Departamento de Ingeniería de Sistemas y Automática

TRABAJO FIN DE GRADO

Interfaz y librería para visión artificial, navegación y seguimiento en robótica

Autor: Santiago Morante Cendrero

Director: Juan Carlos González Víctores
Tutor: Miguel González-Fierro Palacios

Leganés, Mayo 2012

Título: Interfaz y librería para visión artificial, navegación y seguimiento en robótica
Autor: Santiago Morante Cendrero
Director: Juan Carlos González Víctores
Tutor: Miguel González-Fierro Palacios

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día ..... de ..... de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Aunque esta parte parece la más fácil del proyecto, en realidad es la que más debería ocupar, pues es mucha la gente a la que agradecer. Es por esto que como no tengo suficiente espacio (y aunque lo tuviera se me olvidaría alguien), voy a hacer un repaso rápido a los agradecimientos.

En primer lugar, sin duda, a mi familia y amigos, los motivos son obvios. Gracias también a la Universidad Carlos III de Madrid y a todos sus profesores, pero especialmente a dos de ellos: Juan Carlos González Víctores y Miguel González-Fierro Palacios. No sólo por dirigir mi proyecto, sino por los ánimos infundidos y el conocimiento regalado a través de la Asociación de Robótica UC3M.

La gente suele parar aquí de agradecer, pero no podría dejarlo sin reconocer a todos los hombres y mujeres de ciencia, que con su trabajo anónimo a través de los siglos, trabajos a veces insignificantes, trabajos que nunca se hicieron famosos, han conseguido hacer de la ciencia lo que es hoy en día.

Y por último a ti, lector, que vas a leer este proyecto...

Resumen

En este proyecto se han desarrollado un conjunto de algoritmos de visión artificial, que permiten a los robots el seguimiento de objetos, la detección de movimiento y la navegación por entornos no conocidos. En cuanto a los algoritmos de seguimiento, se han utilizado técnicas ya conocidas como SURF, meanshift o template matching y un método propio: Colormax. La detección de movimiento se realiza a través de optical flow. Para la navegación se han creado dos métodos nuevos para arquitecturas reactivas. NaviOtsu, que se basa en la umbralización por el método Otsu para evitar obstáculos, y NaviColor, que utiliza la distribución de colores del entorno para elegir el camino más adecuado.

La implementación de los algoritmos está basada en la librería OpenCV y éstos se han integrado dentro de una interfaz multiplataforma que simplifica su uso y permite que sean utilizados por cualquier robot. Se incluyen también estadísticas y comparativas entre los algoritmos desarrollados para comprender las ventajas e inconvenientes que ofrece cada uno.

Por último, se ha realizado una demostración del software en un pequeño robot humanoide y se han comprobado los resultados de utilizar los algoritmos en una plataforma real.

Palabras clave: robótica, opencv, visión artificial, navegación, seguimiento, detección movimiento.

Abstract

In this project there have been developed a set of vision algorithms that allow object tracking, motion detection and navigation. The algorithms of tracking have been designed using popular techniques like SURF, meanshift or template matching and one new method: Colormax. Motion detection is achieved by using optical flow. There have been designed two new methods for reactive navigation architectures. NaviOtsu, which is based on Otsu thresholding to avoid obstacles and NaviColor, that uses environment colors distribution to choose the best appropriate path.

Algorithms implementation are based on the OpenCV library and they have been integrated into a multi-platform interface, which makes more friendly and useful their use in robots. It also includes statistics and comparisons between algorithms and their advantages and disadvantages.

Finally, a software demostration has been accomplished by using a small humanoid robot and have proven the results of using the algorithms in real environtments.

Keywords: robotics, opencv, computer vision, navigation, tracking, motion detection.

