interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005)*, Vannes, France, 2005.

[15] Andrew Kirillov. Aforge.net framework, 2012. http://goo.gl/LZrOO.

[16] H. Kozono, S. Yokota, and T. Mori. Mechatronics coined by tetsuro mori. In *12th International Conference on Mechatronics Technology (ICMT 2008).*, 2008.

[17] A. Martínez. *Plataforma móvil omnidireccional a partir de dos robots móviles diferenciales*. Facultad de Inegniería, UNAM, February 2012.

[18] MathWorks. Overview, September 2012. http://goo.gl/sv00S.

[19] A. M. Ángeles and D. Lima. *Sistema Semi-autónomo de robots móviles colaborativos para el manejo de materiales*. Facultad de Inegniería, UNAM, March 2011.

[20] Anthony Oliver. Opencv vs. matlab vs. simplecv. Blog, April 2012. http://goo.gl/ak8Gj.

[21] OpenCV. Camera calibration and 3d reconstruction, 2012. http://goo.gl/KBHoK.

[22] Patrick. Family fiducial, July 2009. http://goo.gl/keCbC.

[23] L. Guillermo Restrepo. Una introducción a la visión de máquina, Diciembre 2007. http://goo.gl/trGXG.

[24] J. Rietdijk. Ten propositions on mechatronics. In *Mechatronics in Products and Manufacturing Conference, Lancaster University, UK*, pages 11–13, 1989.

[25] John J. G. Savard. The five seidel aberrations, 2004. http://goo.gl/IMkmt.

[26] L. Shapiro and G.C. Stockman. *Computer Vision. 2001.* Prentice Hall, 2001.

[27] Utkarsh Sinha. Why opencv? Blog, February 2010. http://goo.gl/qF1xo.

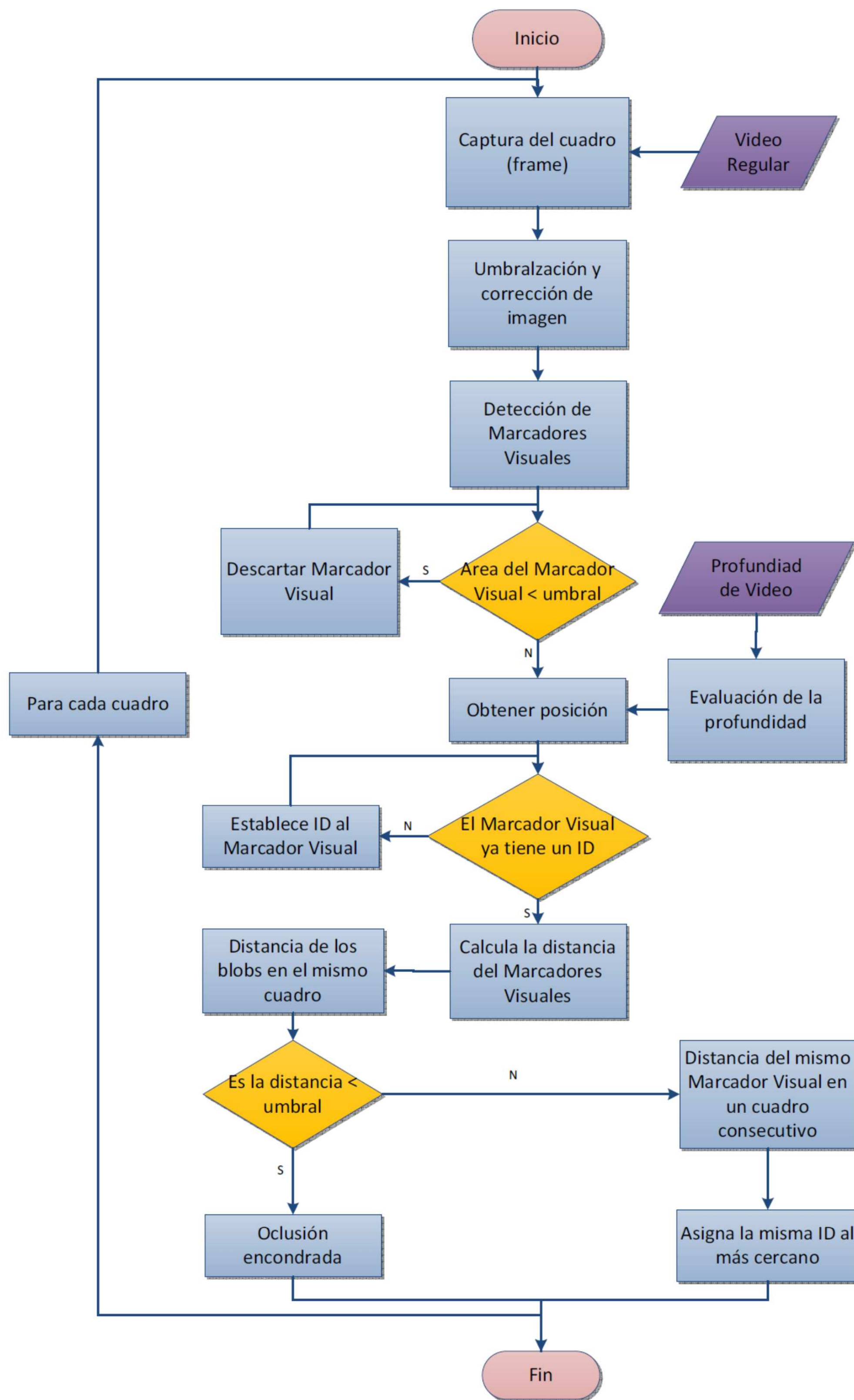
[28] G.P. Stein. Lens distortion calibration using point correspondences. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 602–608. IEEE, 1997.

[29] Wikipedia. Opencv, September 2012. http://goo.gl/wDqd2.

Apéndices

Apéndice I: Diagrama de flujo generalizado de un sistema de visión

.


Inicio
Captura del cuadro (frame)
Video Regular
Umbralzación y corrección de imagen
Detección de Marcadores Visuales
Descartar Marcador Visual
S
Area del Marcador Visual < umbral
Profundiad de Video
N
Para cada cuadro
Obtener posición
Evaluación de la profundidad
Establece ID al Marcador Visual
N
El Marcador Visual ya tiene un ID
S
Distancia de los blobs en el mismo cuadro
Calcula la distancia del Marcadores Visuales
Es la distancia < umbral
N
Distancia del mismo Marcador Visual en un cuadro consecutivo
S
Oclusión encondrada
Asigna la misma ID al más cercano
Fin

