Universidad Carlos III
De Madrid

Ingeniería Técnica Industrial
en Electrónica Industrial

Proyecto Fin de Carrera

Desarrollo de una cámara omnidireccional para un robot mini humanoide.

Proyecto dirigido por:
Alberto Jardon Huete.

Proyecto tutorado por:
Martin Fodstad Stoelen.

Proyecto realizado por:
Félix Rodríguez Cañadillas.

Leganés, Madrid
Mayo 2011

UNIVERSIDAD CARLOS III DE MADRID

Ingeniería Técnica Industrial en Electrónica Industrial

El tribunal aprueba el proyecto fin de carrera titulado “Desarrollo de una cámara omnidireccional para un robot mini humanoide” realizado por Félix Rodríguez Cañadillas.

Fecha: Mayo 2011

Tribunal:

Presidente: Fabio Bonsignorio.

Secretario: Concepción Alicia Monje.

Vocal: Higinio Rubio Alonso.

Agradecimientos

En primer lugar quisiera agradecer a Martin, la oportunidad que me ha brindado para realizar este proyecto y aprender de él, y a la Asociación de Robótica de la Universidad Carlos III el permitirme realizarlo.

Agradecer a mis padres de todo corazón, su incansable trabajo para darme la oportunidad que ellos no tuvieron y por los valores que me ha inculcado a lo largo de mi vida.

A mi familia, por haber confiado en mí, especialmente, a mi hermana María que nunca ha dudado de mis posibilidades.

A mis amigos, porque me han apoyado incondicionalmente y me han hecho sentir especial.

A mis compañeros y amigos de la Universidad, por los buenos momentos pasados y por su ayuda.

A Gloria, por haber estado siempre a mi lado, además de por su cariño, apoyo y confianza depositada en mi.

Índice

Índice de figuras

1 Introducción

El Proyecto de robótica de Mini Humanoides [1] surge dentro de la Asociación de Robótica de la Universidad Carlos III de Madrid, como una alternativa para poder inicializarse de manera directa con la robótica humanoide y competir en concursos nacionales e internacionales. Dicha iniciativa tuvo una acogida positiva dentro del Departamento de Ingeniería de Sistemas y Automática, lo que permitió que inicialmente se emplearan recursos del RoboticsLab para realizar las primeras aproximaciones a los pequeños humanoides para concursos y poder explorar las diferentes posibilidades en el área de los robots mini Humanoides.

A través de esta iniciativa, podemos investigar las distintas áreas relacionadas con la robótica mini humanoide. Una de estas áreas es la interacción con el entorno mediante la utilización de sensores, los cuales dan al robot la autonomía suficiente para el reconocimiento del medio que les rodea.

Los distintos tipos de sensores para la exploración de este área, nos abre una gran variedad de posibilidades así como la posibilidad de hacer un estudio de cualquiera de ellos con el fin de mejorar este área de la robótica mini humanoide. Pero los sensores típicamente utilizados en concursos para robots mini humanoides tienen limitaciones en su cobertura del entorno que les rodea. Por ejemplo los sensores infrarrojos, que miden la distancia hasta un punto en el entorno, requiriendo normalmente hasta 5-10 sensores para una cobertura aceptable.

Aquí es donde nace este proyecto, desde la motivación de crear un sistema de visión con una cobertura más completa para el robot. Este sistema deberá mejorar las técnicas de interacción con el entorno que el robot mini humanoide ya posee, complementando a los sensores de ultrasonido, infrarrojo y presión.

Por tanto, a través de este proyecto intentaremos comenzar el desarrollo de un sistema de visión para un robot mini humanoide, haciendo hincapié en los distintos tipos de algoritmos utilizados en visión, así como la correcta colocación del sistema en el robot.

Por último, destacar que este proyecto abre un gran número de posibilidades en el área de visión, y por tanto, siempre se podrá continuar la implementación de diversos algoritmos de visión más allá de los que veremos en este proyecto. La visión es un área muy grande y de gran interés en el campo de robótica, en el cual siempre podremos avanzar hasta conseguir un sistema más completo de visión, pero teniendo en cuenta las limitaciones de recursos y procesamiento abordo un robot mini humanoide

1.1 Descripción general

Con este proyecto se pretende desarrollar la implementación de una cámara omnidireccional de tamaño pequeño para robots mini humanoides, de tal manera que permita al robot interactuar con el medio de forma autónoma.

Se pretende también que los desarrollos realizados en la asociación y en este proyecto en concreto, sean puestos a prueba y comparados frente a otros en campeonatos de robótica humanoide, participaciones que permitirán ver si se está trabajando de forma adecuada y al mismo tiempo aprender de las experiencias de otras personas y grupos de investigación en el campo.

2 Objetivos generales

2.1 Investigar los algoritmos de visión por computador adecuados

El primer objetivo de este proyecto es la investigación a cerca de los algoritmos de visión por computador. Hacer un estudio sobre las diferentes teorías de visión por computador y cuál de ellas será la mejor para nuestro proyecto, debe ser la primera parte de nuestro proyecto.

2.2 Implementar los algoritmos de visión en Matlab.

Para hacer un correcto estudio sobre los algoritmos de visión por computador deberemos utilizar la herramienta de programación Matlab, con la cual a través de su sistema gráfico conseguiremos probar los algoritmos de visión antes de implementarlos en la cámara.

Para la consecución de este objetivo, no solo será necesario desarrollar el código de programación requerido en Matlab, sino que además deberemos simular distintos tipos de entornos, similares a los que el robot se encontrará posteriormente.

2.3 Implantación de la cámara en el robot

La colocación de la cámara en el robot es otro de los objetivos de este proyecto, teniendo que desarrollar la mejor de las opciones para el correcto funcionamiento de la cámara.

Para el cumplimiento de este objetivo será necesario cumplir las especificaciones de dimensiones propuestas en los campeonatos que más adelante veremos, además del estudio de la mejor opción en cuanto a la colocación de la cámara en el robot.

2.4 Implementar los algoritmos seleccionados en la cámara como sensor independiente

Una vez que hayamos hecho un correcto estudio de las teorías de visión, de cómo aplicarlas y de cómo utilizarlas de la mejor manera posible en el robot, deberemos desarrollar los algoritmos de visión para nuestra cámara.

Conjuntamente necesitaremos familiarizarnos con el entorno de programación utilizado en el software de la cámara y a partir de este, desarrollar los algoritmos de visión por computador que hayamos seleccionado para la consecución de este proyecto.

Una vez concluido esto, la cámara deberá funcionar como un sensor independiente capaz de informar del medio en el que se encuentra el robot.

2.5 Controlar el robot mediante la cámara de visión

Desarrollar el sistema de control e implantarlo en el robot es uno de los objetivos más importantes de este proyecto.

Por tanto, deberemos desarrollar e implantar en el robot la programación necesaria, así como los protocolos de comunicación para que la interacción entre los sistemas de control mecánico del robot y la cámara, como sensor independiente, puedan interactuar consiguiendo de esta manera un sistema de total autonomía.

2.6 Participación en campeonatos de robots humanoides

El objetivo final de nuestro proyecto será, participar en concursos de robótica humanoide que permitan poner a prueba los algoritmos diseñados para el robot según la tarea a desarrollar, como pueden ser, pruebas de caminata con evasión de obstáculos, subir escaleras o sumo. Para cada una de las actividades mencionadas antes es necesario desarrollar no sólo estrategias de control para el movimiento del robot, sino que además es necesario dotar al robot con una lógica robusta que le permita desarrollar las tareas de manera autónoma y eficiente.

Posibles campeonatos:

- CampusBot 2011.
- CEABOT 2011.

A continuación, veremos más de cerca cada una las pruebas que recoge el campeonato CEABOT 2011, cuyas bases podemos consultar en el ANEXO I. Las investigaremos con el fin de saber, que tipo de algoritmo de visión puede ayudarnos más en cada una de estas pruebas.

- **Prueba 1:** *Carrera de obstáculos:*

Esta prueba se basa en completar un recorrido a través de una zona plagada de obstáculos, teniendo que cruzar esta zona desde una línea de salida a otra de llegada. Hay que tener en cuenta que las líneas son de color amarillo, el suelo es verde y los obstáculos son de color blanco, como podemos apreciar en la figura 2.1. Esto nos puede ayudar a diferenciar entre ellos, siempre y cuando podamos diferenciar colores.

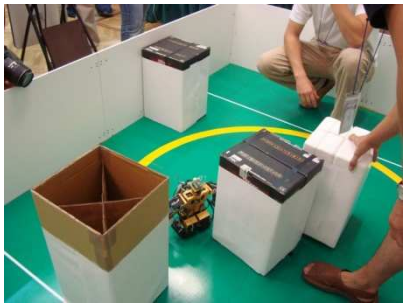


Figura 2.1: Prueba de obstáculos del campeonato CEABOT.

Por tanto, una de las primeras investigaciones que deberemos realizar será la diferenciación de colores e identificación de estos.

También es importante el tiempo de realización del recorrido, por esto, la elección de un recorrido correcto es otro de los puntos que deberemos investigar. Puede que utilizando el campo de visión de la cámara podamos determinar el recorrido más corto, o al menos, el menos plagado de obstáculos.

Por último, también será interesante la investigación del control de nuestro desplazamiento, es decir, si nos estamos moviendo y a qué velocidad lo hacemos. Esto conseguiremos hacerlo mediante la cantidad de movimiento que haya entre el robot y los elementos de alrededor, los cuales están estáticos.

- **Prueba 2:** *Escalera:*

En esta prueba se demuestra la capacidad del robot para subir y bajar escaleras (ver figura 2.2). Una buena manera de apoyar al robot en la consecución de esta prueba, será la localización de la escalera, la cual podremos determinar mediante la detección de bordes.

Figura 2.2: Prueba de escalera del campeonato CEABOT.

Por tanto, el estudio de la localización de elementos será otro de los puntos a investigar en este proyecto, y más específicamente para esta prueba, la detección de elementos no móviles.

- **Prueba 3:** *Lucha (Sumo):*

En la prueba de lucha, como podemos observar en la figura 2.3, se recrea el combate de Sumo mediante la pelea de dos robots adversarios. Con el fin de poder ayudar a nuestro robot a ganar dicho combate, lo apoyaremos utilizando la cámara y aplicando las técnicas de visión necesarias para esta prueba. Una de las técnicas puede ser la detección de movimiento, con la cual, podremos detectar si el robot adversario se está moviendo, a qué velocidad lo hace y hacia donde lo hace.

Figura 2.3: Prueba de lucha del campeonato CEABOT.

Cabe destacar, que el área de la zona de combate está marcada por la normativa del CEABOT y determinada por una línea circular de color amarillo. Será importante la localización de esta línea ya que no debemos sobrepasarla.

Por lo tanto, la correcta localización del adversario y la de nuestro robot será la base de nuestra investigación para esta prueba.

- **Prueba 4:** *Prueba libre-exhibición:*

La prueba de exhibición se basa en una libre demostración de lo que es capaz de hacer el robot mecánicamente, mediante una serie de movimientos como si fuese una coreografía. Al ser una prueba en la cual el robot no interactúa con el medio, no será necesario apoyar al robot por medio de nuestra cámara. Sin embargo, si en esta demostración utilizásemos algún tipo de elemento externo como una pelota o un cubo, si seríamos capaces de ayudar al robot a interactuar con dicho elemento.

Así, la mejor manera de apoyar al robot en esta prueba mediante la cámara será la localización de algún tipo de elemento externo, atendiendo a su forma, color, tamaño, movimiento y situación.

3 Descripción y alternativas del robot mini humanoide

3.1 Descripción general

En este proyecto se desarrollará un robot humanoide y su sistema de control para participar en el concurso CEABOT 2011. Se elegirá el robot que cumpla la mayoría de las siguientes características:

- Kit comercial de robot humanoide.
- Sistema de visión simple que permita reconocer las características del entorno.
- Sensor de distancias, que complemente al sistema de visión, para reconocer la distancia a la que se encuentran los objetos detectados.
- Sensor de inclinación que permita al robot mantener el equilibrio.
- Sistema de comunicación remoto que permita la descarga inalámbrica de programas y el control remoto del dispositivo mediante un mando o controlador.
- Entorno de desarrollo adecuado para el robot con un lenguaje de programación sencillo.

3.2 Alternativas:

En esta sección se resume estas características de las plataformas más típicas, que son importantes para el desarrollo e integración del sistema de visión. Existen una gran cantidad de kits comerciales de robots humanoides. Las principales alternativas son las siguientes:

3.2.1 Bioloid Comprehensive Kit:

Este kit permite construir robots avanzados de hasta 18 grados de libertad con diversas configuraciones, por ejemplo, un humanoide. Se trata de una plataforma robótica con tecnología inteligente servo-controlada en serie, que permite retroalimentación y control sensorial de posición, velocidad, temperatura, corriente y tensión de cada servo. Las principales características del kit son las siguientes:

- Módulo controlador CM-5 basado en el Atmel ATMega128 a 16 MHz.
- 18 servomotores AX-12 (Servomotores Dynamixel controlados en serie).
- Módulo sensor del robot AX-S1 (con 1 sensor infrarrojo, 3 emisores infrarrojos, 1 micrófono y 1 altavoz piezoeléctrico).
- Puertos de conexión serie.
- Batería recargable (9,6V, 2,3Ah, NiMH).
- Alimentador de potencia.
- Cable serie RS-232 de 9 pines.
- Utilidades de software: Editor de movimientos (Motion Editor) y Programador del comportamiento (Behaviour Control Program)
- Precio aproximado: 700 €
- Altura: 34 cm.

Figura 3.1: Bioloid Comprehensive Kit.

3.2.2 Robonova-1:

Este kit permite montar un robot humanoide de aluminio anodizado y plástico, con 16 servomotores digitales con piñonería de Karbonite, que permite colocar manualmente el robot en una posición y posteriormente leer los valores de los servos desde el controlador. Su placa de control admite múltiples ampliaciones. Tiene las siguientes características:

- Altura 30.5 cm -Peso 1,3 kg.
- 16 servos digitales HSR-8498HB (Elevado par de fuerza y muy resistentes y duraderos).
- Circuito controlador MRC-3024 basado en el AtMega128.
- Piezas de aluminio anodizado y plástico.
- Batería recargable de NiMH 6V / 1000 mA y cargador de baterías rápido a 220V.
- Mando a distancia por infrarrojos con sensor IR y altavoz incorporado.
- Cables de conexión Serie RS232.
- Utilidades software intuitivas y prácticas: Robo-Remocon, Robo-Basic y Robo-Script.
- Precio aproximado: 700 €.

Figura 3.2: Robonova-1.

3.2.3 Kondo KHR-2HV:

Este kit permite montar fácilmente un robot de 17 servos muy sofisticados y una estructura de aluminio que le protege de los golpes en las juntas. Incluye dos placas de

control que trabajan simultáneamente. Se puede programar de forma sencilla usando el software que incluye el kit, con una buena interface gráfica de usuario, y puede ser controlado vía control remoto o a través del PC. Las características principales son:

- Altura 34 cm -Peso 1,2 kg.
- 17 grados de libertad (1 en la cabeza-cuello, 3 en cada brazo y 5 en cada pierna)
- 17 servomotores KRS-784ICS.
- 2 Placa de control RCB-1 con 2x128 Kb de memoria (Cada una puede controlar hasta 12 servos).
- Interface serie RS-232.
- Batería Ni Cd 6V 600 mA-h.
- Software de programación Heart-to-Heart3
- Precio aproximado: 800 €

Figura 3.3: Kondo KHR-2HV.

3.3 Plataforma utilizada

El robot utilizado por la Asociación de Robótica para el desarrollo del proyecto de Robótica Mini Humanoide es el Robot Bioloid, fabricado por Robotis, el cual podemos ver en la figura 3.4.

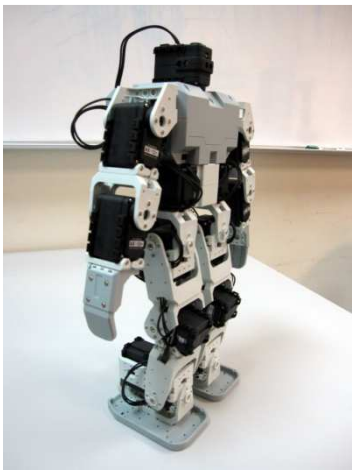


Figura 3.4: Robot Bioloid utilizado en el RoboticLab.

4 Cámara Surveyor Blackfin

Para la realización de este proyecto hemos determinado la utilización de una cámara de visión completa. La cámara elegida para esta labor es la Surveyor Blackfin Cámara, ver figura 4.1, de la cual veremos las principales características a continuación, podemos consultar más detenidamente toda la información de la cámara en la web de Surveyor Corporation [2].

Figura 4.1: Cámara Surveyor Blackfin.

4.1 Características principales

La principal característica de la cámara Surveyor Blackfin es que viene dotada con un microprocesador, en concreto, el Analog Devices Blackfin BF537 Procesor, el cual tiene una velocidad de 500MHz y soporta un desarrollo de software en C. Además, vienes provisto de conectores tales como el JTAG (14 pines) o el External I/O Header (32 pines), éstos nos resultarán de gran ayuda a la hora de conectar de diferentes formas con la cámara.

Respecto al objetivo de la cámara, debemos saber que se trata de un OV655 de 1,3 megapixel. Este objetivo nos da la posibilidad de utilizar dos lentes de diferentes características, una con un ángulo de visión de 90^{o} y otra con un ángulo de 120^{o}. Cabe destacar, que también existe la posibilidad de elegir el ángulo del conector entre la placa de microprocesador y la placa del objetivo, pudiendo elegir entre un conector que nos sitúe las dos placas en paralelo u otro el cual las deje perpendiculares entre sí.

Por último, destacar la gran cantidad de módulos de conexión que existen para esta cámara.

4.2 Módulos utilizados

Una vez vistas las características de la cámara Surveyor Blackfin y sus diversas configuraciones, hemos decidido utilizar los siguientes módulos para la tarea a desarrollar en este proyecto.

La primera de las partes de la cámara es el objetivo OV9655 y su correspondiente placa (ver figura 4.2). Gracias a ella, podemos recoger la imagen captada y por medio del conector de 32 pines que en ella se encuentra, pasaremos la información al microprocesador que posteriormente veremos. Además hemos elegido un conector de

90º con el cual conseguimos que el objetivo de la cámara quede paralelo a la placa del micro procesador.

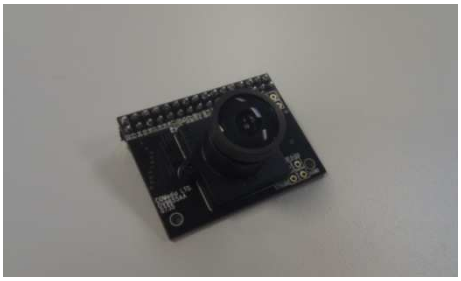


Figura 4.2: Objetivo de la cámara y placa.

Como hemos visto anteriormente en las características de la placa, tenemos la opción de elegir una lente de mayor ángulo de visión. Debido a la utilización y posicionamiento que queremos darle a la cámara, la mejor opción es la utilización de la lente de 120º. A continuación, en la figura 4.3, se muestran ambas lentes.

Lente de 90º de ángulo de visión

Lente de 120º de ángulo de visión

Figura 4.3: Comparativa entre lentes disponibles.

El microprocesador utilizado por la cámara Surveyor Blackfin es el Analog Devices Blackfin BF537 Processor (ver figura 4.4), el cual viene perfectamente implantado en su placa base. Esta placa tiene varios conectores, de los que podemos destacar el conector de 32 pines, procedente de la placa del objetivo de la cámara y el conector de 32 pines al cual le podremos conectar otras placas de extensión como la placa Wifi o la placa de extensión de pines, además del conector JTAG al cual podremos conectar nuestro módulo gnICE+.

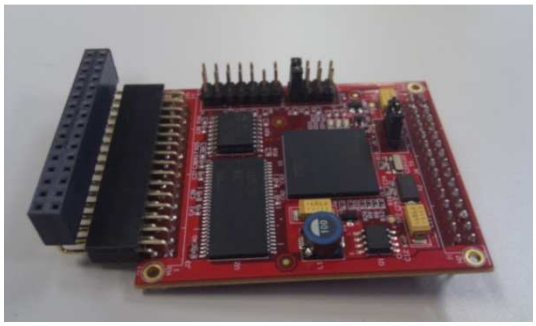


Figura 4.4: Placa del micro procesador.

Como ya hemos citado anteriormente, otro de los módulos que utilizaremos en este proyecto, es el módulo de Wifi, que se muestra en la figura 4.5. Con él,

conseguiremos conectar con la cámara de manera rápida y además tener una conexión en tiempo real, con la cual conseguiremos ver cómo se comportan los algoritmos de visión que posteriormente implementaremos en la cámara.

Figura 4.5: Módulo Wifi.

Además, utilizaremos la placa de extensión de pines (ver figura 4.6) del micro procesador a través de la cual podremos hacer una conexión serie con el robot.

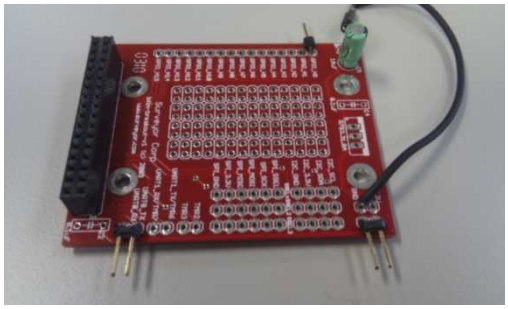


Figura 4.6: Placa de extensión de pines.

Por último, como podemos observar en la figura 4.7, vamos a utilizar el conector gnICE+, con el que accederemos a la Blackfin por medio de su conexión JTAG. Este conector nos será de gran ayuda a la hora de cargar código en el procesador de la Blackfin cámara.

Figura 4.7: Conector gnICE+.

5 Investigación de los algoritmos de visión por computador

5.1 Visión por computador

Como podemos consultar en el libro *'Vision por Computador'* [3], la visión por computador se define como el proceso de extracción de información del mundo físico a partir de imágenes, utilizando para ello un computador.

Existen varios términos que hacen referencia al campo de la visión artificial: visión por computador, visión de máquina, visión de robot, visión computacional, análisis de imágenes, interpretación de escena. Cada uno de estos términos, aunque abordando el mismo problema, posee connotaciones distintas y enfatizan aspectos distintos involucrados en el proceso de la visión artificial.

La visión es, sin duda, nuestro sentido más poderoso y a la vez más complejo. En un principio se pensó que crear un sistema de visión por computador era algo bastante fácil. El argumento para ello era que los computadores eran muy potentes; por ejemplo, aunque resolver un sistema de ecuaciones diferenciales es difícil para los humanos, los computadores lo hacen con apenas dificultad. De esta forma, si se considera una tarea que es trivial para los humanos, como la visión, para los ordenadores debería ser más fácil aún. Cuando se intentaron los primeros sistemas de visión artificiales se fracaso rotundamente. La explicación de este fracaso reside en lo superficial y errado del razonamiento anterior: mientras que los humanos son conscientes de la mayoría de etapas de procesamiento involucradas en la resolución de un sistema de ecuaciones diferenciales y, por tanto, son conscientes de la complejidad del proceso (pudiendo por consiguiente trasladarlo a un computador), la mayoría del procesamiento que se realiza en la percepción visual se lleva a cabo de forma inconsciente, siendo por ello imposible de imitar o transportar a un sistema informático.

Debido a esto, para construir un sistema de visión artificial es necesario conocer y emplear restricciones físicas sobre los objetos del mundo real y sobre su proyección en imágenes. Es conveniente destacar que esta predisposición natural a imitar al humano puede acarrear ciertas desventajas. Un ejemplo de ello son las ampliamente conocidas ilusiones visuales.

5.1.1 Etapas en un proceso de visión por computador

Las etapas a considerar en un proceso de visión artificial dependen del objetivo perseguido: reconocer, localizar, estimar forma, etc. En la figura 5.1 se muestran, en un sentido global, las distintas etapas que suelen considerarse, las cuales se definen a continuación:

- Adquisición de la imagen: tiene por objeto plasmar en una imagen digital el mundo real tridimensional.
- Pre-procesamiento: incluye aquellas operaciones encaminadas a preparar la imagen para posteriores etapas, como son la eliminación de ruido y el realce.
- Detección de bordes: su importancia es vital en muchos de los procesos de visión ya que permite extraer de la imagen los bordes de los objetos.

- Segmentación: tiene por objeto determinar en la imagen regiones cuyos píxeles comparten algún tipo de atributo. Estas regiones, previsiblemente, van a corresponder a objetos de interés de la escena.
- Extracción de características: obtiene una representación matemática de los objetos previamente segmentados.
- Reconocimiento: se clasifican los objetos como pertenecientes a aquella clase o prototipo cuyas características más se asemejen a la del objeto.
- Localización: se procede a localizar al objeto en el espacio 3D. Para ello, es necesario recurrir a técnicas de triangulación o a restringir el espacio de acuerdo con el conocimiento que se tenga de la escena.
- Interpretación: con la información obtenida en las etapas anteriores se procede a interpretar la escena, considerando para ello la relación entre los objetos simples previamente reconocidos y localizados, así como un cierto conocimiento sobre restricciones y reglas que rigen el mundo real.

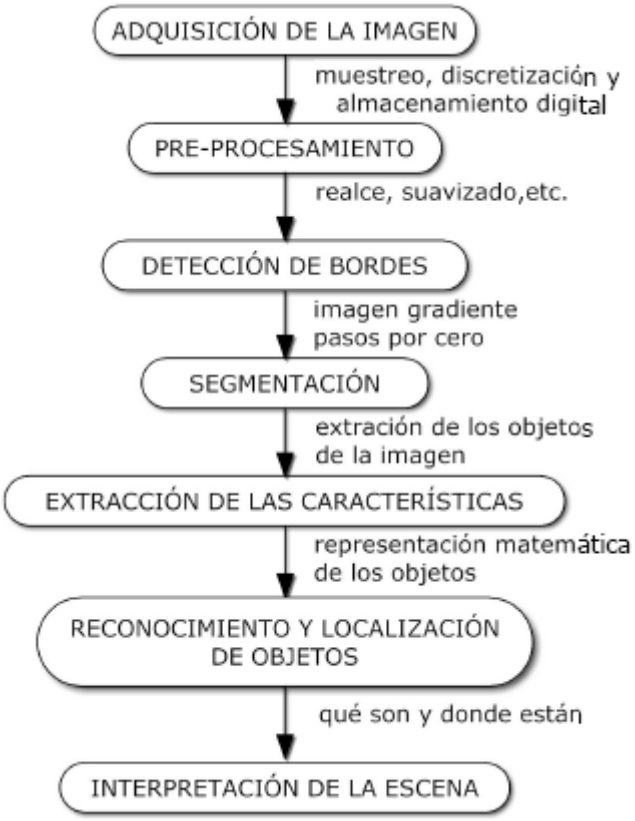


Figura 5.1: Etapas que pueden considerarse en un proceso de visión por computador

5.1.2 Componentes de un sistema de visión

Los componentes básicos de un sistema de visión son los que se muestran en la figura 5.2. La entrada del sistema suelen ser una o varias cámaras de vídeo, aunque también puede ser cualquier otro sensor de imagen como un ecógrafo, escáner óptico, escáner de resonancia magnética, etc.

La información recogida por el sensor es enviada (normalmente de forma analógica) a un computador donde una tarjeta se encarga de la digitalización y, en algunos de los casos, del almacenamiento y procesamiento a bajo nivel de la imagen. Estas tarjetas pueden ser conectadas al computador mediante los buses más comunes, en particular, bus VME, Q- bus (Sun Microsystem), bus PCI, bus ISA y Nu- bus (Apple Mancintosh). Las prestaciones de estas tarjetas varían tremendamente y con ellas, lógicamente, sus precios; desde aquellas que tan sólo permiten la digitalización y el almacenamiento de una única imagen monocromo, hasta otras que incluyen hardware para procesamiento en tiempo real de imágenes en color.

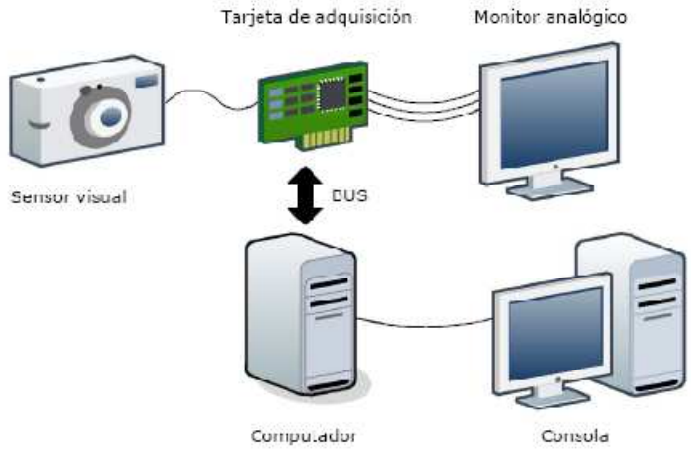


Figura 5.2: Componentes básicos de un sistema de visión por computador.

Para la visualización de las imágenes capturadas puede ser necesario, dependiendo del tipo de tarjeta, un monitor analógico, además del monitor digital del computador. Normalmente, las imágenes son trasferidas por el BUS desde la tarjeta a las RAM del computador para su procesamiento a alto nivel. El computador puede ser cualquiera de los existentes en el mercado, condicionando, lógicamente, el tipo de tarjeta de adquisición y las prestaciones del sistema.

5.2 Ópticas

La función de la óptica de una cámara es la de captar los rayos luminosos para concentrarlos sobre el sensor de imagen. Idealmente, la imagen obtenida debería ser una fiel reproducción de los objetos de la escena, aunque invertida y con diferente tamaño.

La óptica de una cámara se compone de diferentes partes; exteriormente, la óptica u objetivo adopta la forma de un cilindro metálico cuya cara anterior es una lente; en su parte posterior (montura) posee una rosca que permite su fijación a la cámara. En su interior se dispone de una agrupación de lentes de diversas características y por ultimo en la parte anterior de la óptica aparecen impresas una serie de anotaciones que definen alguna de las características de la óptica.

Las ópticas comerciales están compuestas por agrupaciones de lentes positivas o convergentes y negativas o divergentes. Las lentes positivas hacen converger en un punto todos los haces de luz que reciben. Por el contrario, las lentes negativas se caracterizan porque los rayos de luz paralelos que las atraviesan se apartan de este paralelismo, divergen entre sí.

5.2.1 Modelo del agujero o "pin-hole"

Es el modelo más simple de lente. En el modelo "pin-hole" cada punto de un objeto del espacio se proyecta en un punto de un plano denominado plano imagen, donde se forma la imagen de la escena 3D (ver figura 5.3). Puesto que el agujero es de diámetro infinitesimal, de los rayos de luz proceden de cada punto 3D sólo uno pasa por él, por lo que sobre cada elemento del sensor imagen incide un único rayo de luz. De acuerdo con este proceso, todos los puntos del espacio estarán siempre perfectamente enfocados.

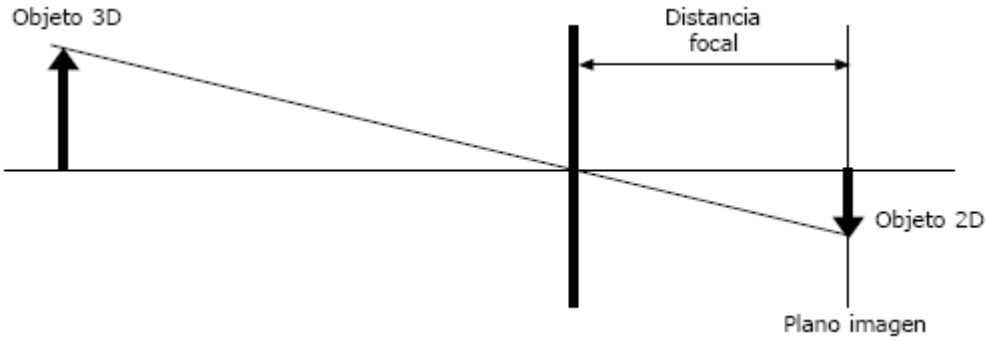


Figura 5.3: Proyección de un punto de un objeto 3D en el plano imagen utilizando el modelo del agujero ("pin-hole").

Por su simplicidad, este modelo es muy utilizado en un gran número de aplicaciones y procesos de visión por computador. Sin embargo, desde un punto de vista radiométrico es incapaz de explicar la formación de imágenes, ya que, un único rayo procedente de un punto del espacio no posee energía suficiente para excitar el elemento del sensor. Si el agujero se hace mayor permitiría una cantidad de rayos suficientes para ello, pero la imagen estaría desenfocada.