Índice general

Agradecimientos v

Resumen vii

Abstract ix

1. Introducción **1**
1.1. Sobre la visión artificial 1
1.2. ASROB, ECRO y motivación del proyecto 5
1.3. Objetivos 6
1.4. Estructura del documento 8

2. Herramientas informáticas utilizadas **11**
2.1. Librería OpenCV 11
2.2. Librerías Qt 13
2.3. YARP 14

3. Software desarrollado **17**
3.1. RETINA 17
3.2. Librería Travis 20

4. Seguimiento de objetivos **23**
4.1. Acerca del seguimiento de objetivos 23
4.2. Meanshift 24
4.3. Camshift 25
4.4. Colormax 26
4.5. Template matching 27
4.6. SURF + Centroid (centroide) 29
4.7. SURF + Homography (homografía) 32

5. Detección de movimiento **35**
5.1. Introducción 35
5.2. Optical flow 36

5.2.1. Lucas-Kanade 37
5.2.2. Lucas-Kanade piramidal 39

6. Navegación **41**
6.1. Sobre las arquitecturas en navegación 41
6.2. NaviOtsu 42
6.3. NaviColor 43

7. Resultados experimentales **47**
7.1. Análisis estadístico 47
7.1.1. Diseño del experimento 47
7.1.2. Evaluación de resultados: Curvas ROC 49
7.1.3. Resultados e interpretación 51
7.2. El robot HOAP-3 54
7.3. Descripción de la demostración 55
7.3.1. Documentación de la demostración 55

Conclusiones **61**
7.4. Conclusión 61
7.5. Desarrollos futuros 62

Bibliografía **65**

Anexo: Documentación de la librería Travis **69**

Índice de figuras

1.1. Maggie, el robot social de la UC3M ©UC3M 3
1.2. Ejemplos de robots espaciales que utilizan sistemas de visión artificial ©NASA 3
1.3. Stanley, vehículo ganador del DARPA Grand Challenge 2005 ©Stanford University 4
1.4. Justin, un avanzado manipulador ©DLR 5
1.5. Logotipo de la asociación de robótica de la UC3M ©ASROB 5
1.6. Líneas de investigación ASROB 6
1.7. ECRO, plataforma para investigación en en el campo de la robótica terrestre ©ASROB 7

2.1. Logotipo de OpenCV ©AdiShavit 11
2.2. Funciones de OpenCV ©WillowGarage 12
2.3. Logotipo de Willow Garage ©Willow Garage 13
2.4. Logotipo de Qt ©Nokia 13
2.5. Módulos de YARP a través de Doxygen 14

3.1. Pantalla inicial de RETINA 18
3.2. Ventana info de RETINA 20
3.3. Logotipo de Travis 20

4.1. Rastreando con Meanshift 25
4.2. Camshift. La distribución de probabilidad adaptativa, modifica el área de búsqueda 26
4.3. Colormax, algoritmo desarrollado con pequeñas funciones combinadas. 27
4.4. Etapas de búsqueda en Template Matching 28
4.5. Imagen integral como suma del valor de sus vecinos de la zona superior e izquierda 29
4.6. Ejemplo de rastreo usando SURF + Centroid 31
4.7. Ejemplo de rastreo usando SURF + Homography . . 32

4.8. Ransac permite estimar modelos a pesar del ruido ©Wikipedia 33

5.1. Representación de Optical Flow ©Huston SJ, Krapp HG 36
5.2. Problema de la apertura 37
5.3. Detectando movimiento con Lucas-Kanade 39
5.4. Pirámide de imágenes 39

6.1. Navegación a través del método NaviOtsu 43
6.2. Navegación usando el método NaviColor 44

7.1. Objeto de color llamativo 48
7.2. Objeto con textura 48
7.3. Matriz de confusión 49
7.4. Curvas ROC: Ejemplo 50
7.5. Curvas ROC: Colormax-Pieza color 52
7.6. Curvas ROC: Colormax-Pieza textura 52
7.7. Curvas ROC: Template Matching 53
7.8. Curvas ROC: SURF-Pieza color 53
7.9. Curvas ROC: SURF-Pieza textura 54
7.10. HOAP-3 en distintas posturas ©Fujitsu 54
7.11. HOAP-3 preparado para la demostración 55
7.12. Demostración: detección de movimiento 56
7.13. RETINA versión HOAP-3: Detectando Movimiento . 56
7.14. Demostración: rastreo por color I 57
7.15. RETINA versión HOAP-3: Camshift 57
7.16. Demostración: rastreo por color II 58
7.17. Robot dando un paso hacia adelante ©UCMERCED 58