Apéndice II: Código fuente del programa de visión (en C++)

```
#pragma region Inclusions & Definitions
#include <winsock.h>
//Input-Output
//Entrada-Salida
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
//#include<math.h>
//#include<fstream>//Stream class to both read and write from/to files
//#include<string>
#include "resource.h"//Include source files
//OpenCV Headers
//Encabezados de OpenCV
#include <cv.h>
#include <cvaux.h>
#include <cxcore.h>
#include <highgui.h>
//Blob Library Headers
//Encabezado de Biblioteca de Blobs
#include <cvblob.h>
#include <string>   // Used for C++ strings
#include "ImageUtils.h"     // easy image cropping, resizing, rotating, etc
//Inclusions for XML writting
//Inclusiones para escribir en XML
#include "xml.h"
#include "json.hpp"
#pragma comment(lib,"wininet.lib")
#pragma comment(lib,"crypt32.lib")
#pragma comment(lib,"ws2_32.lib")
//Definitions of the screen size
//Definiciones del tamaño de pantalla
#define h 768
#define w 1024
//Definitions of constants
//Definicion de constantes
#define Pi 3.14159265358979323846
#pragma endregion
#pragma region NameSpaces
using namespace cvb;
using namespace std;
#pragma endregion

int main()
{
    #pragma region Open Settings.Xml
    //Declare the file with its name
    //Declara el archivo con su nombre
    char* file = ".\\Settings.xml";
    //Load file
    //Carga el archivo
    XML* x = new XML(file);
    //Parse status check
    //Checa el estado de lectura
```

```
int iPS = x->ParseStatus(); // 0 OK , 1 Header warning (not fatal) , 2 Error in parse (fatal)
bool iTT = x->IntegrityTest(); // True: OK
//Show erros message if unable to open
//Muestra mensaje de error si no se pudo abrir
if (iPS == 2 || iTT == false)
{
    fprintf(stderr,"Error: XML file %s is corrupt (or not a XML file).\r\n\r\n",file);
    delete x;
    return 0;
}
//Load intrinsec and distortion matrices back
//Carga las matrices de instrínsica y de distorción
CvMat *intrinsic = (CvMat*)cvLoad( "Intrinsics.xml" );
CvMat *distortion = (CvMat*)cvLoad( "Distortion.xml" );

#pragma endregion
#pragma region UDP Socket
    //Structure for UDP socket definition
    //Estructua para definir el Socket UDP
    WSADATA wsaData;
    SOCKET SendSocket;
    sockaddr_in RecvAddr;
    //Variables for the comunication
    //Variables para la comunicación
    int Puerto = XMLGetInt("UDP","Port",0,0,x);
    char ip[15] = "";
    XMLGetString("UDP","IP",0,ip,13,0,x);

#pragma endregion
#pragma region OpenCV Camera Configuration
//Structure to get information from CAM
//Estructura para obtener información de la CAM
int ID = XMLGetInt("Camera","ID",0,0,x);
//CvCapture* capture = cvCreateCameraCapture(0);//El numero determina la camara (e.g. 0 es
la camara por default,1 camara extra, 3 Webcam Max, 8 IP WebCam)
CvCapture* capture = cvCaptureFromCAM(ID);
//Check if the capture was succesfull
//Checa si la captura fue exitosa
if( !capture )
{
    fprintf(stderr,"Could not initialize capturing...\n");
    return -1;
}
//Windows that shows the capture
//Ventana que muestra la captura
cvNamedWindow("Live",CV_WINDOW_AUTOSIZE);
//Image Variables
//Variables de Imagen
IplImage *frame = cvCreateImage(cvSize(w,h),8,3);//Original Image //Imagen original
IplImage *hsvframe = cvCreateImage(cvSize(w,h),8,3);//Image in HSV color space // Imagen en
espacio de color HSV
IplImage *labelImg = cvCreateImage(cvSize(w,h),IPL_DEPTH_LABEL,1);//Image Variable for
blobs // Imagen variable para los blobs
```

```
    IplImage *threshy = cvCreateImage(cvSize(w,h),8,1); //Threshold image of yellow color
    //Umbral imagen de color amarillo
    IplImage *mhi = cvCreateImage(cvSize(w,h),32,1);    //Motion History Image update
    IplImage *orient = cvCreateImage(cvSize(w,h),32,1); //Orientation Image

    //Structure to hold Blobs
    //Estructura para retener Blobs
    CvBlobs blobs;

#pragma endregion
    #pragma region Variables Declaration
    //Get the screen information
    //Obten información de pantalla
    int screenx = GetSystemMetrics(SM_CXSCREEN);
    int screeny = GetSystemMetrics(SM_CYSCREEN);
    //Variables para definir el sistema de referencia
    float OriginXX, OriginXY, OriginYX, OriginYY, OriginYXpos, OriginYYpos;
    float CamzeroX, CamzeroY, CenterX, CenterY, CorrectionX, CorrectionY;
    //Variables for counters and flags
    //Variables para contadores y banderas
    int n=0, i=1, savedblobsflag=0, k=0, signX=1, signY=1;//positivos
    //Variables for Blobs discrimination
    //Variables para discriminar los blobs
    double blobarea[100]={0}, a1[100]={0}, a2[100]={0};
    //Variables for planar correction
    //Variables para la corrección de planos
    float angle, axisangle, axisref, scale, distance, blobangle, blobsdistance;
    float correctionangle, blobangleslope, axisslope, blobslope, angleblob;
    float Xv, Xr, Z, Z1, Z2, C, theta1;
    //Variables for color trackbars
    //Variables para las barras deslizables del color
    int MinHue=0, MinSat=0, MinVol=0;
    int MaxHue=0, MaxSat=0, MaxVol=0;
    //Variables for settings
    //Variables para ajustes
    int flip=0, threshold=0, MinArea=0, MaxArea=0;
    //Variables for menu flags
    //Variables para las banderas del menu
    bool menu=false, colorcalibration=false, areacalibration=false;

    //Variables para la rueda de colores
    const int HUE_RANGE = 180;  // OpenCV just uses Hues between 0 to 179!
    //const int HUE_RANGE = 256;    // OpenCV just uses Hues between 0 to 179!
    const int WIDTH = 620;      // Window size
    const int HEIGHT = 300;     //      "
    const int HUE_HEIGHT = 25;      // thickness of Hue chart
    const int WHEEL_TOP = HUE_HEIGHT + 20;      // y position for top of color wheel (= Hue
    height + gap)
    const int WHEEL_BOTTOM = WHEEL_TOP + 255;   // y position for bottom of color wheel
    const int TILE_LEFT = 280;  // Position of small tile showing highlighted color
    const int TILEmax_TOP = 100;    //      "
    const int TILEmin_TOP = 200;    //      "
    const int TILE_W = 60;      //      "
```

```
    const int TILE_H = 60;       //       "

    #pragma endregion
    #pragma region Set Configurations
    //Load Flip configuration
    //Carga la configuración del Flip
    flip= XMLGetInt("Camera","Flip",0,0,x);
    //Load Color configurations
    //Carga las configuraciones de Color
    MinHue= XMLGetInt("Color\\Hue","Min",0,0,x);
    MaxHue= XMLGetInt("Color\\Hue","Max",0,0,x);
    MinSat= XMLGetInt("Color\\Saturation","Min",0,0,x);
    MaxSat= XMLGetInt("Color\\Saturation","Max",0,0,x);
    MinVol= XMLGetInt("Color\\Volume","Min",0,0,x);
    MaxVol= XMLGetInt("Color\\Volume","Max",0,0,x);
    //Load Area configurations
    //Carga las configuraciones de Area
    MinArea= XMLGetInt("Area","Min",0,0,x);
    MaxArea= XMLGetInt("Area","Max",0,0,x);

    #pragma endregion

    while(1)
    {
        #pragma region Vision Restrictions & Undistortion aplication
        //Get the current frame
        //Obten el cuadro actual
        IplImage *source=cvQueryFrame(capture);
        //If failed to get break the loop
        //Si falla la obtención rompe el ciclo
        if (!source)
            break;
        IplImage *fram = cvCreateImage(cvSize(source->width, source->height), IPL_DEPTH_8U, 3);
        cvUndistort2(source, fram, intrinsic, distortion);
        //Resize the capture
        //Redimensiona la captura
        cvResize(fram,frame,CV_INTER_LINEAR );
        //Flip the frame
        //Voltea el cuadro
        cvFlip(frame,frame,flip);
        //Change the color space
        //Cambia el espacio de color
        cvCvtColor(frame,hsvframe,CV_BGR2HSV);
        //Threshold the frame for the color of the blobs
        //Umbralización del marco para el color de los marcadores
        cvInRangeS(hsvframe,cvScalar(MinHue,MinSat,MinVol),cvScalar(MaxHue,MaxSat,MaxVol),
        threshy);
        //Filter the frame
        //Filtra el cuadro
        cvSmooth(threshy,threshy,CV_MEDIAN,7,7);
        //Find the blobs
        //Encuentra los blobs
        unsigned int result=cvLabel(threshy,labelImg,blobs);
```

```
        //Render the blobs
        //Renderizar los blobs
        //cvRenderBlobs(labelImg,blobs,frame,frame, CV_BLOB_RENDER_ANGLE
        |CV_BLOB_RENDER_CENTROID |CV_BLOB_RENDER_BOUNDING_BOX);
        cvRenderBlobs(labelImg,blobs,frame,frame, CV_BLOB_RENDER_CENTROID);
        //Filter the blobs
        //Filtrar los blobs
        cvFilterByArea(blobs,MinArea,MaxArea);
        //cvFilterByLabel(blobs,cvGreaterBlob(blobs));
        //cvFilterByLabel(blobs,2);

        #pragma endregion
        #pragma region Menu
        //Show the menu
        //Despliega el menu
        CvFont font;
        cvInitFont(&font, CV_FONT_HERSHEY_SIMPLEX, 0.5, 0.5, 0, 1, 3);
        cvPutText(frame,"Press 'm' to display options menu.", cvPoint(10,20), &font, cvScalar(0,
        255,0));
        char key=cvWaitKey('m'||'x'||'t'||'p'||'c'||'f'||'a'||'s');
        switch (key)
        {
            case 'm':
                menu= !menu;
            break;
            case 'x':
                exit(0);
            break;
            case 'c':
                colorcalibration= !colorcalibration;
            break;
            case 'a':
                areacalibration= !areacalibration;
            break;
            case 't':
                threshold++;
                if (threshold>3)
                {
                    threshold=0;
                }
            break;
            case 'f':
                flip++;
                if (flip>1)
                {
                    flip=-1;
                }
            break;
                    case 'p':
                cvPutText(frame,"Sistem Paused", cvPoint(200,40), &font, cvScalar(255,255,255));
                key=cvWaitKey(0);
            break;
            case 's':
```

```
            //Define in one variable the location of the root element on the .xml file
            //Define en una variable la ubicacion del elemento raíz en el archivo .xml
            XMLElement* r = x->GetRootElement();

            r->FindElementZ("Camera",true)->FindVariableZ("Flip",true)->SetValueInt(flip);
            r->FindElementZ("Color",true)->FindElementZ("Hue",true)->FindVariableZ("Max",
            true)->SetValueInt(MaxHue);
            r->FindElementZ("Color",true)->FindElementZ("Hue",true)->FindVariableZ("Min",
            true)->SetValueInt(MinHue);
            r->FindElementZ("Color",true)->FindElementZ("Saturation",true)->FindVariableZ(
            "Max",true)->SetValueInt(MaxSat);
            r->FindElementZ("Color",true)->FindElementZ("Saturation",true)->FindVariableZ(
            "Min",true)->SetValueInt(MinSat);
            r->FindElementZ("Color",true)->FindElementZ("Volume",true)->FindVariableZ("Max",
            true)->SetValueInt(MaxVol);
            r->FindElementZ("Color",true)->FindElementZ("Volume",true)->FindVariableZ("Min",
            true)->SetValueInt(MinVol);
            r->FindElementZ("Area",true)->FindVariableZ("Max",true)->SetValueInt(MaxArea);
            r->FindElementZ("Area",true)->FindVariableZ("Min",true)->SetValueInt(MinArea);

            // XML object save
            // Manipulate export format
            XMLEXPORTFORMAT xf = {0};
            xf.UseSpace = true;
            xf.nId = 2;
            x->SetExportFormatting(&xf);
            if (x->Save(file) == 1)
                fprintf(stdout,"%s saved.\r\n",file);

            cvPutText(frame,"Settings Saved", cvPoint(200,40), &font, cvScalar(255,255,255));
        break;
    }
    #pragma endregion
    #pragma region Set Menu Ajustments

    if (menu)
    {
        //Show the menu on the Live screen
        //Muetsra el menú en la pantalla Live
        cvPutText(frame,"    t - Threshold/Source View", cvPoint(10,60), &font, cvScalar(0,
        255,0));
        cvPutText(frame,"    f - Flip Image", cvPoint(10,80), &font, cvScalar(0,255,0));
        cvPutText(frame,"    c - Color Calibration", cvPoint(10,100), &font, cvScalar(0,255,0
        ));
        cvPutText(frame,"    a - Area Calibration", cvPoint(10,120), &font, cvScalar(0,255,0
        ));
        cvPutText(frame,"    p - Pause capture", cvPoint(10,140), &font, cvScalar(0,255,0));
        cvPutText(frame,"    s - Save Settings", cvPoint(10,160), &font, cvScalar(0,255,0));
        cvPutText(frame,"    x - Exit", cvPoint(10,180), &font, cvScalar(0,255,0));
    }
    if (areacalibration)
    {
        cvRenderBlobs(labelImg,blobs,frame,frame);
```

```
        cvNamedWindow("Calibrate Area",CV_WINDOW_KEEPRATIO);
        IplImage* img=cvCreateImage(cvSize(w,h),8,3);
        cvRectangle(img,cvPoint((w/2)-sqrt(double (MaxArea)),(h/2)-sqrt(double(MaxArea))),
        cvPoint((w/2)+sqrt(double(MaxArea)),(h/2)+sqrt(double(MaxArea))),CV_RGB(255,255,255
        ),-1,8,0);
        cvRectangle(img,cvPoint((w/2)-sqrt(double (MinArea)),(h/2)-sqrt(double(MinArea))),
        cvPoint((w/2)+sqrt(double (MinArea)),(h/2)+sqrt(double (MinArea))),CV_RGB(0,0,0),-1,
        8,0);
        cvCreateTrackbar("Min Area","Calibrate Area",&MinArea,(w*h),0);
        cvCreateTrackbar("Max Area","Calibrate Area",&MaxArea,(w*h),0);
        cvShowImage("Calibrate Area", img);
        cvReleaseImage( &img );
    }
    else
    {
        cvDestroyWindow("Calibrate Area");
    }
    if (colorcalibration)
    {
        #pragma region Color Calibration Window
        //Show the blobs
        //Muestra los blobs
        cvRenderBlobs(labelImg,blobs,frame,frame);
        //Crea la ventana
        //Create a GUI window
        cvNamedWindow("Color Calibration", CV_WINDOW_AUTOSIZE);
        //Create the trackbars
        //Crea las barras deslizables
        cvCreateTrackbar( "Hue min", "Color Calibration", &MinHue, HUE_RANGE-1 );
        cvCreateTrackbar( "Hue max", "Color Calibration", &MaxHue, HUE_RANGE-1 );
        cvCreateTrackbar( "Sat min", "Color Calibration", &MinSat, 255 );
        cvCreateTrackbar( "Sat max", "Color Calibration", &MaxSat, 255 );
        cvCreateTrackbar( "Bright min", "Color Calibration", &MinVol, 255 );
        cvCreateTrackbar( "Bright max", "Color Calibration", &MaxVol, 255 );
        //Create a blank image
        //Crea una imagen en blanco
        IplImage *imageHSV = cvCreateImage(cvSize(WIDTH, HEIGHT), 8, 3);
        int h2 = imageHSV->height;          // Pixel height
        int w2 = imageHSV->width;           // Pixel width
        int rowSize = imageHSV->widthStep;  // Size of row in bytes, including extra padding
        char *imOfs = imageHSV->imageData;  // Pointer to the start of the image HSV pixels.
        //Clear the image to grey (Saturation=0)
        //Limpia la imagen a gris (Saturación=0)
        cvSet(imageHSV, cvScalar(0,0,210, 0));
        //Draw the hue chart on the top, at double width.
        //Dibuja la gráfica de hue en la parte superior, con el doble de ancho
        for (int y=0; y<HUE_HEIGHT; y++)
        {
            for (int x=0; x<HUE_RANGE; x++)
            {
                uchar h2 = x;       // Hue (0 - 179)
                uchar s = 255;      // max Saturation => most colorful
                uchar v = 255;      // max Value => brightest color
```

```
            //Highlight the current value
            //Destaca el valor actual
            if ((h2 == MinHue-2 || h2 == MinHue+2) && (y < HUE_HEIGHT/2))
            {
                s = 0;  // Make it white instead of the color
            }

            if ((h2 == MaxHue-2 || h2 == MaxHue+2) && (y < HUE_HEIGHT/2))
            {
                v = 0;  // Make it black instead of the color
            }
            // Set the HSV pixel components
            // Establece los componentes de pixeles HSV
            *(uchar*)(imOfs + y*rowSize + ((x+65)*2+0)*3 + 0) = h2;
            *(uchar*)(imOfs + y*rowSize + ((x+65)*2+0)*3 + 1) = s;
            *(uchar*)(imOfs + y*rowSize + ((x+65)*2+0)*3 + 2) = v;
            *(uchar*)(imOfs + y*rowSize + ((x+65)*2+1)*3 + 0) = h2;
            *(uchar*)(imOfs + y*rowSize + ((x+65)*2+1)*3 + 1) = s;
            *(uchar*)(imOfs + y*rowSize + ((x+65)*2+1)*3 + 2) = v;
        }
    }
    //Draw the color wheel: Saturation on the x-axis and Value (brightness) on the y-axis
    //Dibuja la rueda de color: la Saturación en el eje x, y Value (brillo) en el eje y
    for (int y=0; y<255; y++)
    {
        for (int x=0; x<255; x++)
        {
            uchar h2 = MinHue;  // Hue (0 - 179)
            uchar s = x;        // Saturation (0 - 255)
            uchar v = (255-y);  // Value (Brightness) (0 - 255)
            //Highlight the current value
            //Destaca el valor actual
            if ((s == MinSat-2 || s == MinSat-3 || s == MinSat+2 || s == MinSat+3) && (v
             == MinVol-2 || v == MinVol-3 || v == MinVol+2 || v == MinVol+3))
            {
                s = 0;  // make it white instead of the color
                v = 0;  // bright white
            }
            //Set the HSV pixel components
            //Establece los componentes de pixeles HSV
            *(uchar*)(imOfs + (y+WHEEL_TOP)*rowSize + x*3 + 0) = h2;
            *(uchar*)(imOfs + (y+WHEEL_TOP)*rowSize + x*3 + 1) = s;
            *(uchar*)(imOfs + (y+WHEEL_TOP)*rowSize + x*3 + 2) = v;
        }
    }
    //Draw a small tile of the highlighted color
    //Dibuja un pequeño recuadro del color destacado
    for (int y=0; y<TILE_H; y++)
    {
        for (int x=0; x<TILE_W; x++)
        {
            //Set the HSV pixel components
            //Establece los componentes de pixeles HSV
```

```
            *(uchar*)(imOfs + (y+TILEmin_TOP)*rowSize + (x+TILE_LEFT)*3 + 0) = MinHue;
            *(uchar*)(imOfs + (y+TILEmin_TOP)*rowSize + (x+TILE_LEFT)*3 + 1) = MinSat;
            *(uchar*)(imOfs + (y+TILEmin_TOP)*rowSize + (x+TILE_LEFT)*3 + 2) = MinVol;
        }
    }
    //Draw the second color wheel: Saturation on the x-axis and Value (brightness) on
    the y-axis
    //Dibuja la segunda rueda de color: la Saturación en el eje x, y Value (brillo) en
    el eje y