Además de todo esto, el inconveniente fundamental de este modelo es que no contempla la mayoría de las características y parámetros típicos de una óptica como el desenfoque de objetos, el control de la cantidad de luz incidente en el sensor, etc.

En realidad el único parámetro que modela es la distancia focal, que controla los aumentos o "zoom".

5.2.2 Modelo de lente delgada

Es un sistema más completo que el anterior que permite modelar la mayoría de los parámetros de las ópticas reales. Este sistema modela la óptica como una única lente que concentra en un punto los infinitos rayos luminosos procedentes de un determinado punto del espacio. De acuerdo con este modelo, todos los rayos paralelos al eje óptico de la lente convergen en un punto, que es el foco F como podemos observar en la figura 5.4

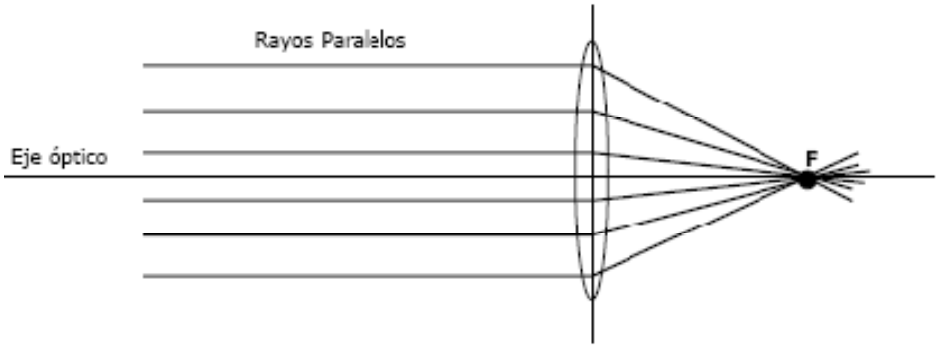


Figura 5.4: Todos los rayos paralelos al eje óptico convergen en el foco F.

En la figura 5.5 se muestra el proceso de formación de la imagen con este modelo. La proyección de un punto dado A_o se determina en base a dos rayos principales: uno paralelo al eje óptico de la lente (r_1) y otro que pasa por el centro óptico (r_2). El punto de inserción de ambos determina el punto A_i donde convergerán los infinitos rayos que provienen de A_o pasan a través de la lente. Asimismo, el foco F de la lente viene dado por el punto donde el rayo r_1', refractado de r_1, corta al eje óptico.

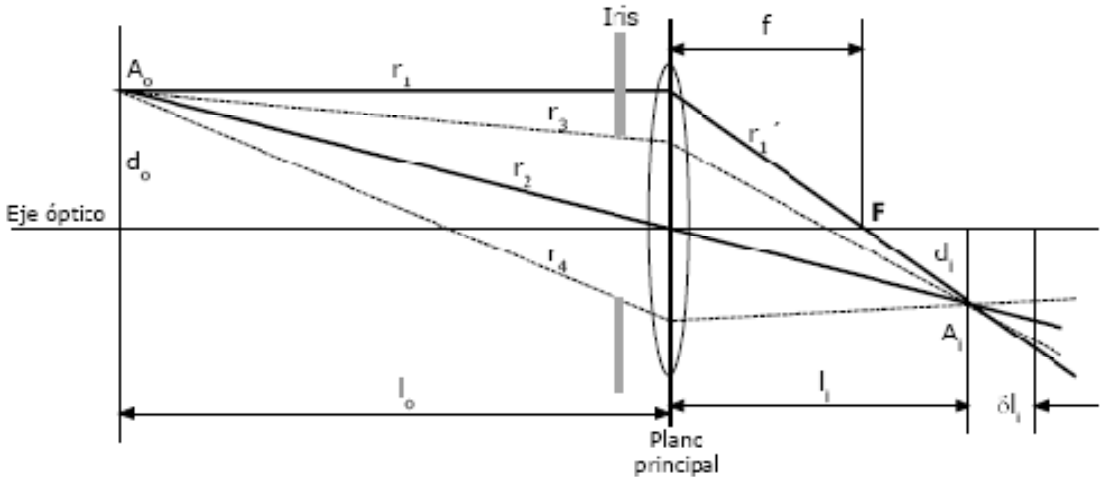


Figura 5.5: Proceso de formación de la imagen en el modelo de lente delgada.

Los parámetros que intervienen en este modelo son:

1. Distancia focal

La distancia focal es la distancia "f" del centro de la lente al foco, y determina el tamaño de la imagen formada en el sensor (efecto "zoom"). En concreto, una distancia focal pequeña reduce el tamaño de los objetos en la imagen, mientras que una grande la aumenta. Puesto que el tamaño del sensor es fijo, la distancia focal controla también el ángulo de visión: a mayor distancia focal menor ángulo de visión.

2. Distancia de enfoque

Es la distancia, medida desde el plano de la lente, a la que se encuentra el plano del espacio que permanece enfocado en el plano imagen ("l_o" en la figura 5.5).

Con referencia a la figura 5.5, si el plano imagen se desplaza ligeramente un $\delta 1_i$, el punto del espacio A_o ya no se proyecta en un único punto A_i, sino en un área de

tamaño δA_i denominada círculo de confusión. Esto origina el desenfoque del punto A_o. El efecto se produce si es el propio punto A_o, manteniendo fijo el plano imagen, el que se aleja o acerca del plano principal.

La distancia de enfoque 1_o y la distancia focal están relacionados mediante la siguiente expresión (ver figura 5.5):

$$\frac{1}{f} = \frac{1}{l_o} + \frac{1}{l_i}$$

3. Profundidad de campo

Determina la anchura de la zona enfocada, es decir, es el rango de distancia delante y detrás del objeto que parece estar enfocada en el plano imagen. La profundidad de campo depende de la resolución del sensor imagen, de la apertura del diafragma, y de la distancia de enfoque.

4. Apertura

Las ópticas van provistas de un sistema que regula el paso de los rayos luminosos a través de las lentes. Este sistema se denomina diafragma o iris. El más utilizado consiste en una serie de delgadas láminas montadas en anillo alrededor del objetivo. El mayor o menor diámetro del orificio se selecciona girando una rosca de mando, que abre o cierra las hojas.

Por último, destacar que podemos utilizar proyección perspectiva de la cámara para la transformación de las distancias de la imagen en distancias reales. Esta ecuación desarrollada y descompuesta en los ejes de coordenadas queda de la siguiente manera:

$$x = f\,\frac{X}{Z} \qquad\qquad y = f\,\frac{Y}{Z}$$

siendo las coordenadas en la imagen las variables (x,y) y las coordenadas reales las variables (X,Y). Además, "f" es la distancia focal y "Z" es la altura que se encuentra nuestra cámara.

Pero, esta representación de las distancias reales es realizada sobre una imagen completamente plana en la cual no afecta la curvatura de la lente para el cálculo de las distancias reales. Por tanto, debemos utilizar los parámetros intrínsecos de la óptica sobre estas ecuaciones para en correcto cálculo de las distancias reales, de tal forma que, ahora sí, conseguiremos un cálculo de las distancias reales teniendo en cuanta la curvatura de la lente. Por tanto, podemos concretar que la proyección perspectiva de la cámara atendiendo a los parámetros intrínsecos de la óptica, se describe de la siguiente manera.

$$x = -(x_{im} - o_x) * s_x \qquad\qquad y = -(y_{im} - o_y) * s_y$$

donde (o_x, o_y) son las coordenadas en pixel del centro de la imagen y (s_x, s_y) el tamaño de los pixel en milímetros en la dirección horizontal y vertical de la imagen respectivamente.

5.3 Pre-procesamiento de imágenes

Antes de poder realizar cualquier procesamiento de imágenes, debemos reconocer los diferentes tipos de imágenes además de la utilización de filtros para la eliminación del ruido.

A continuación, veremos una introducción a los tipos de imágenes y posteriormente la implementación matemática de los filtros más utilizados.

5.3.1 Tipos de imágenes

Como podemos observar en el libro *'Visión por Computador. Imágenes digitales y aplicaciones* [4], el uso del color en el procesamiento de imágenes está motivado por dos factores principales. En primer lugar el análisis de imágenes el color es un potente descriptor que a menudo simplifica la identificación y extracción de una escena. En segundo lugar, el ojo humano el ojo humano puede distinguir una amplia gama de colores comparado con los niveles de grises.

En el tratamiento de imágenes el procesamiento de las imágenes en color se divide en dos áreas fundamentales: color y pseudocolor. En la primera categoría se procesan las imágenes obtenidas con un sensor de color o multiespectral, mientras que en la segunda, las imágenes monocromas son coloreadas por asignación de un determinado color a un determinado nivel de intensidad.

Existen tres rezones fundamentales por las cuales los sistemas basados en color son menos utilizados: mayor volumen de datos (normalmente, 3 veces más), encarecimiento del sistema (tanto de la cámara como del digitalizador) y, porque en muchas ocasiones no aporta prácticamente ninguna información relevante.

Por tanto, podemos destacar dos tipos de imágenes de especial relevancia, las imágenes en color, normalmente RGB, y las imágenes monocromáticas, de la cuales destacan las imágenes en escala de grises. A continuación describiremos estos tipos de imágenes.

La representación RGB descompone la imagen color en tres imágenes correspondientes a las componentes de rojo (R), verde (G) y azul (B). Esta representación, esta habitualmente empleada en el mundo informático, corresponde a su vez con la percepción física del color por parte del ojo humano.

Por su parte, la imagen en escala de grises solo se descompone en una imagen de similar color (gris) donde se produce una variación de intensidad en un rango de 0 a 255 de valor.

5.3.2 Filtrado de la imagen

Las operaciones de suavizado tienen por objeto reducir efectos espurios que pueden presentarse en una imagen a consecuencia del proceso de captura, digitalización y transmisión. A continuación, veremos un resumen de los diferentes tipos de filtros. Podemos consultar más información sobre estos filtros en [3].

- **Filtro de la mediana**

Una de las dificultades principales del promediado del entorno es que se desdibujan los contornos y otros detalles de la imagen. Una alternativa es el uso del filtro de la mediana, que consiste en remplazar el nivel de gris de cada píxel por la mediana de los niveles de grises de los píxel vecinos.

Sea $\{a_1,a_2,\ldots,a_N\}$ una secuencia discreta de un número impar de valores ordenados crecientemente o decrecientemente. Se define la mediana de esta secuencia como el elemento $a_{(N-1)/2}$. Por ejemplo, si se consideran los 4-vecinos de "p" cuyas intensidades son 1, 2, 9, 4 y 5, el píxel "p" se remplazara por el valor 4, ya que éste es el valor central en la secuencia ordenada 1, 2, 4, 5 y 9.

Puede afirmarse que, en general, el comportamiento del filtro de la media es mejor que el del promediado del entorno para eliminar efectos espurios y para preservar los bordes de la imagen. Presenta, sin embargo, el inconveniente de que pierde detalles finos como líneas delgadas, puntos aislados y redondea las esquinas.

- **Filtro de Gauss**

Una distribución gaussiana con desviación típica σ y media μ viene dada por:

$$g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{3.1}$$

La convolución de la función $g_\sigma(x)$ dada en 3.1 con una señal f(x) da lugar a una nueva señal visualizada h(x), donde el valor en cada punto es el resultado de promediar con distintos pesos los valores vecinos a ambos lados de dicho punto. En este suavizado, la desviación típica σ juega un papel importante a la hora de controlar el grado de suavizado de este operador. Cuanto mayor sea σ, mas se tienen en cuenta los puntos lejanos, y, por tanto, se realizara un mayor suavizado. Conforme σ disminuye, los valores de los puntos vecinos van recibiendo menor ponderación, con lo que la reducción de ruido será menor.

Desarrollando la función 5.3 a dos dimensiones obtendremos:

$$g_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)} \tag{3.2}$$

donde se ha supuesto que la campana de Gauss se centra sobre la media.

En el caso discreto, la convolución de una imagen f (i,j) con equivalente discreto de la ecuación 3.2 viene determinada por la expresión:

$$F(i,j) = \frac{1}{2\pi\sigma^2}\sum_k\sum_l e^{-\frac{1}{2}\frac{k^2+l^2}{\sigma^2}} f(i+k,j+l) \tag{3.3}$$

Aunque este tipo de suavizado tiene el problema del suavizado de bordes, no es tan acusado como en otros tipos de filtrado tales como el caso de la mediana simple.

En la práctica, los sumatorios de 3.3 se limitan a un determinado entorno de velocidad de píxel analizado, lo que se traduce en la convolución de la imagen con una máscara cuadrada cuyo tamaño dependerá de la desviación típica σ escogida. A medida que aumenta el valor de σ, los términos de la función gaussiana alejados de la media son mayores y no deben, por tanto, ser despreciados.

Se han propuesto diferentes relaciones entre σ y el tamaño W. Una de las más comúnmente aceptadas es:

$$W \geq 3c$$

donde c = $2\sqrt{2\sigma}$ es el tamaño del lóbulo central de la gaussiana.

Como puede apreciarse, incluso desviaciones típicas pequeñas llevan a mascara de un tamaño enorme. Afortunadamente, el operador g (x,y) es separable por filas y columnas, esto es:

$$g_\sigma(x,y) = g_1(x) * g_2(y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} * \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y}{\sigma}\right)^2} \right)$$

De forma equivalente, en el caso discreto, un operador g (i,j) se dice que es separable si puede ponerse como:

$$g_\sigma(i,j) = g_1(i) * g_2(j)$$

siendo g_1 un vector de filas y g_2 un vector de columnas. De esta manera, el filtro gaussiano se aplica de dos maneras: en la primera se realiza una convolución por filas y al resultado obtenido se le aplica una convolución por columnas. Así se consigue reducir el número de operaciones N^2 a 2N. Es recomendable que a la hora de programar este filtro, realicemos la discretización de cada componente como un promediado entre tres valores contiguos de la función gaussiana, con lo cual, conseguiremos una mayor exactitud del operador.

5.4 Detección de bordes

Los bordes de una imagen se pueden definir como transiciones entre dos regiones de niveles de gris significativamente distintos. Éstos suministran una valiosa información sobre las fronteras de los objetos que puede ser utilizada para la segmentación de la imagen, reconocimiento de objetos, visión estéreo, etc.

Como se sabe, la derivada de una señal continua proporciona las variaciones locales con respecto a la variable, de tal forma que el valor de la derivada es mayor cuanto más rápidas son estas variaciones.

Considere una señal continua f(x) que representa un borde en una dimensión. Parece lógico considera que la localización de este borde venga dada por el punto de inflexión de f(x), es decir, por el máximo de la función f´(x). De esta manera la derivada de f(x) permite determinar "x_0" de una manera fácil, al tiempo que proporciona una estimación sobre la magnitud y sentido de la variación (creciente se f´(x) es positivo y, decreciente, en caso contrario).

En las funciones bidimensionales f(x,y), la derivada es un vector que apunta en la dirección de máxima variación de f(x,y) y cuyo módulo es proporcional a dicha variación. Este vector, denotado como ∇f(x,y), se denomina gradiente y se define como:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix} = \begin{bmatrix} f_x(x,y) \\ f_y(x,y) \end{bmatrix}$$

Obsérvese que el gradiente en un punto (x,y) viene dado por las derivadas de f(x,y) a lo largo de los ejes coordenados ortogonales "x" e "y". El módulo y dirección vienen dados por:

$$|\nabla f(x,y)| = \sqrt{(f_x(x,y))^2 + (f_y(x,y))^2}$$

$$\alpha(x,y) = \operatorname{atan}\left(\frac{f_y(x,y)}{f_x(x,y)}\right)$$

En este proyecto, debido a la utilización de un entorno de gran simplicidad, únicamente captaremos los bordes mediante el paso a binario de la imagen, ya que nuestro entorno se caracteriza por la diferenciación de dos colores. Uno de estos colores será el verde para la identificación del suelo y el otro será el blanco para la identificación de los obstáculos. Por tanto, mediante el paso a binario conseguiremos una imagen en la cual se distinguirá entre suelo (color negro) y obstáculos (color blanco), y nuestros bordes serán la frontera entre estos dos colores.

5.5 Detección de movimiento

Como podemos observar en el libro '*Visión por Computador*' [3], cualquier movimiento perceptible en la escena se traduce en la secuencia de imágenes tomadas de dicha escena, por lo cual, si tales cambios son detectados, se pueden analizar las características de este movimiento. En el caso general del movimiento tridimensional de un objeto, sólo será posible obtener estimaciones cualitativas de dicho movimiento. De hecho, cuando un objeto se mueve en la dirección de su línea de observación, el movimiento no es perceptible; aunque sí existen ciertos indicios de movimiento, como la variación de su tamaño en la imagen, el movimiento de la sombra arrojada, etc. Sin embargo, cuando el objeto se mueve en un plano paralelo al plano imagen se pueden obtener buenas estimaciones de las componentes del movimiento.

Una teoría utilizada para la detección de movimiento es la utilización de la imagen diferencia. Esta se basa en la resta de imágenes con la cual podremos calcular la cantidad de movimiento del objeto ya que para un único objeto en movimiento la diferencia entre imágenes será el movimiento de dicho objeto.

Por tanto, se define la imagen diferencia f_d como:

$$f_d(p, t_1, t_2) = f(p, t_2) - f(p, t_1) \qquad (5.1)$$

donde p = (x,y) es un pixel genérico de la imagen y t_1, t_2 son los instantes de tiempo de dos imágenes consecutivas. Nótese que los valores o intensidades que se obtienen mediante 5.1 pueden ser negativos.

Algunos puntos de interés a destacar en relación a la unidad de la imagen diferencia son:

1. Puede ser utilizada como una aproximación a la derivada de la función imagen. Esta aproximación de *df(p,t)/dt* se produce en el punto medio del intervalo t_2-t_1.
2. Cuando la imagen diferencia se obtiene a partir de dos imágenes ligeramente desplazadas, ésta muestra una imagen de bordes, como consecuencia de su analogía con un operador gradiente.

3. Segmentando la imagen diferencia puede estimarse la dirección del movimiento del objeto.

El aspecto más atractivo de esta técnica es su extremada simplicidad y que se presta a una cómoda implantación paralela. La información que proporciona, sin embargo, no es demasiado descriptiva sobre la forma o movimiento de los objetos, aunque si permite detectar la zona de la imagen donde se están produciendo cambios, pudiéndose así concentrar el posterior esfuerzo computacional en el área detectada.

5.6 Flujo Óptico

El flujo óptico juega un papel importante en la estimación y descripción del movimiento, por lo cual es comúnmente utilizado en tareas de detección, segmentación y seguimiento de objetos móviles en una escena a partir de un conjunto de imágenes. En este trabajo se seleccionó una estrategia de flujo óptico, para estimar el movimiento relativo de la imagen.

A continuación se describe el principio de flujo óptico utilizado en nuestro proyecto, en particular la técnica de flujo óptico constante.

Como podemos observar en '*Introductory Techniques for 3-D Computer Vision*' [5], se describe el flujo óptico como el campo vectorial sujeto a la condición de la ecuación de brillo constante y definido como el movimiento aparente del patrón de brillo de la imagen. Antes de continuar con la explicación del flujo óptico, debemos detenernos en la comprensión de la ecuación de brillo constante. Esta viene definida por:

$$(\nabla E)^T * v + E_t = 0 \qquad (6.1)$$

Como observamos en la ecuación, para una imagen $E = E(x,y,t)$, y un vector v de campo de movimiento; la suma del producto del gradiente de la imagen por el vector representativo del movimiento, más la imagen tomada dependiente del tiempo E_t ha de ser igual a cero para un entorno sin cambio de brillo, es decir, brillo constante.

Por tanto, el flujo óptico es la aproximación del campo de movimiento que puede ser calculado a partir de secuencias de imágenes variables en el tiempo. Siempre y cuando asumamos las siguientes condiciones:

- Superficies Lambertianas: Una superficie perfectamente difusora es aquella que emite o refleja el flujo luminoso en forma tal que ésta presenta la misma luminancia independientemente del ángulo de visión. Tal superficie se denomina lambertiana porque responde a la ley de Lambert.

- Fuente puntual de luz en el infinito: Hace referencia a la fuente de luz utilizada situando esta a larga distancia respecto a nuestra zona de captación de movimiento.

- Sin distorsión fotométrica: La distorsión fotométrica representa la variación de contraste entre imágenes, por tanto para la correcta utilización del flujo óptico deberemos tener un entorno en el cual la variación de contraste en el tiempo se nula.

Además, debemos tener en cuenta los siguientes errores debidos a la aproximación:

- Error en pequeños puntos con un alto gradiente espacial.

- La ecuación de brillo constante será exactamente cero sólo para el movimiento de traslación o por cualquier movimiento rígido de tal manera que la dirección de iluminación sea paralela a la velocidad angular.

Por tanto, para la implementación de nuestro algoritmo de visión deberemos tomar los siguientes supuestos.

1. La ecuación de brillo constante produce una buena aproximación de la componente normal del campo de movimiento.
2. El campo de movimiento se aproxima bien al campo vector constante dentro de cualquier pequeña porción del plano de imagen.

Tomando el supuesto 1, para cada punto p_i dentro de una región de pequeñas *Q*, de dimensiones *N x N*, puede escribirse como la ecuación 6.1; donde las derivadas espaciales y temporales del brillo de la imagen se calculan en p_1, $p_2 ... p_{NxN}$. Típicamente se utiliza una pequeña región de 5x5.

Por lo tanto, el flujo óptico se puede estimar con Q como el vector constante, *v*, que minimiza la funcionalidad de la siguiente ecuación:

$$\Psi[v] = \sum_{p_i \in Q} [(\nabla E)^T v + E_t]^2 \tag{6.2}$$

La solución a este problema de mínimos cuadrados puede ser determinada mediante el paso a un del sistema lineal, esto lo haremos de la siguiente manera:

$$A^T A\, v = A^T b \tag{6.3}$$

La fila *i-enésima* de la matriz A $(N^2 x2)$ es el gradiente de la imagen espacial evaluado en punto p_i,

$$A = \begin{bmatrix} \nabla E(p_1) \\ \nabla E(p_2) \\ \vdots \\ \nabla E(p_{N\,x\,N}) \end{bmatrix} \tag{6.4}$$

y b es la dimensión N^2 del vector de las derivadas parciales temporales del brillo de la imagen, evaluada en $p_1, \ldots p_{NxN}$, después de un cambio de signo:

$$b = -[E_t(p_1), \ldots, E_t(p_{NxN})]^T \tag{6.5}$$

La solución de mínimos cuadrados del sistema a través de restricciones 6.3 se puede obtener como

$$v = (A^T A)^{-1} A^T b \tag{6.6}$$

donde *v* es el flujo óptico (la estimación del campo de movimiento) en el centro de la región *Q*; repitiendo este proceso para todos los puntos de la imagen, conseguiremos un flujo óptico completo.

Por último, es conveniente describir algunas de las utilidades del flujo óptico usadas en el mundo de la robótica.

Como podemos observar en Coombs & Roberts [6], una de las utilidades del flujo óptico es el centrado de un robot mediante la visión periférica. Esto se basa en la utilización del flujo óptico máximo observado a la izquierda y derecha del campo visual periférico, para indicar la proximidad de los obstáculos. Si el flujo óptico máximo es superior en la derecha, los objetos están más próximos por este lado y por tanto, deberemos girar nuestro robot hacia la derecha con el fin de evitar la colisión con el obstáculo.

En la figura 5.6, podemos observar la trayectoria de un robot utilizando este sistema de centrado en función del flujo óptico máximo a ambos lados.

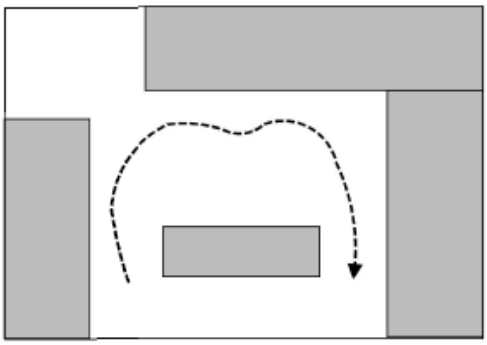


Figura 5.6: Trayectoria de un robot utilizando el centrado en función del flujo óptico captado a ambos lados.

Cabe destacar que este sistema de visión está inspirado en la visión utilizada por los insectos. El principio consiste sencillamente en que si el insecto vuela describiendo una línea recta, los objetos próximos parecerían pasar mucho más rápido en el ojo que los objetos que están alejados. De ese modo, la distancia a un objeto puede deducirse a partir de la velocidad de su imagen en el ojo: cuanto mayor es la velocidad, más cerca se encuentra el objeto.

Otra utilización del flujo óptico, es la utilizada por Souhila & Karim [7], en la cual su utiliza el flujo óptico para evitar colisiones con los obstáculos. Esto se realiza a través del cálculo del tiempo de contacto, que es el tiempo que tardamos en colisionar con un obstáculo que se encuentra en nuestra dirección en función de la velocidad de nuestro robot.

Para determinar el tiempo para el contacto se debe primeramente conseguir las coordenadas del foco de expansión (FOE) del campo de flujo óptico, lo cual significa el punto en la imagen a partir del cual todos los vectores emergen. Una vez calculado el FOE, podemos describir el tiempo de contacto (TTC) como:

$$TTC = \frac{A_i}{|V_t|}$$

donde A_i es la distancia del punto considerado (x_i, y_i) en la imagen respecto del FOE y *Vt* es la velocidad de traslación de la imagen.

Este algoritmo de visión nos puede ser de gran ayuda en este proyecto a la hora de realizar una detección dinámica de los obstáculos con el fin de no colisionar con ellos.

6 Implementación de los algoritmos de visión en Matlab

6.1 Introducción a Matlab

MATLAB es un entorno de cálculo técnico de altas prestaciones para cálculo numérico y visualización, del cual podemos ampliar nuestra información accediendo a la página web de MathWorks [8].

Este entorno de cálculo integra entre sus características principales la capacidad de realizar análisis numérico, cálculo matricial, procesamiento de señales y gráficos; en un entorno fácil de usar, donde los problemas y las soluciones son expresados como se escriben matemáticamente, sin la programación tradicional.

MATLAB fue escrito originalmente para proporcionar un acceso sencillo al software matricial utilizando un sistema interactivo cuyo elemento básico de datos es una matriz que no requiere dimensionamiento. Esto permite resolver muchos problemas numéricos en una fracción del tiempo que llevaría hacerlo en lenguajes como C, BASIC o FORTRAN.

MATLAB ha evolucionado en los últimos años a partir de la colaboración de muchos usuarios. En entornos universitarios se ha convertido en la herramienta de enseñanza estándar para cursos de introducción en álgebra lineal aplicada, así como cursos avanzados en otras áreas. En la industria, MATLAB se utiliza para investigación y para resolver problemas prácticos de ingeniería y matemáticas, con un gran énfasis en aplicaciones de control y procesamiento de señales. MATLAB también proporciona una serie de soluciones específicas denominadas TOOLBOXES. Estas son muy importantes para la mayoría de los usuarios de MATLAB y son conjuntos de funciones MATLAB que extienden el entorno MATLAB para resolver clases particulares de problemas como el procesamiento de señales, diseño de sistemas de control, simulación de sistemas dinámicos e identificación de sistemas.

Probablemente la característica más importante de MATLAB es su capacidad de crecimiento. Esto permite convertir al usuario en un autor contribuyente, creando sus propias aplicaciones.

En resumen, las prestaciones más importantes de MATLAB son:

- Escritura del programa en lenguaje matemático.
- Implementación de las matrices como elemento básico del lenguaje, lo que permite una gran reducción del código, al no necesitar implementar el cálculo matricial.
- Implementación de aritmética compleja.
- Un gran contenido de órdenes específicas, agrupadas en TOOLBOXES.
- Posibilidad de ampliar y adaptar el lenguaje, mediante ficheros de script y funciones .m.

6.2 Implementación del algoritmo de campo de visión en Matlab

La utilización del campo de visión para determinar las características de un entorno es de especial relevancia, sobretodo en este proyecto, donde la utilización de un sistema de visión omnidireccional nos dará una especial ventaja en cuanto a la detección de objetos se refiere. Esto se debe a que nuestro robot camina sobre el mismo plano al que está enfocada la cámara y por tanto, es más fácil de definir la dirección en la que se encuentra el objeto. A continuación, describiremos el código de detección de obstáculos implementado en Matlab, el cual podemos consultar en el ANEXO II.

Comenzamos con la lectura de una imagen de tamaño '320 x 240'. En ésta imagen podemos apreciar dos áreas; una interior de color verde y otra exterior de color blanco. Con ésta imagen (ver figura 6.1), intentamos simular la captura en imagen de un entorno en el cual el suelo es de color verde y los objetos que nos rodean son de color blanco.

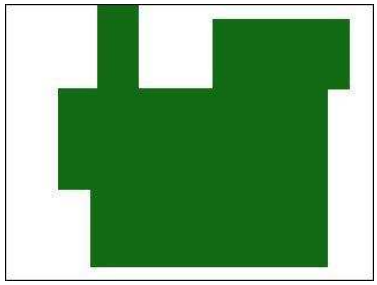


Figura 6.1: Imagen de simulación de un entorno ideal (A).

Además de cargar la imagen, la pasaremos a binario mediante la utilización de un umbral, con lo cual conseguiremos una imagen de color negro en las zonas más oscuras y blanco en las zonas con más brillo. Utilizando la función "*im2bw*" pasaremos la imagen A a binario. Esta función crea un umbral automáticamente en función de las características de la imagen. Sera interesante ver el funcionamiento de este umbral en un entorno real, ya que para un entorno ideal como el que estamos utilizando, no tiene problemas de ruido. La resultante del paso a binario de la imagen *A* podemos observarla en la figura 6.2.

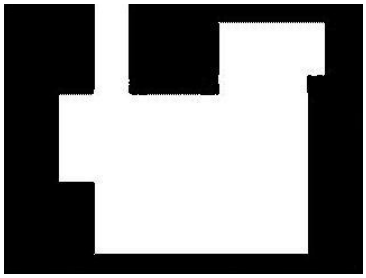


Figura 6.2: Resultante del paso a binario de la imagen A.

Una vez diferenciadas las zonas de nuestra imagen, es momento de calcular las distancias a los objetos que nos rodean. Esto lo conseguiremos mediante la utilización de rectas con origen en el centro de la imagen y separadas de un ángulo θ constante entre ellas. Por tanto, lo primero que debemos calcular serán los ángulos de cada recta, los cuales vienen definidos por la siguiente ecuación,

$$Ang(n) = \frac{2 * pi}{n_{total}} * n$$

donde $n = 1,2,\ldots n_{total}$ es el numero de rectas que queremos utilizar para el cálculo de distancias. Aplicando esta ecuación en Matlab conseguiremos un vector con todos los ángulos de nuestras rectas.

Atendiendo a la ecuación general de una recta $y = ax + b$ donde b es la y_0 inicial, para nuestro caso '0', y donde a es la pendiente de la recta definida para nuestro caso como $a = tan\,(Ang(n))$, por tanto, podemos determinar que

$$y = x \tan\,(Ang(n))$$

Utilizando esta ecuación, recorreremos los valores de x por la recta de ángulo n para conseguir un valor y. Un vez calculados x e y, comprobamos el valor de la imagen en binario para el pixel (x,y). Si el valor del pixel (x,y) es igual a '0' significa que no hay objetos y por tanto no será necesario calcular la distancia, pero si el valor del pixel (x,y) es igual a '1', significa que hemos encontrado objeto y por tanto ya no es necesario seguir recorriendo la recta. Para calcular la distancia al objeto debemos calcular el módulo del vector generado por x e y, el cual podemos determinar como

$$Dist(n) = \sqrt{x^2 + y^2}$$

siendo *Dist(n)* la distancia al objeto para la recta n. Repitiendo el proceso para todas las rectas conseguiremos un mapeo de distancias de todo nuestro entorno. Este mapeo de todas las distancias será guardado en un vector *Dist(n)* el cual contendrá todas las distancias calculadas en función de la recta utilizada.

En la figura 6.3, podemos observar la representación de los vectores *Dist(n)* con origen en el centro de la imagen.

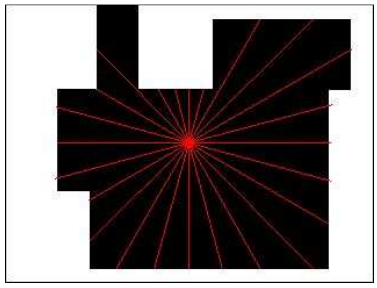


Figura 6.3: Representación del cálculo de distancias.