Capítulo 1

Introducción

En este capítulo se ofrece una introducción a la visión por computador, a través de ejemplos de proyectos de varias universidades y empresas. También se detallan los objetivos que se esperan conseguir una vez terminado este proyecto. Por otro lado y para finalizar el capítulo, se desglosa la estructura del documento.

1.1. Sobre la visión artificial

La visión artificial, o por computador, es una rama de investigación que estudia como extraer información de una imagen y hacer que una máquina la entienda o ejecute una determinada tarea con éxito. Si bien esta disciplina se inició en los años 60, no fue hasta los 80, con el desarrollo de ordenadores más potentes, cuando se convierte en un campo emergente de estudio [21].

Enmarcada a veces como un subcampo de la inteligencia artificial [18], lo cierto es que sus aplicaciones potenciales abarcan una gran variedad de sistemas. Como se puede comprobar en la siguiente lista, la visión artificial es aplicable en prácticamente cualquier industria que necesite extraer o procesar información del entorno.

- **Control de calidad en industrias**. Este fue uno de los primeros usos y también de los más extendidos actualmente. Desde distinguir productos alimenticios en buen o mal estado, hasta la clasificación de material industrial como tornillos y tuercas.
- **Vigilancia y seguridad**. Un campo también con gran experiencia, especialmente en la detección de movimiento. La mayoría de cámaras de seguridad modernas incluyen pequeños programas para detectar el acceso no autorizado en espacios controlados.

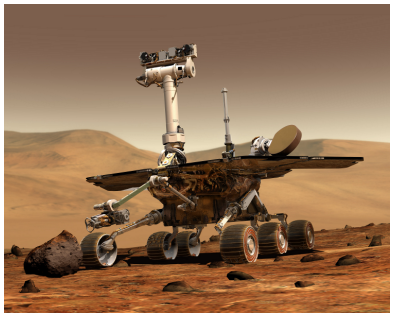
- **Navegación y guiado** de vehículos, ya sea terrestres, aéreos o marítimos. Todavía en pruebas de laboratorio en la mayoría de los casos, es uno de los campos con más posibilidades de verse implantado a corto plazo. En algunos países, se permite actualmente incluso la circulación por carretera de vehículos autónomos sin conductor.
- **Interacción humano-robot**. Imprescindible para construir futuros robots con capacidades avanzadas. Desde la localización de personas, hasta encontrar medicamentos, las posibilidades son infinitas.
- **Realidad aumentada**. Las aplicaciones móviles han sido las pioneras en este terreno. Acciones como mostrar la ubicación de un restaurante, o la tienda de moda más cercana usando como base la cámara del teléfono, son funciones habituales hoy en día.
- **Seguimiento de trayectorias**. Un ejemplo sería el famoso ojo de halcón de los partidos de tenis. Este permite reconstruir el movimiento de la pelota en el aire, en tiempo real.
- **Biometría**. El reconocimiento de caras es ya una realidad en la electrónica actual. Tanto es así que el sistema operativo Android de Google en su versión 4.0 (llamado Ice Cream Sandwich) ha introducido el desbloqueo del teléfono a través del reconocimiento de la cara del dueño del terminal.

En el mundo de la robótica, esta disciplina ha servido para contribuir notablemente a facilitar la integración de robots en entornos, no sólo urbanos, sino incluso domésticos. Buena prueba de ello es el nacimiento de los llamados robots sociales, como por ejemplo *Maggie* [20] (figura 1.1)cuyo desarrollo corre a cargo del grupo de investigación **Robotics Lab** perteneciente a la **Universidad Carlos III de Madrid**. Estos robots se diseñan pensando en la interacción con humanos, lo que sería imposible sin un sistema de visión adecuado. De este mismo grupo de investigación han surgido trabajos de visión relacionados con robots asistenciales como ASIBOT [23]. Otra línea de investigación de la misma universidad, a cargo del **Laboratorio de sistemas inteligentes**, busca completar un sistema de ayuda a la conducción, llamado IvvI, basándose principalmente en las imágenes obtenidas de las cámaras [3].