    for (int y=0; y<255; y++)
    {
        for (int x=365; x<620; x++)
        {
            uchar h2 = MaxHue;  // Hue (0 - 179)
            uchar s = x-365;        // Saturation (0 - 255)
            uchar v = (255-y);  // Value (Brightness) (0 - 255)
            //Highlight the current value
            //Destaca el valor actual
            if ((s == MaxSat-2 || s == MaxSat-3 || s == MaxSat+2 || s == MaxSat+3) && (v
             == MaxVol-2 || v == MaxVol-3 || v == MaxVol+2 || v == MaxVol+3))
            {
                s = 0;  // make it white instead of the color
                v = 0;  // bright white
            }
            //Set the HSV pixel components
            //Establece los componentes de pixeles HSV
            *(uchar*)(imOfs + (y+WHEEL_TOP)*rowSize + x*3 + 0) = h2;
            *(uchar*)(imOfs + (y+WHEEL_TOP)*rowSize + x*3 + 1) = s;
            *(uchar*)(imOfs + (y+WHEEL_TOP)*rowSize + x*3 + 2) = v;
        }
    }
    //Draw the second small tile of the highlighted color
    //Dibuja el segundo pequeño recuadro del color destacado
    for (int y=0; y<TILE_H; y++)
    {
        for (int x=0; x<TILE_W; x++)
        {
            //Set the HSV pixel components
            //Establece los componentes de pixeles HSV
            *(uchar*)(imOfs + (y+TILEmax_TOP)*rowSize + (x+TILE_LEFT)*3 + 0) = MaxHue;
            *(uchar*)(imOfs + (y+TILEmax_TOP)*rowSize + (x+TILE_LEFT)*3 + 1) = MaxSat;
            *(uchar*)(imOfs + (y+TILEmax_TOP)*rowSize + (x+TILE_LEFT)*3 + 2) = MaxVol;
        }
    }
    //Convert the HSV image to RGB (BGR) for displaying
    //Convierte la imagen HSV a RGB (BRG) para mostrar
    IplImage *imageRGB = cvCreateImage(cvSize(imageHSV->width, imageHSV->height), 8, 3);
    cvCvtColor(imageHSV, imageRGB, CV_HSV2BGR); // (note that OpenCV stores RGB images
    in B,G,R order

    cvPutText(imageRGB,"Max", cvPoint(295,95), &font, cvScalar(0,0,0));
    cvPutText(imageRGB,"Min", cvPoint(295,195), &font, cvScalar(0,0,0));
    //Show Image
```

```
            //Muestra la imagen
            cvShowImage("Color Calibration", imageRGB);
            cvReleaseImage( &imageRGB );
            cvReleaseImage( &imageHSV );

            #pragma endregion
        }
        else
        {
            cvDestroyWindow("Color Calibration");
        }
        #pragma endregion
        #pragma region Blob interpretation
        //Blob interpretation loop
        //Ciclo de interpretación de los blobs
        for (CvBlobs::const_iterator it=blobs.begin(); it!=blobs.end(); ++it)
        {
            double moment10 = it->second->m10;
            double moment01 = it->second->m01;
            double area = it->second->area;

            if (savedblobsflag==0)
            {
                //Get reference values (Number of Blobs, Distance between blobs axis, Camera
                height and Robot height)
                //Obtiene valores de referencia (Numero de Blobs, Distancia entre Blobs del
                eje, Atura de la Cámara y Altura del Robot)
                n= XMLGetInt("Blobs","Number",0,0,x);
                axisref= XMLGetInt("Blobs","Distance",0,0,x);
                Z= XMLGetInt("Blobs","Camera_height",0,0,x);
                Z2= XMLGetInt("Blobs","Robot_height",0,0,x);
                savedblobsflag=1;
            }
            if (i<=n && area>0)
            {
                blobarea[i]=area*1.3;
                i++;
            }
            else
            {
                savedblobsflag=2;
            }
            if (savedblobsflag==2)
            {
                //Variable for holding position
                //Variable para mantener posición
                double x1;
                double y1;
                //Calculate the curretn position
                //Calcula la posición actual
                x1 = moment10/area;
                y1 = moment01/area;
                //Show to the screen coordinates
```

```
                //Muestra ubicación de las coordenadas de pantalla
                int x=(int)(x1*screenx/w);
                int y=(int)(y1*screeny/h);

                while (k<=n)
                {
                    if(area <= blobarea[k] && area >= blobarea[k-1])
                    {
                        a1[k]= x;
                        a2[k]= y;

                    }
                    k++;
                }
            k=0;
            }
        }
        #pragma endregion
        #pragma region Blob Sorting

        if (savedblobsflag==1)
        {
            //Sort the saved areas
            //Ordena las areas guardadas
            sort(blobarea,blobarea+n+1);//The sort() function requires the end to be indicated
            with the address of the element beyond the last element that is to be sorted.
            //Show the saved areas
            //Muestra las areas guardadas
            for (int j=1 ; j<=n ; j++)
            {
                cout<<"\tSAVED AREA "<<j<<": "<<blobarea[j]<<endl;
            }
        }
        #pragma endregion
        #pragma region Workspace Settings
        //When Blobs are sorted
        //Cuando los Blobs ya están ordenados
        if (savedblobsflag==2)
        {
            #pragma region Blob Labeling
            stringstream labelx;
            labelx << "-X";
            cvPutText(frame,labelx.str().c_str(), cvPoint(a1[1]*w/screenx,a2[1]*h/screeny), &
            font, cvScalar(0,0,255));
            stringstream labelxx;
            labelxx << "X";
            cvPutText(frame,labelxx.str().c_str(), cvPoint(a1[2]*w/screenx,a2[2]*h/screeny), &
            font, cvScalar(0,0,255));
            //Label cycle
            //Ciclo para etiquetar
            for (int j=3 ; j<=n ; j++)
            {
                stringstream label;
```

```
            label << j;
            cvPutText(frame,label.str().c_str(), cvPoint(a1[j]*w/screenx,a2[j]*h/screeny), &
            font, cvScalar(0,255,0));
        }
        #pragma endregion
        #pragma region Definition of the Cartesian System
        //Loop for the coordinated axes
        //Ciclo para los ejes coordenados
        for (int j=2 ; j<=2 ; j++)
        {
                //Assign the position (x,y) of the first two blobs as references for the X
                axis
                //Asigna la posición (x,y) de los dos primeros blobs como referencias del
                eje X
                float Xref1=a1[j-1]*w/screenx;
                float Yref1=a2[j-1]*h/screeny;
                float Xref2=a1[j]*w/screenx;
                float Yref2=a2[j]*h/screeny;
                //Draw the line between the points that define the X axis
                //Dibuja la linea entre los puntos que definen al eje X
                cvLine(frame,cvPoint(Xref1,Yref1),cvPoint(Xref2,Yref2),CV_RGB(255,0,0),1,8);
                //Calculate the angle of X axis
                //Calcula el ángulo del eje X
                axisslope=(Yref2-Yref1)/(Xref2-Xref1);
                axisangle= atan(axisslope);
                //Calculate the distance between the point that define the X axis
                //Calcula la distancia entre los puntos que definen al eje X
                distance= (sqrt((pow((Xref2-Xref1),2))+(pow((Yref2-Yref1),2))))/2;
                //Direction correction of the distance if the points that define the X axis
                are inverted
                //Corrección de la dirección de la distancia si se invierten los puntos que
                definen al eje X
                if(Xref2 < Xref1)
                {
                    distance=distance*-1;
                }
                //Rotation and translation Matrix
                //Matriz de Rotación y traslación
                OriginXX=Xref1+(distance*(cos(axisangle)));
                OriginXY=Yref1+(distance*(sin(axisangle)));
                OriginYX=OriginXX-(distance*(sin(axisangle)));
                OriginYY=OriginXY+(distance*(cos(axisangle)));
                OriginYXpos=OriginXX+(distance*(sin(axisangle)));
                OriginYYpos=OriginXY-(distance*(cos(axisangle)));
                //Draw the positive line perpendicular to the X axis (Y)
                //Dibuja la linea perpendicular positiva al eje X (Y)
                cvLine(frame,cvPoint(OriginXX,OriginXY),cvPoint(OriginYX,OriginYY),CV_RGB(
                255,0,0),1,8);
                //Draw the negative line perpendicular to the X axis (-Y)
                //Linea perpendicular negativa al eje X (-Y)
                cvLine(frame,cvPoint(OriginXX,OriginXY),cvPoint(OriginYXpos,OriginYYpos),
                CV_RGB(255,0,0),1,8);
                //Label for the origin
```

```
            //Etiqueta del Origen
            stringstream label;
            label << "O";
            cvPutText(frame,label.str().c_str(), cvPoint(OriginXX,OriginXY), &font,
            cvScalar(0,0,255));

    }
    #pragma endregion
    #pragma region Object Localization (with corrections)
    //Object localization loop
    //Ciclo para localizar los objetos
    for (int j=4 ; j<=n ; j++)
    {
        //Assign the position (x,y) of the blobs as references for the objects
        //Asigna la posición (x,y) de los blobs como referencias de los objetos
        float Xref1=a1[j-1]*w/screenx;
        float Yref1=a2[j-1]*h/screeny;
        float Xref2=a1[j]*w/screenx;
        float Yref2=a2[j]*h/screeny;
        //Draw the line between the points that define the objects
        //Dibuja la linea entre los puntos que definen a los objetos
        cvLine(frame,cvPoint(Xref1,Yref1),cvPoint(Xref2,Yref2),CV_RGB(0,0,255),1,8);
        //Calculate the angle of the object
        //Calcula el ángulo del objeto
        float angle2= atan((Yref2-Yref1)/(Xref2-Xref1));
        //Calculate the distance between the point that define the X axis
        //Calcula la distancia entre los puntos que definen al eje X
        blobsdistance = (sqrt((pow((Xref2-Xref1),2))+(pow((Yref2-Yref1),2))))/2;
        //Direction correction of the distance if the points that define the object are
        inverted
        //Corrección de la dirección de la distancia si se invierten los puntos que
        definen al objeto
        if(Xref2 < Xref1)
            {
            blobsdistance=blobsdistance*-1;
            }
        //Definition of the object center
        //Definición del centro del objecto
        CenterX=Xref1+(blobsdistance*(cos(angle2)));
        CenterY=Yref1+(blobsdistance*(sin(angle2)));
        //Define the object centers
        //Define los centros del objeto
        float Xref=CenterX;
        float Yref=CenterY;
        //Draw the line from the origin to the object
        //Dibuja la linea del Origen al objeto
        cvLine(frame,cvPoint(OriginXX,OriginXY),cvPoint(Xref,Yref),CV_RGB(0,255,255),1,8
        );
        //Position of the camera center
        //Posición del centro de la camara
        CamzeroX= 1360*w/(2*screenx);
        CamzeroY= 760*h/(2*screeny);
        //Draw the line from the camera center to the object center
```

```
//Dibuja la linea del centro de la camara al centro del objeto
cvLine(frame,cvPoint(CamzeroX,CamzeroY),cvPoint(Xref,Yref),CV_RGB(255,0,0),1,8);
//Position correction
//Corrección de la posición
//Calculate the distance between the camera and the object
//Calcula la distancia ente la cámara y el objeto
Z1= Z-Z2;
//Calculate the distance between the camera center and the object center
//Calcula la distancia entre el centro real de la camara y el centro del objeto
Xv= sqrt((pow(Xref-CamzeroX,2))+(pow((Yref-CamzeroY),2)));
//Calculate the angle between the camera center and the object center
//Calcula el angulo ente el centro de la camara y el centro del objeto
blobangle=atan((CamzeroY-Yref)/(CamzeroX-Xref));
//Calculate the hypotenuse formed between the virtual center of the object and
the camera
//Calcula la hipotenusa formada entre el centro virtual del objeto y la cámara
C= sqrt((pow(Z,2))+(pow(Xv,2)));
//Calculate the angle between the hypotenuse and the virtual distance from the
center of the camera to the object center
//Calcula el ángulo entre la hipotenusa y la distancia virtual desde el centro
de la cámara hasta el centro del objeto
theta1= asin(Xv/C);
Xr= (tan(theta1))*(Z1);
//Angle correction
//Corrección del angulo
if (Xref < CamzeroX)
{
    blobangle= blobangle + Pi;
}
//Definition if the corrected object center
//Definición del centro del objeto corregido
CorrectionX= CamzeroX + (Xr*(cos(blobangle)));
CorrectionY= CamzeroY + (Xr*(sin(blobangle)));
//Calculate the angle between the origin and the fixed center of the object
//Calcula el ángulo entre el origen y el centro corregido del objeto
float correctionangleslope= (OriginXY-CorrectionY)/(OriginXX-CorrectionX);
correctionangle=atan(abs((correctionangleslope-axisslope)/(1+
correctionangleslope*axisslope)));
//Scale factor
//Factor de escala
scale= axisref/(distance*2);
//Draw the line from the real camera center to the corrected object center
//Dibuja la linea del centro real de la camara al centro corregido del objeto
cvLine(frame,cvPoint(CamzeroX,CamzeroY),cvPoint(CorrectionX,CorrectionY),CV_RGB(
0,255,0),1,8);
//Draw the line from the origin to the corrected object center
//Dibuja la linea del origen al centro corregido del objeto
cvLine(frame,cvPoint(OriginXX,OriginXY),cvPoint(CorrectionX,CorrectionY),CV_RGB(
255,0,255),1,8);
//Definition of the fixed and scaled center of the object
//Definición del centro corregido y escalado del objeto
float correctiondistance= sqrt((pow((CorrectionX-OriginXX),2))+(pow((CorrectionY
-OriginXY),2)));
```

```
        float PositionX=abs(correctiondistance*(cos(correctionangle)))*scale;
        float PositionY=abs(correctiondistance*(sin(correctionangle)))*scale;
        //Definicion de la pendiente del objeto con respecto de la cámara
        blobslope=(Yref2-Yref1)/(Xref2-Xref1);
        //Definición del ángulo del objeto
        angleblob=atan(abs((blobslope-axisslope)/(1+blobslope*axisslope)));
        angleblob=angleblob*(180/Pi);
        //Normalización del angulo
        if(Yref1>Yref2 && Xref1>Xref2)
            angleblob=360-angleblob;
        else if (Yref1>Yref2 && Xref1<Xref2)
            angleblob=180+angleblob;
        else if (Yref1<Yref2 && Xref1<Xref2)
            angleblob=180-angleblob;
        else if (Yref1<Yref2 && Xref1>Xref2);
            angleblob=360-angleblob;
        //Conditions for normalizing the measurement of the angle
        //Condiciones para normalizar la medición del ángulo
        //Condition for desigate the position X
        //Condición para signar la posicion en X

        if (CorrectionX>OriginXX && CorrectionY<OriginXY)//Primer cuadrante
        {
            correctionangle=correctionangle;
            PositionX=PositionX;
            PositionY=PositionY;
            signX=1;//positivo
            signY=1;//positivo
        }
        else if (CorrectionX<OriginXX && CorrectionY<OriginXY)//Segundo cuadrante
        {
            correctionangle=Pi-correctionangle;
            PositionX=PositionX*-1;
            PositionY=PositionY*1;
            signX=0;//negativo
            signY=1;//positivo
        }
        else if (CorrectionX<OriginXX && CorrectionY>OriginXY)//3er cuadrante
        {
            correctionangle=Pi+correctionangle;
            PositionX=PositionX*-1;
            PositionY=PositionY*-1;
            signX=0;//negativo
            signY=0;//negativo
        }
        else if (CorrectionX>OriginXX && CorrectionY>OriginXY)//4to cuadrante
        {
            correctionangle=(2*Pi)-correctionangle;
            PositionX=PositionX*1;
            PositionY=PositionY*-1;
            signX=1;//positivo
            signY=0;//negativo
        }
```

```
            #pragma endregion
            #pragma region Data display and UDP send
            //Show the values
            //Muesta los valores
            cout<<j<<" D: "<<correctiondistance<<"\t Oaxis: "<<axisangle*(180/Pi)<<"\t Or: "
            <<correctionangle*(180/Pi)<<"\n";
            cout<<j<<" X: "<<PositionX<<"\t Y: "<<PositionY<<"\t Theta: "<<360-angleblob<<
            "\n";
            char SendBuf[]="";
            //Assign values ??to a string vector (with printf formats)
            //Asigna valores a un vector string (con formatos de printf)

            //sprintf(SendBuf,";%d;%03.0f;%03.0f;%03.0f;%d%d",j,abs(PositionX)*10,abs(Positio
            nY)*10,correctionangle*(180/Pi),signX,signY);
            sprintf(SendBuf,";%d;%03.0f;%03.0f;%03.0f;%d%d",j,abs(PositionX)*10,abs(
            PositionY)*10,360-angleblob,signX,signY);
            //Structure for sending data via UDP
            //Estructura para enviar datos via UDP
            WSAStartup(MAKEWORD(2,2), &wsaData);
            SendSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
            RecvAddr.sin_family = AF_INET;
            RecvAddr.sin_port = htons(Puerto);
            RecvAddr.sin_addr.s_addr = inet_addr(ip);
            sendto(SendSocket,SendBuf,strlen(SendBuf)+1,0,(SOCKADDR *) &RecvAddr,sizeof(
            RecvAddr));
            WSACleanup();
            #pragma endregion
            //Increment the counter one more to analize pairs
            //Incrementa el contador uno más para analizar pares
            j++;
        }
    }
    #pragma endregion
    #pragma region Window Set
    //Show the images
    //Muestra las imagenes de acuerdo a la visualización elegida
    switch (threshold)
    {
        case 0:
            cvShowImage("Live",frame);
        break;
        case 1:
            cvShowImage("Live",fram);
        break;
        case 2:
            cvShowImage("Live",threshy);
        break;
        case 3:
            cvShowImage("Live",hsvframe);
        break;
    }
    cvReleaseImage(&fram);
```

```
        //Escape Sequence
        //Secuencia de salida y retardo
        char c=cvWaitKey(33);
        if(c==27)
        break;
        #pragma endregion
    }
    //Cleanup
    //Limpieza
    cvReleaseCapture(&capture);
    cvDestroyAllWindows();
}
```