Por último, debemos pasar el valor de las distancias en la imagen al valor de las distancias real. Esto los realizaremos mediante la aplicación de la proyección perspectiva para los parámetros intrínsecos de nuestra cámara.

Finalmente, podemos determinar el tiempo de ejecución de nuestro algoritmo de cálculo de distancias. Para un procesador Intel Core Duo de 1,66GHz el tiempo de ejecución de este algoritmo para estas imágenes ha sido de 1,33segundos.

A continuación, en la figura 6.4, podemos observar el flujograma del código de visión para el cálculo de distancias descrito anteriormente.

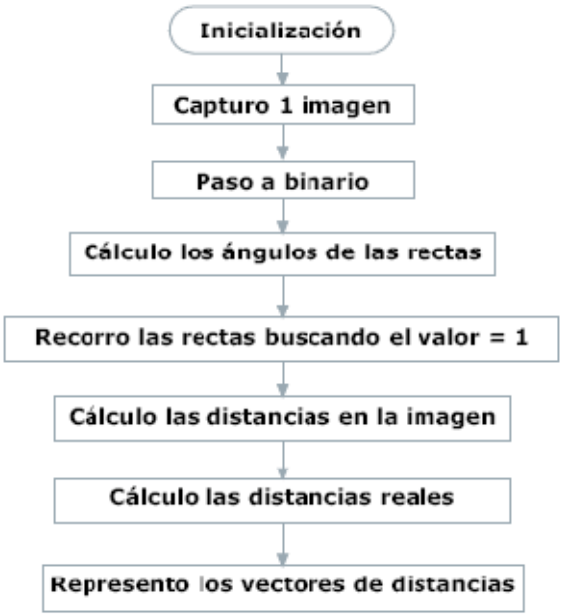


Figura 6.4: Flujograma del código de campo de visión.

6.3 Implementación del algoritmo de detección de movimiento en Matlab

En este apartado nos dedicaremos a la explicación del código generado en Matlab para la implementación del algoritmo de visión de detección de movimiento. Podemos consultar este código en el ANEXO III.

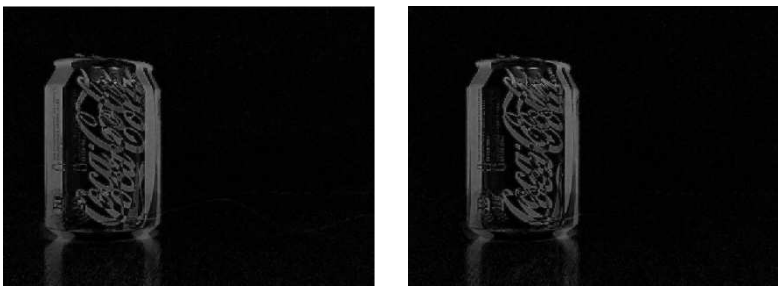
En primer lugar, capturaremos tres imágenes (ver figura 6.5) con un intervalo de tiempo finito y de corto valor. Estas imágenes las almacenamos en matrices RGB, una por cada imagen tomada y tendrá un tamaño similar a la imagen '320 x 240'.

(1) (2) (3)

Figura 6.5: Secuencia de imágenes capturadas

Mediante la función 'rgb2gray' pasamos cada una de las imágenes a escala de grises obteniendo por cada imagen una matriz de tamaño '320 x 240', compuesta por valores del rango 0 a 255.

A continuación, restaremos las imágenes en el orden que han sido tomadas, es decir, restaremos la primera imagen con la segunda y posteriormente restaremos la segunda imagen con la tercera (ver figura 6.6). Esto nos proporcionará dos imágenes en escala de grises. Dichas imágenes nos muestran únicamente el movimiento captado por la secuencia de imágenes, siendo más blancas las zonas con mucho movimiento y negras las zonas que carecen de movimiento.



(a) (b)

Figura 6.6: Resultado de la resta de imágenes: a) Imagen 1 y 2; b) Imagen 2 y 3

En éste punto es conveniente el filtrado de las imágenes ya que existe ruido debido a los cambios en la iluminación, posición, etc.

Mediante la función 'spatial-gaussian-filter', implantaremos el filtro gaussiano de la imagen con el cual conseguiremos una imagen más suavizada y con menos pixel de movimiento erróneos. En la figura 6.7 podemos apreciar el efecto que ejerce el filtro en nuestra imagen.

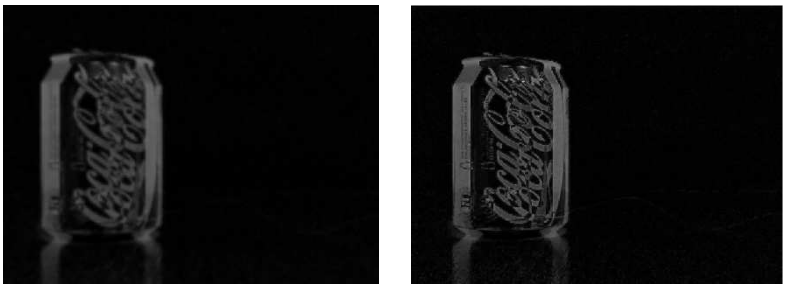


Imagen filtrada Imagen original

Figura 6.7: Comparativa entre la imagen filtrada y la imagen original.

Una vez filtrada las imágenes, necesitamos pasarlas a binario. Dado que tenemos las imágenes en escala de grises de 0 a 255, necesitaremos crear un umbral que será la referencia para saber si cada pixel de la imagen pasa a ser '1 o 0'. Utilizando la función 'graythresh' calcularemos el umbral de nuestras imágenes. Una vez calculado nuestro umbral, pasaremos la imagen a binario mediante la función 'im2bw'. En éste momento, tendremos dos imágenes con áreas blancas sobre fondo negro. Estas áreas blancas nos muestran las zonas de la imagen dónde hay movimiento, pero para un único objeto nos puede mostrar varias áreas, de tal manera, que deberemos identificar todas las regiones de la imagen y determinar sus propiedades, como conjunto.

(A) (B)

Figura 6.8: Resultados del paso a binario de las imágenes (a) y (b)

Mediante la función 'bwlabel', a la cual introducimos la imagen en binario, conseguiremos identificar todas las regiones de la imagen que tienen elementos conectados, devolviéndonos el número de regiones encontradas y una matriz *L* del tamaño de la imagen. En esta matriz, cada uno de los elementos pertenecientes a una misma región tienen el mismo valor, siendo éste distinto para cada una de las regiones.

Utilizando la función 'regionprops' sobre la matriz *L* podemos determinar las principales propiedades de cada una de nuestras regiones, tales como; el área, el centroide, el perímetro, etc. Guardaremos estas propiedades para cada una de nuestras imágenes.

Uno de los puntos más importantes en la captación de un elemento en movimiento es concretar la dirección, sentido y velocidad del movimiento. Para esto necesitaremos saber la posición del centroide del movimiento captado.

Como ya hemos mencionado anteriormente, para un único objeto en movimiento podemos captar varias regiones de movimiento, por tanto, necesitaremos saber el centroide de todo el movimiento captado. Este centroide ha de ser proporcional al área de cada región de movimiento dentro del conjunto total del movimiento captado, es

decir, para una región determinada su aporte al centroide total será proporcional al tamaño de su área respecto al área total del movimiento captado. Este algoritmo se realiza aplicando la siguiente fórmula:

$$Centroide_{TOTAL}(x,y) = \sum_{i=1}^{n} \left(\frac{Area(i)}{Area_{TOTAL}} \right) * Centroide(i)$$

Por tanto, lo primero que debemos calcular será el área total de las regiones de movimiento captado, esto lo haremos creando un sumatorio de todas las áreas de todas nuestras regiones.

Posteriormente, implementaremos la fórmula anteriormente mostrada para cada una de las dos imágenes. Esto nos creará dos *centroides (x,y),* uno para cada imagen del conjunto de todas las regiones de movimiento captado. Con estos dos puntos formaremos un vector el cual nos aportará toda la información necesaria para poder concretar el sentido, la dirección y velocidad del objeto en movimiento. A continuación, en la figura 6.9, podemos observar este vector representado sobre la imagen original.

Figura 6.9: Representación del vector movimiento sobre la última imagen.

La velocidad del movimiento, no será una velocidad real, ya que únicamente utilizamos dos dimensiones, pero si conseguiremos concretar la velocidad del movimiento respecto a nuestro campo visual, es decir, a la velocidad que se mueve nuestro objeto por la imagen, independientemente de la distancia que haya entre el objeto en movimiento y la cámara. Por tanto, siempre y cuando el robot se mantenga estático, podremos determinar dónde estará el objeto en un corto espacio de tiempo.

Finalmente, podemos determinar el tiempo de ejecución de nuestro algoritmo de detección de movimiento. Para un procesador Intel Core Duo de 1,66GHz el tiempo de ejecución de este algoritmo y para estas imágenes ha sido de 1,99 segundos.

Por último, podemos observar en la figura 6.10, en la cual se muestra el flujograma del código de detección de movimiento, con el fin comprender de una forma más grafica el funcionamiento de este.

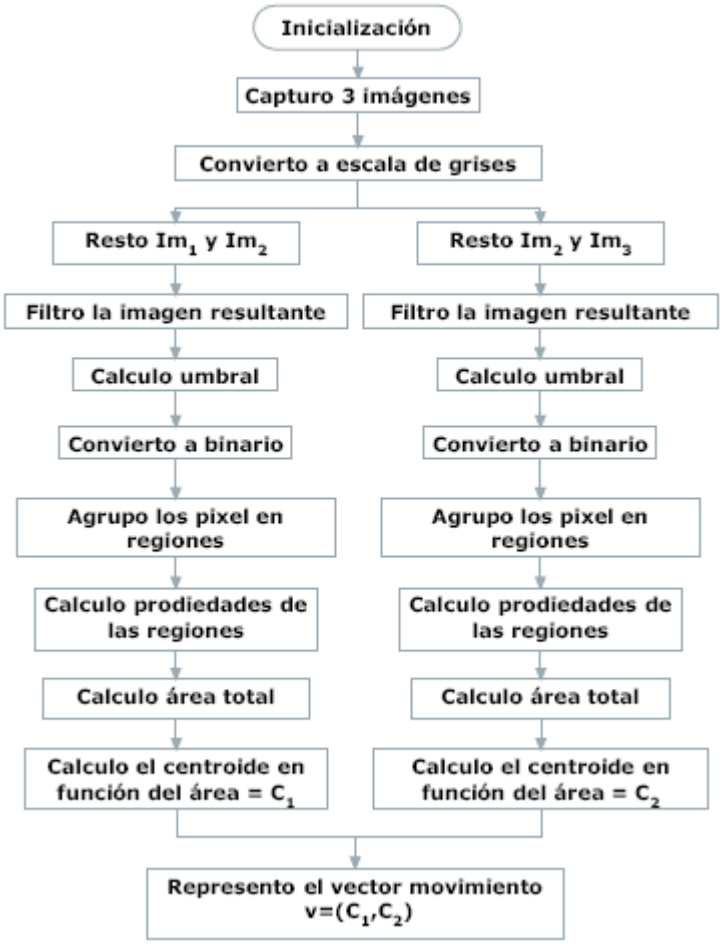


Figura 6.10: Flujograma del código de detección de movimiento.

6.4 Implementación del algoritmo de flujo óptico en Matlab

El programa desarrollado en Matlab, el cual podemos consultar en el ANEXO IV, comienza con el reseteo de todas las variables y con la posterior inicialización de éstas. Una vez realizada la inicialización, comenzamos con la captura de una secuencia de imágenes, realizando una serie de cinco capturas con una diferencia temporal entre ellas. Estas imágenes serán tomadas en escala de grises. Como podemos observar en la figura 6.11, cada imagen debe ser tomada aproximándonos cada vez más adelantada, ya que de no ser así, todas las imágenes serán iguales y no tendremos flujo óptico.

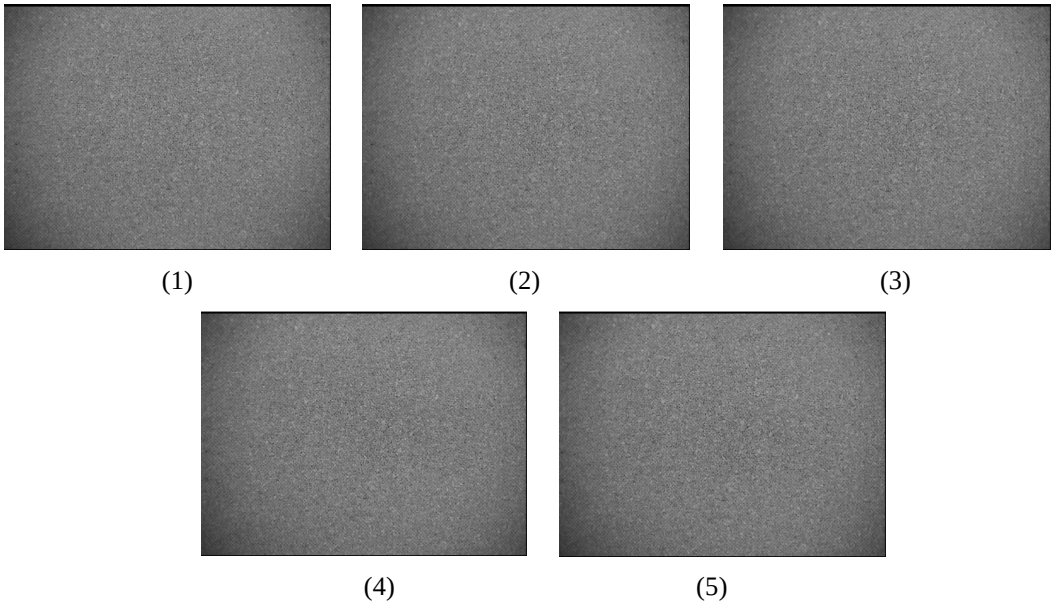


Figura 6.11: Secuencia de 5 imágenes en escala de grises.

Aunque en la figura anterior no se aprecia bien el acercamiento hacia el objeto a capturar, podemos observar la figura 6.12, la cual nos da una situación de cómo hemos realizado la secuencia de imágenes anteriores.

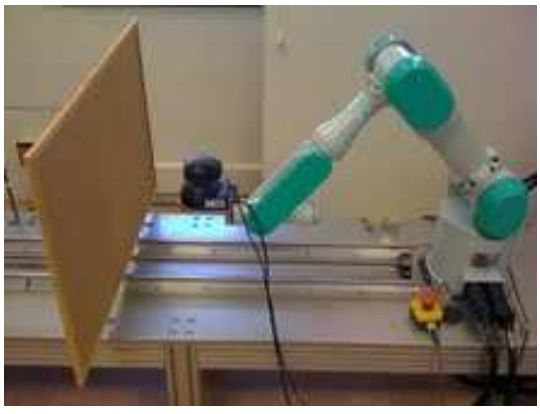


Figura 6.12: Entorno ideal de captura de imágenes para flujo óptico.

Una vez tomada la secuencia de imágenes, es momento para el filtrado de las imágenes. Comenzaremos con el filtrado espacial, el cual se realizara sobre cada imagen con el fin de equiparar cada pixel de la imagen con los pixel de alrededor. Para realizar este filtrado utilizamos la función '*spatian_gaussian_filter*', la cual tendrá como entrada cada imagen de nuestra secuencia además de otros valores como

la desviación típica (normalmente 1,5 pixel) y el valor de mascara (*maskWidthSpatial = 9).* Finalmente obtendremos la secuencia de imágenes con el filtrado ya realizado, el que podemos observar en la figura 6.13.

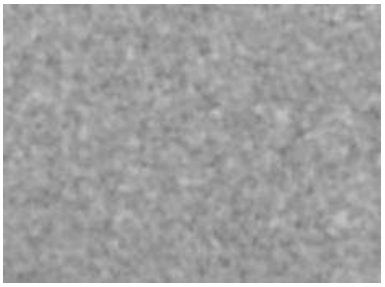


Imagen filtrada

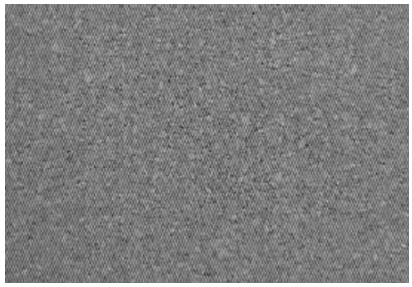


Imagen original

Figura 6.13: Comparativa entre imagen filtrada espacialmente e imagen original

A continuación, necesitamos realizar el filtrado temporal de imágenes. Sobre las imágenes filtradas espacialmente, ejecutaremos un filtrado temporal para que la diferencia entre el mismo pixel de diferente imagen no sea muy elevada. Utilizando de nuevo la función *'spatian_gaussian_filter'*, a la cual le reportaremos las imágenes filtradas anteriormente en grupos de tres imágenes, es decir, filtraremos temporalmente las imágenes 1, 2 y 3; luego las imágenes 2, 3 y 4 y por último las imágenes 3, 4 y 5. Esta es la forma de filtrar imágenes con la desviación típica de 1,5 imágenes, la cual podemos observar sus resultados en la figura 6.14.

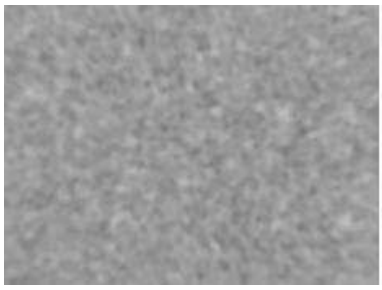


Imagen con filtrado temporal.

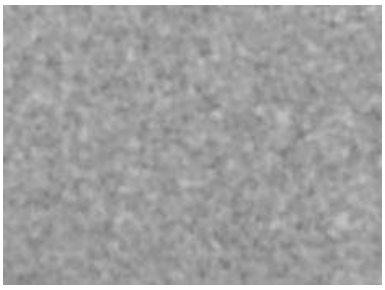


Imagen con filtrado espacial.

Figura 6.14: Comparativa entre imagen filtrada temporalmente e imagen filtrada espacialmente.

Una vez filtradas las imágenes, es el momento de aplicar el algoritmo flujo óptico constante sobre ellas. Comenzaremos con el seccionamiento de cada imagen en regiones de *NxN* pixel (típicamente *N = 5*) sobre las cuales aplicaremos la ecuación de flujo óptico constante. Mediante la utilización de un bucle recorreremos los pixel de cada imagen, para los cuales, formaremos una región de *NxN* donde el pixel en el que estamos situados sea nuestro pixel central. Cada vez que definamos una región es conveniente aplicar la ecuación de flujo óptico constante sobre ella, ya que, de no ser así, deberíamos guardar todas las regiones de la imagen y nos supondría una mayor carga computacional. Utilizando la función *"constant_flow_patch"* sobre cada región de la imagen conseguiremos el valor del flujo óptico sobre esa región, o lo que es lo mismo, la cantidad de movimiento que se encuentra en esa región. Además, conseguiremos la dirección del movimiento captado. Por tanto, guardaremos las posiciones de las regiones en una matriz además de los valores de flujo óptico captado para cada región.

En este momento, tenemos una matriz con todo el movimiento captado en las regiones de la imagen, pero esta matriz es demasiado grande para ser representada ya que tenemos un valor por cada pixel de la imagen. Es conveniente la segmentación de la matriz en regiones mayores y por lo tanto visibles para su representación. Esto lo realizaremos agrupando los valores de flujo óptico en regiones y calculando la media de cada región, de esta forma, como podemos apreciar en la figura 6.15, conseguiremos apreciar la cantidad de movimiento captado para en cada zona de la imagen.

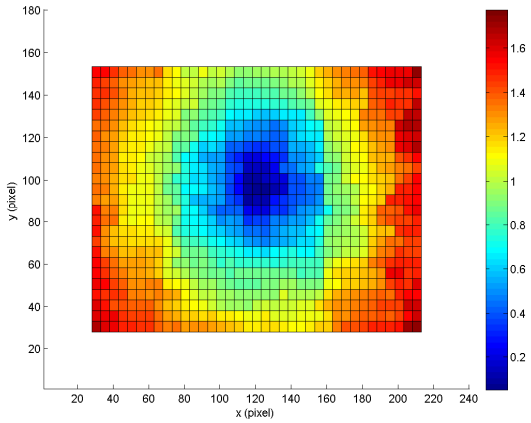


Figura 6.15: Representación del valor flujo óptico en cada región.

Además podemos representar el flujo óptico con su módulo y dirección. Haciendo una representación vectorial sobre cada zona de la imagen con su correspondiente promediado conseguiremos visualizar el movimiento captado en módulo y dirección, como se aprecia en la figura 6.16.

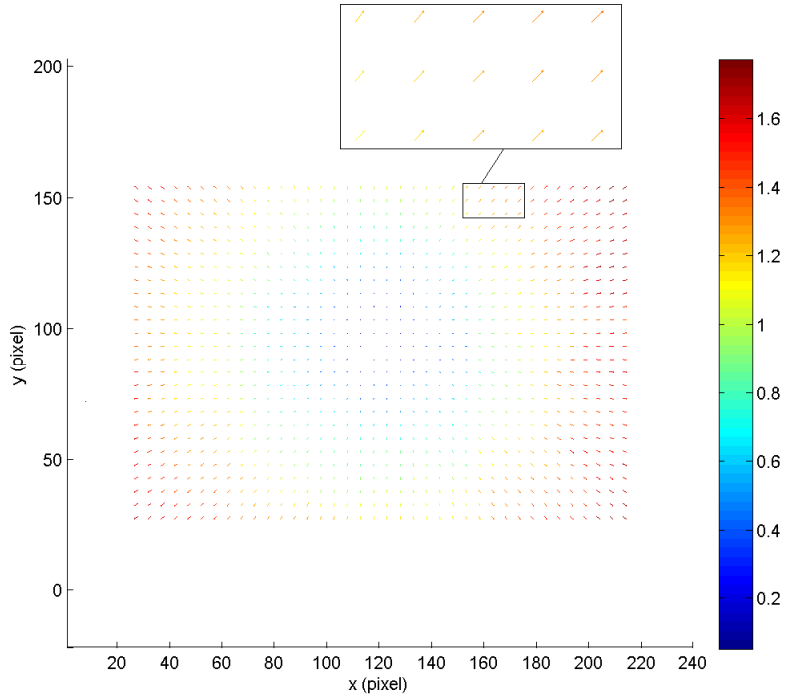


Figura 6.16: Representación del valor y la dirección del flujo óptico en cada región.

A continuación, en la figura 6.17, podemos observar el flujograma del código de flujo óptico, donde se muestra de una manera más simple los pasos necesarios para el desarrollo de este programa:

Figura 6.17: Flujograma del código de flujo óptico.

Finalmente, podemos determinar el tiempo de ejecución de nuestro algoritmo de flujo óptico. Para un procesador Intel Core Duo de 1,66GHz el tiempo de ejecución de este algoritmo y para estas imágenes ha sido de 2,37 segundos.

7 Pruebas en un entorno real similar al utilizado en CEABOT

En este punto, nos centraremos en probar los códigos anteriormente generados en Matlab en un entorno real similar al utilizado en CEABOT. En la figura 7.1, se muestra como hemos realizado la simulación de este entorno.

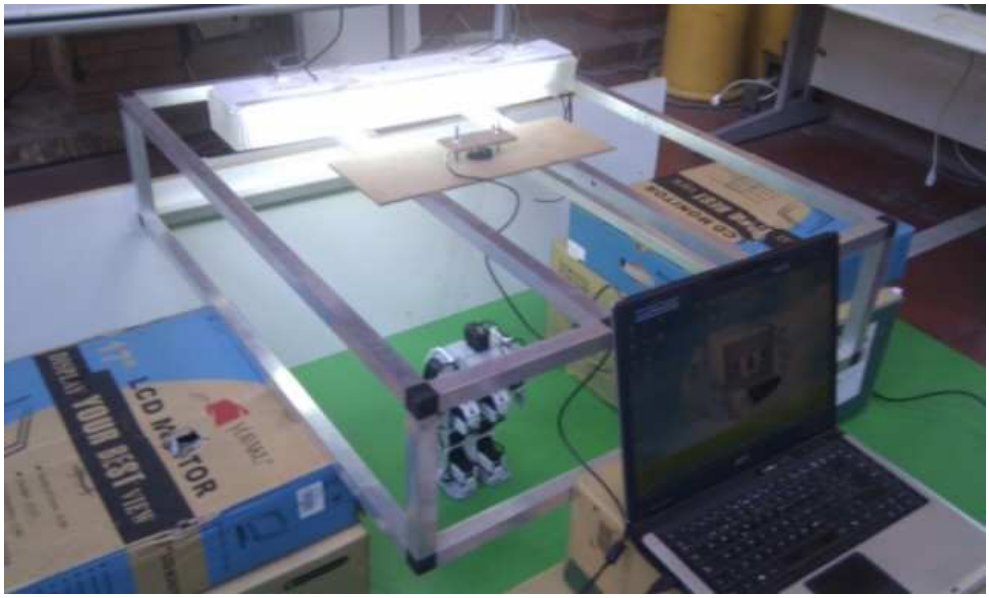


Figura 7.1: Entorno de simulación del campo utilizado en CEABOT.

Como podemos apreciar, para la simulación de este entorno hemos necesitado la utilización de una cámara web conectada a nuestro ordenador con la cual realizaremos la captura de imágenes necesaria para la realización de estas pruebas. Cabe destacar, que esta cámara tiene un ángulo de visión reducido e inferior al de la Blackfin. Por tanto, hemos necesitado aumentar la distancia entre el objetivo y el suelo para obtener un campo de visión adecuado para el ensayo de las pruebas. En la siguiente figura, podemos observar la captura realizada al robot desde nuestra posición omnidireccional.

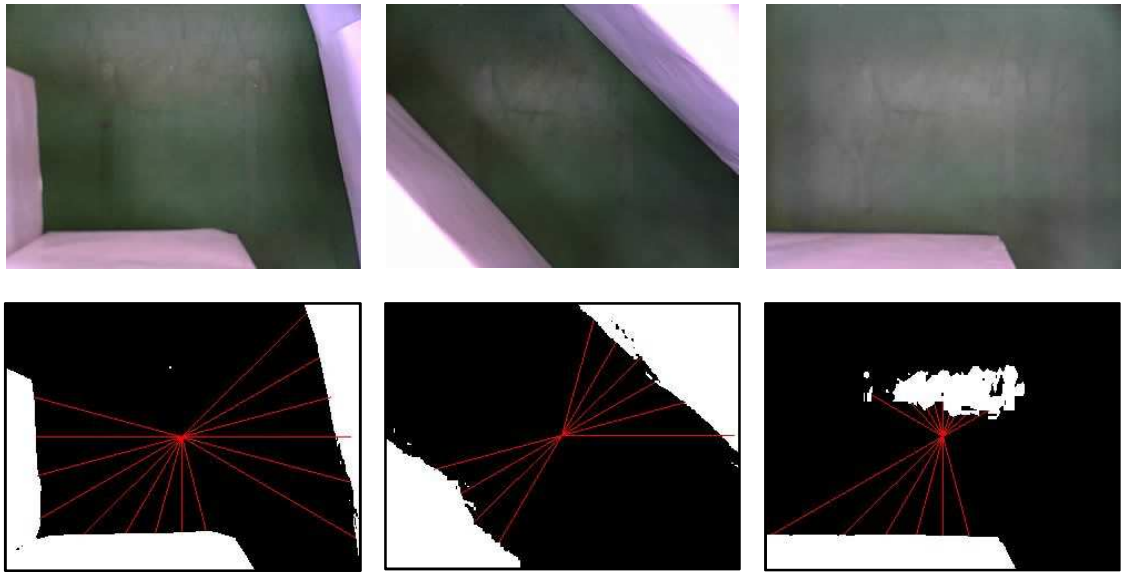
Figura 7.2: Captura de una imagen para un entorno omnidireccional

Además, hemos instalado la cámara web sobre una plataforma plana, de tal forma que se pueda desplazar paralelamente al suelo sobre las barras centrales de la estructura que lo soporta. Esto resulta de vital importancia para los ensayos de flujo óptico ya que este algoritmo es muy sensible a cambios en la dirección del movimiento.

El campo está iluminado con luz artificial de interior, debido a que es lo más uniforme posible. Es muy importante prestar atención a cómo afecta la iluminación a nuestros algoritmos de visión ya que es uno de los factores que más puede variar.

7.1 Pruebas del algoritmo de campo de visión

En esta prueba, nos dedicaremos a la investigación del comportamiento de nuestro algoritmo de cálculo de distancias en un entorno real. Para ello, necesitamos de la utilización de obstáculos de color blanco que colocaremos en diferentes posiciones. A continuación, podemos observar las imágenes utilizadas en este ensayo con sus respectivos resultados, los cuales comentaremos más tarde.



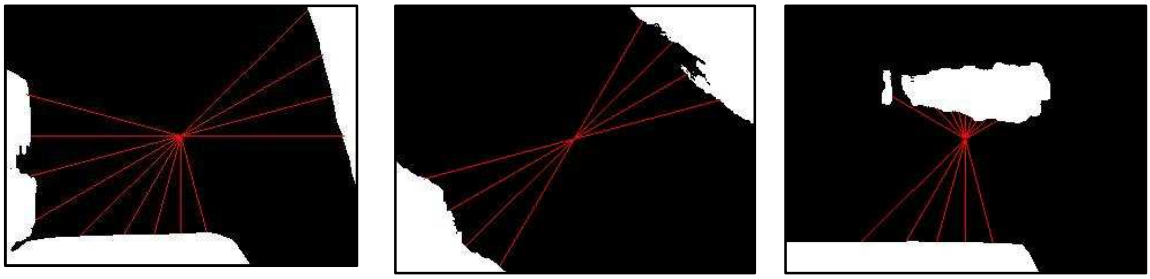
(A) (B) (C)

Figura 7.3: Comparativa entre los resultados de código de campo de visión para distintos casos.

Observando las tres imágenes podemos comprobar que el algoritmo de cálculo de distancias no funciona bien en todos nuestros casos. Podemos comprobar que en las imágenes A y B el algoritmo funciona correctamente, aunque no se concrete correctamente la frontera entre el suelo y los obstáculos. Esto se debe al paso de la imagen a binario de manera automática, es decir, a la creación automática de un umbral en función de los valores totales de la imagen. Fijándonos en la imagen C, vemos más claramente el error anteriormente citado ya que, al ser una imagen más iluminada, el umbral de paso a binario es más alto, y por tanto, habrá ciertos pixel más iluminados que produzcan este error. Este error, puede ser a causa de la poca distancia entre el foco de luz y el suelo (ver figura 7.1) la cual, nos produce reflejos que si estuviésemos en un entorno real no serian tan pronunciados ya que, cuanto mayor sea la distancia entre el foco de luz y el suelo, mayor será la dispersión de esta. De todas formas, solventar este problema nos será de gran ayuda para dar robustez a nuestro algoritmo de cálculo de distancias.

Es conveniente que cambiemos nuestro código para la realización de un filtrado espacial anterior al paso a binario de la imagen. Con este filtrado trataremos de suavizar la imagen con el fin de que el paso a binario de la imagen se realice de una manera más fina y con menos pixel dispersos.

Como en otros códigos, utilizaremos el filtro de Gauss para el filtrado espacial de las imágenes, utilizando una desviación típica de 1,5 pixel y una máscara de valor igual a 9. A continuación, en la figura 7.4, podemos observar los resultados del algoritmo de cálculo de distancias con la aplicación del filtro de Gauss sobre las imágenes A, B y C.



(A) (B) (C)

Figura 7.4: Resultados del cálculo de distancias utilizando filtro de Gauss.

Observando los resultados, comprobamos que el filtro de Gauss no nos es de gran ayuda a la hora de enfrentarnos al error que sucede en la imagen C. Si podemos visualizar unas fronteras más suavizadas y homogéneas aunque se siguen produciendo errores en algunas partes, como podemos observar en las imágenes A y B.