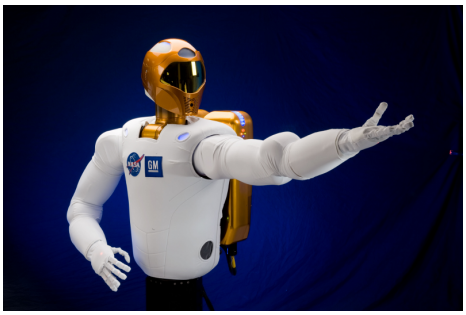
Utilizados también en un vehículo, pero en este caso extraterrestre, se encuentran los sistemas de navegación de los *Mars Rover* de la NASA [17]. Una de las dificultades del funcionamiento de estos

Figura 1.1: Maggie, el robot social de la UC3M ©UC3M

robots se basa en la inmensa distancia que separa la Tierra y Marte. Esto hace que la teleoperación sea prácticamente imposible y tengan que ser bastante autónomos. También por parte de la NASA tenemos el robot *R2 Robonaut* [2]. Su misión consiste en ayudar a los astronautas en tareas peligrosas para el ser humano, ya sea en los transbordadores espaciales o en la Estación Espacial Internacional (ISS por sus siglas en inglés). La versión 1 de este robot contiene un sistema de visión estéreo que filtra las imágenes mediante un método (laplaciana de gausianas) que enfatiza los bordes de los objetos. Esto facilita la reconstrucción 3D de las imágenes.



(a) Mars Rover



(b) R2 Robonaut

Figura 1.2: Ejemplos de robots espaciales que utilizan sistemas de visión artificial ©NASA

Un ejemplo que demuestra el claro potencial de la visión por computador lo tenemos en *Stanley* [22] (figura 1.3).

Figura 1.3: Stanley, vehículo ganador del DARPA Grand Challenge 2005 ©Stanford University

Este vehículo creado por la **Universidad de Stanford** y equipado con múltiples cámaras, ganó en 2005 el DARPA Grand Challenge. Esta carrera, organizada por DARPA (Defense Advanced Research Projects Agency), tiene como objetivo que vehículos autónomos viajen a través de los Estados Unidos hasta un determinado punto sin intervención humana. Para conseguir el premio (2 millones de dólares), se tuvo que dotar a *Stanley* de cámaras que creaban un mapa 3D del entorno, lo que unido a un GPS permitía la navegación sin ayuda.

Viendo los últimos ejemplos expuestos, podría parecer que esta disciplina se centra principalmente en el ámbito de la navegación. Nada mas lejos de la realidad. Como podemos ver en el ejemplo siguiente, la manipulación de objetos es perfectamente asumible utilizando cámaras de vídeo. Un robot avanzado en este línea es *Justin* de la agencia aeroespacial alemana DLR (figura 1.4). Puede ejecutar tareas complejas (como cálculos de trayectorias de pelotas en tiempo real) basándose en su sistema de visión.

Con estos ejemplos se ha intentado hacer una panorámica general de algunos de los proyectos más punteros del mundo de la tecnología que necesitan, completa o parcialmente, de un elemento de visión artificial que les ayude a extraer información del entorno. Contribuir en esta rama de conocimiento es un reto desde el punto de vista técnico y social, pues su desarrollo puede suponer grandes mejoras en las condiciones de funcionamiento de las máquinas actuales, lo que conllevaría a un aumento de la seguridad, la comodidad y la accesibilidad para los usuarios. Es por esto que en este proyecto se va a recorrer un pequeño camino que espero ayude y anime a otros

Figura 1.4: Justin, un avanzado manipulador ©DLR

en el futuro a involucrarse en este campo.