Apéndice III: Código fuente del programa de calibración de la cámara (en C++)

```
#pragma region Inclusions & Definitions
#include <time.h>
//Input-Output
//Entrada-Salida
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include "resource.h"//Include source files
//OpenCV Headers
//Encabezados de OpenCV
#include <cvaux.h>
#include <cxcore.h>
#include <cv.h>
#include <highgui.h>
//Inclusions for XML writting
//Inclusiones para escribir en XML
#include "xml.h"
#include "json.hpp"
#pragma comment(lib,"wininet.lib")
#pragma comment(lib,"crypt32.lib")
#pragma endregion
#pragma region NameSpaces
using namespace std;
#pragma endregion
//Capture mouse coordinates
//Captura las coordenadas del mouse
void on_mouse(int event, int x, int y, int flags, void *param);
CvPoint location = {0, 0};

int main(int argc , char *argv[])
{
    #pragma region Open Settings.Xml
    //Declare the file with its name
    //Declara el archivo con su nombre
    char* file = ".\\Settings.xml";
    //Load file
    //Carga el archivo
    XML* x = new XML(file);
    //Parse status check
    //Checa el estado de lectura
    int iPS = x->ParseStatus(); // 0 OK , 1 Header warning (not fatal) , 2 Error in parse (fatal)
    bool iTT = x->IntegrityTest(); // True: OK
    //Show erros message if unable to open
    //Muestra mensaje de error si no se pudo abrir
    if (iPS == 2 || iTT == false)
    {
        fprintf(stderr,"Error: XML file %s is corrupt (or not a XML file).\r\n\r\n",file);
        delete x;
        return 0;
    }
    #pragma endregion
    #pragma region Variables Declaration
    //Variables for counters
```

```
    //Variables para contadores
    int i, j;
    //Variables of the distortion matrix coefficients
    //Variables de los coeficientes de la matriz de distorción
    float fx,fy,cx,cy,k1,k2,p1,p2;
    //Variables for chessboard interpretation
    //Variables para interpetar el chessboard
    int corner_counts, pattern_was_found, cal_images_num;
    int corner_row, corner_col, valid = 0, total_num = 0, current_num = 0;
    float distance1 = 0, distance2 = 0;

    int *points_counts = NULL;
    //Variables for the names of the images
    //Variables para los nombres de las imágenes
    char cal_images_name[30], read_image[40];
    char c;

    IplImage *inputImage, *corner_image, *calibrated_image, *corner_draw, *test_image, *
    corner_image_grey;
    CvPoint2D32f *corners;
    //Variables for the matrixes intrinsec and distortion
    //Variables para las matrices intrínseca de distorción
    CvMat *intrinsic_matrix, *distortion_coeffs;

    CvMat **object_points = NULL, **image_points = NULL;
    CvMat *object_points_update, *image_points_update, *points_counts_update;

    //Variable for flag the first image
    //Variable de bandera de la primera imagen
    bool init = true;

    #pragma endregion
    #pragma region OpenCV Camera Configuration
    //Structure to get information from CAM
    //Estructura para obtener información de la CAM
    CvCapture* capture = cvCreateCameraCapture(0);
    //Check if the capture was succesfull
    //Checa si la captura fue exitosa
    if( !capture )
    {
        fprintf(stderr,"Could not initialize capturing...\n");
        return -1;
    }
    //Windows that shows the capture
    //Ventana que muestra la captura
    cvNamedWindow( "Image", 1 );
    cvMoveWindow( "Image", 40, 40 );
    cvNamedWindow( "Output", 1 );
    cvMoveWindow( "Output", 440, 40 );
    #pragma endregion
    #pragma region Chessboard Settings
    printf("CALIBRATING AND CORRECTING CAMERA DISTORTIONS\n");
    printf("(Keep output window active for all key presses)\n");
```

```
    printf("\nPresent camera with chessboard images in order to calculate the intrinsic
    parameters of the camera\n");
    //Definition of the chessboar read from Settings.xml
    //Definición del chessboard leído desde Settings.xml
    cal_images_num = XMLGetInt("Chessboard","Pictures_number",0,0,x);
    corner_row = XMLGetInt("Chessboard","Rows",0,0,x);
    corner_col = XMLGetInt("Chessboard","Colums",0,0,x);
    corners = (CvPoint2D32f *)calloc(corner_row * corner_col, sizeof(CvPoint2D32f));
    //Create intrinsec and distortion matrixes
    //Crea matrices intrínseca y de distorción
    intrinsic_matrix = cvCreateMat(3, 3, CV_32FC1);
    distortion_coeffs = cvCreateMat(1, 4, CV_32FC1);
    #pragma endregion
    //Load the chessboard images.
    //Carga las imágenes del tablero de ajedrez

    // As the image points on the chessboard images need to match the object
    // points, so you need to point out manually which corner is the start of
    // the object points. You could see a color line go through all the detected
    // corners. Sometimes the start of the color line just matches your start
    // object points, but sometime the end of the line matches your start object
    // points. Choose one end in the image.

    // Los puntos de la imagen de las imágenes de tablero de ajedrez necesitan coincidir con
    los puntos
    // del objeto, por lo que se tienen que señalar manualmente qué esquina es el comienzo de
    // los puntos del objeto. Se puede ver una línea de color de pasar por todas las esquinas
    // A veces, el inicio de la línea de color sólo coincide con el inicio de los
    // puntos del objeto, pero en algún momento al final de la línea coincide con los puntos de
inicio
    // del objeto. Elegir uno de los extremos de la imagen.

    #pragma region Image Reading
    //Captures name
    //Nombre de las capturas
    sprintf(cal_images_name, "Camera_Capture_");
    //Image captures loop
    //Ciclo de captura de imagenes
    for (i = 1; i <= cal_images_num; i++)
    {
        sprintf(read_image, "%s%02d", cal_images_name, i);
        printf("\nInput: %s\n", read_image);
        int grab = cvGrabFrame(capture);
        inputImage = cvRetrieveFrame(capture);
        //Run once when first image is loaded
        //Corre una vez cuando la primera imagen es cargada
        if (init)
        {
            corner_image = cvCreateImage(cvSize(inputImage->width, inputImage->height),
            IPL_DEPTH_8U, 3);
            corner_image_grey = cvCreateImage(cvSize(inputImage->width, inputImage->height),
            IPL_DEPTH_8U, 1);
            corner_draw = cvCreateImage(cvSize(inputImage->width, inputImage->height),
```

```
            IPL_DEPTH_8U, 3);
            calibrated_image = cvCreateImage(cvSize(inputImage->width, inputImage->height),
            IPL_DEPTH_8U, 3);
            test_image = cvCreateImage(cvSize(inputImage->width, inputImage->height),
            IPL_DEPTH_8U, 3);
            init = false;
        }
        //Flip the image if camera is inverting
        //Gira la imagen si la camara esta invirtiendo
        cvConvertImage(inputImage,corner_image,CV_CVTIMG_FLIP);

        cvShowImage("Image", inputImage);
        cvZero(corner_draw);

        //Find the corners and store their coordinates in 'corners' array
        //Encuentra las esquinas y almacena sus coordenadas en el arreglo 'corners'
        pattern_was_found = cvFindChessboardCorners(corner_image, cvSize(corner_row, corner_col
        ), corners, &corner_counts,
            CV_CALIB_CB_ADAPTIVE_THRESH+CV_CALIB_CB_NORMALIZE_IMAGE+CV_CALIB_CB_FILTER_QUADS);

        if (pattern_was_found)
        {
            cvCopyImage(corner_image, corner_draw);

            //Draw all the detected corners
            //Dibuja todas las esquinas detectadas
            cvCvtColor(corner_image, corner_image_grey, CV_BGR2GRAY);

            //Refine the corners
            //Refina las esquinas
            cvFindCornerSubPix( corner_image_grey, corners,
                corner_row*corner_col, cvSize(11,11), cvSize(-1,-1),
                cvTermCriteria( CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ) );

            cvDrawChessboardCorners(corner_draw, cvSize(corner_row, corner_col), corners,
            corner_counts, pattern_was_found);
            cvShowImage("Output", corner_draw);

            //printf("Choose your start point from two ends of the color line as origin of
            object points\n");
            printf("Click near the start of the coloured, zig-zag line\n");
            printf("Then press any key on the image to continue\n");

            // Get the coordinates of chosen point
            // Obtiene las coordenadas del punto elegido
            cvSetMouseCallback( "Output", on_mouse, 0 );

            c=cvWaitKey(-1);
            printf("The start point is x:%d y:%d\n", location.x, location.y);

            //As corners can't be detected in some chessbord images, so here is used 'valid' to
            record how many useful chessboard images
            //Como las esquinas no puede ser detectadas en algunas imágenes del chessbord, asi
```

```
        se usa 'valid' para registrar la cantidad de imágenes útiles
        valid++;
        object_points = (CvMat **)realloc(object_points, sizeof(CvMat *)*valid);
        image_points = (CvMat **)realloc(image_points, sizeof(CvMat *)*valid);
        points_counts = (int *)realloc(points_counts, sizeof(int)*valid);

        object_points[valid-1] = cvCreateMat(corner_counts, 3, CV_32FC1);
        image_points[valid-1] = cvCreateMat(corner_counts, 2, CV_32FC1);

        //See the chosen matches the start of the corners
        //Ve los puntos elegidos al comienzo de las esquinas
        distance1 = (location.x-corners[0].x)*(location.x-corners[0].x)+(location.y-corners[
        0].y)*(location.y-corners[0].y);
        distance2 = (location.x-corners[corner_counts-1].x)*(location.x-corners[
        corner_counts-1].x)+(location.y-corners[corner_counts-1].y)*(location.y-corners[
        corner_counts-1].y);

        if (distance1 < distance2)
        {
            for (j = 0; j < corner_counts; j++)
            {
                CV_MAT_ELEM(*object_points[valid-1], float, j, 0) = (float)(j%corner_row);
                CV_MAT_ELEM(*object_points[valid-1], float, j, 1) = (float)(j/corner_row);
                CV_MAT_ELEM(*object_points[valid-1], float, j, 2) = 0;
                CV_MAT_ELEM(*image_points[valid-1], float, j, 0) = corners[j].y;
                CV_MAT_ELEM(*image_points[valid-1], float, j, 1) = corners[j].x;
            }
        }
        else
        {
            for (j = 0; j < corner_counts; j++)
            {
                CV_MAT_ELEM(*object_points[valid-1], float, j, 0) = (float)(j%corner_row);
                CV_MAT_ELEM(*object_points[valid-1], float, j, 1) = (float)(j/corner_row);
                CV_MAT_ELEM(*object_points[valid-1], float, j, 2) = 0;
                CV_MAT_ELEM(*image_points[valid-1], float, j, 0) = corners[corner_counts-j-1
                ].y;
                CV_MAT_ELEM(*image_points[valid-1], float, j, 1) = corners[corner_counts-j-1
                ].x;
            }
        }
        points_counts[valid-1] = corner_counts;
    }
    else
    {
        //Not all the chessboard images are useful :(
        //Draw the points that were found in any case
        cvCopyImage(corner_image, corner_draw);
        cvDrawChessboardCorners(corner_draw, cvSize(corner_row, corner_col), corners,
        corner_counts, pattern_was_found);

        cvShowImage("Output", corner_draw);
        printf("Image %s does not contain chessboard pattern, try again\n", read_image);
```