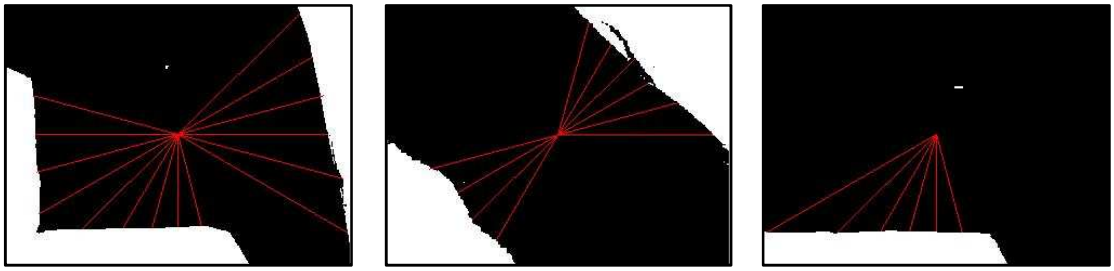
Prestando atención a la imagen B y comparándola con la imagen original (ver figura 7.3), observamos que la frontera real entre el suelo y los obstáculos se encuentra más cercana al centro de la imagen que lo que nos muestra el resultado del cálculo de distancias. Esto se debe a que los obstáculos en su zona baja tienen menos iluminación que en la parte superior y, por tanto, al pasar a binario desde una imagen en escala de grises obtenemos resultados erróneos.

Este error, es similar al que nos sucede en la imagen C pero sucede por fenómenos inversos, ya que en la imagen C el error es debido al exceso de iluminación en una zona del suelo siendo esta zona confundida con obstáculo. Sin embargo, en la imagen B, la falta de iluminación en zonas bajas de los obstáculos hace que zonas de obstáculos poco iluminadas sean confundidas como suelo. Estos errores son producto de la utilización de la imagen en escala de grises previa al paso a binario, debido a que este tipo de imágenes solo se centran en la captación de iluminación. Por tanto, será necesario cambiar la manera de realizar el paso a binario ya sea cambiando la imagen desde la cual se realiza esta conversión o utilizando valores del umbral adecuados a nuestra situación.

En primer lugar, se omitió la posibilidad de utilizar imágenes RGB por su gran carga computacional además de por su poco aporte de información ya que en la imagen de simulación (figura 6.1) no requeríamos de tanta información. Como acabamos de ver, la información aportada por una imagen en escala de grises para este algoritmo no es válida debido a sus grandes problemas con los cambios de iluminación. Por tanto, es conveniente la extracción de información a partir de imágenes RGB en las cuales lograremos hacer diferenciación de colores.

Observando la ayuda proporcionada por Matlab sobre la función "*graythresh*", la cual crea un umbral de intensidad sobre una imagen, podemos comprobar que esta función es aplicable a una imagen creada a partir de tres matrices, como sucede con las imágenes RGB. Este umbral podemos utilizarle posteriormente para el paso a binario, usándolo como segundo parámetro de la función "*im2bw*". Además, esta función de paso a binario, también admite la entrada de una imagen RGB. Por tanto, aplicaremos estas dos funciones a nuestra imagen en RGB.

A continuación, en la figura 7.5, podemos comprobar los resultados de la aplicación de las funciones anteriormente descritas.



(A) (B) (C)

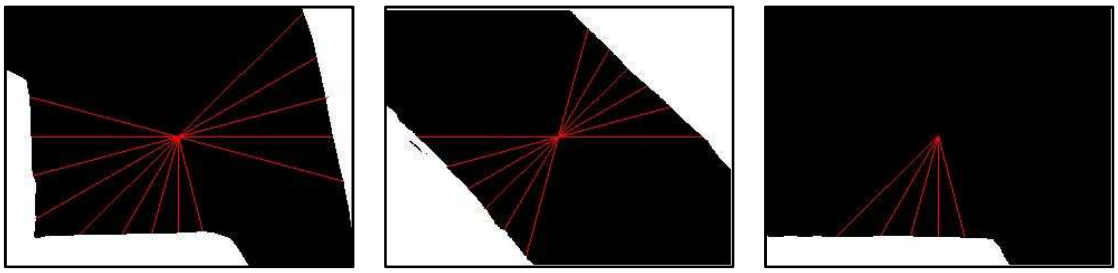
Figura 7.5: Resultados de la aplicación del cálculo de distancias sobre una imagen RGB.

Sorprendentemente, la aplicación del cálculo de distancias sobre una imagen RGB apenas sufre problemas con la iluminación. Como podemos observar en la imagen C, la zona de sobre iluminación que teníamos anteriormente en la zona superior de la imagen a desaparecido casi por completo. Además, atendiendo a la imagen B y comparándola con la imagen B de la figura 7.4, comprobamos que los problemas que teníamos con las zonas de baja iluminación se reduce ampliamente siendo más exacto el cálculo de distancias.

Cabe destacar que estos resultados han sido tomados sin la aplicación del filtro de Gauss, ya que en el anterior ensayo no suponía de gran relevancia a la hora de resolver nuestro problema con la iluminación. Pero una vez resuelto nuestro problema con la iluminación, sería conveniente ver el efecto del filtro de Gauss sobre nuestra imagen RGB, con el fin de suavizar las fronteras de nuestra imagen y conseguir un cálculo de distancias de mejor precisión.

Para poder aplicar el filtro de Gauss sobre una imagen RGB, es necesaria la aplicación de la función *'spatian_gaussian_filter'*, sobre cada una de las tres matrices RGB de nuestra imagen, de manera individual. Esto nos reportará el resultado de las tres matrices ya filtradas por separado, con lo cual, deberemos volver a unirlas en una única imagen.

Una vez ejecutado todos los pasos anteriormente descritos, podemos observar sus resultados en la figura 7.6.



(A) (B) (C)

Figura 7.6: Resultado final del algoritmo de cálculo de distancias.

Observamos que el filtro de Gauss si nos es de ayuda en este momento, limpiándonos la imagen de pixel erróneos, como los que teníamos en la imagen A y C de la figura 7.5. Además, comprobamos en la imagen B que los problemas que teníamos en las zonas de baja iluminación también se han eliminado siendo la frontera entre el suelo y los obstáculos los más ajustada posible con la realidad.

Por tanto, todos nuestros problemas de iluminación se han suprimido, consiguiendo un sistema de cálculo de distancias muy fiable y de gran robustez.

Por último, veremos en la figura 7.7, otros dos ejemplos en los cuales comprobamos que nuestro algoritmo de cálculo de distancias sigue comportándose de manera fiable.

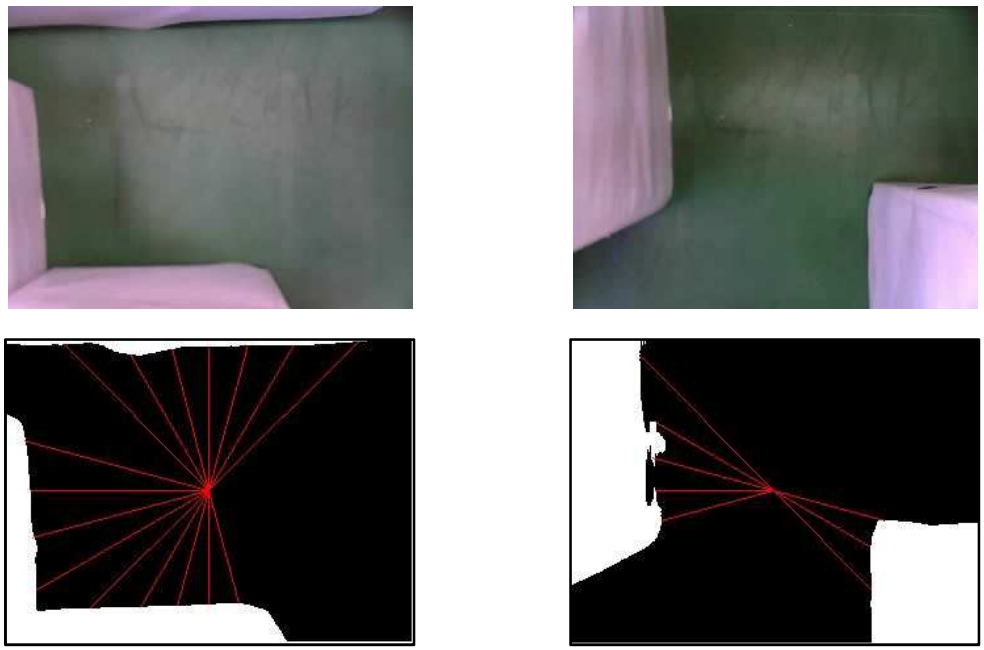
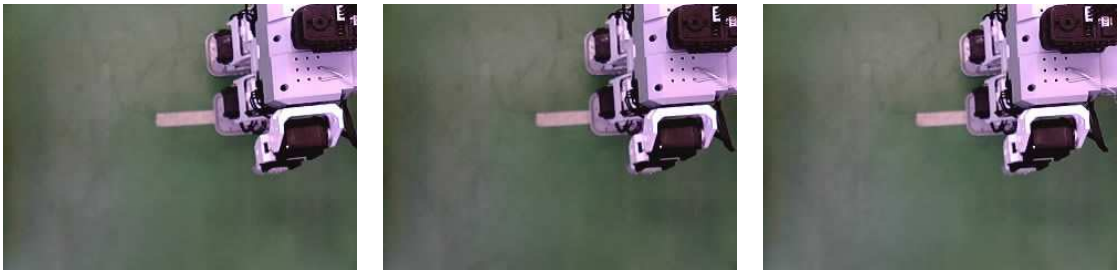


Figura 7.7: Ejemplos del funcionamiento del algoritmo de cálculo de distancias.

7.2 Pruebas del algoritmo de detección de movimiento

En el ensayo del algoritmo de detección de movimiento, nos centraremos en cómo nos afecta la velocidad del objeto a detectar, de tal forma que el resultado de nuestro calculo sea lo más correcto posible. En un entorno similar a CEABOT, en concreto el de la prueba de sumo de este campeonato, probaremos nuestro código de detección de movimiento mediante la captura de una serie de imágenes, en las cuales, el objeto captado, en nuestro caso el robot Bioloid, se desplaza por nuestro campo de visión durante la secuencia de imágenes.

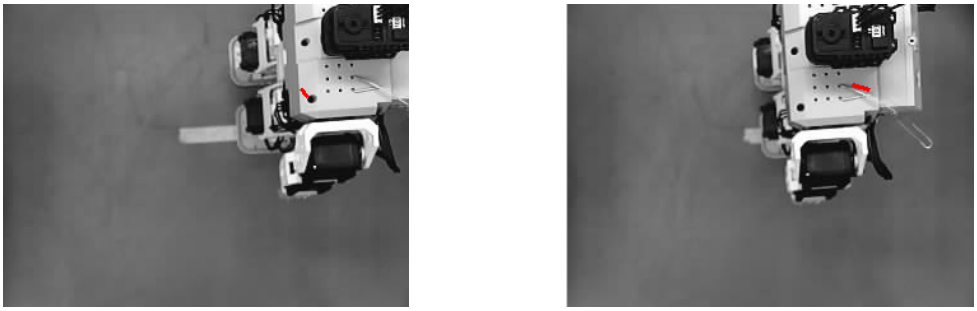
A continuación, en la figura 7.8, podemos observar las 3 primeras imágenes de una secuencia total de 12 imágenes tomadas. En cada imagen, el robot se adelanta 5mm de tal forma que podremos utilizar la serie de imágenes para probar como afecta la velocidad en nuestro algoritmo, por ejemplo, podemos probar como afecta un velocidad baja del objeto captado utilizando las 3 primeras imágenes o podemos ver cómo afecta la alta velocidad del objeto captado tomando las imágenes 1,6 y 12.



(1) (2) (3)

Figura 7.8: Primeras 3 imágenes de la secuencia para el ensayo del código de detección de movimiento.

Una vez tomada la secuencia de imágenes, es momento de probar nuestro código para diferentes series de imágenes. Comenzaremos probando los casos más extremos, es decir, los casos en los cuales el desplazamiento del robot por nuestra imagen es mayor y menor. Para ello, cargaremos las imágenes 1, 2 y 3 para el ensayo de baja velocidad, y las imágenes 1, 6 y 12 para el ensayo de alta velocidad. En la figura 7.9 podemos observar los resultados de nuestro algoritmo de detección de movimiento.



Desplazamiento total de 1,5mm (A) Desplazamiento total de 6mm (B)

Figura 7.9: Comparativa entre resultados para los casos más extremos de desplazamiento.

Comparando los dos resultados, comprobamos que la detección de movimiento se realiza de una manera más exacta para la secuencia de imágenes de mayor distancia recorrida, por tanto, será conveniente que la secuencia de imágenes tenga un intervalo de tiempo entre cada captura relativamente alto. Para saber cuál debe ser el valor a utilizar para el tiempo de espera entre captura y captura, debemos saber cuál es la velocidad máxima y mínima a la que se desplaza un robot mini humanoide. Buscando

información de cuál es la velocidad media a la que un robot mini humanoide se desplaza, hemos determinado que la velocidad media es de 20mm/s. Por tanto, un tiempo de espera entre capturas de 2 segundos, será un valor valido para el correcto funcionamiento de nuestro algoritmo de visión de detección de movimiento.

Además, podemos comprobar que para el ensayo A, en el cual el desplazamiento es mínimo, la dirección del vector velocidad no es paralela a la dirección del movimiento, esto se debe a que mientras el robot anda produce velocidad angular, es decir, para andar, adelanta primero una pierna y luego la otra, esto da lugar a un mayor movimiento en la parte adelantada y por esto, el vector velocidad no es paralelo a la dirección del movimiento. Sin embargo, podemos comprobar que en la imagen de mayor desplazamiento, este fenómeno se reduce siendo el vector velocidad bastante más paralelo al sentido del movimiento.

Este fenómeno, también se produce por el poco ángulo de visión de la cámara web utilizada ya que al estar a mucha altura podemos observar el movimiento angular del robot al andar. De estar en una situación omnidireccional como la propuesta en este proyecto, no veríamos este fenómeno. Para comprobar esto, realizaremos otra secuencia de imágenes en la cual el robot estará más alejado del centro de la imagen, siendo menor la captación de la parte superior del robot.

En la figura 7.10, podemos observar la secuencia de imágenes utilizadas actualmente. Destacar que hemos utilizado las imágenes que nos proporcionan un desplazamiento total de 60mm.

Figura 7.10: Secuencia de imágenes utilizadas en la prueba de detección de movimiento.

Utilizando estas imágenes en nuestro código de detección de movimiento, hemos conseguido los siguientes resultados, ver figura 7.11.

Figura 7.11: Resultados de algoritmo de detección de movimiento para un movimiento alejado.

Como podemos observar, la dirección del vector velocidad si se corresponde con la dirección del movimiento captado. Por tanto, podemos concluir que este código tiene un buen comportamiento ante un entorno como el utilizado en CEABOT. Siendo

mejor el comportamiento que tendrá la cámara Blackfin que la utilizada en estas pruebas, debido a la colocación de la cámara.

Por último, destacar que, a diferencia de nuestro anterior código, el algoritmo de detección de movimiento es muy robusto en cuanto a lo que a la iluminación se refiere. Únicamente podríamos tener problemas de iluminación si los focos de luz están en movimiento, ya que de ser así, nos provocaría movimiento de sombras o reflejos a lo largo de la secuencia de imágenes. Pero, no debemos preocuparnos por esto, ya que la iluminación utilizada en CEABOT es estática, y por tanto, este efecto desaparece completamente.

7.3 Pruebas del algoritmo de flujo óptico

En este apartado, nos dedicaremos al ensayo del algoritmo de flujo óptico sobre un entorno real, utilizando una colocación omnidireccional. Para ello, hemos realizado una secuencia de imágenes en la cual hemos desplazado la cámara hacia delante para conseguir así captar flujo óptico.

El desplazamiento realizado sobre la cámara ha de ser paralelo al suelo y progresivo, tomando secuencias de 5 imágenes para desplazamientos de 1, 3, 5 y 10mm. En la figura 7.12 observamos la secuencia de 5 imágenes con desplazamiento de 10mm. En la cual, observamos como un obstáculo estático penetra en nuestro campo de visión a causa del desplazamiento de la cámara.

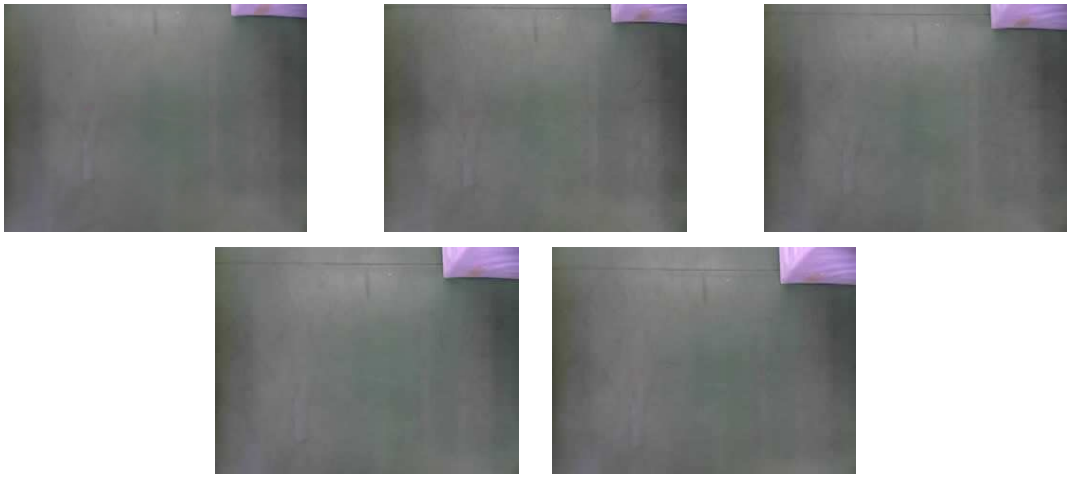
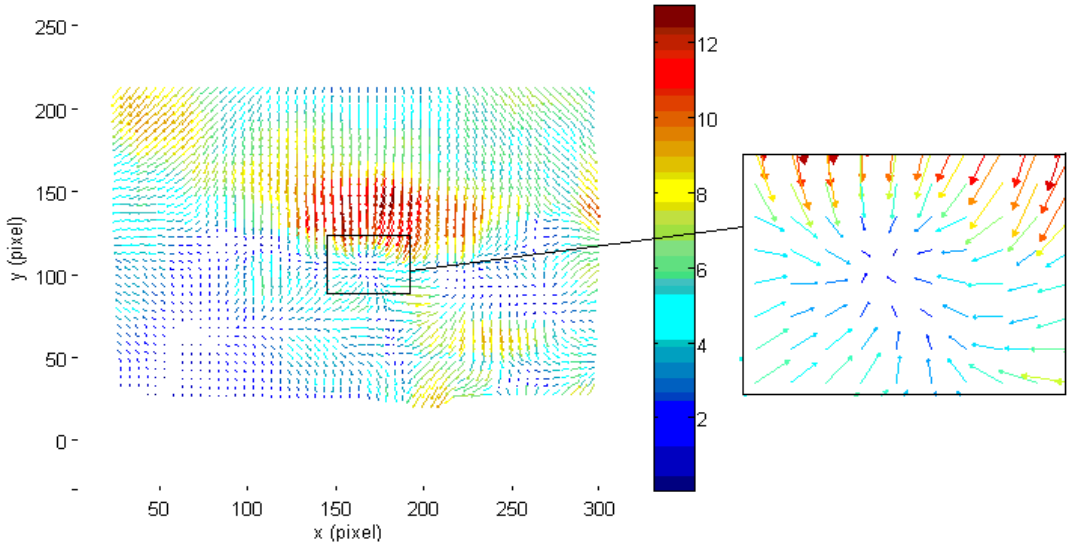
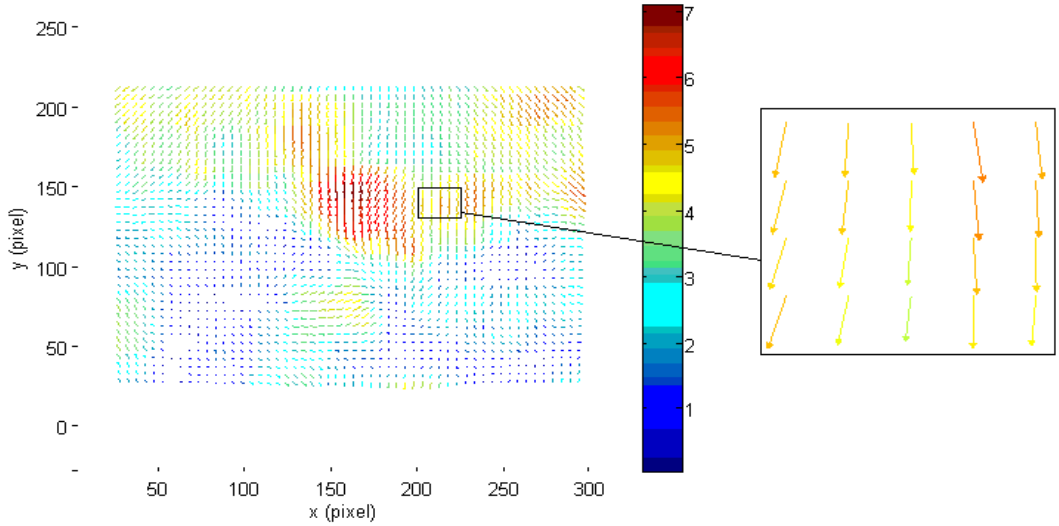


Figura 7.12: Secuencia de imágenes utilizadas para el ensayo del algoritmo de flujo óptico con desplazamiento de 10mm.

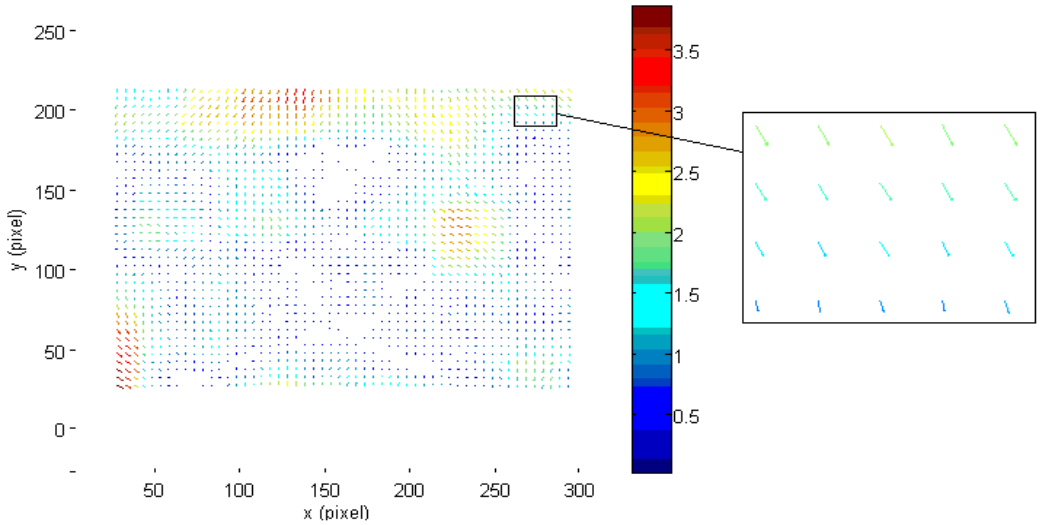
Antes de realizar cualquier tipo de prueba en la cual cambiemos el entorno, es necesario comprobar cuál es el mejor desplazamiento de la cámara para el correcto cálculo del flujo óptico. Para realizar esto, aplicaremos nuestro código de flujo óptico sobre todas las secuencias de imágenes anteriormente nombradas. En la figura 7.13, podemos observar los resultados en función del desplazamiento de la cámara.



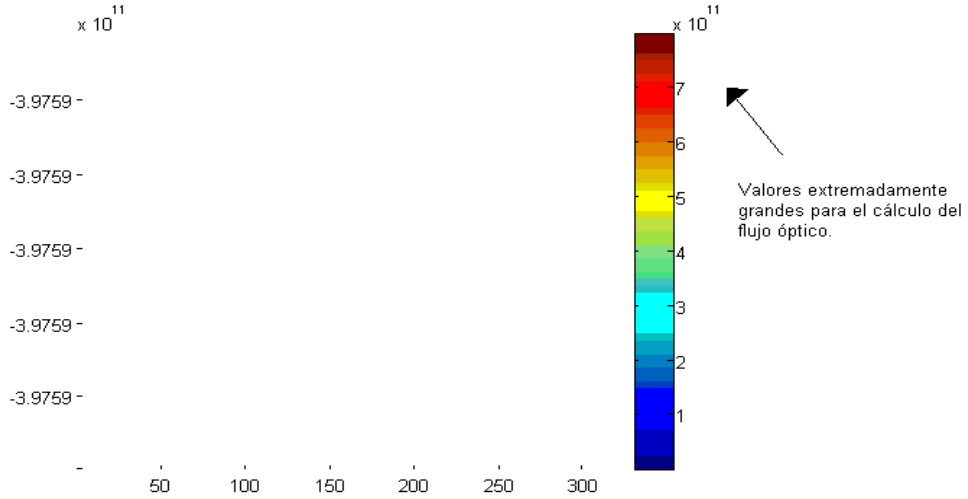
1. Resultado del algoritmo flujo óptico para un desplazamiento de 1mm.



2. Resultado del algoritmo flujo óptico para un desplazamiento de 3mm.



3. Resultado del algoritmo flujo óptico para un desplazamiento de 5mm.



4. Resultado del algoritmo flujo óptico para un desplazamiento de 10mm.

Figura 7.13: Comparativa entre los resultados del algoritmo de flujo óptico en función del desplazamiento

Comprobando los resultados, observamos que el resultado más valido es el de la imagen 2 siendo su desplazamiento de 3mm, pero veamos más detenidamente cada resultado en función del desplazamiento. Para un desplazamiento demasiado pequeño, como el de 1mm, el flujo óptico es apenas captado además de producir resultados erróneos en la dirección del flujo óptico. Opuestamente, en un desplazamiento demasiado grande, como el de 10mm, el flujo óptico es demasiado grande e imposible de calcular por superar con creces los valores de flujo ópticos capaces de representar. Además, podemos observar en la imagen 3 que para un desplazamiento de 5mm el flujo óptico sigue siendo demasiado grande llevándonos a errores en ciertas zonas de la imagen. Por tanto, para próximos ensayos de flujo óptico, utilizaremos un desplazamiento de 3mm. Cabe destacar, que este movimiento deberá realizarlo el robot Bioloid ya que necesitamos movimiento propio para la captación del flujo óptico en obstáculos estáticos.

Antes de efectuar otro ensayo, esta vez en función del entorno, debemos observar los resultados del flujo óptico en esta primera prueba. En ella, podemos observar que apenas apreciamos flujo óptico en las zonas uniformes como el suelo y que únicamente captamos flujo óptico en las zonas de desplazamiento de los obstáculos. Esto puede deberse a la falta de textura en las zonas uniformes del entorno. Además, observamos como captamos una gran región en el centro de la imagen, cuando en esta zona no hay nada que detectar. Esto es porque captamos el movimiento de los reflejos generados por la iluminación. Por tanto, es conveniente que nuestro siguiente ensayo se base en la comparativa entre entornos con mucha textura y entornos que carecen de esta.

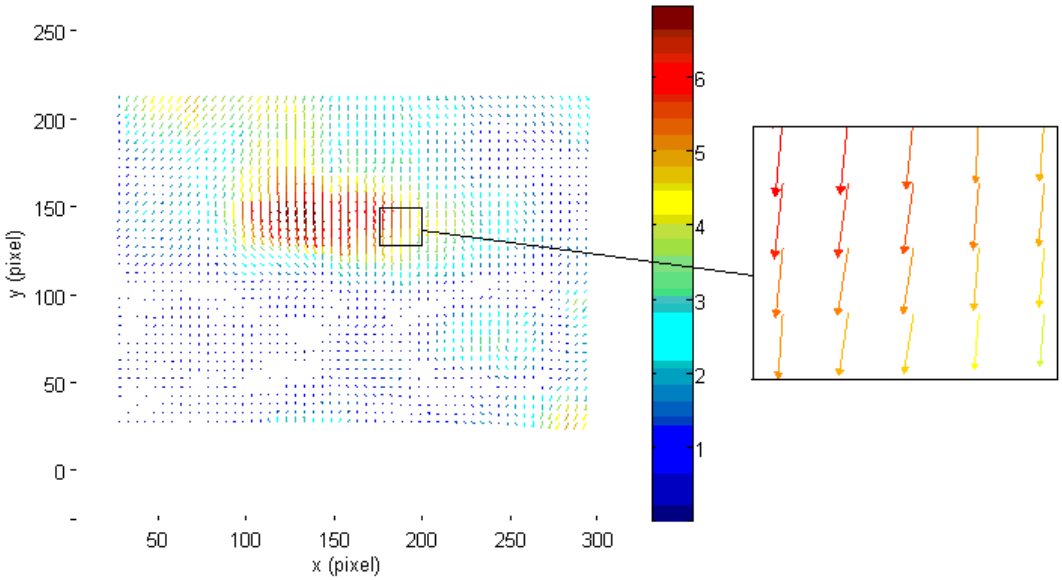
A continuación, en la figura 7.14 podemos observar los dos entornos sobre los que probaremos nuestro algoritmo de flujo óptico. Uno será nuestro entorno similar a CEABOT libre de obstáculos, donde hemos supuesto que carece de textura. Y otro, en el cual podemos captar muchas texturas al deberse de un entorno con gran variedad de colores y formas. Destacar que hemos realizado un desplazamiento de 3mm en la dirección superior de la cámara y por tanto de la imagen.

Captura de un entorno de poca textura.

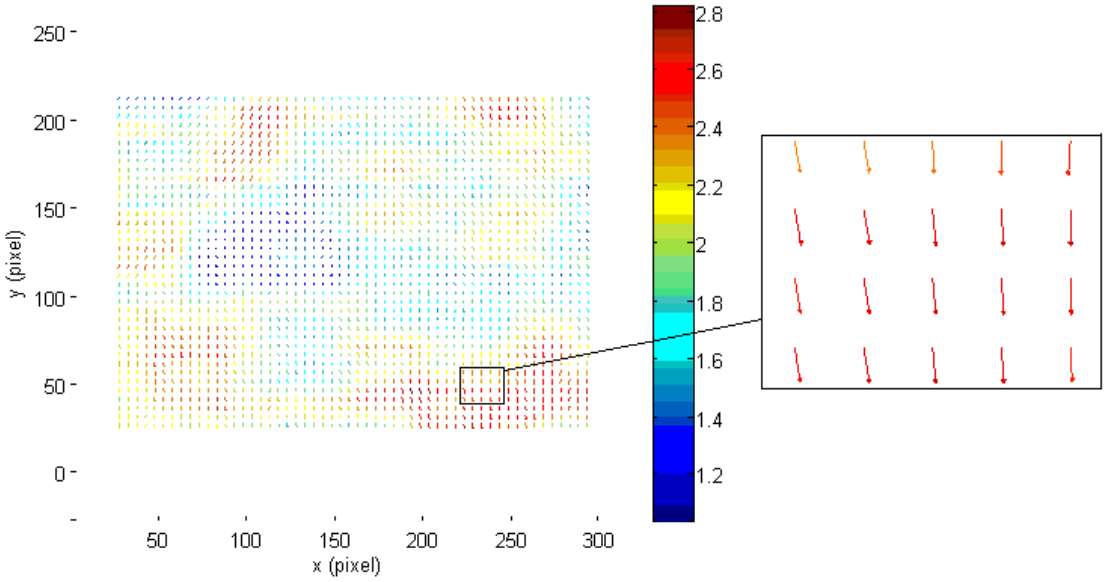
Captura de un entorno con mucha textura

Figura 7.14: Capturas de diferentes entornos en función de la textura para el ensayo del código de flujo óptico

Aplicando nuestro código sobre la secuencia de imágenes tomada para diferentes entornos en función de la textura, hemos obtenido los siguientes resultados, los cuales podemos apreciar en la figura 7.15. Además se muestra un zoom de las zonas más relevante del cálculo del flujo óptico.



Resultado de la prueba de flujo óptico para un entorno con poca textura.



Resultado de la prueba de flujo óptico para un entorno con mucha textura.

Figura 7.15: Comparativa entre los resultados de flujo óptico para entornos con diferentes texturas.

Como podemos apreciar en los resultados, para una superficie, aparentemente uniforme como es la del entorno de CEABOT, el flujo óptico no es continuo a lo largo de toda la imagen. Esto es debido a que no estamos captando textura y lo que realmente captamos son reflejos de luz, los cuales se desplazan por la imagen a causa de nuestro movimiento. Sin embargo, en el resultado de la imagen de mayor textura el flujo óptico se reparte más uniformemente por la imagen, lo cual, nos da una mejor aproximación del flujo óptico ideal a calcular.