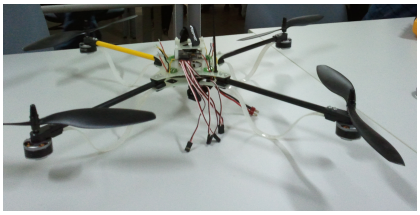
1.2. ASROB, ECRO y motivación del proyecto

La idea de crear un conjunto de algoritmos que permitiesen la navegación autónoma surgió dentro de la **Asociación de Robótica UC3M** o ASROB [13] (figura 1.5).

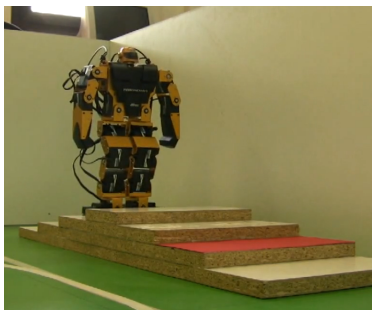
Figura 1.5: Logotipo de la asociación de robótica de la UC3M ©ASROB

Los miembros de la asociación se involucran en los proyectos que

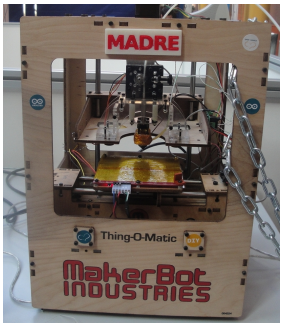
allí se desarrollan y, en concreto, uno de los proyectos fue el origen de este trabajo. Actualmente (2012) la asociación cuenta con cuatro líneas principales de investigación (figura 1.2): UAV, minihumanoides, impresoras 3D y el aquí citado, ECRO.



(a) UAV



(b) Robonova



(c) Impresora 3D

Figura 1.6: Líneas de investigación ASROB

Una de las líneas de investigación, el proyecto ECRO (Earth Civil RObot o robot civil de tierra) (figura 1.7), trata el tema de la robótica terrestre. Una de las características de ECRO es la utilización de una cámara Kinect que, junto a varios algoritmos de visión artificial, le permiten circular sin colisionar con el entorno. Fue esta idea de navegación reactiva la que, unida a la técnica de control *Visual Servo* (usar los sensores de visión para controlar el movimiento del robot) [14], desembocó en los algoritmos que se presentan en este proyecto.

1.3. Objetivos

Uno de los objetivos, no ya de contenido, sino de forma, era utilizar en la medida de lo posible el mayor número de herramientas libres y multiplataforma. Con ello se buscaba que el software que

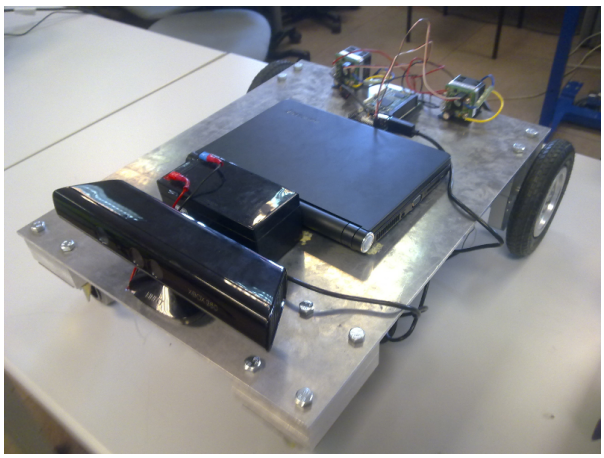


Figura 1.7: ECRO, plataforma para investigación en en el campo de la robótica terrestre ©ASROB

se fuese a generar pudiera ser compatible con cualquier robot que pretendiera utilizarlo. Por ello se utilizó OpenCV para la programación y Qt para la interfaz. Cualquier ordenador puede utilizar estas librerías sin tener requisitos especiales.