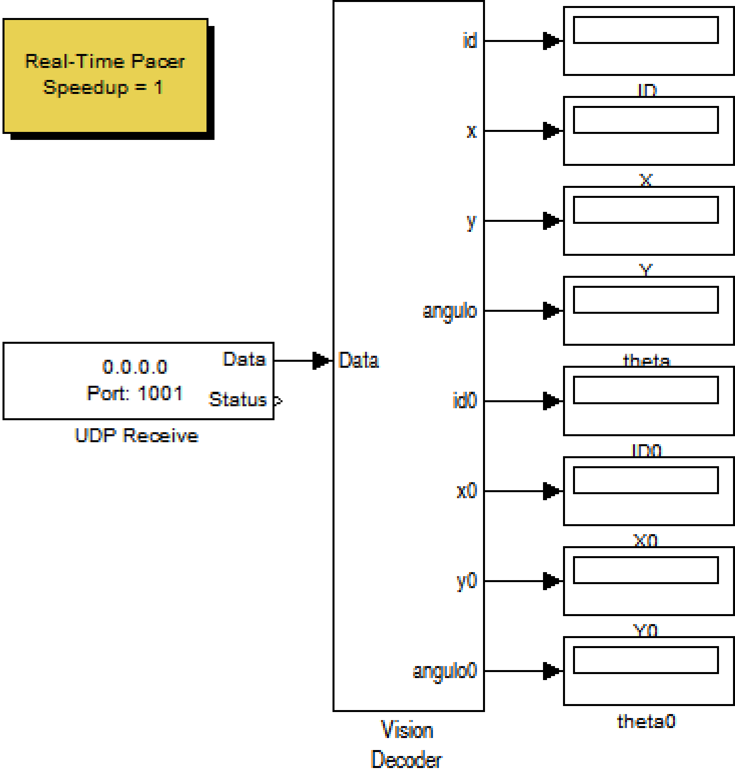
```
            printf("Press any key to proceed\n", read_image);
            c=cvWaitKey(-1);

            i--;
        }
    }
    #pragma endregion
    #pragma region Matrix Construction
    points_counts_update = cvCreateMat(valid, 1, CV_32SC1);
    total_num = 0;
    for (i = 0; i < valid; i++)
    {
        total_num += points_counts[i];
        CV_MAT_ELEM(*points_counts_update, int, i, 0) = points_counts[i];
    }

    object_points_update = cvCreateMat(total_num, 3, CV_32FC1);
    image_points_update = cvCreateMat(total_num, 2, CV_32FC1);

    //Put all matrices in joint matrices
    //Pone todas las matrices en matrices de conjuntos
    current_num = 0;
    for (i = 0; i < valid; i++)
    {
        for (j = 0; j < points_counts[i]; j++)
        {
            CV_MAT_ELEM(*object_points_update,  float, current_num+j, 0) = CV_MAT_ELEM(*
            object_points[i], float, j, 0);
            CV_MAT_ELEM(*object_points_update,  float, current_num+j, 1) = CV_MAT_ELEM(*
            object_points[i], float, j, 1);
            CV_MAT_ELEM(*object_points_update,  float, current_num+j, 2) = CV_MAT_ELEM(*
            object_points[i], float, j, 2);
            CV_MAT_ELEM(*image_points_update,  float, current_num+j, 0) = CV_MAT_ELEM(*
            image_points[i], float, j, 0);
            CV_MAT_ELEM(*image_points_update,  float, current_num+j, 1) = CV_MAT_ELEM(*
            image_points[i], float, j, 1);
        }
        current_num += points_counts[i];
        cvReleaseMat(&object_points[i]);
        cvReleaseMat(&image_points[i]);
    }
    free(object_points);
    free(image_points);
    free(points_counts);
    #pragma endregion
    #pragma region Camera Undistortion
    //Calculate the intrinsic parameters and distortion coefficients
    //Calcula los parámetros intrinsecos y los coeficientes de distorción
    cvCalibrateCamera2(object_points_update, image_points_update, points_counts_update, cvSize(
    corner_image->width, corner_image->height), intrinsic_matrix, distortion_coeffs, NULL, NULL,
     0);
    //Save the intrinsic matrix and distortions coefficients
    //Guarda la matrz intrínseca y los coeficientes de distorción
```

```
    cvSave( "Intrinsics.xml", intrinsic_matrix );
    cvSave( "Distortion.xml", distortion_coeffs );
    //Define variables for intrinsec parameters
    //Define variables de parámetros intrínsecos
    fx=CV_MAT_ELEM(*intrinsic_matrix, float, 0 , 0);
    fy=CV_MAT_ELEM(*intrinsic_matrix, float, 1 , 1);
    cx=CV_MAT_ELEM(*intrinsic_matrix, float, 0 , 2);
    cy=CV_MAT_ELEM(*intrinsic_matrix, float, 1 , 2);
    k1=CV_MAT_ELEM(*distortion_coeffs, float,0, 0);
    k2=CV_MAT_ELEM(*distortion_coeffs, float,0, 1);
    p1=CV_MAT_ELEM(*distortion_coeffs, float,0, 2);
    p2=CV_MAT_ELEM(*distortion_coeffs, float,0, 3);
    //Display on screen instrinsec parameters
    //Despliga parámetros instrinsecos en pantalla
    printf("\n\nFocal length:  [fx fy] = [ %.4f %.4f ]\n",fx, fy);
    printf("Principal point:   [cx cy] = [ %.4f %.4f ]\n",cx, cy);
    printf("Distortion coeffs: [k1 k2 p1 p2] = [ %.4f %.4f %.4f %.4f ]\n",k1, k2, p1, p2);
    //Save on xml file
    //Guarda en el archivo xml
    XMLElement* r = x->GetRootElement();
    r->FindElementZ("Camera_Calibration",true)->FindElementZ("Focal_length",true)->FindVariableZ
    ("fx",true)->SetValueFloat(fx);
    r->FindElementZ("Camera_Calibration",true)->FindElementZ("Focal_length",true)->FindVariableZ
    ("fy",true)->SetValueFloat(fy);
    r->FindElementZ("Camera_Calibration",true)->FindElementZ("Principal_point",true)->
    FindVariableZ("cx",true)->SetValueFloat(cx);
    r->FindElementZ("Camera_Calibration",true)->FindElementZ("Principal_point",true)->
    FindVariableZ("cy",true)->SetValueFloat(cy);
    r->FindElementZ("Camera_Calibration",true)->FindElementZ("Distortion_coeffs",true)->
    FindVariableZ("k1",true)->SetValueFloat(k1);
    r->FindElementZ("Camera_Calibration",true)->FindElementZ("Distortion_coeffs",true)->
    FindVariableZ("k2",true)->SetValueFloat(k2);
    r->FindElementZ("Camera_Calibration",true)->FindElementZ("Distortion_coeffs",true)->
    FindVariableZ("p1",true)->SetValueFloat(p1);
    r->FindElementZ("Camera_Calibration",true)->FindElementZ("Distortion_coeffs",true)->
    FindVariableZ("p2",true)->SetValueFloat(p2);
    // XML object save
    // Manipulate export format
    XMLEXPORTFORMAT xf = {0};
    xf.UseSpace = true;
    xf.nId = 2;
    x->SetExportFormatting(&xf);
     if (x->Save(file) == 1)
        fprintf(stdout,"%s saved.\r\n",file);

    cvNamedWindow( "Uncorrected Image", 1 );
    cvMoveWindow( "Uncorrected Image", 40, 400 );
    cvNamedWindow( "Corrected Image", 1 );
    cvMoveWindow( "Corrected Image", 440, 400 );

    printf("\nPress esc key on the image to exit\n");
    //Undistortion implementation loop
    //Ciclo de implementación de la distorción
```