Además, atendiendo a los valores máximos del flujo óptico nos damos cuenta de que para el entorno de poca textura el valor de flujo óptico máximo es muy superior al de mucha textura. Esto se debe a que para entornos de poca textura la diferencia entre el flujo óptico máximo y mínimo es mayor, ya que para las zonas sin reflejos apenas

captamos movimiento y por tanto, donde hay reflejos captamos mucho flujo óptico en relación a este primero. Sin embargo, en el entorno de mucha textura como todo está en movimiento, la diferencia entre el flujo óptico máximo y mínimo es más reducida.

A la vista de los resultados, podemos determinar que el algoritmo de flujo óptico es muy sensible a los cambios en la iluminación y además de tener un mal funcionamiento en entornos de poca textura como el utilizado en CEABOT. Pero, cuando utilicemos la cámara Blackfin con una posición omnidireccional, debido a su mayor ángulo de visión, podremos observar imágenes como la que se muestra en la figura 7.16, en la cual podemos ver las paredes de los obstáculos de una forma más vertical. De tal forma que, conseguiremos tener más textura gracias a la diferencia de color entre el suelo y los obstáculos.

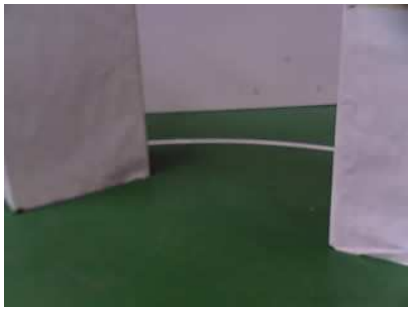


Figura 7.16: Imagen utilizada en el ensayo de flujo óptico emulando la parte exterior de la imagen de la cámara Blackfin.

Para poder concretar flujo óptico en una situación como la propuesta anteriormente, hemos realizado una secuencia de imágenes en la cual nos desplazamos hacia la izquierda con el fin de poder captar flujo óptico. El resultado del cálculo de flujo óptico sobre esta secuencia de imágenes podemos observarlo en la figura 7.17.

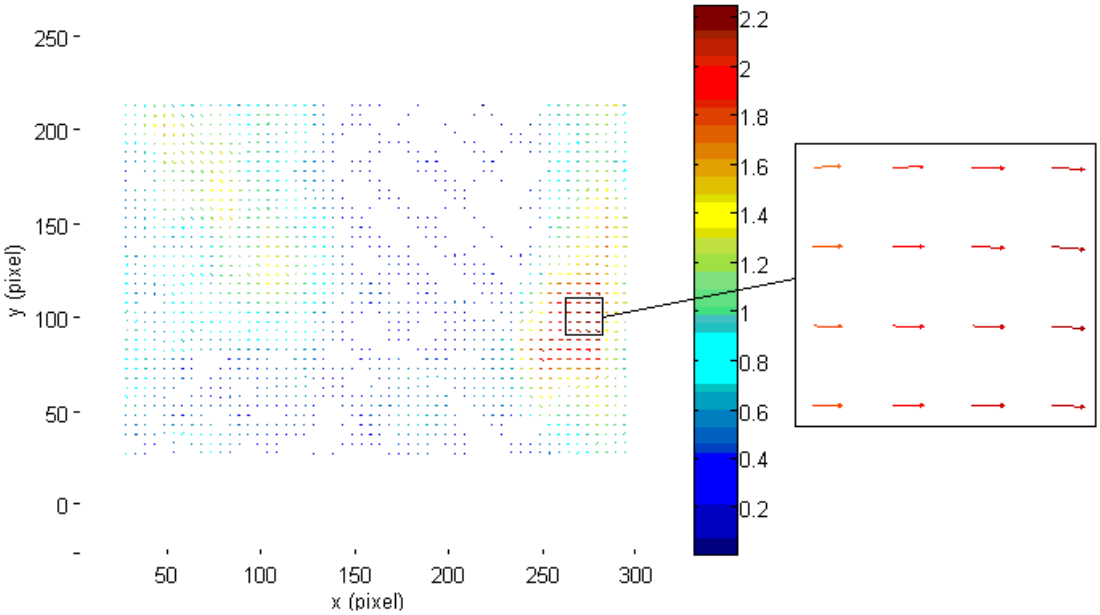


Figura 7.17: Resultado del flujo óptico aplicado a un movimiento lateral.

Como podemos observar, en las zonas de más textura como las fronteras verticales de los obstáculos, captamos correctamente flujo óptico y en las zonas de poca textura como el suelo y el fondo, apenas lo captamos. Este es un buen resultado

de la aplicación del algoritmo de visión de flujo óptico, ya que, como veremos a continuación, nos aporta gran información del entorno que nos rodea.

Por ejemplo, cómo podemos apreciar en la figura 7.18, las zonas más destacas de flujo óptico se corresponden con los obstáculos del entorno. Esto nos puede servir de gran ayuda a la hora de dar utilidad a este código en un entorno con apenas textura, ya que podremos utilizar el algoritmo de flujo óptico como apoyo al cálculo de distancias en la prueba de obstáculos del CEABOT.

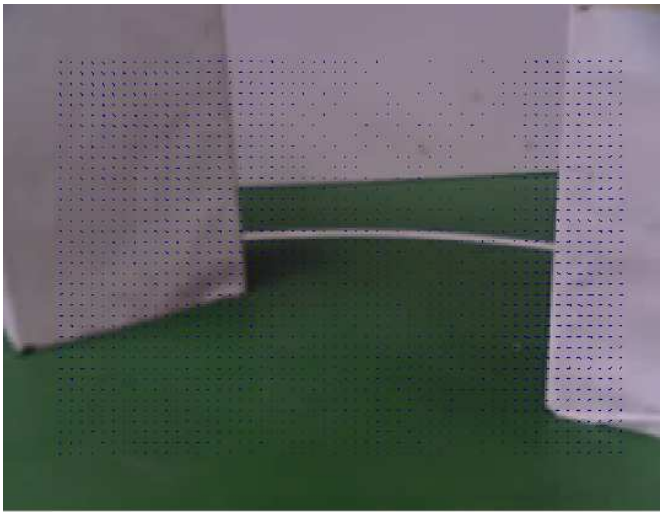


Figura 7.18: Representación del flujo óptico sobre la imagen original.

Además, como podemos observar en la representación de la intensidad del flujo óptico (figura 7.19), en la cual podemos observar la variación del flujo óptico en función de la proximidad entre el obstáculo y la cámara.

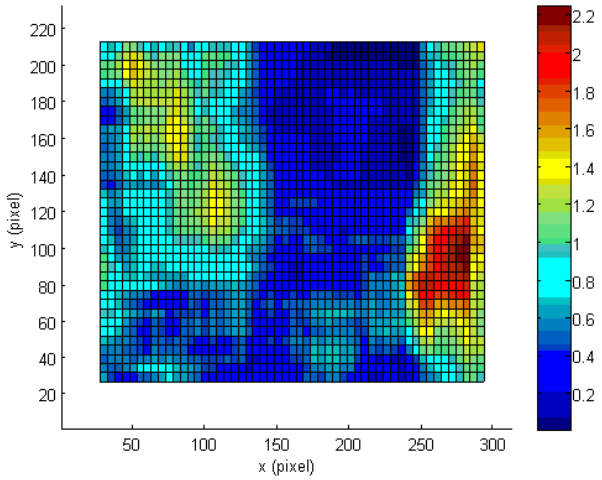


Figura 7.19: Representación de la intensidad del flujo óptico captado.

Como observamos, en la zona de la derecha, donde se sitúa el obstáculo más cercano, el flujo óptico es de mayor intensidad respecto al obstáculo del lado izquierdo el cual se encuentra el obstáculo más alejado de la cámara. Esto nos es de gran ayuda a la hora de diferenciar las distancias entre obstáculos.

Si realizamos el cálculo de la media de los valores de flujo óptico sobre la imagen de la figura 7.19, además del cálculo de la desviación estándar conseguiremos los siguientes valores.

media_izq = 0.6123 media_der = 0.6437

desvi_izq = 0.0625 desvi_der = 0.1406

Donde podemos observar que en la zona derecha de nuestra imagen la desviación estándar de nuestro flujo óptico es mayor, debido a la mayor diferencia entre los valores de máximo flujo y mínimo. Además vemos que la media es superior en el lado derecho que en el izquierdo.

Por tanto, podemos concretar que el algoritmo de flujo óptico nos será de gran ayuda como apoyo al cálculo de distancias entre obstáculos y la cámara.

Por último, podemos comprobar si el algoritmo de flujo óptico es aplicable al apoyo del algoritmo de visión de detección de movimiento.

Aplicando la secuencia de imágenes utilizada en el primer ensayo del detector de movimiento (figura 7.8) sobre en nuestro código de flujo óptico hemos conseguido los siguientes resultados.

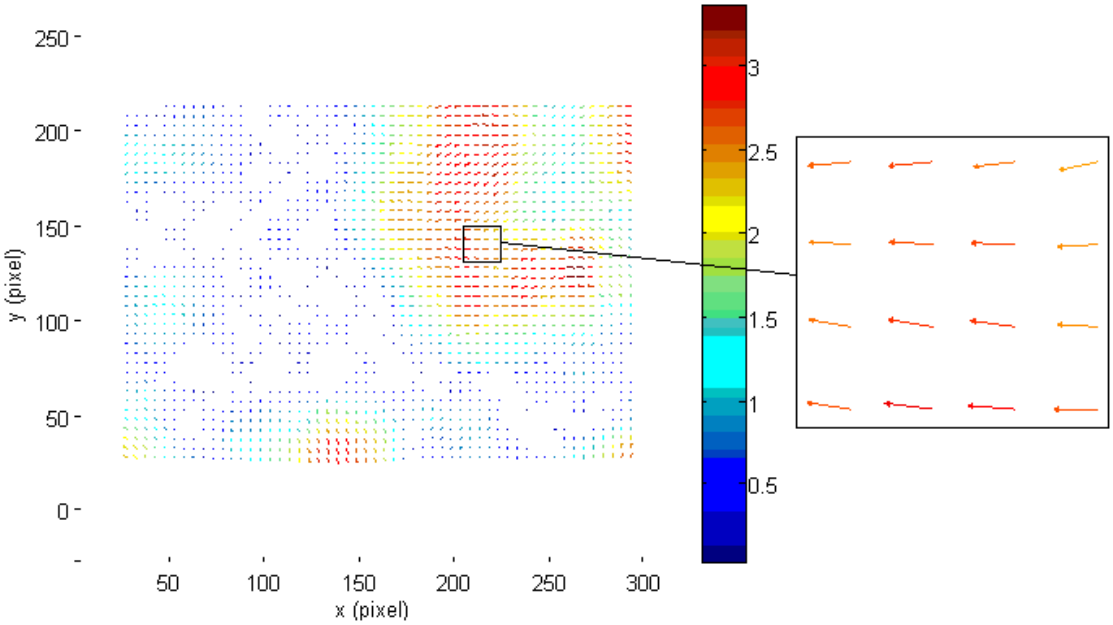


Figura 7.20: Resultado de la aplicación del flujo óptico sobre las imágenes del ensayo de detección de movimiento.

Como podemos apreciar en el zoom de la imagen, la dirección del movimiento captado mediante el flujo óptico es correcta respecto de la dirección real del movimiento del robot. Pero si nos fijamos en la totalidad de la imagen, comprobamos que existen zonas erróneas de flujo óptico como la que se encuentra en la zona baja de la imagen.

Por tanto, a la vista de los resultados, podemos concretar que ante la posibilidad de utilizar el flujo óptico como apoyo al algoritmo de detección de movimiento, es conveniente desestimar esta posibilidad. Esto es debido a que el algoritmo de detección de movimiento realizado por nosotros tiene gran fiabilidad con respecto a los resultados concretados por flujo óptico.

8 Implantación de la cámara en el robot

La colocación de la cámara en el robot necesita de un estudio de las especificaciones que tenemos tanto de la cámara como de los campeonatos donde queremos participar.

Tras el estudio de las distintas opciones para la colocación de la cámara, hemos determinado que la mejor opción para la implantación física de la cámara en el robot es la colocación de esta de manera omnidireccional. Esto quiere decir que la cámara debe estar colocada de tal forma que nos permita tener un área de visión de 360^o respecto de la vertical del robot. De esta forma conseguiremos un sistema de visión completo no influenciado por la dirección a la que mira el robot pudiendo detectar objetos que se aproximen por cualquier ángulo.

Una manera de conseguir esto, es colocar la cámara paralela al suelo, es decir, con foco del objetivo perpendicular al suelo y mirando hacia este.

Para que esto sea viable, es necesario que la cámara esté colocada arriba del robot, como si fuese una especie de "cabeza" y con una distancia suficiente al cuerpo de este. Esta distancia deberá librar todo lo posible el cuerpo del robot, ya que de estar demasiado cerca puede que solo podamos captar el cuerpo del robot y no el área que le rodea.

A continuación, estudiaremos si esta opción es viable y la mejor forma de desarrollarla.

8.1 Estudio del área de visión en modo de visión omnidireccional

Para la consecución del sistema de visión omnidireccional utilizaremos la lente de 120^o de ángulo de visión que nos dará un área de visión de mayor diámetro o con la misma área de visión, conseguiremos reducir la distancia entre la cámara y el robot.

Cabe destacar que el campeonato CEABOT 2011 tiene en la normativa una altura máxima marcada en 50cm. También sabemos que nuestro robot Bioloid tiene una altura de 31cm. sin la cabeza, con lo cual llegar a la altura máxima permitida por el campeonato será desproporcionado para el robot e innecesario ya que nos proporcionaría un área de visión excesivo para nuestras necesidades.

Hemos concluido, como veremos posteriormente, que la mejor opción es colocar el foco del objetivo de la cámara a una distancia de 40cm. del suelo, es decir, a 9cm. de la parte alta del cuerpo del robot. Con esto el robot tendrá una altura máxima de unos 42cm. debido a que la placa que contiene el procesador de la cámara nos dará unos 3cm de altura extra por encima del objetivo de la cámara.

Como veremos a continuación, esto nos proporcionará un área de visión de 69,3cm. de radio. Además, veremos cómo nos afecta el volumen del robot a nuestro campo de visión, reduciendo nuestro campo de visión por las zonas más cercanas al centro de la imagen.

En la figura 8.1 observamos el alzado del robot con el ángulo de visión de 120^o de la lente y con la distancia que alcanzaremos a ver. Además vemos el ángulo que no podremos ver a causa de los hombros del robot con su respectiva distancia de visión nula.

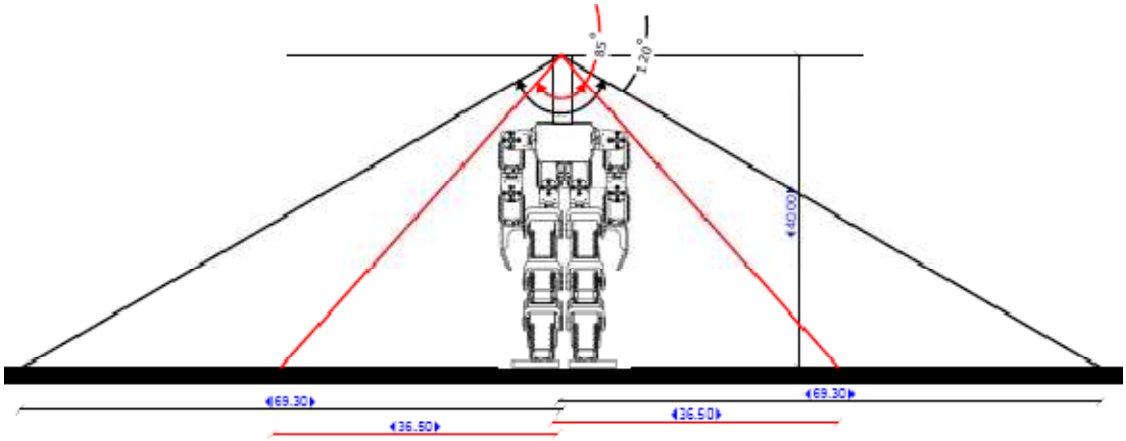


Figura 8.1: Alzado del área de visión del Robot Bioloid.

Como en la figura anterior, en la figura 8.2 veremos las mismas cotas pero esta vez desde el perfil del Robot Bioloid.

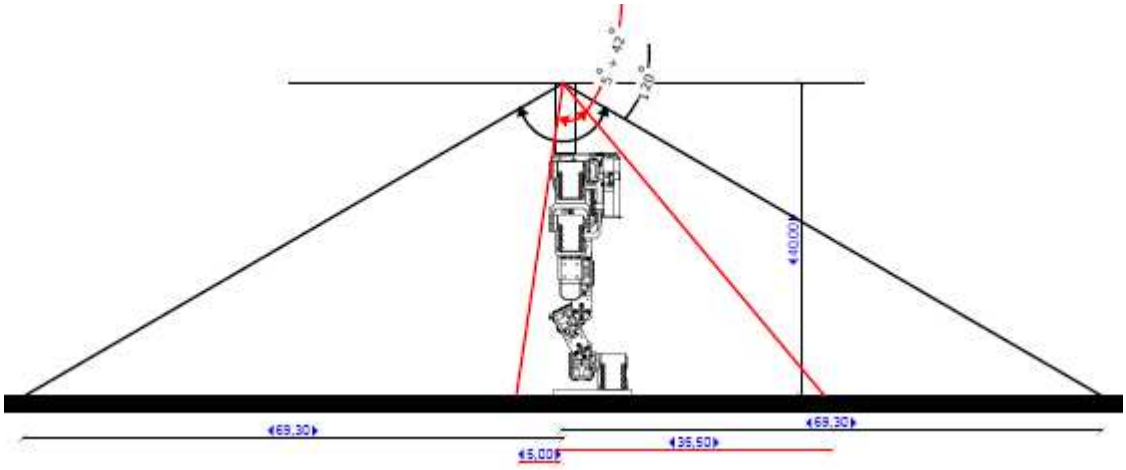


Figura 8.2: Perfil del área de visión del Robot Bioloid.

Por último, comprobamos en la figura 8.3 como será el área completa de visión, a través de la planta del Robot Bioloid.

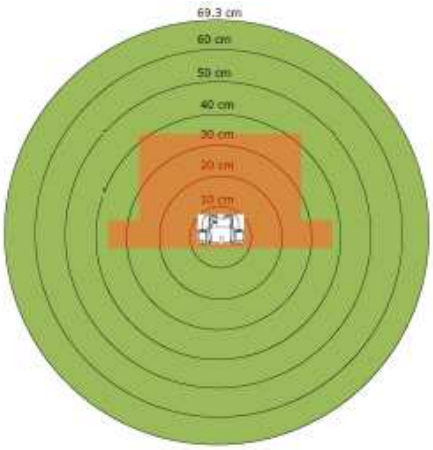


Figura 8.3: Planta el área de visión del Robot Bioloid.

Tras esta investigación, hemos determinado que la opción de colocar la cámara en el robot de forma omnidireccional es viable y por tanto, podemos pasar a la creación de un soporte para la cámara con el cual obtendremos esta configuración física.

8.2 Soporte de la cámara

Para la realización de un soporte omnidireccional para la cámara es necesaria la utilización de un material transparente, rígido y no frágil. Hemos escogido el metacrilato por ser el material que más responde a esas características además de por su bajo coste.

Atendiendo a las características físicas de la cámara y del robot, hemos determinado la siguiente estructura de metacrilato para la creación del soporte. Podemos consultar el ANEXO V donde veremos los planos acotados de esta estructura. En la figura 8.4 podemos observar esta estructura en perspectiva.

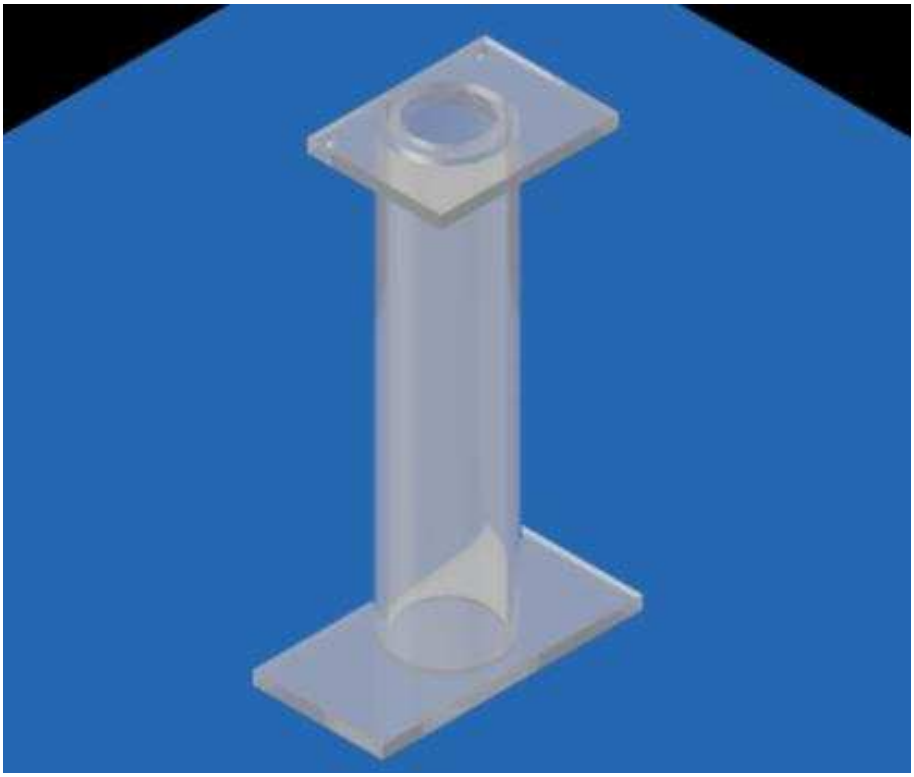


Figura 8.4: Perspectiva isométrica del soporte de la cámara.

Como podemos observar, el soporte consta de tres partes: la base, el cuerpo y la placa superior. La base es una placa de 60x30x3mm la cual irá en contacto con el robot y se ocupará de que el soporte de la cámara y el robot no se separen. El cuerpo se trata de un cilindro de 100mm de longitud, 20mm de diámetro interior y 2mm de espesor, con el cual, aumentamos la distancia entre el robot y el objetivo de la cámara. Y por último la placa superior, la cual es de similares medidas a la placa que soporta el objetivo de la cámara (41x31x3mm). Esta placa tiene una abertura cilíndrica para la introducción del objetivo siendo concéntrica al cilindro que soporta la placa. Además, hemos realizado unos taladros situados en igual posición a los que trae la placa del objetivo (véase figura 4.2), mediante los cuales aseguraremos la cámara al soporte por medio de tornillos y espaciadores.

Finalmente, podemos observar en la imagen 8.5 el soporte de la cámara una vez construido y con la cámara situada en su correcta posición.

Figura 8.5: Soporte con la cámara instalada.

9 Implementación los algoritmos seleccionados en la cámara como sensor independiente

En este último apartado de nuestra memoria, veremos cómo configurar nuestro ordenador para desarrollar los códigos de nuestros algoritmos de visión, anteriormente desarrollados en Matlab, en una versión reducida de C (microC o µC). Este lenguaje de programación es el que soporta el procesador de nuestra cámara Blackfin y por tanto, deberemos aprender a desarrollar, depurar y cargar código en µC sobre nuestra cámara.

Cabe destacar que podemos descargarnos todo el firmware de cámara a través de la página web de Surveyor Corporation [2]. El firmware de la cámara contiene toda la estructura del código fuente necesaria para el correcto funcionamiento de la cámara tales como la iniciación, el código principal y funciones básicas de nuestro procesador.

Observando la web de Surveyor Corporation [2], el apartado del SRV-1 C Interpreter, podemos aprender mediante ejemplos como utilizar las principales funciones definidas por los desarrolladores de la cámara, además de un breve resumen de cómo utilizar µC.

Una vez que hemos aprendido como desarrollar código en µC, podemos comprender como es el funcionamiento del código principal de nuestro firmware. Para ello, necesitamos abrir el archivo *main.c* que se encuentra dentro de la carpeta *srv* de nuestro firmware, en un programa de desarrollo de C si estamos utilizando Windows o sobre un editor de texto si utilizamos Linux. Un buen desarrollador de C para Windows es el programa Eclipse, aunque nosotros utilizaremos Linux para el desarrollo y depuración de nuestros algoritmos de visión ya que es la mejor herramienta de desarrollo de cualquier lenguaje de programación basado en C. Además, como comprobaremos posteriormente, la forma de depurar nuestro código es muy sencilla de ejecutar en Linux.

Aunque en este apartado no nos centraremos en cómo crear nuestros algoritmos de visión en µC, si es conveniente que entendamos cómo funciona el código del SRV-1 Robot, ya que nos muestra que estructura debemos seguir para el correcto funcionamiento de la cámara Blackfin. Observando el código *main.c* vemos que para poder implementar nuestros algoritmos de visión, deberemos crearlos como funciones independientes y añadirlas al archivo *myfunc.h.* Por lo demás, podremos editar nuestro archivo principal *main.c* y adaptarlo a nuestras necesidades, es decir, cambiar el código para poder realizar las llamadas a las funciones de nuestros algoritmos de visión, además del envío de los resultados por un puerto serie.

Antes de poder depurar nuestro código, debemos configurar Linux para el correcto funcionamiento de los compiladores del procesador Blackfin. Para ello, debemos actualizar todos los paquetes ya instalados Linux. Esto lo conseguiremos mediante la apertura de un terminar y la ejecución de la siguiente instrucción.

```
admin@ubuntu:~$ sudo apt-get upgrade
```

Una vez actualizado todo nuestro sistema, debemos instalar los paquetes que nos faltan para el correcto funcionamiento de los depuradores, mediante la introducción del siguiente comando en nuestro terminal.

```
admin@ubuntu:~$ sudo apt-get install autoconf automake bash binutils
bison bzip2 \  coreutils flex gawk gcc gettext grep intltool iputils-
ping libtool libz-dev \linux-libc-dev liblzo1 liblzo-dev liblzo2-dev
liblzo2-2 libncurses5 libreadline5 \libreadline5-dev libncurses5-dev
m4 make pax-utils pkg-config rpm texinfo zlib1g \zlib1g-dev
```

En este punto, tendremos preparado nuestro sistema de Linux para la ejecución de los compiladores del procesador Blackfin, pero aún no los tenemos instalados. Para instalarlos de una forma sencilla, es necesario descargarnos los compiladores Bfin-elf, Bfin-uClinux y Bfin-linux-uclibc en sus correspondientes carpetas, no en formato ejecutable ni como paquete de Linux. Podemos descargar las carpetas de los tres compiladores en la web de Surveyor Corporation [2] en formato comprimido. Posteriormente, debemos descomprimir cada compilador y crear una carpeta común en la cual guardaremos las carpetas de los tres compiladores. Esta carpeta la guardaremos en la carpeta principal de Linux.

Una vez situados nuestros compiladores es necesario añadirlos a un PATH, que es una variable de nuestro soporte de Linux, que nos reportara al directorio que escribamos posteriormente. Como deseamos configurar el PATH de forma permanente, debemos editar el archivo de configuración de nuestro soporte de conexión. Por lo general el soporte BASH es el más utilizado y podemos editar su código mediante la apertura de este en un editor de texto. Esto podemos hacerlo directamente desde el terminal de Linux mediante el siguiente comando:

```
admin@ubuntu:~$ gedit .bashrc &
```

Una vez abierto el editor de texto, nos desplazamos hasta la última línea de nuestro BASH y generamos un PATH con los directorios de las carpetas de nuestros compiladores utilizando la siguiente instrucción de código:

```
export PATH=$PATH:/home/administrador/uClinux/bfin-
uclinux/bin:/home/administrador/uClinux/bfin-linux-
uclibc/bin:/home/administrador/uClinux/bfin-elf/bin
```

Con todo esto, hemos realizado la instalación del depurador del Blackfin y ya estamos preparados para poder depurar nuestro código y crear los ejecutables necesarios para poder realizar la carga del código en la cámara.

Para depurar nuestro código, debemos abrir un terminal y acceder al directorio donde reside nuestro código principal *main.c* del firmware, es decir, la carpeta srv del firmware. Sobre nuestro directorio, ejecutaremos el comando *make*, este comando nos depurará el código y nos generará los ejecutables que utilizaremos para la carga del código en la cámara. En función de cómo conectemos con la cámara, utilizaremos el formato *".ldr"* utilizado para la conexión a través del Matchport o el formato *".bin"* utilizado para la conexión a través de gnICE+. El formato *".ldr"*, es un formato especial utilizado para la transferencia de datos mediante una transmisión "X Modem" como la que realizaremos con el Matchport. Este formato contiene en su interior el formato binario *".bin"* del ejecutable de nuestro código, que es realmente el que se carga en nuestro procesador. Es por esto, por lo que es utilizado este formato en una conexión directa como la que puede ser a través del gnICE+.

Cabe destacar, que es conveniente crear nuestra conexión del Matchport sobre el sistema operativo de Windows ya que es más sencillo a la hora de instalar los complementos necesarios para la configuración y conexión mediante el Matchport.

Antes de poder conectar con nuestra cámara a través de Matchport, es necesaria la configuración de este. Para ello, es necesaria la utilización de un convertidor serie-USB el cual conectaremos por el terminal de serie a la cámara y por el USB a nuestro ordenador. En concreto, nosotros hemos utilizado el convertidor CP210x de Silicon Labs, el cual podemos observar en la figura 9.1.

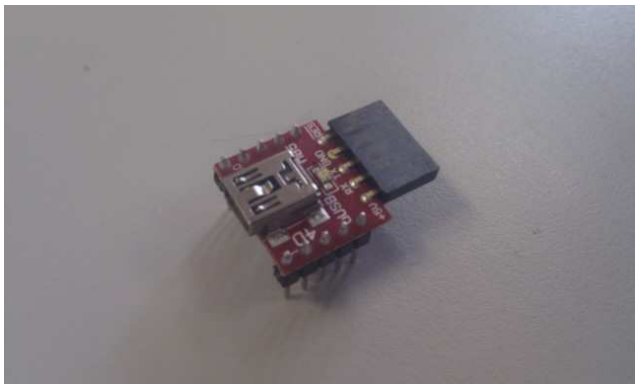


Figura 9.1: Convertidor serie-USB CP210x.

Para conectar este convertidor a nuestro módulo Wifi (Matchport), debemos conectar los pines RX y TX de nuestro convertidor, con los pines RX1 y TX1 de nuestro Matchport (pines 4 y 3 respectivamente del J4). Además, debemos conectar los GND del convertido y del Matchport (pin 2 del J4), como podemos apreciar en la figura 9.2.

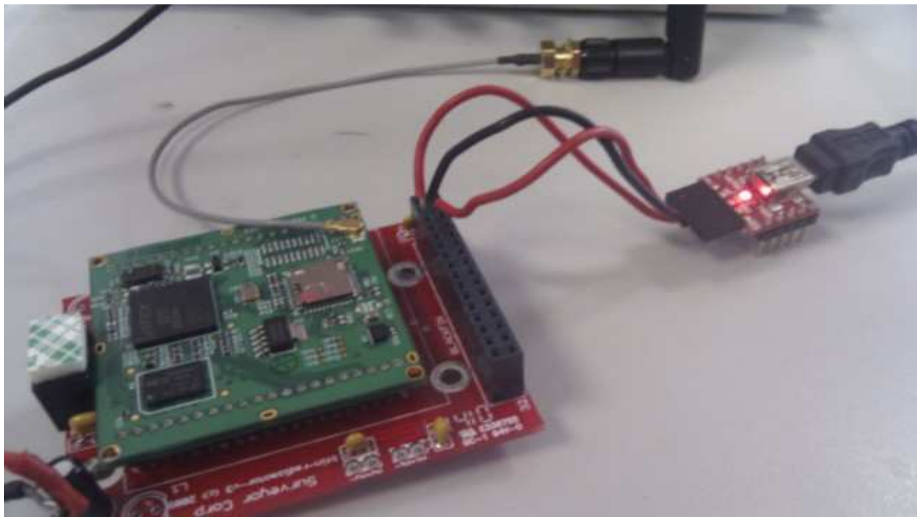


Figura 9.2: Conexión entre el módulo Wifi y el convertidor serie-USB.

Una vez conectado el módulo Wifi y el convertidor serie-USB, debemos conectar el convertidor a nuestro ordenador e instalar los drivers. Cuando hayamos instalado el convertidor, debemos buscar el puerto de nuestro ordenador al que hemos conectador el convertidor. Esto lo haremos pinchando sobre las propiedades del icono *Mi PC*, posteriormente pinchamos sobre *Hardware* y por ultimo en *Administrar Dispositivos*, allí buscaremos nuestro convertidor y nos mostrará el puerto utilizado. Como podemos apreciar en la figura 9.3, en nuestro caso hemos utilizado el puerto COM7,

por el cual conectaremos con nuestro módulo a través de un terminal con conexión serie.