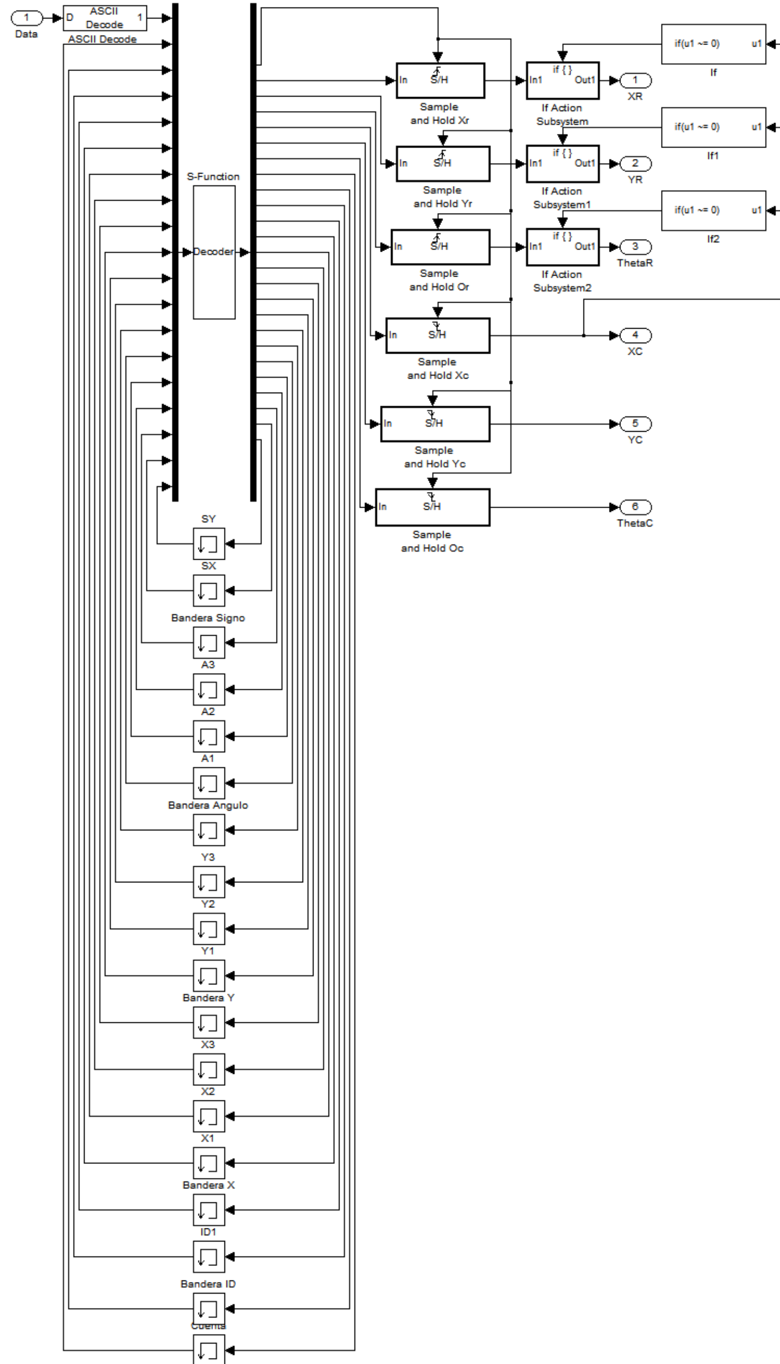
```
    while (1)
    {
        inputImage = cvQueryFrame(capture);
        //Flip the image if camera is inverting
        //Gira la imagen si la camara esta invirtiendo
        cvConvertImage(inputImage,test_image,CV_CVTIMG_FLIP);
        cvUndistort2(test_image, calibrated_image, intrinsic_matrix, distortion_coeffs);
        cvShowImage("Uncorrected Image", test_image);
        cvShowImage("Corrected Image", calibrated_image);
        c=cvWaitKey(1);
        // Exit if Esc key is pressed
        // Sale si se presiona Esc
        if( c == 27 )
            break;
    }
    // Clean up memory
    // Limpieza de memoria
    cvReleaseMat(&object_points_update);
    cvReleaseMat(&image_points_update);
    cvReleaseMat(&points_counts_update);

    if (calibrated_image != NULL) cvReleaseImage(&calibrated_image);
    if (corner_draw != NULL) cvReleaseImage(&corner_draw);

    cvReleaseCapture(&capture);

    return(0);
    #pragma endregion
}

void on_mouse(int event, int x, int y, int flags, void *param){
    switch (event){
        case CV_EVENT_LBUTTONDOWN:
            location = cvPoint(x,y);
    }
}
```

Apéndice IV: Diagrama de bloques del programa “socket” (en Simulink)

Esquema exterior del diagrama



Bloque Visión Decoder



Apéndice V : Código fuente de la función-S “Decoder.m” (en Simulink)

```
function [sys,x0,str,ts,simStateCompliance] = Decoder(t,x,u,flag)
% The following outlines the general structure of an S-function.
%
switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes;

  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1,
    sys=mdlDerivatives(t,x,u);

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2,
    sys=mdlUpdate(t,x,u);

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u);

  %%%%%%%%%%%%%%%%%%%%%%%
  % GetTimeOfNextVarHit %
  %%%%%%%%%%%%%%%%%%%%%%%
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  %%%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%%
  case 9,
    sys=mdlTerminate(t,x,u);

  %%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%
  otherwise
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));

end

% end sfuntmpl

%
%=============================================================================
```

```
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes

%
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
%
% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.
%
sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 25;
sizes.NumInputs      = 19;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
x0  = [];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
%ts  = [0.01 0];
ts  = [0.007 0];

% Specify the block simStateCompliance. The allowed values are:
%    'UnknownSimState', < The default setting; warn and assume DefaultSimState
%    'DefaultSimState', < Same sim state as a built-in block
%    'HasNoSimState',   < No sim state
%    'DisallowSimState' < Error out when saving or restoring the model sim state
simStateCompliance = 'UnknownSimState';

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
```

```
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u)

sys = [];

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%
function sys=mdlOutputs(t,x,u)
global id Xr Yr Or Xc Yc Oc angulo cuenta y contador flagid id1 flagx x1 x2 x3 flagy y1 y2 y3
flagang a1 a2 a3 flagsigno sx sy xx data cont flagid0 id0 flagx0 x0 x00 x000 flagy0 y0 y00 y000
flagang0 a0 a00 a000 flagsigno0 s0 s00;

data=u(1);
cont=u(2);
flagid0=u(3);
id0=u(4);
flagx0=u(5);
x0=u(6);
x00=u(7);
x000=u(8);
flagy0=u(9);
y0=u(10);
y00=u(11);
y000=u(12);
flagang0=u(13);
a0=u(14);
a00=u(15);
a000=u(16);
flagsigno0=u(17);
s0=u(18);
s00=u(19);
```

```
Xr=0;
Yr=0;
Or=0;
Oc=0;
Yc=0;
Xc=0;


[cuenta,id,xx,y,angulo,contador,flagid,id1,flagx,x1,x2,x3,flagy,y1,y2,y3,flagang,a1,a2,a3,
flagsigno,sx,sy] = fcn(data,cont,flagid0,id0,flagx0,x0,x00,x000,flagy0,y0,y00,y000,flagang0,a0,
a00,a000,flagsigno0,s0,s00);

switch id
    case 2
        if ((xx ~= 0) && (y ~= 0))
            Xr=xx/100;
            Yr=y/100;
            Or=(angulo*3.1416)/180;
        end;
    case 0
        if ((xx ~= 0) && (y ~= 0))
            Xc=xx/100;
            Yc=y/100;
            Oc=(angulo*3.1416)/180;
        end;
end;

sys = [id Xr Yr Or Xc Yc Oc contador flagid id1 flagx x1 x2 x3 flagy y1 y2 y3 flagang a1 a2 a3
flagsigno sx sy];

% end mdlOutputs

%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sys = [];
%sampleTime = 1;    %  Example, set the next hit to be one second later.
%sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
```

```
%
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

Apéndice VI : Código fuente de la función-M “fcn.m” (en Simulink)

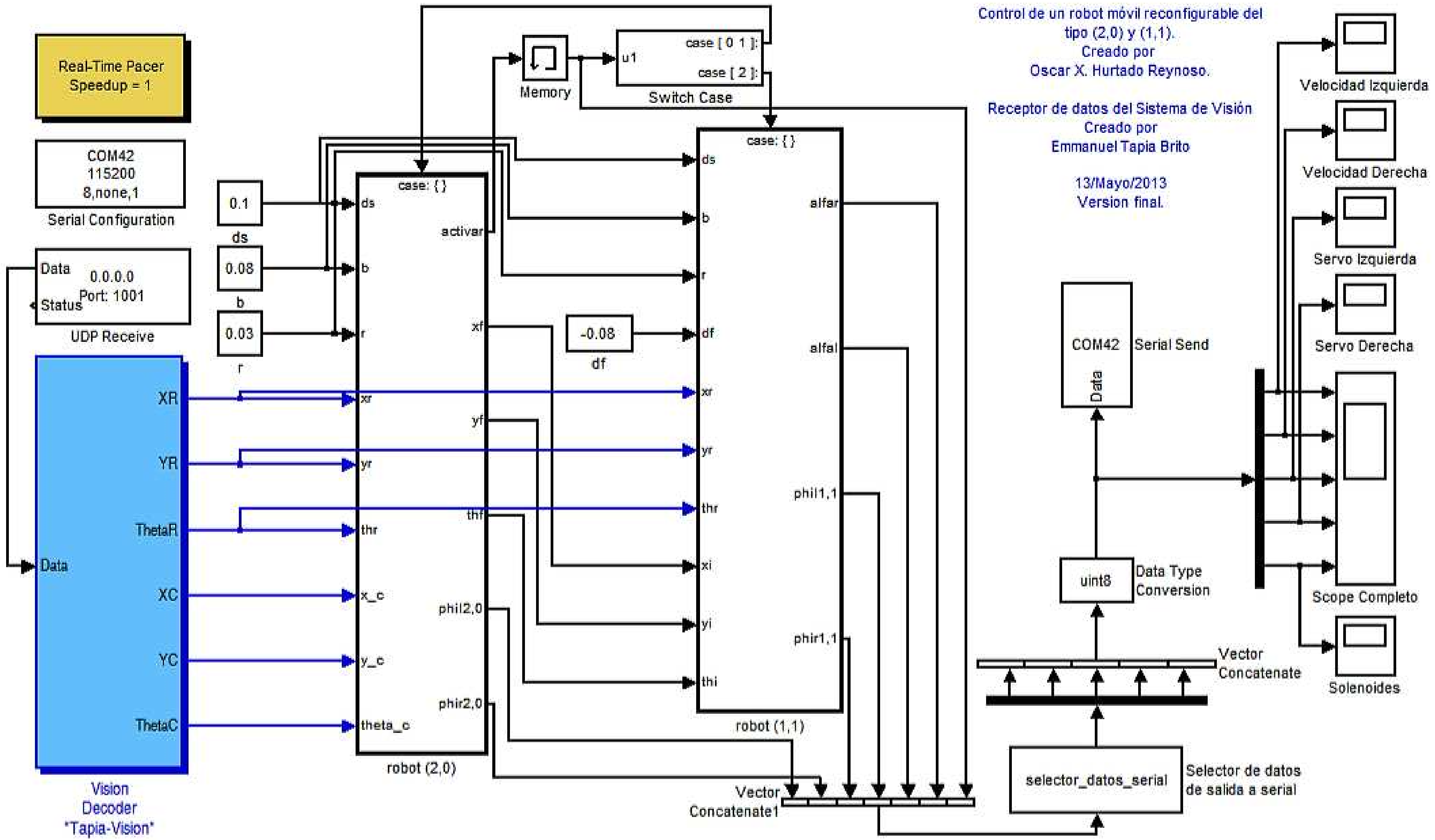
```
function [cuenta,id,x,y,angulo,contador,flagid,id1,flagx,x1,x2,x3,flagy,y1,y2,y3,flagang,a1,a2,
a3,flagsigno,sx,sy] = fcn(data,cont,flagid0,id0,flagx0,x0,x00,x000,flagy0,y0,y00,y000,flagang0,
a0,a00,a000,flagsigno0,s0,s00)
%#codegen
%Variables de retroalimentación
contador=cont;
id1=id0;
flagid=flagid0;
flagx=flagx0;
flagy=flagy0;
flagang=flagang0;
flagsigno=flagsigno0;
x1=x0;x2=x00;x3=x000;
y1=y0;y2=y00;y3=y000;
a1=a0;a2=a00;a3=a000;
sx=s0;sy=s00;

%Variables de transferencia a la salida
cuenta=contador;
%id=0;
%y=0;x=0;signox=0;signoy=0;angulo=0;
signox=0; signoy=0;
        if (s00==0)
            signoy=-1;
        elseif (s00==1)
            signoy=1;
        end;
        if (s0==0)
            signox=-1;
        elseif (s0==1)
            signox=1;
        end;
id=id1-4;
x=(((x3*100)+(x2*10)+x1))*signox/10;
y=(((y3*100)+(y2*10)+y1))*signoy/10;
angulo=(a3*100)+(a2*10)+a1;

%Detección del separador
if (data>100)
    contador=contador+1;
end;

%Detección de caracteres normales
if (data<100)
    if (contador==0)
        if (s00==0)
            signoy=-1;
        elseif (s00==1)
            signoy=1;
        end;
        if (s0==0)
            signox=-1;
        elseif (s0==1)
```

```
            signox=1;
        end;
        id=id0-4;
        x=(((x000*100)+(x00*10)+x0)*signox)/10;
        y=(((y000*100)+(y00*10)+y0)*signoy)/10;
        angulo=(a000*100)+(a00*10)+a0;
    elseif (contador==1)
        flagid=1;
        contador=2;
    elseif(contador==3)
        flagx=1;
        contador=4;
    elseif(contador==5)
        flagy=1;
        contador=6;
    elseif(contador==7)
        flagang=1;
        contador=8;
    elseif (contador==9)
        flagsigno=1;
        contador=0;
    end;

    %Decodificación de ID
    if(flagid==1 && contador==2)
        id1=data;
        flagid=2;
    end;

    %Decodificación de X
    if(flagx==1)
        x3=data;
        flagx=2;
    elseif(flagx==2)
        x2=data;
        flagx=3;
    elseif(flagx==3 && contador==4)
        x1=data;
    end;

    %Decodificación de Y
    if(flagy==1)
        y3=data;
        flagy=2;
    elseif(flagy==2)
        y2=data;
        flagy=3;
    elseif(flagy==3 && contador==6)
        y1=data;
    end;

     %Decodificación de Angulo
    if(flagang==1)
```

```
        a3=data;
        flagang=2;
    elseif(flagang==2)
        a2=data;
        flagang=3;
    elseif(flagang==3 && contador==8)
        a1=data;
    end;