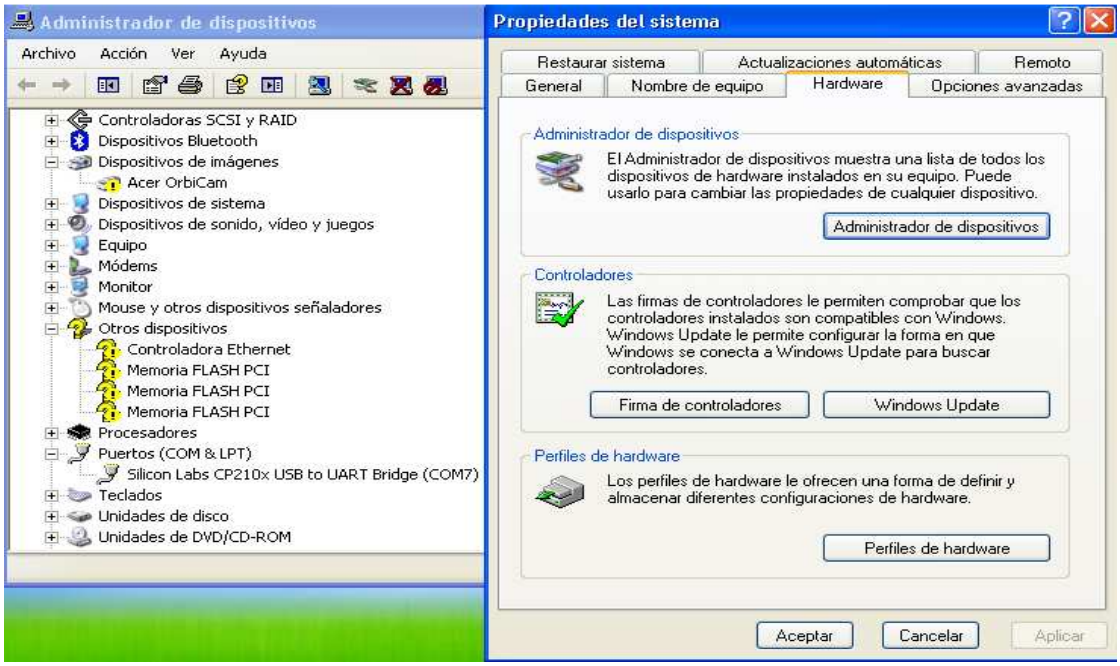


Figura 9.3: Dispositivos hardware utilizados en nuestro ordenador.

En este momento tenemos casi todos los elementos necesarios para la conexión con el Matchport mediante un convertidor serie-USB. Pero, aun necesitamos un terminal con conexión serie para poder realizar la conexión. El terminal utilizado por nosotros sobre el sistema operativo de Windows ha sido el terminal Putty el cual podemos descargar de la página http://www.putty.org/ .

Cuando tengamos instalado el terminal Putty, estamos preparados para realizar la conexión. Para ello, alimentamos la placa de Matchport con 9,3 V de continua y abrimos nuestro terminal Putty. Configuramos el terminal para una conexión serie como podemos apreciar en las siguientes imágenes (figura 9.4) en la cuales se muestra las dos pestañas que debemos configurar en nuestro terminal.

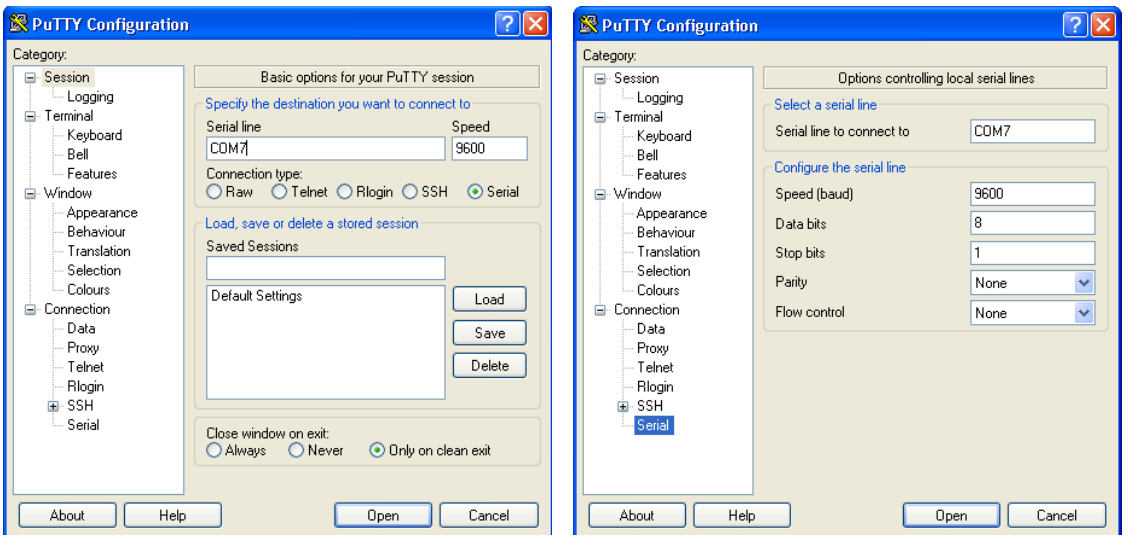


Figura 9.4: Configuración del terminal Putty para una conexión serie.

Si nuestra configuración del terminal es correcta se nos abrirá un terminal como el que se muestra en la figura 9.5.

Figura 9.5: Terminal Putty conectado mediante conexión serie.

Para poder acceder a la configuración del Matchport debemos reiniciar el módulo Wifi mediante un corte en la alimentación mientras mantenemos pulsado el carácter "x" de nuestro ordenador. Esto nos mandará un mensaje, el cual nos muestra que debemos pulsar ENTER para acceder a la configuración del Matchport.

Una vez que hemos entrado en la configuración del Matchport, veremos una imagen como la que se muestra a continuación (figura 9.6).

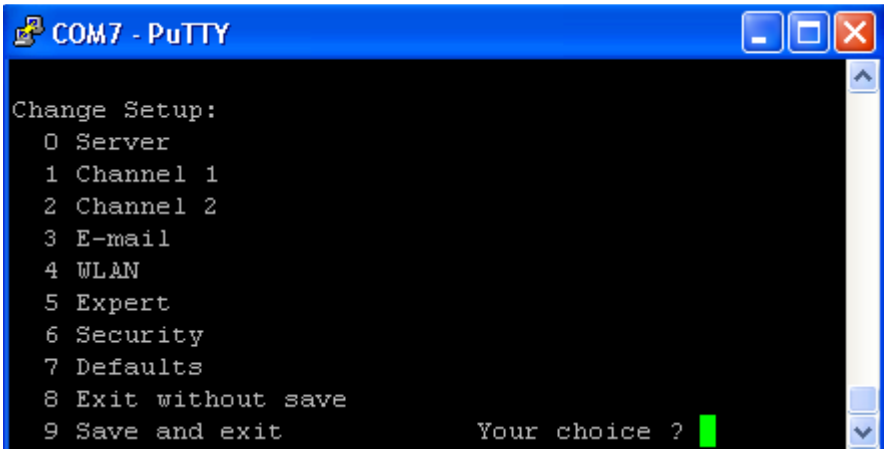


Figura 9.6: Menú principal de la configuración del Matchport mediante una conexión serie.

Como podemos apreciar, se nos muestra un menú en el cual podemos elegir entre las distintas partes de la configuración del Matchport. Los puntos que debemos cambiar para la correcta configuración del Matchport son los siguientes.

- Seleccionamos 5 (Expert): Cambiamos CPU speed a High (2).
- Seleccionamos 4 (WLAN): Nos aseguramos que el Network esta en modo adhoc y cambiamos la ID a SRV1.

- Seleccionamos 1 (Channel 1): Cambiamos la velocidad del canal 1 a 921600 baudios y cambiamos el flujo de control a 2 (hardware cts/rts).
- Seleccionamos 0 (Network): Cambiamos a Wirelles Only e introducimos la IP a la cual posteriormente nos conectaremos, en nuestro caso, 169.254.0.10.

Los puntos de la configuración que no se muestran anteriormente, hay que dejarlos en su configuración por defecto, de no ser así, la configuración no será correcta y tendremos que volver a efectuarla.

Por último, pulsamos "9" (Save & Exit) y ENTER. Con esto ya tendremos configurado nuestro Matchport y podremos conectarlo a nuestra cámara Blackfin.

Manteniendo nuestra alimentación sobre el módulo Wifi a 9,3V y conectando la cámara al Matchport mediante el J4, podemos conectar por wifi con nuestra Blackfin además de poder conectar por wifi con el Matchport y por tanto, con su configuración.

Como podemos apreciar en la figura 9.7, utilizando un navegador, en nuestro caso Mozilla, e introduciéndole la IP anteriormente seleccionada en la configuración del Matchport podemos acceder a la misma configuración del módulo wifi anteriormente explicada pero mediante una interfaz gráfica debido a la conexión wifi.

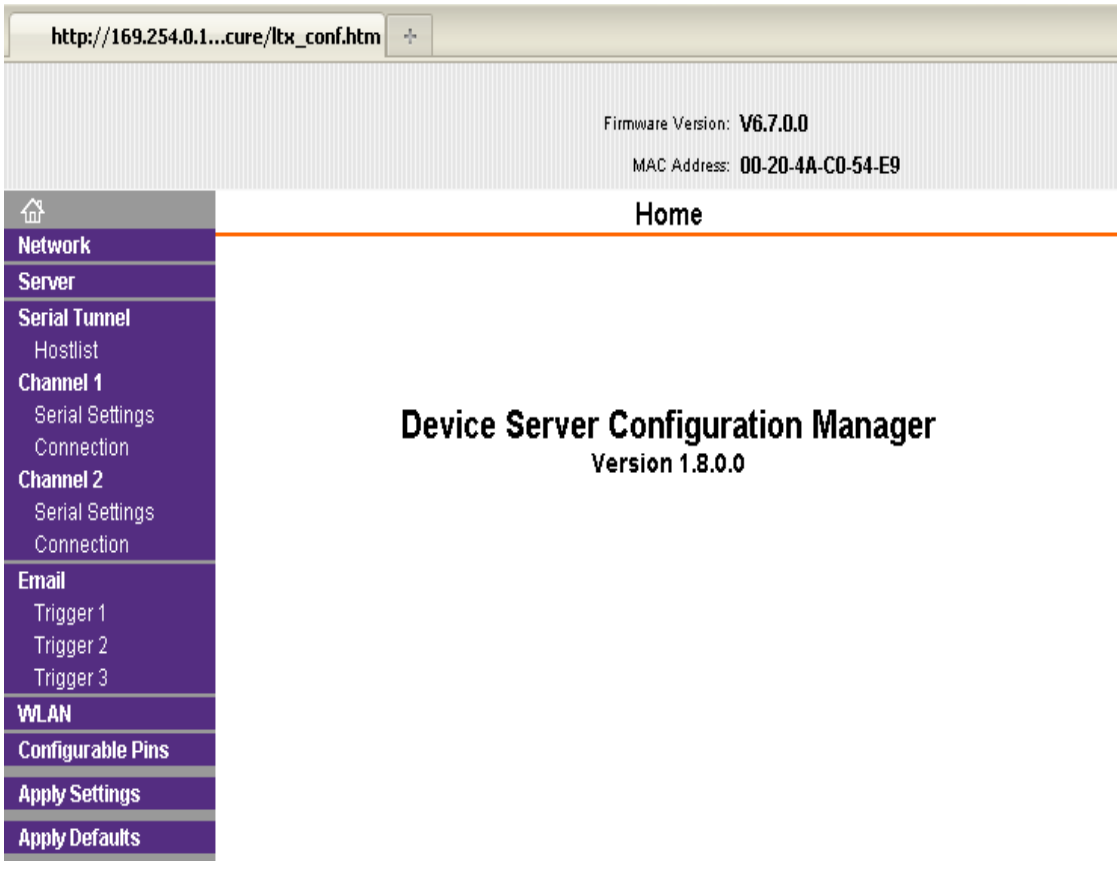


Figura 9.7: Menú de configuración del Matchport mediante una conexión wifi.

Para cargar código sobre nuestra Blackfin, abrimos un navegador y le introducimos la siguiente dirección http://169.254.0.10:10001/admin. De esta forma,

accederemos a la carga del firmware sobre la cámara Blackfin mediante una interfaz gráfica como la que podemos apreciar a continuación, ver figura 9.8.

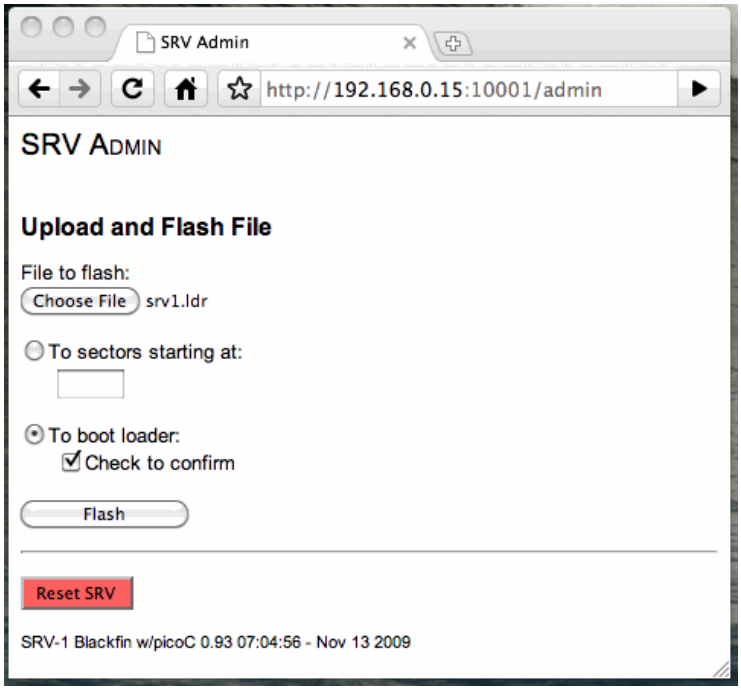


Figura 9.8: Interfaz gráfica utilizada para la carga del firmware sobre la Blackfin.

A través de esta interfaz, seleccionamos el código a cargar en formato *".ldr"*, y pinchamos sobre FLASH. Si el firmware se ha cargado correctamente veremos una respuesta como la que podemos observar en la figura 9.9.

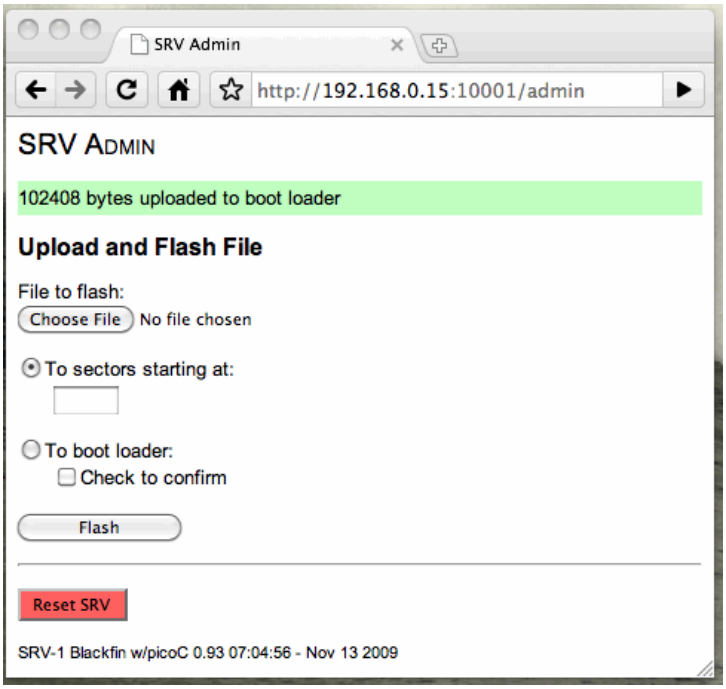


Figura 9.9: Respuesta correcta de la carga del firmware sobre la cámara Blackfin.

De esta forma conseguiremos cargar código sobre nuestra cámara Blackfin de manera sencilla.

Destacar, que existe otra forma de cargar código sobre nuestra cámara Blackfin, aunque de una manera algo más complicada. Como podemos observar en la página web de Analog Devices [9], por medio del conector gnICE+ y del conector JTAG de la placa de nuestra cámara Blackfin, podemos realizar la cargar del firmware, desde un sistema operativo basado en Linux. En este caso, al tratarse de una carga directa sobre el flash del procesador deberemos cargar el ejecutable de formato *".bin"* sobre nuestro dispositivo.

Pero, antes de cargar un archivo sobre nuestra cámara debemos descargarnos los complementos necesarios para la realización de esta carga. Estos complementos son el programador del flash (vdsp-flash-programmer), con el cual ejecutaremos la carga de nuestro código y el driver de nuestro procesador (BF537EzFlashDriver.dxe), el cual aporta al programador flash la información necesaria sobre nuestro procesador ya que se trata de un driver de este. Estos dos archivos son necesarios para la carga de código a través de una conexión JTAG podemos descargárnoslos de la pagina web de Analog Devices [9]. Una vez descargados estos dos archivos los guardamos en una carpeta conjunta en el directorio principal de nuestro Linux.

Antes de poder ejecutar nuestro programador flash, debemos conectar la cámara con nuestro ordenador. Para ello, sobre un terminal de Linux, deberemos introducir el siguiente comando:

```
admin@ubuntu:~$ bfin-gdbproxy bfin --connect='cable gnICE+'
```

Mediante esto, iniciaremos la conexión entre el gnICE+ y nuestro ordenador, aunque aun no hayamos conectado con la Blackfin ya que para ello, deberemos enviar el siguiente comando desde un GDB (depurador) de los compiladores.

```
(gdb) target remote :2000
```

Aunque no será necesario ejecutar ningún GDB de los compiladores ya que como lo que realmente queremos es cargar código sobre nuestra cámara, al ejecutar el flash

Para ejecutar el programador flash debemos acceder a través del terminal de Linux, al directorio donde se encuentra este y ejecutar el siguiente comando sobre el terminal:

```
admin@ubuntu:~$   ./vdsp-flash-programmer.sh BF537EzFlashDriver.dxe
```

Esto nos arrancará un GDB, con el cual mediante la instrucción anterior lograremos conectar con la cámara Blackfin. Además, si nos fijamos en la información proporcionada por el programador flash durante su ejecución, podemos observar los comandos más significativos de la utilización de este, los cuales podemos observar a continuación.

```
fldrvinfo
flinfo
flstat
flcmd
flerase_sector
flerase
flwrite
flrestore
flread
fldump
```

Antes de enviar el ejecutable de nuestro código al flash del procesador debemos cargarlo sobre el depurador del programador flash. Utilizando los comandos *file* y *load* en el GDB del programador flash seguidos del nombre del ejecutable en formato *".bin"*, en nuestro caso *"srv1.bin"*.

Una vez cargado el ejecutable, ya podemos cargar el código en el flash del procesador. Para ello, utilizaremos el comando *flwrite*, el cual guarda nuestro ejecutable en la Blackfin. Si la transmisión se ha realizado correctamente recibiremos un mensaje de carga completada.

Por último, podemos conectar con la cámara en tiempo real de dos formas distintas. Mediante la utilización de un navegador introduciéndole la dirección http://169.254.0.10:10001/view podremos observar la interfaz que se muestra en la figura 9.10.

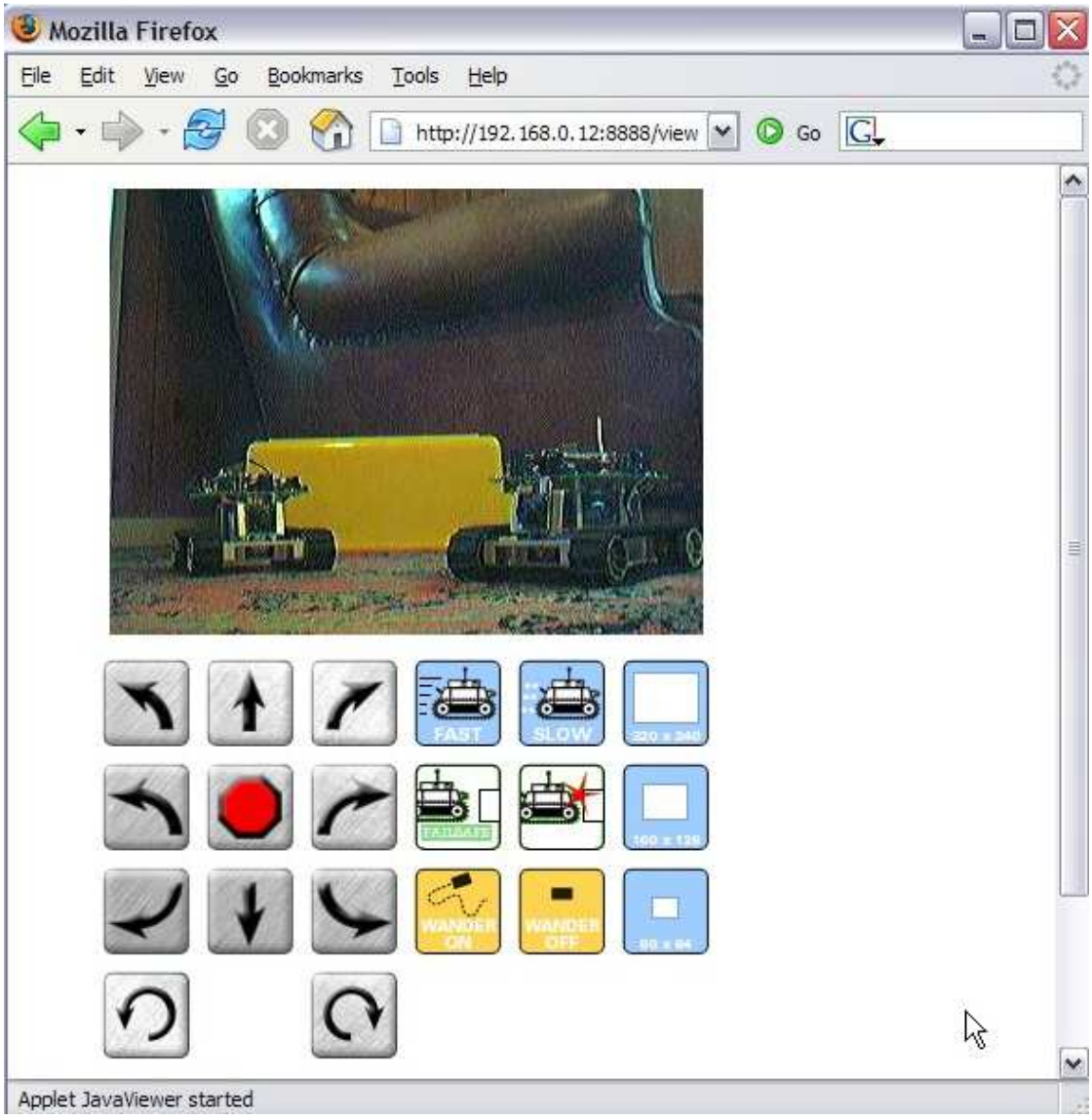


Figura 9.10: Visualización de la captura de la cámara a través de un navegador.

O, mediante la utilización de la SRV1Console, la cual podemos descargar de la página de Surveyor Corporation [2]. Ejecutando esta interfaz grafica y pinchando sobre CONNECT podremos ver la captura de la cámara en tiempo real como se muestra en la figura 9.11.

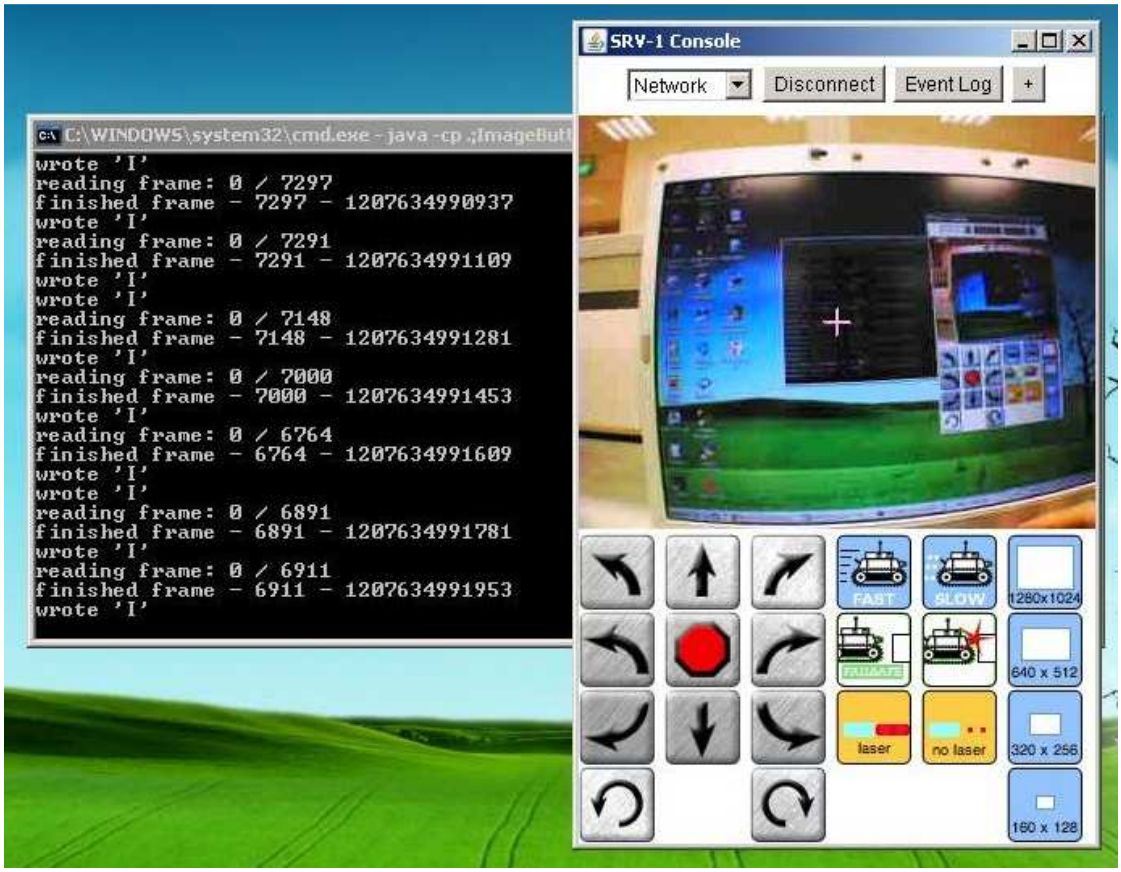


Figura 9.11: Visualización de la captura de la cámara Blackfin a través de la interfaz grafica SRV1Console

Con todo esto, estamos preparados para el desarrollo de los algoritmos de visión en nuestra Blackfin. Únicamente tendremos que pasar los códigos generados en Matlab al lenguaje de programación soportado por nuestra cámara Blackfin, es decir, en µC.

10 Conclusiones

En primer lugar, la realización de este proyecto me ha aportado muchos conocimientos en un campo, hasta ahora desconocido para mí, como es la visión por computador. En general, este proyecto me ha servido para darme cuenta de lo amplio que puede llegar a ser este campo y tras realizar este proyecto, me ha entusiasmado este mundo.

Al mismo tiempo, comentar que me ha sido de gran dificultad la obtención de los objetivos planteados, pero que poco a poco he sabido solventar de manera satisfactoria. Por otro lado, he de destacar que ha sido un proyecto lleno de pros y contras; los pros han sido la obtención satisfactoria de los resultados finales en el desarrollo de nuestros algoritmos de visión, y contras, por los excesivos problemas de conexión con la cámara Blackfin.

Es conveniente que veamos paso a paso las conclusiones, con el fin de ver detalladamente todas las partes de las que consta el proyecto.

En el comienzo de este proyecto nos hemos centrado en el estudio de la normativa del CEABOT, al igual que del estudio de las características de la cámara Blackfin y el robot Bioloid. Mediante estos tres estudios, hemos obtenido factores determinantes a la hora de desarrollar este proyecto, como la altura del robot con la cámara, las dimensiones del entorno del CEABOT y por tanto la elección de la mejor lente para nuestro proyecto.

Cabe destacar, que el estudio teórico de los algoritmos de visión nos ha sido de vital importancia a la hora de desarrollar los algoritmos en Matlab. Es fundamental tener una base teórica firme de los principios fundamentales en el área de visión, además de la comprensión de los algoritmos que hemos implantado, tales como la detección de movimiento y el flujo óptico.

La implementación de los algoritmos de visión en Matlab, nos ha sido de gran ayuda a la hora de aprender como implementar algoritmos a través de la visión por computador. Gracias al cálculo matricial de Matlab y las funciones de visión implantadas en este, hemos podido desarrollar nuestros algoritmos de forma sencilla.

Sin embargo, podemos concluir que nuestros mejores resultados han sido los extraídos de los ensayos en un entorno real, donde hemos actualizado nuestros algoritmos de visión hasta que han sido fiables y de gran utilidad para nosotros.

Los resultados obtenidos del algoritmo de cálculo de distancias han sido satisfactorios, ya que, a través de los ensayos realizados en un entorno real, hemos demostrado la fiabilidad de éste. Lo más destacado de éste algoritmo ha sido los avances que hemos realizado en el ensayo real, gracias a esto hemos podido obtener unos resultados reales consiguiendo así uno de nuestro principales objetivos del proyecto. Una posible ampliación de este algoritmo sería, el estudio para determinar colores, es decir, estudiar cómo podemos distinguir entre varios colores y utilizar esta información para crear un sistema de detección más completo.

Respecto al algoritmo de detección de movimiento concluimos que, pese a su sencillez, es de gran ayuda a la hora de detectar objetos de movimiento. Hay que resaltar la gran estabilidad de este algoritmo debido a la solidez que presenta, respecto de la iluminación. Además, hemos solucionado el problema del movimiento tridimensional, gracias al posicionamiento de la cámara de forma omnidireccional.

Para ampliar este algoritmo podríamos estudiar como detectar varios objetos en movimiento y segmentar la información obtenida para cada uno de ellos.

Del algoritmo de flujo óptico, podemos determinar que, pese a su complejidad, al final hemos conseguido unos resultados bastante convincentes, aunque todavía se vea afectado por la iluminación del entorno o por la textura de este. De todas formas, el flujo óptico ha sido fundamental a la hora de aprender a desarrollar algoritmos de visión, ya que su dificultad nos ha servido para un mayor aprendizaje y por tanto, hemos podido mejorar otros algoritmos más sencillos. Como aplicación, deberíamos incidir en la utilización que se está haciendo del flujo óptico en el mundo de la robótica, tales como el centrado de un robot mediante el flujo óptico lateral o como el cálculo del tiempo de colisión con un obstáculo.

La colocación de la cámara en el robot ha sido vital para poder desarrollar este proyecto, ya que la utilización de un posicionamiento omnidireccional nos ha ayudado a tener un sistema completo de visión además de solucionarnos problemas referidos al posicionamiento en 3D. Podríamos mejorar nuestro sistema de visión, mediante la utilización de un cono reflectante. Para ello la colocación de la cámara debe ser inversa, es decir, con el objetivo hacia arriba, y con el cono sobre el objetivo, de esta forma conseguiremos una visión paralela a la altura del robot, así como una reducción de la altura de este.

Sobre la conexión con la cámara Blackfin debo comentar que ha sido muy complicada ya que los problemas de conexión se han sucedido hasta la conclusión de este proyecto.

Por último, como ampliación a este proyecto, sería interesante el desarrollo del sistema de control del robot Bioloid en sincronización con nuestra cámara Blackfin. De tal forma que utilicemos la cámara como único sensor del robot, al tratarse de un sensor idóneo.

11 Referencias

1. Asociación de Robótica de la Universidad Carlos III de Madrid, [Online, fecha de acceso 26/04/11].

URL: http://asrob.uc3m.es/index.php/Mini-Humanoide

2. *"Surveyor SRV-1 Blackfin Camera"*, Surveyor Corporation, [Online, fecha de acceso 14/04/11]

URL: http://www.surveyor.com/

3. Javier González Jiménez (1999), *"Visión por computador"*. PARANINFO.

URL: http://www.paraninfo.es

4. Gonzalo Pajares & Jesús M. de la Cruz (2007), *"Visión por Computador. Imágenes digitales y aplicaciones"*. Ra-Ma Editorial.

URL: http://www.ra-ma.es

5. Emanuele Trucco & Alessandro Verri (1998), *"Introductory Techniques for 3-D Computer Vision"*, Prentice Hall.

URL: http://www.prenhall.com

6. David Cooms & Karen Roberts (1993), *"Centering Behavior Using Peripheral Vision"*, in proceedings of Computer Vision and Pattern Recognition (CVPR ´93), New York, pp. 440-445.

7. Kahlouche Souhila and Achour Karim (2008). *"Optical Flow based Robot Obstacle Avoidance"*, International Journal of Advanced Robotic Systems, ISSN: 1729-8806, InTech. pp. 13-16.

8. *"MATLAB - The Language of Technical Computing"*. MathWorks, [Online, fecha de acceso 21/03/11].

URL: http://www.mathworks.com/products/matlab/

9. *"hw:jtag:gnice-plus [Analog Devices | Mixed-signal and Digital Signal Processing ICs]"*, Analog Devices, [Online, fecha de acceso 21/04/11].

URL: https://docs.blackfin.uclinux.org/doku.php?id=hw:jtag:gnice-plus

ANEXOS

ANEXO I: Normativa del CEABOT 2010

Comité Español de Automática

V Concurso de Robots Humanoides

Normativa

Alberto Jardón Huete **
Pedro J. Sanz Valero *
Fernando Gómez Bravo ***
Javier Felip León *
Juan C. García Sánchez *
*{cesnep, jfelip, garciaju@uji.es}
**{ajardon@ing.uc3m.es}
***{fernando.gomez@diesio.uhu.es}

Jaén, Septiembre 2010

NORMATIVA GENERAL

Objetivo.

El objetivo del concurso es mostrar las habilidades que cada robot humanoide posee mediante el desarrollo de varias pruebas que serán realizadas por separado. El número y contenido de las pruebas puede variar en cada convocatoria, según recoge la normativa específica de cada edición.

Equipos.

Los equipos podrán estar formados por un máximo de tres personas, no pudiendo pertenecer una misma persona a equipos distintos. En cada equipo habrá una persona que hará de portavoz representando al equipo y a quien se le informará tanto de los posibles cambios en las bases como de las decisiones de los jueces durante el transcurso del concurso.

El portavoz de cada equipo será la persona encargada de depositar y poner en marcha el robot durante el desarrollo de las pruebas. No se podrá cambiar de portavoz durante la competición a no ser que exista una causa de fuerza mayor que lo justifique.

Cada equipo se identifica por el robot o robots que haya inscrito oficialmente,

Cada equipo no puede inscribir más de un robot por cada prueba. Por tanto, el número máximo de robots por cada equipo será menor o igual al de pruebas en cada edición. En caso de mayor número de robots se inscribirán como equipos distintos.

Inscripción.

Los participantes deberán inscribirse por email (concurso.ceabot@gmail.com) indicando el nombre, email y teléfono de cada uno de los miembros del equipo.

Así mismo, indicarán tanto quién va a ejercer de responsable, datos el centro y/o universidad por el que se presentan como las características técnicas de cuantos robots vayan a usar.

Bases.

Estas normas se tienen como fundamentales y se han de respetar. La participación en el “Concurso de Robots Humanoides” implica la total aceptación de estas bases. Se dividen en dos secciones: la primera: NORMATIVA GENERAL, es aplicable en cada edición y rige los aspectos comunes; la segunda: NORMATIVA REGULADORA DE LAS PRUEBAS: se revisa casa año para ajustarla a las nuevas pruebas propuestas.

Cambio de reglas.

Estas bases pueden ser modificadas por la Organización. Esta comunicará a los equipos toda modificación que se pudiese realizar con suficiente antelación (al menos 30 días).

Los jueces.

Los jueces se encargarán de tomar todas las decisiones oportunas durante el transcurso de la competición, referentes a descalificaciones, ganadores o pruebas nulas. Es por tanto, de ellos la última palabra en la interpretación de estas bases.

Expulsión de la competición.

En casos extremos, los jueces se reservan el derecho a expulsar de la competición a quienes se crean merecedores de dicha penalización.

Objeciones.

Los jugadores pueden presentar sus objeciones al árbitro, antes de que acabe el juego, si se tiene cualquier duda en interpretación de las normas.

Presentación oficial.

La presentación oficial es obligatoria. La no asistencia implica la descalificación del equipo. Se realizará durante la celebración del concurso, y siempre antes de que comience la competición de sumo. En este acto se realizarán los sorteos de grupo, así como la explicación del desarrollo de la competición.

Excepciones.

En caso de que ocurra cualquier circunstancia no contemplada en los artículos anteriores de la prueba, los jueces adoptarán la decisión oportuna.

Robots.

Los robots han de tener una constitución antropomórfica, es decir dos piernas un tronco y dos brazos articulados. La altura máxima es de 50 cm. y la máxima longitud del pie de 11 cm. Se entiende por altura máxima la distancia desde el suelo a la parte más alta del robot cuando éste se encuentra completamente estirado. Se entiende por máxima longitud del pie a la distancia entre sus dos puntos más alejados. En la Figura 1 se puede ver algunos ejemplos de esto. El peso máximo permitido es de 3 Kg.

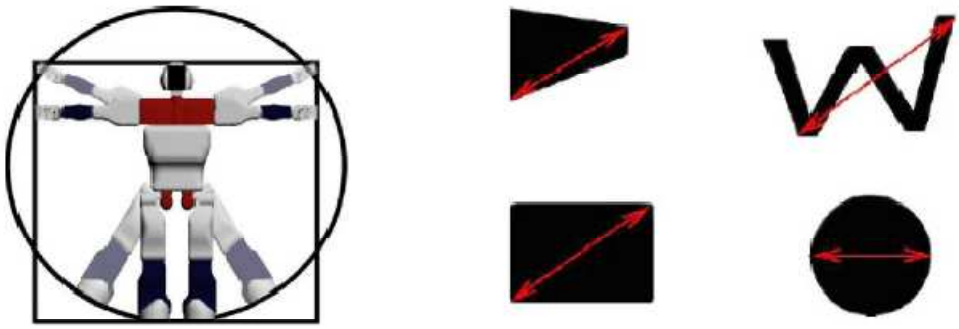


Figura 1: Ejemplo de Humanoide antropomórfico, y máxima longitud del pie.

Locomoción.

El modo de locomoción deberá ser andar o correr a dos patas, no pudiéndose utilizar ruedas, patines o similares.

Autonomía.

Cada robot debe ser completamente autónomo a nivel de locomoción, sensorización y procesamiento. Actuadores, sensores, energía y procesado deben estar incorporados en el robot, debiendo éste tomar sus propias decisiones.

No se podrá dar ninguna instrucción directa o indirectamente al robot tras encenderlo. Además, tras encender el robot, este deberá esperar 5 segundos antes de realizar cualquier movimiento.

Modificaciones sobre el robot.

- Las pruebas son llevadas a cabo por un robot humanoide por cada equipo.
- Los equipos podrán utilizar un robot diferente para cada prueba, siempre y cuando en la inscripción se hayan inscrito todos los robots que presenta en el equipo. Por tanto, los puntos para cada prueba se imputan al equipo.
- Sólo se podrá cambiar de robot durante una prueba, en caso de incapacitación del primer robot utilizado para dicha prueba y con el consentimiento del jurado.
- El código del robot no podrá ser modificado una vez haya comenzado cada una de las pruebas de la competición. Para ello, los robots deberán permanecer en la mesa del jurado hasta el momento de su participación.

Seguridad.

- El robot no puede poseer ningún elemento que suponga un peligro para él, los otros robots, el campo de pruebas o las personas.

NORMATIVA PARTICULAR, V EDICIÓN 2010

Reparto de puntuación entre pruebas

Durante las ediciones pasadas se ha comprobado que la preparación precisa para las pruebas de habilidad/movilidad es mayor que para la de sumo, por lo a partir de esta edición se da más peso a estas pruebas frente la de sumo quedando en 70% / 30%.

- La prueba de movilidad supone un 35% (Prueba 1) del total de puntos posibles.
- La prueba de habilidad supone un 35% (Prueba 2) del total de puntos posibles.
- Por lo tanto, la prueba de sumo supone un 30% de los puntos totales.

Prueba 1: Carrera de obstáculos

Artículo 1.1. Objetivo.

Los robots irán desde un extremo del campo al otro, y vuelta al punto de partida, caminando de cara. El robot deberá esquivar los obstáculos, sin tirarlos ni desplazarlos de su posición.

Los robots saldrán desde la zona central, situada en la Zona de Salida, debiendo llegar a la Zona de Llegada Parcial. Una vez allí, el robot deberá darse la vuelta de forma autónoma e iniciar el proceso de vuelta, una vez haya traspasado por completo la línea de la Zona de Llegada Parcial. Se puntuarán tanto el tiempo de llegada a la Zona de Llegada Parcial, como el tiempo total.

Los jueces decidirán la posición de los obstáculos a esquivar, antes de cada uno de los dos intentos de los que disponen los equipos. La configuración de los obstáculos, será igual para todos los equipos en cada intento. En el apéndice, se pueden observar algunas configuraciones posibles y algunos casos especiales a tener en cuenta en la programación del robot.

Habrá un máximo de 6 obstáculos paralepípedos rectángulos, cuyas medidas serán de 20x20x50cm. (base x altura) y su color será blanco.

Novedad: Se ha reducido el tamaño de la base de los obstáculos, para facilitar el paso de los robots, que en ocasiones era muy difícil superarlos sin tocar los obstáculos.

Artículo 1.2. Campo.

El campo de pruebas es una superficie nivelada, plana y rígida, de 2.5 m. de largo por 2 m. de ancho, formado típicamente por tableros de conglomerado y revestimiento de melamina o pintados. El color de la superficie será verde y homogéneo en la medida de lo posible, (Pantone Code: 16C606 (R:22:G:198:B:6)).

El campo está dividido por líneas blancas en tres zonas como se aprecia la figura 2. Alrededor del campo habrá una pared de 50 cm. de altura y 1 cm. de grosor, de color blanco.

Nota: Es posible que existan otras marcas o líneas correspondientes al campo de sumo, y/o que el campo esté compuesto por tableros ensamblados entre sí. En la medida de lo posible las uniones entre tableros se realizarán con machihembrado o centradores para que no presenten escalones y desniveles entre ellos.

El campo estará iluminado con luz artificial de interior, que será lo más uniforme posible. Se debe tener en cuenta que durante la realización de las pruebas puede haber flashes procedentes de cámaras fotográficas de público o prensa y alterar posibles sistemas de visión a bordo de los robots.

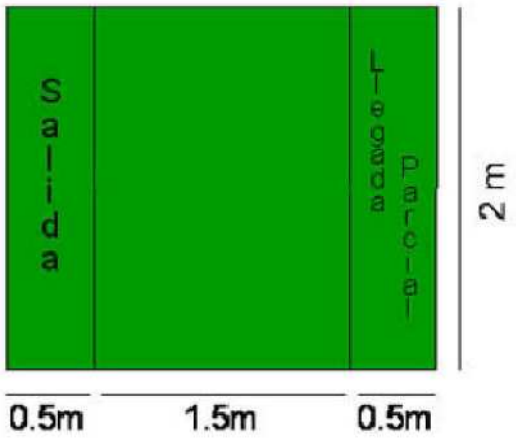


Figura 2: Esquema del Campo durante la Prueba 1.

Artículo 1.3. Tiempo máximo.

El tiempo máximo para esta prueba es de 5 minutos en total. El tiempo comenzará a contar cuando el robot, después de realizar la pausa de 5 segundos, se ponga en movimiento. Se considera que el robot ha terminado un parcial cuando haya cruzado completamente la línea de final de ese tramo.

Artículo 1.4. Manipulación del robot y penalizaciones.

Se obtendrá una penalización por cada manipulación del robot, y por desplazar o tirar los obstáculos a sortear. Se considerará que se ha desplazado un obstáculo, cuando se haya movido su centro más de 5 cm. Tras esto el juez encargado de la prueba volverá a colocar el obstáculo en la posición original, desplazando al robot al menos 10 cm para que pueda reiniciar la maniobra.

Artículo 1.5. Puntuación.

En primer lugar se tendrán en cuenta el número de penalizaciones, de menor a mayor. En caso de empate a penalizaciones, se tendrá en cuenta el tiempo para decidir las posiciones finales de los equipos.

Se puntuará, por una parte teniendo en cuenta el orden de llegada a la zona Llegada Parcial y por otra parte el orden de llegada a la Salida. La puntuación se muestra en el cuadro 1. Si un robot no llega siquiera a la Llegada Parcial no recibirá ningún punto. Si no llega al final pero sí realiza el primer tramo recibirá únicamente los puntos correspondientes a este tramo.

Posición Llegada parcial	Puntos
1º	10
2º	8
3º	7
4º	6
5º	5
6º	4
7º	3
8º	2
9º	1

Posición Llegada final	Puntos
1º	25
2º	21
3º	18
4º	15
5º	12
6º	9
7º	6
8º	3
9º	1

Cuadro 1: Puntuaciones para la primera prueba.

Por tanto, el robot que complete LIBRE DE PENALIZACIONES los dos recorridos completos en un tiempo menor que sus oponentes, recibirá 35 puntos.

Prueba 2: Escalera

Artículo 2.1. Objetivo.

En esta prueba, se añade una escalera al escenario de la primera prueba, una vez retirados los obstáculos.

Para superar la prueba, los robots deberán alcanzar la Zona de Llegada indicada por el juez, superando una serie de escalones de subida y de bajada.

Se puntuará tanto el número de escalones superados como el tiempo empleado.

Los robots deberán subir y bajar la escalera caminando, no siendo permitido ningún tipo de salto o acrobacia. La escalera solo se recorre en un sentido, siendo éste elegido por los jueces. Se finalizará la prueba cuando se haya sobrepasado totalmente la línea de Salida o Llegada Parcial, según el sentido de comienzo de la prueba indicado por los jueces. Se puntuará la habilidad de superar la escalera de forma autónoma, valorando el que el robot supere escalones sin que caiga o se desvíe y penalizando cualquier intervención por parte del portavoz del equipo.

Artículo 2.2. Campo.

Los jueces decidirán la colocación de la escalera antes de cada ronda de intentos, teniendo en cuenta que uno de los extremos deberá cubrir completamente una de las líneas sin sobrepasarla. De este modo, quedarán por el otro extremo 20 cm. hasta la línea de Salida o Llegada Parcial.

Las escaleras tendrán unos escalones de 3 cm. de altura y de longitud 25, 15 y 50 cm. según se indica en la figura 3. Además las escaleras, tendrán un ancho de 99 cm.).

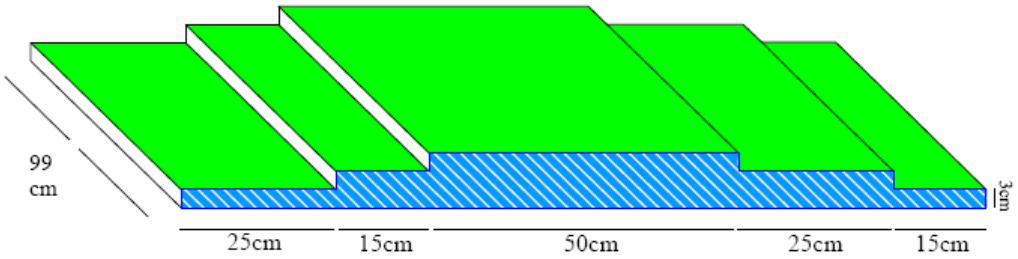


Figura 3: Esquema de las escaleras.

Artículo 2.3. Tiempo máximo.

El tiempo máximo de cada carrera es de 5 minutos. El tiempo comenzará a contar cuando el robot, después de realizar la pausa de 5 segundos, se ponga en movimiento. Se considera que el robot ha terminado un parcial cuando haya cruzado completamente la línea de final de ese tramo.

Artículo 2.4. Manipulación del robot y penalizaciones.

En caso que el robot caiga y no pueda levantarse, el representante del equipo colocará el robot siempre sobre el peldaño desde dónde ha caído, obteniendo una penalización.

Si el robot toca con mano/brazo la superficie de la escalera será penalizado aunque no llegue a caer. Nótese que si el robot, cae pero se recupera y vuelve a quedarse de pie en el mismo escalón del que se cayó no será penalizado.

Artículo 2.5. Puntuación.

En primer lugar se tendrán en cuenta el número de penalizaciones, de menor a mayor. En caso de empate a penalizaciones, se tendrá en cuenta, primero el número de escalones superadas y a igualdad de estos, el tiempo para decidir las posiciones finales de los equipos.

Se puntuará, por una parte teniendo en cuenta el orden de llegada a la zona Llegada Parcial. La puntuación se muestra en el cuadro 2. Si un robot no llega a la Llegada Parcial, solo recibirá puntos por los escalones superados.

Posición Llegada parcial	Puntos
1º	10
2º	8
3º	7
4º	6
5º	5
6º	4
7º	3
8º	2
9º	1

Nº escalones superados	Puntos
1	3
2	5
3	10
4	18
5	22
6	25

Cuadro 2: Puntuaciones para la segunda prueba.

Por tanto, el robot que complete LIBRE DE PENALIZACIONES todos los escalones (6) en menos tiempo que sus oponentes, recibirá 35 puntos.

Prueba 3: Lucha (Sumo)

Articulo 3.1. Objetivo de la prueba.

En la prueba luchan dos robots de dos equipos diferentes, dentro del Área de Combate según las normas que a continuación se expondrán, para obtener puntos

efectivos (llamados puntos Yuhkoh). Se valora el comportamiento competitivo del robot, pudiéndose penalizar actitudes pasivas e inmóviles.

Artículo 3.2. Definición del área de combate.

Se denomina Área de Combate a la tarima de juego (Ring). Cualquier espacio fuera del Área de Combate se denomina Área Exterior, que deberá ser de al menos 0.5 m. alrededor del Ring (Figura 3).

Artículo 3.3. Ring de sumo.

El Ring será circular, de color verde, homogéneo en la medida de lo posible y de 150 cm. de diámetro. Señalando el límite exterior del Ring, habrá una línea blanca o amarilla circular de 5 cm. de ancho. El ring de sumo podrá ubicarse en el en centro del campo usado para las pruebas.

En el centro del Ring habrá dos líneas paralelas separadas 20 cm., llamadas líneas Shikiri. Las líneas Shikiri serán de color negro o blanco de 2 cm. de ancho y 20 cm. de largo. Estas líneas marcarán las posiciones iniciales de los robots.

El campo estará iluminado con luz artificial de interior, que será lo más uniforme posible.

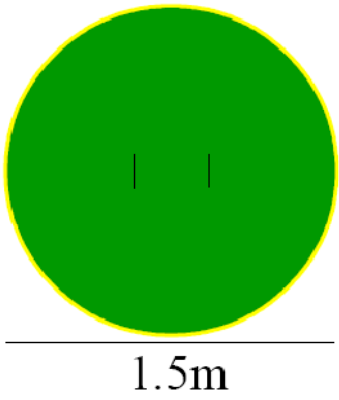


Figura 3. Estructura del Ring

Artículo 3.4. Combates de sumo.

- Los combates consistirán en 2 asaltos de 3 minutos cada uno. Entre asalto y asalto habrá un tiempo máximo de 1 minuto.

- Para el comienzo del combate se llamará a los dos equipos participantes. Se realizarán como máximo tres avisos, y si en el plazo de 1 minuto desde el último aviso uno de los equipos no compareciera se otorgaría directamente la victoria al equipo compareciente.

- Ganará el combate el robot con más puntos Yuhkoh en el total de los dos asaltos. En caso de empate a puntos, se realizará un asalto extra donde el ganador será el primero que consiga un punto Yuhkoh. De mantenerse el empate, al no puntuar ninguno, los árbitros decidirán el ganador del asalto.

Artículo 3.5. Rutina del combate.

Siguiendo las indicaciones de los jueces, los equipos se saludarán en el Área Exterior, a continuación, sólo entrará el responsable del equipo en el Área de Combate

y situará el robot inmediatamente detrás de la línea Shikiri, a la derecha o izquierda según decidan los jueces.

Cuando el juez lo indique se activarán los robots, que deberán permanecer parados durante 5 segundos. Tras dicha pausa, comenzará el asalto.

Únicamente se podrá acceder dentro del Área de Combate cuando el asalto esté parado y/o den permiso los jueces. Cuando el árbitro dé por finalizado el combate, los dos responsables de equipo retirarán los robots del Área de Combate.

Artículo 3.7. Puntos Yuhkoh.

Se otorgarán puntos Yuhkoh cuando:

- El robot contrario toca el espacio fuera del Ring, 1 punto.
- El robot contrario toca con alguna mano el suelo sin caer, 1 punto.
- El robot contrario cae al suelo por si mismo, 1 punto.
- Por tumbar al robot contrario dentro o fuera del Ring, 2 puntos.

Cuando se cumplan varias condiciones, sólo se otorgarán los puntos de una de ellas, siendo ésta la de mayor puntuación. Sólo se otorgarán puntos a uno de los robots a la vez. En caso de duda, se otorgarán al robot que inició la acción. En caso de empate, no se otorgará ningún punto.

Artículo 3.8. Parada del combate.

Cada asalto durará tres minutos, pudiéndose parar y reanudarse hasta agotar el tiempo, cuando:

- Cuando el juez otorgue algún punto Yuhkoh.
- Los dos robots permanezcan 30 seg. sin moverse.
- Los dos robots permanezcan 30 seg. sin tocarse.
- Los dos robots permanezcan 30 seg. empujándose pero sin que el movimiento favorezca a ninguno de los equipos.

Para reanudar el combate, tras cada asalto o tras una pausa, se colocarán de nuevo los robots en las líneas Shikiri.

Artículo 3.9. La organización del concurso.

El número de equipos por grupo y las clasificaciones que dan derecho a pasar a la fase final, se decidirán en función del número de inscritos, y se comunicarán antes del comienzo de la prueba.

Los grupos y turnos de combate se sortearán antes del comienzo de la prueba. Dependiendo del número de rondas que se realicen (fases previas, octavos, cuartos, etc.) se podrán establecer cabezas de serie, de forma que sea mayor la competitividad.

Artículo 3.10. Puntuación.

La puntuación de las pruebas eliminatorias, será como sigue:

Posición	Puntos
1º	30
2º	25
3º	21
4º	18
5º	15
6º	12
7º	9
8º	6
9º	3

Cuadro 3: Puntuación de la tercera prueba

En caso de quedar campeón en la prueba de sumo, el robot recibirá 30 puntos; Los robots calificados por debajo del noveno no reciben puntos en esta prueba.

PRUEBA LIBRE -EXHIBICIÓN

El objetivo de esta prueba es que los equipos que lo deseen puedan demostrar las habilidades y capacidades que han sido capaces de programar en su robot humanoide, bien porque sean diferentes de aquellas demostradas en las pruebas regladas, o bien por que los robots excedan alguno de los parámetros de tamaño y peso especificados en la normativa. El tema o exhibición será libre y la única limitación será el tiempo, que no superará los 5 minutos. Será el equipo inscrito en esta prueba el responsable de la infraestructura necesaria para el desarrollo de la misma.

Para participar en la modalidad de prueba libre será preciso en el momento de la inscripción notificarlo expresamente, especificando el tipo de robot (dimensiones y breve descripción del mismo si no es un modelo comercial) y habilidades o pruebas que van a realizar. Durante la ejecución del concurso, el jurado asignará un horario para la realización de estas exhibiciones en función del número de inscripciones.

EQUIPO GANADOR

Tras completar las tres pruebas el equipo el jurado presentará el recuento de puntos otorgados a cada equipo en cada prueba y que haya obtenido más puntos será nombrado ganador.

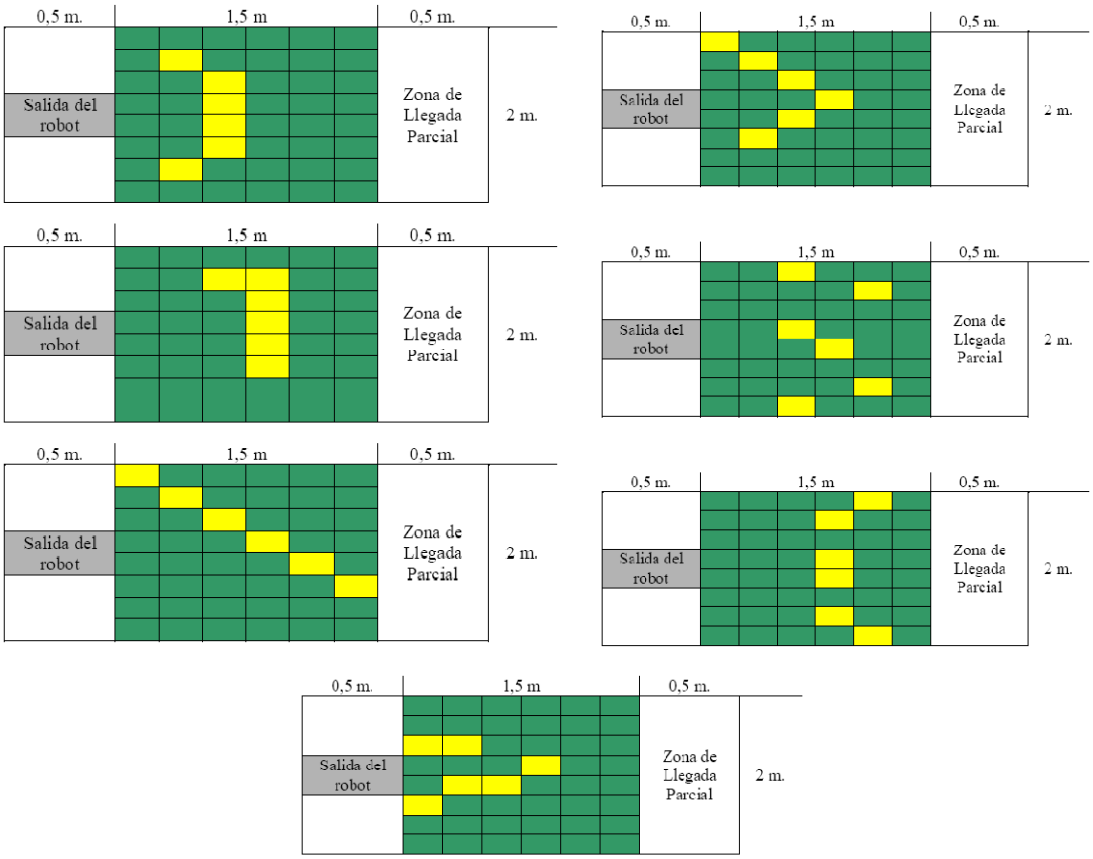
La organización se reserva el derecho de otorgar un reconocimiento al equipo que haya causado mejor impresión en la realización de la prueba libre.

Apéndice: Configuraciones para la carrera de obstáculos

Tal y como se indica en la normativa de la prueba, los jueces elegirán una combinación de obstáculos, antes de cada intento. Los participantes, cogerán el robot (que deberá estar encima de la mesa del jurado) y sin ningún tipo de comando externo (es decir, no se puede usar el mando a distancia) activarán el robot. Una vez finalizado el intento, el robot volverá a colocarse encima de la mesa de los jueces. Cuando acabe la primera tanda de intentos, los jueces modificarán la posición de los obstáculos y se procederá a la siguiente tanda.

A continuación se muestran posibles configuraciones del campo, a tener en cuenta por los participantes, los jueces tienen plena capacidad para distribuir los obstáculos según crean conveniente sin necesidad de utilizar alguna de las configuraciones que se

exponen a continuación. Las celdas en amarillo indican la localización aproximada de un obstáculo.