    %Decodificación de Signos
    if(flagsigno==1)
        sx=data;
        flagsigno=2;
    elseif(flagsigno==2 && contador==0)
        sy=data;
    end;

end;
```

Apéndice VII: Diagrama de bloques del control del robot (en Simulink)

.


Real-Time Pacer
Speedup = 1
COM42
115200
8,none,1
Serial Configuration
Data
0.0.0.0
Port: 1001
Status
UDP Receive
Data
XR
YR
ThetaR
XC
YC
ThetaC
Vision
Decoder
"Tapia-Vision"
0.1
ds
0.08
b
0.03
r
case: { }
ds
b
r
xr
yr
thr
x_c
y_c
theta_c
activar
xf
yf
thf
phil2,0
phir2,0
robot (2,0)
Memory
u1
case [0 1]:
case [2]:
Switch Case
-0.08
df
case: { }
ds
b
r
df
xr
yr
thr
xi
yi
thi
alfar
alfal
phil1,1
phir1,1
robot (1,1)
Vector
Concatenate1
Control de un robot móvil reconfigurable del
tipo (2,0) y (1,1).
Creado por
Oscar X. Hurtado Reynoso.
Receptor de datos del Sistema de Visión
Creado por
Emmanuel Tapia Brito
13/Mayo/2013
Version final.
COM42
Data
Serial Send
uint8
Data Type
Conversion
Vector
Concatenate
selector_datos_serial
Selector de datos
de salida a serial
Velocidad Izquierda
Velocidad Derecha
Servo Izquierda
Servo Derecha
Scope Completo
Solenoides

Apéndice VIII: Secuencia de pasos para implementar el sistema de visión (instructivo)

1. Ubicar los archivos "Camera Calibration.exe", "Configuration File.exe" y "Blob tracking.exe"
2. Asegurarse que la cámara esté conectada e identificada. Pueden hacerse mejoras de la imagen accediendo a sus Configuraciones (a través de la Aplicación "Cámara" en Windows 8, Skype, etc.)

CONFIGURACIÓN

1. Abrir el programa "Configuration File.exe"
2. Llenar todos los espacios del formulario y al final presionar "Save Settings"
3. Se generará o actualizará el archivo "Settings.xml"

CALIBRACIÓN

1. Abrir el programa "Camera Calibration.exe".
2. Seleccionar la Cámara que se va a emplear para su calibración y hacer click en OK
3. Colocar frente a la cámara, el tablero de calibración (Apéndice XI) de forma que sea completamente visible. Cuando es completamente visible, aparecerá un zigzag compuesto por líneas de colores que marcan los vértices del tablero. Si no es completamente visible, el zigzag no aparecerá.
4. En caso de que el zigzag no aparezca, hay que presionar la barra espaciadora hasta que el tablero sea reconocido. Al presionar la barra espaciadora se hará una nueva captura.

5. Cuando aparezca el zigzag en la ventana “Image”, hacer click en el punto de inicio del mismo.

6. Hay que repetir esto, en tantas capturas como se haya indicado en el programa “Configuration File”, colocando el tablero a diferentes distancias y en distintas posiciones e inclinaciones. (Nota: Entre mayor sea el número de capturas, mejor será la calibración).

7. Se generarán o actualizarán los archivos “Intrinsics.xml” y “Distortion.xml”

VISIÓN

(Nota: Para mejores resultados asegurarse que ningún disopsitivo (wifi, bluetooth, Ethernet, etc.) está estableciendo conexión con ninguna red.)

1. Verificar que los todos los marcadores visuales se encuentren visibles para la cámara desde el momento de arranque

2. Abrir el programa “Blob Tracking.exe”

3. Presionar la tecla ‘m’(minúscula) para desplegar el menú y las letras que indique para acceder a los submenus. Para cerrar dichos submenús hay que oprimir la misma letra de acceso.

4. Al presionar la tecla “t”, la función Treshold muestra los distintos filtros utilizados

5. Al presionar la tecla “f”, la función Flip Image rota la imagen capturada

6. Al presionar la tecla “c”, la función Color Calibration despliega un menú de calibración del color, donde se pueden manipular las barras de Hue (), Saturation (Saturación), Value (Valor) máximas y mínimas. En la parte inferior se encuentra los recuadros ’Max’ y ’Min’ dónde se muestran los límites del umbral de color discriminado.

7. Al presionar la tecla “a”, la función Area Calibration despliega un menú que permite ajustar las areas minima y máxima de discriminación de figuras, mediante las barras ’Min Area’ y ’Max Area’.

8. Al presionar la tecla “p”, la función Pause Capture detiene la captura

9. Al preisonar la tecla “s”, la función Save Settings guarda las modificaciones hechas a la configuración en el documento “Settings.xml”

10. Al presionar la tecla “x”, la función Exit cierra la ventana de captura

Apéndice IX: Hojas de especificaciones de la tarjeta MD-25

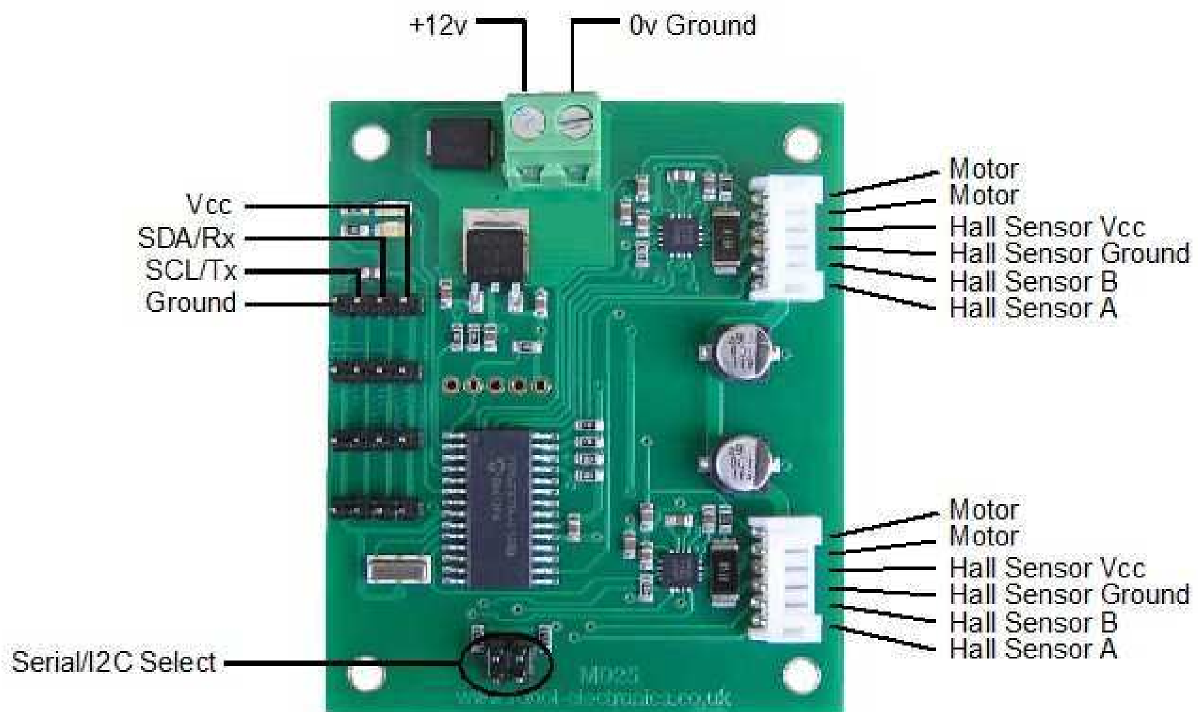
.

MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive

Overview
The MD25 is a robust I2C or serial, dual motor driver, designed for use with our EMG30 motors. Main features are:

1. Reads motors encoders and provides counts for determining distance traveled and direction .
2. Drives two motors with independent or combined control.
3. Motor current is readable.
4. Only 12v is required to power the module.
5. Onboard 5v regulator can supply up to 1A peak, 300mA continuously to external circuitry
6. Steering feature, motors can be commanded to turn by sent value.
7. Variable acceleration and power regulation also included

Connections



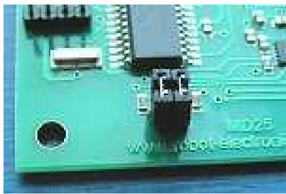
Jumper Selection

I2C mode with no jumpers installed, up to 100 khz clock.
Full Details of I2C Mode is here

Serial mode at 9600 bps, 1 start bit, 2 stop bits, no parity
Full Details of Serial Mode is here

Serial mode at 19200 bps, 1 start bit, 2 stop bits, no parity

Full Details of Serial Mode is here



Serial mode at 38400 bps, 1 start bit, 2 stop bits, no parity
Full Details of Serial Mode is here

Motor Voltage
The MD25 is designed to work with a 12v battery. In practical terms, this means the 9v-14v swing of a flat/charging 12v battery is fine. Much below 9v and the under-voltage protection will prevent any drive to the motors.

Motor Noise Suppression
When using our EMG30 encoded motors, you will find that a 10n noise suppression capacitor has already been fitted. Other motors may require suppression. This is easily achieved by the addition of a 10n snubbing capacitor across the motors. The capacitor should also be capable of handling a voltage of twice the drive voltage to the motor.

Leds
The Red Power Led indicates power is applied to the module.
A Green Led indicates communication activity with the MD25. In I2C mode the green led will also initially flash the address it has been set to. See I2C documentation for further details.

Board dimensions

MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive

I2C mode documentation

(Click here for Serial Mode)

Automatic Speed regulation
By using feedback from the encoders the MD25 is able to dynamically increase power as required. If the required speed is not being achieved, the MD25 will increase power to the motors until it reaches the desired rate or the motors reach there maximum output. Speed regulation can be turned off in the command register.

Automatic Motor Timeout
The MD25 will automatically stop the motors if there is no I2C communications within 2 seconds. This is to prevent your robot running wild if the controller fails. The feature can be turned off, if not required. See the command register.

Controlling the MD25
The MD25 is designed to operate in a standard I2C bus system on addresses from 0xB0 to 0xBE (last bit of address is read/write bit, so even numbers only), with its default address being 0xB0. This is easily changed by removing the Address Jumper or in the software see Changing the I2C Bus Address.

I2C mode allows the MD25 to be connected to popular controllers such as the PICAXE, OOPic and BS2p, and a wide range of micro-controllers like PIC's, AVR's, 8051's etc.
I2C communication protocol with the MD25 module is the same as popular EPROM's such as the 24C04. To read one or more of the MD25 registers, first send a start bit, the module address (0XB0 for example) with the read/write bit low, then the register number you wish to read. This is followed by a repeated start and the module address again with the read/write bit high (0XB1 in this example). You are now able to read one or more registers. The MD25 has 17 registers numbered 0 to 16 as follows;

Register	Name	Read/Write	Description
0	Speed1	R/W	Motor1 speed (mode 0,1) or speed (mode 2,3)
1	Speed2/Turn	R/W	Motor2 speed (mode 0,1) or turn (mode 2,3)
2	Enc1a	Read only	Encoder 1 position, 1st byte (highest), capture count when read
3	Enc1b	Read only	Encoder 1 position, 2nd byte
4	Enc1c	Read only	Encoder 1 position, 3rd byte
5	Enc1d	Read only	Encoder 1 position, 4th (lowest byte)
6	Enc2a	Read only	Encoder 2 position, 1st byte (highest), capture count when read
7	Enc2b	Read only	Encoder 2 position, 2nd byte
8	Enc2c	Read only	Encoder 2 position, 3rd byte
9	Enc2d	Read only	Encoder 2 position, 4th byte (lowest byte)
10	Battery volts	Read only	The supply battery voltage
11	Motor 1 current	Read only	The current through motor 1
12	Motor 2 current	Read only	The current through motor 2
13	Software Revision	Read only	Software Revision Number
14	Acceleration rate	R/W	Optional Acceleration register
15	Mode	R/W	Mode of operation (see below)
16	Command	R/W	Used for reset of encoder counts and module address changes

Speed1 Register
Depending on what mode you are in, this register can affect the speed of one motor or both motors. If you are in mode 0 or 1 it will set the speed and direction of motor 1. The larger the number written to this register, the more power is applied to the motor. A mode of 2 or 3 will control the speed and direction of both motors (subject to effect of turn register).

Speed2/Turn Register
When in mode 0 or 1 this register operates the speed and direction of motor 2. When in mode 2 or 3 Speed2 becomes a Turn register, and any value in this register is combined with the contents of Speed1 to steer the device (see below).

Turn mode
Turn mode looks at the speed register to decide if the direction is forward or reverse. Then it applies a subtraction or addition of the turn value on either motor.

so if the direction is forward
motor speed1 = speed - turn

motor speed2 = speed + turn

else the direction is reverse so
motor speed1 = speed + turn
motor speed2 = speed - turn

If the either motor is not able to achieve the required speed for the turn (beyond the maximum output), then the other motor is automatically changed by the program to meet the required difference.

Encoder registers
Each motor has its encoder count stored in an array of four bytes, together the bytes form a signed 32 bit number, the encoder count is captured on a read of the highest byte (registers 2, 6) and the subsequent lower bytes will be held until another read of the highest byte takes place. The count is stored with the highest byte in the lowest numbered register. The registers can be zeroed at any time by writing 32 (0x20) to the command register.

Battery volts
A reading of the voltage of the connected battery is available in this register. It reads as 10 times the voltage (121 for 12.1v).

Motor 1 and 2 current
A guide reading of the average current through the motor is available in this register. It reads approx ten times the number of Amps (25 at 2.5A).

Software Revision number
This register contains the revision number of the software in the modules PIC16F873 controller - currently 1 at the time of writing.

Acceleration Rate
If you require a controlled acceleration period for the attached motors to reach there ultimate speed, the MD25 has a register to provide this. It works by using a value into the acceleration register and incrementing the power by that value. Changing between the current speed of the motors and the new speed (from speed 1 and 2 registers). So if the motors were traveling at full speed in the forward direction (255) and were instructed to move at full speed in reverse (0), there would be 255 steps with an acceleration register value of 1, but 128 for a value of 2. The default acceleration value is 5, meaning the speed is changed from full forward to full reverse in 1.25 seconds. The register will accept values of 1 up to 10 which equates to a period of only 0.65 seconds to travel from full speed in one direction to full speed in the opposite direction.

So to calculate the time (in seconds) for the acceleration to complete :

if new speed > current speed
steps = (new speed - current speed) / acceleration register

if new speed < current speed
steps = (current speed - new speed) / acceleration register

time = steps * 25ms

For example :

Acceleration register	Time/step	Current speed	New speed	Steps	Acceleration time
1	25ms	0	255	255	6.375s
2	25ms	127	255	64	1.6s
3	25ms	80	0	27	0.675s
5 (default)	25ms	0	255	51	1.275s
10	25ms	255	0	26	0.65s

Mode Register
The mode register selects which mode of operation and I2C data input type the user requires. The options being:
0, (Default Setting) If a value of 0 is written to the mode register then the meaning of the speed registers is literal speeds in the range of 0 (Full Reverse) 128 (Stop) 255 (Full Forward).

1, Mode 1 is similar to Mode 0, except that the speed registers are interpreted as signed values. The meaning of the speed registers is literal speeds in the range of -128 (Full Reverse) 0 (Stop) 127 (Full Forward).

2, Writing a value of 2 to the mode register will make speed1 control both motors speed, and speed2 becomes the turn

value.
Data is in the range of 0 (Full Reverse) 128 (Stop) 255 (Full Forward).

3, Mode 3 is similar to Mode 2, except that the speed registers are interpreted as signed values.
Data is in the range of -128 (Full Reverse) 0 (Stop) 127 (Full Forward)

Command register

Command		Action
Dec	Hex	
32	20	Resets the encoder registers to zero
48	30	Disables automatic speed regulation
49	31	Enables automatic speed regulation (default)
50	32	Disables 2 second timeout of motors (Version 2 onwards only)
51	33	Enables 2 second timeout of motors when no I2C comms (default) (Version 2 onwards only)
160	A0	1st in sequence to change I2C address
170	AA	2nd in sequence to change I2C address
165	A5	3rd in sequence to change I2C address

Changing the I2C Bus Address
To change the I2C address of the MD25 by writing a new address you must have only one module on the bus. Write the 3 sequence commands in the correct order followed by the address. Example; to change the address of an MD25 currently at 0xB0 (the default shipped address) to 0xB4, write the following to address 0xB0; (0xA0, 0xAA, 0xA5, 0xB4). These commands must be sent in the correct sequence to change the I2C address, additionally, no other command may be issued in the middle of the sequence. The sequence must be sent to the command register at location 16, which means 4 separate write transactions on the I2C bus. Because of the way the MD25 works internally, there MUST be a delay of at least 5mS between the writing of each of these 4 transactions. When done, you should label the MD25 with its address, however if you do forget, just power it up without sending any commands. The MD25 will flash its address out on the green communication LED. One long flash followed by a number of shorter flashes indicating its address. Any command sent to the MD25 during this period will still be received and writing new speeds or a write to the command register will terminate the flashing.

Address		Long Flash	Short Flashes
Decimal	Hex		
176	B0	1	0
178	B2	1	1
180	B4	1	2
182	B6	1	3
184	B8	1	4
186	BA	1	5
188	BC	1	6
190	BE	1	7

Take care not to set more than one MD25 to the same address, there will be a bus collision and very unpredictable results.

Apéndice X: Planos del robot

.

35 15 110 15 45

35
35
90
330,4
64,5
35
46
200
124
60
55
30 145 25
161,4
63,16
43,16

	NOMBRE
PLANO POR:	EMMANUEL TAPIA BRITO
MODELADO POR:	OSCAR X. HURTADO REYNOSO
DIBUJO	OSCAR X. HURTADO REYNOSO

DIMENSIONES EN MM

TITULO

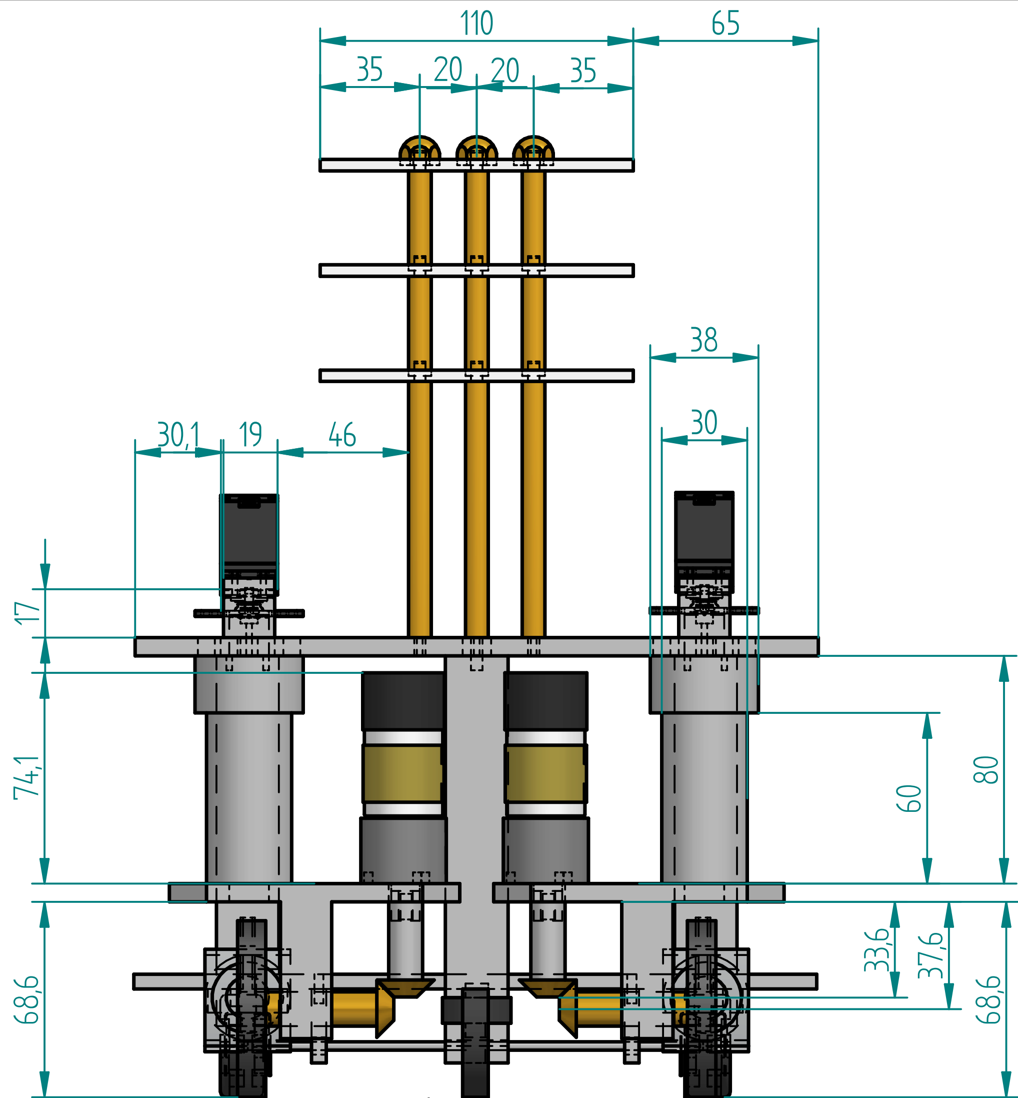
Vista lateral del robot ensamblado

TAMAÑO	PLANO NUMERO	REV
A4	1	

ESCALA: 1 : 2.5

HOJA 1 DE 4

.



	NOMBRE
PLANO POR:	EMMANUEL TAPIA BRITO
MODELADO POR:	OSCAR X. HURTADO REYNOSO
DIBUJO	OSCAR X. HURTADO REYNOSO

DIMENSIONES EN MM

TITULO

Vista frontal del robot ensamblado

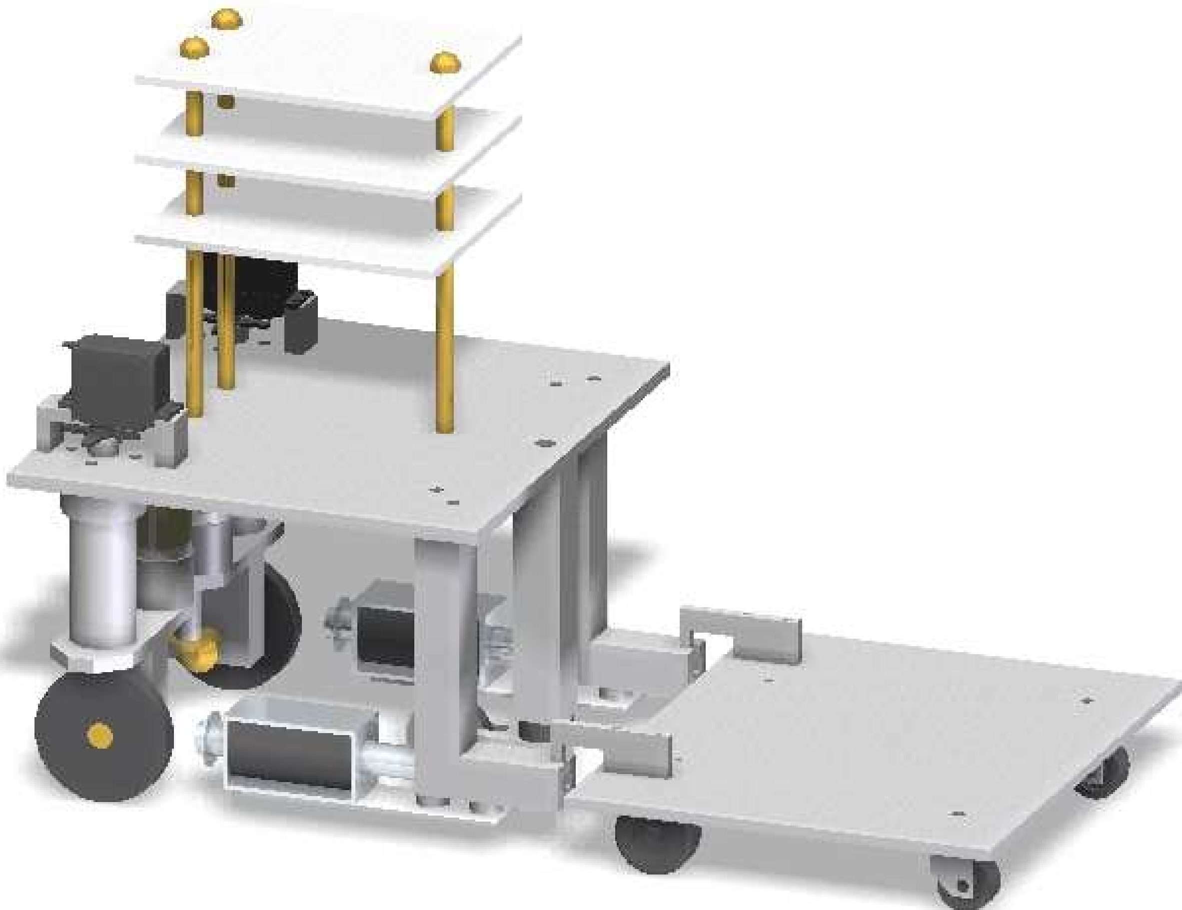
TAMAÑO	PLANO NUMERO	REV
A4	2	

ESCALA: 1 : 2.5

HOJA 2 DE 4

	NOMBRE	TITULO
PLANO POR:	EMMANUEL TAPIA BRITO	Vista en sagital del robot ensamblado
MODELADO POR:	OSCAR X. HURTADO REYNOSO	TAMAÑO: A4 — PLANO NUMERO: 3 — REV
DIBUJO	OSCAR X. HURTADO REYNOSO	
DIMENSIONES EN MM		ESCALA: 1 : 2.5 — HOJA 3 DE 4

.



	NOMBRE	TITULO
PLANO POR:	EMMANUEL TAPIA BRITO	Vista dimétrica del robot ensamblado
MODELADO POR:	OSCAR X. HURTADO REYNOSO	TAMAÑO A4 / PLANO NUMERO 4 / REV
DIBUJO	OSCAR X. HURTADO REYNOSO	
DIMENSIONES EN MM		ESCALA: 1 : 3 / HOJA 4 DE 4

Apéndice XI: Tablero de calibración