ANEXO II: Código fuente del calculador de distancias.

```
%% ********************************************************************
%  Código de cálculo de distancias para una cámara omnidirreccional
%  Realizado por: Félix Rodríguez Cañadillas
%  Universidad Carlos III de Madrid
%
********************************************************************
clear all;
sigmaSpatial = 1.5;
maskWidthSpatial = 9;

%% Leo la imagen y la paso a binario
% bw_img = im2bw(imread('Vision4.jpg'));
% figure
% pause(1)
% imshow(bw_img)
% center_image = [size(bw_img,1)/2 ; size(bw_img,2)/2];

%% Leo la imagen en RGB
img(:,:,:) = imread('Vision1.jpg');
% figure
% pause(1)
% imshow(img)

%% Filtrado espacial de la imagen
for i = 1:1:3
    filt_img = mat2gray(spatial_gaussian_filter(img(:,:,i),
sigmaSpatial, maskWidthSpatial));
%     figure
%     pause (1)
%     imshow(filt_img);%title('Imagen Filtrada')
%   saveas(h(1),['output/ImageFilt' int2str(i) '.jpg'])
end

%% Paso imagen filtrada a binario
[umb EM] = graythresh(filt_img)
bw_img = im2bw(filt_img, umb);
figure
pause (1)
imshow(bw_img);
center_image = [size(bw_img,1)/2 ; size(bw_img,2)/2];

%% Calculo los limites de las regiones (por ejemplo, cada 15º;
Nregion = 24)
Nregion = 24;
for i = 1:1:Nregion
   Ang(i) = (2*pi/Nregion)*i;
end

%% Busco el pixel de valor 1 por cada recta
% Primero compruebo los valores que crean error:
% Ang(i) = n*pi/2 siendo n = 1,2,3,4.
% Posteriormente para todas las rectas
hold on
for i =1:1:Nregion
    if (Ang(i) == pi/2)
        y = center_image(1);
        for x = center_image(2):1:center_image(2)*2
```

```
            if (bw_img(y,x) == 1)
                plot([center_image(2);x],[center_image(1);y],'r')
                Dist(i) = x - center_image(2);
                break
            else
                Dist(i)=0;
            end
        end
    elseif (Ang(i) == pi)
        x = center_image(2);
        for y = center_image(1):-1:1
            if (bw_img(y,x) == 1)
                plot([center_image(2);x],[center_image(1);y],'r')
                Dist(i) = center_image(1) - y;
                break
            else
                Dist(i)=0;
            end
        end
    elseif (Ang(i) == 3*pi/2)
        y = center_image(1);
        for x = center_image(2):-1:1
            if (bw_img(y,x) == 1)
                plot([center_image(2);x],[center_image(1);y],'r')
                Dist(i) = center_image(2) - x;
                break
            else
                Dist(i)=0;
            end
        end
    elseif (Ang(i) == 2*pi)
        x = center_image(2);
        for y = center_image(1):1:center_image(1)*2
            if (bw_img(y,x) == 1)
                plot([center_image(2);x],[center_image(1);y],'r')
                Dist(i) = y - center_image(1);
                break
            else
                Dist(i)=0;
            end
        end
    else
                if (Ang(i) < (pi/2) || Ang(i) > (3*pi/2))
            for y_aux = 1:1:center_image(1)
                x_aux = y_aux * tan(Ang(i));
                x_aux = floor(x_aux);
                if (x_aux >= center_image(2) || x_aux <= -
center_image(2) || y_aux >= center_image(1))
                    Dist(i) = 0;
                    break
                end
                x = x_aux + center_image(2);
                y = y_aux + center_image(1);
                if (bw_img(y,x) == 1)
                    plot([center_image(2);x],[center_image(1);y],'r')
                    Dist(i) = sqrt(x_aux^2 + y_aux^2);
                    break
                end
            end
        elseif ((pi/2) < Ang(i) < (3*pi/2))
```

```
            for y_aux = -1:-1:-center_image(1)+1
                x_aux = y_aux * tan(Ang(i));
                x_aux = floor(x_aux);
                x_aux = x_aux + 1;
                if (x_aux >= center_image(2) || x_aux <= -
center_image(2) || y_aux <= -center_image(1))
                    Dist(i) = 0;
                    break
                end
                x = center_image(2) + x_aux;
                y = center_image(1) + y_aux;
                if (bw_img(y,x) == 1)
                    plot([center_image(2);x],[center_image(1);y],'r')
                    Dist(i) = sqrt(x_aux^2 + y_aux^2);
                    break
                else
                    Dist(i) = 0;
                end
            end
        end
    end
end
```

ANEXO III: Código fuente del detector de movimiento.

```
%% ********************************************************************
%  Código de detección de movimiento para una cámara omnidirreccional
%  Realizado por: Félix Rodríguez Cañadillas
%  Universidad Carlos III de Madrid
%
********************************************************************
clear all;
%% Leo y paso a escala de grises
counter = 1;
for i = 0:6:12
    Im_gris(:,:,counter) = rgb2gray(imread(['Movimiento2/move'
int2str(i) '.jpg']));
    counter = counter +1;
end
% h = figure();
% imshow(i1(:,:));title('Detector de movimiento 2')
% saveas(h(1),'output/im1.jpg')
% pause (1)

%% Resto secuencialmente las imágenes
Im_move(:,:,1)=abs(double(Im_gris(:,:,1))-
double(Im_gris(:,:,2)))/256;
Im_move(:,:,2)=abs(double(Im_gris(:,:,2))-
double(Im_gris(:,:,3)))/256;
% h = figure();
% Imshow(Im_move(:,:,1));%title('Detector de movimiento 1')
% saveas(h(1),'output/ImageMovegray1.jpg')
% pause (1)
% h = figure();
% imshow(Im_move(:,:,2));%title('Detector de movimiento 2')
% saveas(h(1),'output/ImageMovegray2.jpg')
% pause (1)

%% Filtro para depurar la imagen
    sigmaSpatial = 1;
    maskWidthSpatial = 9;
for i = 1:1:2
    Im_filtrada(:,:,i) = spatial_gaussian_filter(Im_move (:,:,i) ,
sigmaSpatial, maskWidthSpatial);
%     h = figure();
%     imshow(Im_filtrada(:,:,i));%title('Imagen Filtrada')
%     saveas(h(1),['output/ImageFilt' int2str(i) '.jpg'])
%     pause (1)
end

%% Umbral Threshold y paso a binario
for i = 1:1:2
    umb=graythresh(Im_filtrada(:,:,i));
    umb_Input = umb
    Im_bordes(:,:,i)= im2bw(Im_filtrada(:,:,i), umb);
%     h = figure();
%     imshow(Im_bordes(:,:,i));%title('Imagen Final')
%     saveas(h(1),['output/ImageMovebw' int2str(i) '.jpg'])
%     pause (1)
end
```

```
%% Etiquetar elementos conectados y calculo sus propiedades
for i =1:1:2
    [L N]=bwlabel(Im_bordes(:,:,i));
    if (i == 1)
    propied1 = regionprops(L)
    N1 = N;
    elseif (i == 2)
    propied2 = regionprops(L)
    N2 = N;
    end
end

%% Calculo el area total
for i = 1:1:2
    if (i == 1)
        Area_T1 = 0;
        for n = 1:1:N1
            Area_T1 = Area_T1 + propied1(n).Area;
        end
    elseif (i == 2)
        Area_T2 = 0;
        for n = 1:1:N2
            Area_T2 = Area_T2 + propied2(n).Area;
        end
    end
end
Area_T1
Area_T2
%% Calculo el sumatorio del producto del centroide y la
proporcionalidad
for i = 1:1:2
    if (i == 1)
        Centroide1 = 0;
        for n = 1:1:N1
            Centroide1 = Centroide1 + (propied1(n).Area/Area_T1) *
propied1(n).Centroid;
        end
    elseif (i == 2)
        Centroide2 = 0;
        for n = 1:1:N2
            Centroide2 = Centroide2 + (propied2(n).Area/Area_T2) *
propied2(n).Centroid;
        end
    end
end
Centroide1
Centroide2
h = figure();
imshow(Im_gris(:,:,i));%title('Imagen Final')
saveas(h(1),['output/ImageFinal' int2str(i) '.bmp'])
pause (1)
imshow(Im_gris(:,:,3))
hold on
plot ([Centroide1(1);Centroide2(1)], [Centroide1(2);Centroide2(2)]
,'-r',...
'LineWidth',2.5)
```

ANEXO IV: Código fuente del flujo óptico.

```
%% ****************************************************************
%  Código de flujo óptico para una cámara omnidirreccional
%  Realizado por: Félix Rodríguez Cañadillas
%  Universidad Carlos III de Madrid
%
****************************************************************
%% Inicialización:
clear all;
sizeX_section = 320;
sizeY_section = 240;

% Variables del filtro gaussiano
sigmaSpatial = 1.5;
maskWidthSpatial = 9;

% Variables para el algoritmo de flujo óptico constante
areaSize = 5;
pInvTolerance = 1E-66;

% Variables para la media de los componentes del flujo óptico
flowAveragingFlag = 1; % 1 - use averaged flow
% sigmaAveraging = 1.5;
maskWidthAveraging = 9;

%% Leo las imágenes y paso a escala de grises
counter = 1;
for i = 4:1:8
    mat_inputImages(:,:,counter) = rgb2gray(imread(['FlujoRobot/move'
int2str(i) '.jpg']));
    counter = counter + 1;
end

%% Filtrado espacial de las imagenes
for i = 1:1:size(mat_inputImages,3)
    mat_spatialImages(:,:,i) =
spatial_gaussian_filter(mat_inputImages(:,:,i), sigmaSpatial,
maskWidthSpatial);
end

%% Filtrado temporal de las imagenes
mat_Images(:,:,1) =
temporal_gaussian_filter(mat_spatialImages(:,:,1:1:3));
mat_Images(:,:,2) =
temporal_gaussian_filter(mat_spatialImages(:,:,2:1:4));
mat_Images(:,:,3) =
temporal_gaussian_filter(mat_spatialImages(:,:,3:1:5));

% % % No temporal filtering
% mat_images(:,:,1) = mat_spatialImages(:,:,2);
% mat_images(:,:,2) = mat_spatialImages(:,:,3);
% mat_images(:,:,3) = mat_spatialImages(:,:,4);
% % % % % DEBUG
% %h = imtool(mat_images(:,:,1), [0 255]);
% h = figure();
% imshow(mat_images(:,:,1), [0 1]);
% saveas(h(1),['output/image_filtered_1.fig'])
% saveas(h(1),['output/image_filtered_1.png'])
```

```
% %h = imtool(mat_images(:,:,2), [0 255]);
% h = figure();
% imshow(mat_images(:,:,2), [0 1]);
% saveas(h(1),['output/image_filtered_2.fig'])
% saveas(h(1),['output/image_filtered_2.png'])
% %h = imtool(mat_images(:,:,3), [0 255]);
% h = figure();
% imshow(mat_images(:,:,3), [0 1]);
% saveas(h(1),['output/image_filtered_3.fig'])
% saveas(h(1),['output/image_filtered_3.png'])
% % % % % END DEBUG

%% Busco las regiones donde aplicar el flujo óptico constante y lo
aplico
%% para cada región
% Loop through all areas in image except (n+1)/2 pixels along edges
xCounter = 1;
border = (areaSize + 1)/2;
for x = (border+1):areaSize:(size(mat_Images,2) - border)
%for x = (border+1):1:(size(mat_Images,2) - border)

   yCounter = 1;
   for y = (border+1):areaSize:(size(mat_Images,1) - border)
   %for y = (border+1):1:(size(mat_Images,1) - border)

       xStart = x - border;
       xEnd = x + border;
       yStart = y - border;
       yEnd = y + border;

       % ***** Get 5+2 by 5+2 patch around current pixel *****
       mat_patchWithBorders = mat_Images(yStart:1:yEnd,
xStart:1:xEnd, 1:1:3);

       % /***** Apply CONSTANT_FLOW algorithm to 5 by 5 patch *****
       mat_v(:,yCounter,xCounter) =
constant_flow_patch(mat_patchWithBorders, pInvTolerance);


% Also taking into account reduction in size due to spatial filtering
       mat_pos(:,yCounter,xCounter) = [x+((maskWidthSpatial-1)/2);
y+((maskWidthSpatial-1)/2)];

       yCounter = yCounter + 1;
   end
   xCounter = xCounter + 1;
end

% Matrices of optical flow components
mat_vX(:, :) = mat_v(1,:,:);
mat_vY(:, :) = mat_v(2,:,:);

% Size of optical flow field
sizeX = size(mat_vX,2);
sizeY = size(mat_vX,1);
```

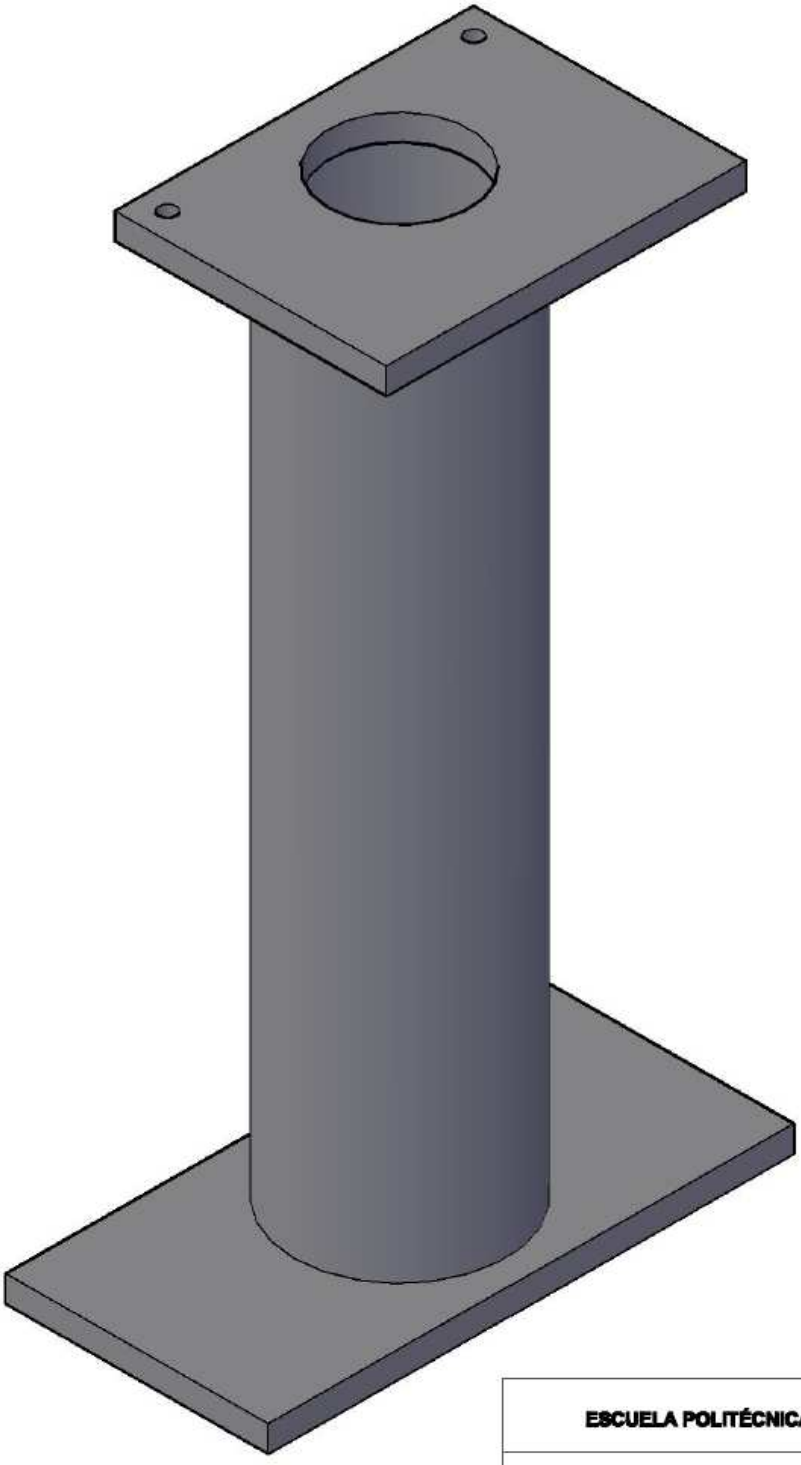
```
% Matrix of optical flow component magnitude
for x = 1:1:sizeX
    for y = 1:1:sizeY
        mat_vSize(y,x) = sqrt(mat_vX(y,x)^2 + mat_vY(y,x)^2);
    end
end

% Matrices of normalized flow components
mat_vSizeNormalized = mat_vSize/max(max(mat_vSize));
mat_vXNormalized = mat_vX/max(max(mat_vX));
mat_vYNormalized = mat_vY/max(max(mat_vY));


% ***** Averaged optical flow components *****
if flowAveragingFlag == 1
%     mat_vX_av(:,:) = spatial_gaussian_filter(mat_vX,
sigmaAveraging, maskWidthAveraging);
%     mat_vY_av(:,:) = spatial_gaussian_filter(mat_vY,
sigmaAveraging, maskWidthAveraging);

    mat_vX_av(:,:) = spatial_averaging(mat_vX, maskWidthAveraging);
    mat_vY_av(:,:) = spatial_averaging(mat_vY, maskWidthAveraging);

%     mat_vX_av(:,:) = spatial_median(mat_vX, maskWidthAveraging);
%     mat_vY_av(:,:) = spatial_median(mat_vY, maskWidthAveraging);

    sizeX_av = size(mat_vX_av,2);
    sizeY_av = size(mat_vX_av,1);

    for x = 1:1:sizeX_av
        for y = 1:1:sizeY_av
            mat_vSize_av(y,x) = sqrt(mat_vX_av(y,x)^2 +
mat_vY_av(y,x)^2);
        end
    end
    temp_i = (maskWidthAveraging - 1)/2;
    mat_pos_av(1,:,:) =
mat_pos(1,(1+temp_i):1:(sizeY_av+temp_i),(1+temp_i):1:(sizeX_av+temp_
i));
    mat_pos_av(2,:,:) =
mat_pos(2,(1+temp_i):1:(sizeY_av+temp_i),(1+temp_i):1:(sizeX_av+temp_
i));
else
    mat_vX_av = mat_vX;
    mat_vY_av = mat_vY;
    sizeX_av = sizeX;
    sizeY_av = sizeY;
    mat_vSize_av = mat_vSize;
    mat_pos_av = mat_pos;
end




% Surface plot of optical flow intensity
```

```
h = figure();
hold on;
mat_pos_av_x(1:1:sizeY_av,1:1:sizeX_av) =
mat_pos_av(1,1:1:sizeY_av,1:1:sizeX_av);
mat_pos_av_y(1:1:sizeY_av,1:1:sizeX_av) =
mat_pos_av(2,1:1:sizeY_av,1:1:sizeX_av);
surf(mat_pos_av_x, size(mat_inputImages,1)-mat_pos_av_y,
mat_vSize_av), colorbar;
%plot3(p_0_im(1), p_0_im(2), 3, 'Markersize', 30, 'Color', 'r');
hold off;
axis([1 size(mat_Images,2) 1 size(mat_Images,1) 0
max(max(mat_vSize_av))]);
xlabel('x (pixel)');
ylabel('y (pixel)');
view(0, 90);
saveas(h(1),['output/flow_surface.fig'])
saveas(h(1),['output/flow_surface.bmp'])

% Vector field plot of optical flow

h = figure();
vfield(mat_pos_av(1,:,:), size(mat_inputImages,1)-mat_pos_av(2,:,:),
mat_vX_av, -mat_vY_av, mat_vSize_av, 'fill', 1, colorbar);
axis([1 size(mat_Images,2) 1 size(mat_Images,1)]);
axis equal;
xlabel('x (pixel)');
ylabel('y (pixel)');
saveas(h(1),['output/flow_field.fig'])
saveas(h(1),['output/flow_field.bmp'])

% h = figure();
% hold on
% imshow(imread(['FlujoLateral/Flujo3.jpg']))
% %imshow(mat2gray(mat_Images(:,:,1)));
% vfield(mat_pos_av(1,:,:), mat_pos_av(2,:,:), mat_vX_av, mat_vY_av,
mat_vSize_av, 'fill', 1, colorbar);
% axis([1 size(mat_Images,2) 1 size(mat_Images,1)]);
% axis equal;
% xlabel('x (pixel)');
% ylabel('y (pixel)');
% hold off;

%% ********************************************************************
%  Funciones utilizadas en el flujo óptico para una cámara
%  omnidirreccional
%  Realizado por: Félix Rodríguez Cañadillas
%                 Martin Fodstad Stoelen
%  Universidad Carlos III de Madrid
%
**********************************************************************

%% Función utilizada para la aplicación del filtro de Gauss

function mat_imageOut = spatial_gaussian_filter(mat_imageIn, sigma,
maskWidth)

% ***** Initialize *****
numRows = size(mat_imageIn,1);
numCols = size(mat_imageIn,2);
```

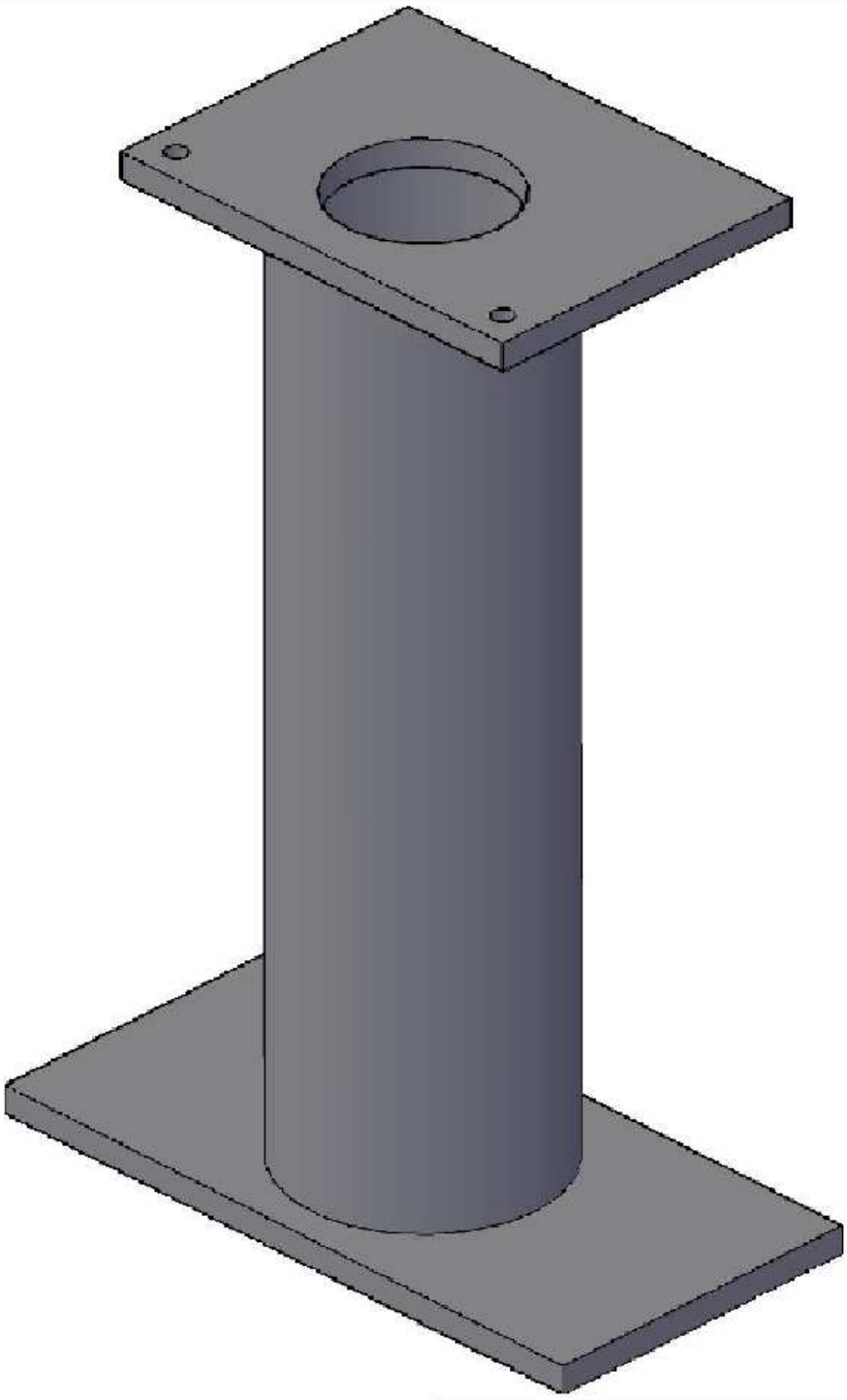
```
% ***** Build 1-D Gaussian mask *****
% Mask width, w (assuming odd)
% maskWidth = sigma * 5;

% Gaussian float mask
for i = 1:1:maskWidth
    xGaussian = i - (maskWidth+1)/2;
    vec_gFloat(i) = (1/(sigma*sqrt((2*pi)))) * exp(-
((xGaussian^2)/(2*sigma^2)));
end

% Gaussian integer mask
% TODO: Much faster?

% ***** Convolve each row of image with mask *****
for r = 1:1:numRows
    pStart = (maskWidth+1)/2;
    pEnd = numCols - (maskWidth+1)/2 + 1;

    pR = 1; % pixel for new shrinked image
    for p = pStart:1:pEnd

        for i = 1:1:maskWidth
             vec_temp(i) = vec_gFloat(i) *  mat_imageIn(r,(p -
(maskWidth+1)/2 + i));
        end

        mat_imageR(r,pR) = sum(vec_temp);

        pR = pR + 1;
    end
end

% ***** Convolve each column of image with mask *****
for c = 1:1:size(mat_imageR,2);
    pStart = (maskWidth+1)/2;
    pEnd = numRows - (maskWidth+1)/2 + 1;

    pC = 1; % pixel for new shrinked image
    for p = pStart:1:pEnd

        for i = 1:1:maskWidth
             vec_temp(i) = vec_gFloat(i) *  mat_imageR((p -
(maskWidth+1)/2 + i),c);
        end

        mat_imageOut(pC,c) = sum(vec_temp);

        pC = pC + 1;
    end
end

%% Función de cálculo del flujo óptico  %%

function vec_v = constant_flow_patch(mat_patch, pinvTol)
```

```
% Initialize
i = 1;
sizePatch = size(mat_patch,1);

% Find spatial and temporal derivatives for all pixels in patch
for x = 2:1:(sizePatch-1) % Ignore borders

  for y = 2:1:(sizePatch-1) % Ignore borders

      % Spatial derivative
      gradE = [(mat_patch(y,x-1,2) - mat_patch(y,x+1,2))
(mat_patch(y-1,x,2) - mat_patch(y+1,x,2))]; % TODO: Check [1 0 -1]
kernel used

      % Temporal derivative
      Et = mat_patch(y,x,1) - mat_patch(y,x,3); % TODO: Check [1 0 -
1] temporal kernel used

      % Fill in matrices and vectors
      A(i,:) = gradE;
      bTrans(i) = -Et;

      % Increment pixel count
      i = i + 1;

  end

end

% % Check for singular A, if all spatial gradients are null or
paralell
% if rank((A') * A) == 0
%     %error('Matrix singular');
%     vec_v = [0; 0];
% else
%     % Find optical flow for center of patch
%     vec_v = inv((A') * A) * (A') * bTrans';
% end

% Using pseudo inverse to compute optical flow
% TODO: verify tolerance settings
vec_v = pinv(((A') * A), pinvTol) * (A') * bTrans';
```

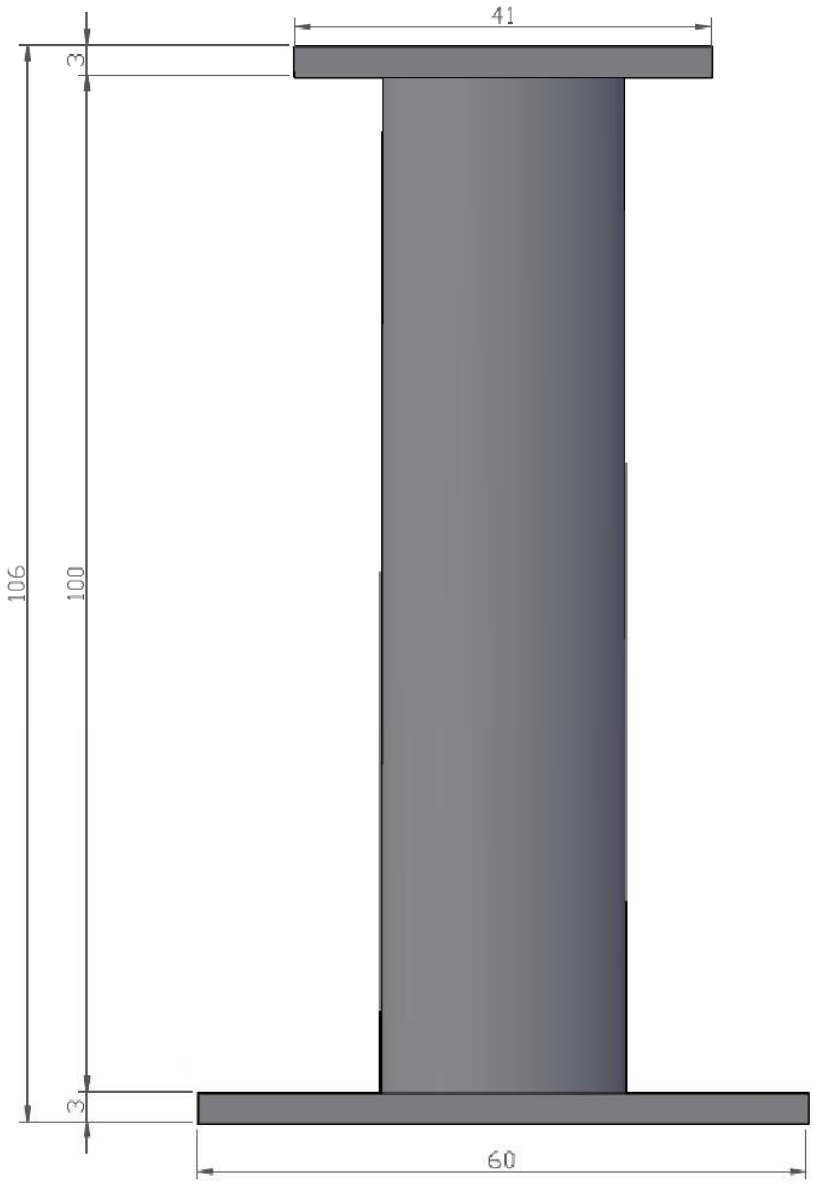
ANEXO V: Planos del soporte de la cámara.



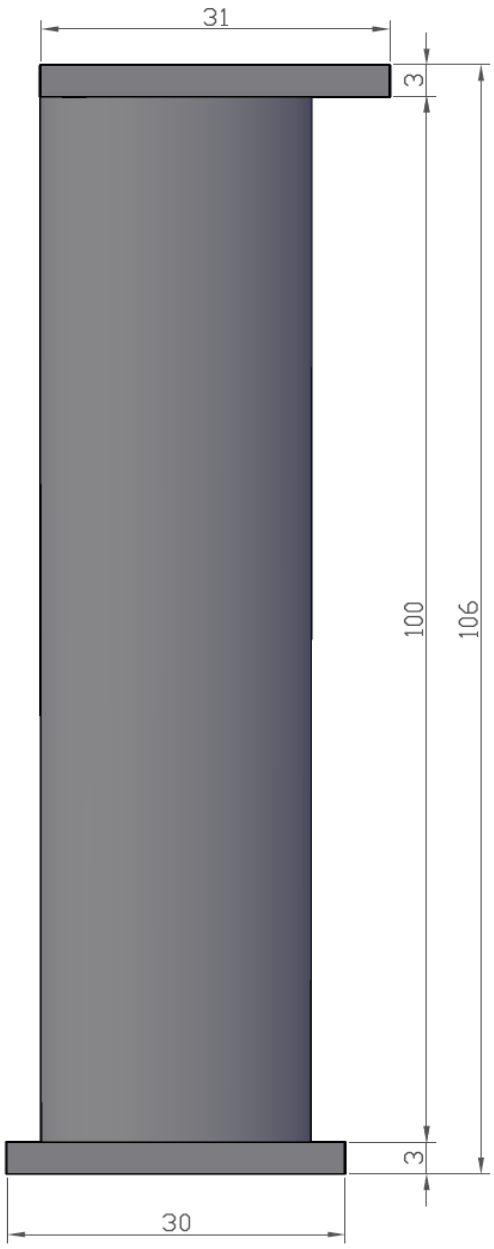
ESCUELA POLITÉCNICA SUPERIOR. CARLOS III MADRID	
DESARROLLO CÁMARA OMNIDIRECCIONAL PARA UN ROBOT MINI HUMANOIDE	
DNI: 70355484 W	FÉLIX RODRÍGUEZ CAÑADILLAS
PLANO 1	PERSPECTIVA ISOMÉTRICA DEL SOPORTE DE LA CÁMARA



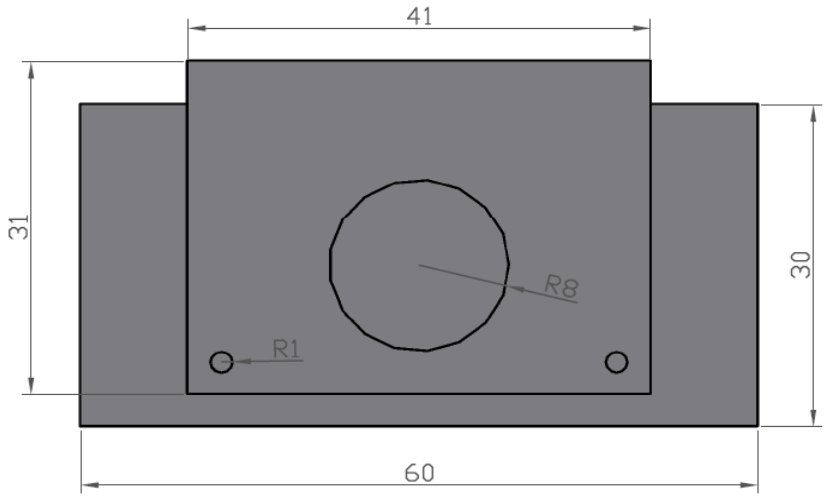
ESCUELA POLITÉCNICA SUPERIOR. CARLOS III MADRID	
DESARROLLO CÁMARA OMNIDIRECCIONAL PARA UN ROBOT MINI HUMANOIDE	
DNI: 70355484 W	FÉLIX RODRÍGUEZ CAÑADILLAS
PLANO 2	PERSPECTIVA ISOMÉTRICA DEL SOPORTE DE LA CÁMARA



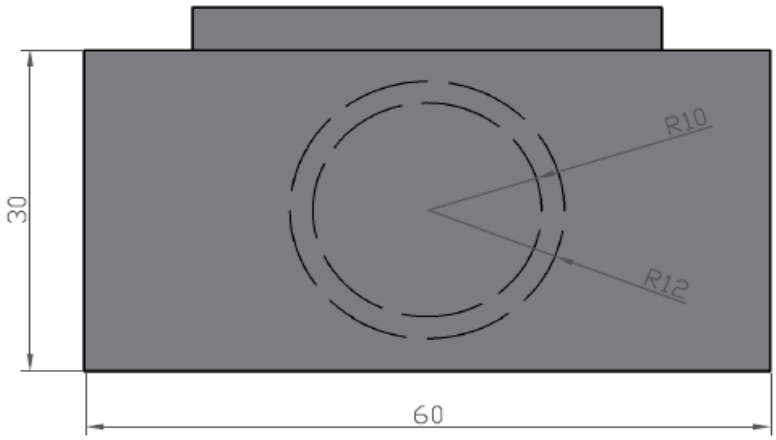
ESCUELA POLITÉCNICA SUPERIOR. CARLOS III MADRID	
DESARROLLO CÁMARA OMNIDIRECCIONAL PARA UN ROBOT MINI HUMANOIDE	
DNI: 70355484 W	FÉLIX RODRÍGUEZ CAÑADILLAS
PLANO 3	PERFIL DEL SOPORTE DE LA CÁMARA



ESCUELA POLITÉCNICA SUPERIOR. CARLOS III MADRID	
DESARROLLO CÁMARA OMNIDIRECCIONAL PARA UN ROBOT MINI HUMANOIDE	
DNI: 70355484 W	FÉLIX RODRÍGUEZ CAÑADILLAS
PLANO 4	PERFIL DEL SOPORTE DE LA CÁMARA



ESCUELA POLITÉCNICA SUPERIOR. CARLOS III MADRID	
DESARROLLO CÁMARA OMNIDIRECCIONAL PARA UN ROBOT MINI HUMANOIDE	
DNI: 70355484 W	FÉLIX RODRÍGUEZ CAÑADILLAS
PLANO 5	PLANTA SUPERIOR DEL SOPORTE DE LA CÁMARA



ESCUELA POLITÉCNICA SUPERIOR. CARLOS III MADRID	
DESARROLLO CÁMARA OMNIDIRECCIONAL PARA UN ROBOT MINI HUMANOIDE	
DNI: 70355484 W	FÉLIX RODRÍGUEZ CAÑADILLAS
PLANO 6	PLANTA INFERIOR DEL SOPORTE DE LA CÁMARA