Dentro ya de la parte técnica o de programación, se buscaba conseguir tres conjuntos de algoritmos: Seguimiento, navegación y detección de movimiento. En cuanto a seguimiento, se han conseguido 6 funciones, cada una de las cuáles se basa en un principio distinto, buscando abarcar un amplio espectro de funcionalidades:

- Meanshift
- Camshift
- Colormax
- Template matching
- SURF + Centroid
- SURF + Homography

La detección de movimiento hace uso de los principios del flujo óptico. Se pretende que, a través de éste, se acoten los puntos que varían en la imagen, permitiendo la localización de los elementos dinámicos. Por último, en cuanto a la parte de programación se refiere, se quieren desarrollar dos algoritmos de navegación completamente nuevos, que permitan al robot cierta autonomía en su movimiento. Estos dos métodos han sido diseñados íntegramente en este trabajo:

- Naviotsu
- Navicolor

Otro objetivo ha sido también que el software consuma pocos recursos para permitir su ejecución en tiempo real. Pensando en la accesibilidad y en la comodidad del usuario, se ha diseñado una interfaz fácil e intuitiva, a través de botones indicativos y un apartado de instrucciones de uso.

Como objetivo final, se propone probar en una plataforma robótica real (un robot humanoide) el software conseguido y analizar los resultados.

1.4. Estructura del documento

A continuación y para facilitar la lectura del documento, se detalla el contenido de cada capítulo.

- En el capítulo 1 se realiza una introducción, se explica la motivación y el origen del proyecto y se muestra, a través de ejemplos reales, el mundo de la visión artificial.
- En el capítulo 2 se hace un repaso de las herramientas informáticas utilizadas. Entre ellas destaca la librería de visión artificial, la de creación de interfaces y la de comunicaciones.
- En el capítulo 3 se presenta y explica el software surgido del proyecto. Estos son la interfaz RETINA y la librería Travis.
- En el capítulo 4 se desarrollan los fundamentos de los algoritmos de seguimiento. Las matemáticas involucradas en el proceso son también descritas.
- En el capítulo 5 se explican los algoritmos de detección de movimiento y los principios del flujo óptico.
- En el capítulo 6 se presentan dos métodos para la navegación reactiva de robot móviles. Aquí se introducen dos métodos originales, pero en especial destaca Navicolor, un nuevo método de navegación reactiva innovador, diseñado para este proyecto.
- En el capítulo 7 se muestra el cómo y el por qué de la implementación de los algoritmos en un robot real. En nuestro caso, la plataforma utilizada ha sido el HOAP-3. Se ofrece también documentación de una demostración con el robot y un análisis estadístico de varios algoritmos de reconocimiento a través una experimento.

- Para finalizar, se aportan unas conclusiones y se pincelan posibles trabajos futuros derivados de éste. Se adjunta también la bibliografía utilizada para las referencias.
- Como anexo se ofrece la documentación generada de la librería Travis. El generador automático de documentación Doxygen ha sido el elegido para este propósito.

Capítulo 2

Herramientas informáticas utilizadas

A continuación se presentan las herramientas que han servido para construir este proyecto. Gracias a la comunidad del software libre, estas librerías se encuentran a disposición de todo el mundo y permiten un avance muy rápido en el desarrollo de aplicaciones. Destacar que, aparte de libres en cuanto al acceso al código, son también gratuitas, lo que las hace más atractivas para los desarrolladores.

2.1. Librería OpenCV

OpenCV (figura 2.1) es una librería que contiene funciones de visión artificial. Fue lanzada en 1999 y estaba desarrollada inicialmente por Intel, aunque ahora su desarrollo está mantenido por Willow Garage (figura 2.3) [8].

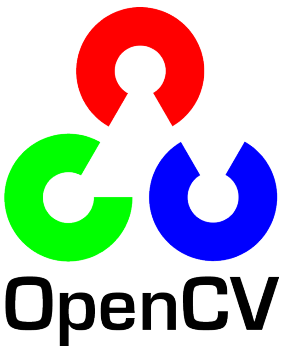


Figura 2.1: Logotipo de OpenCV ©AdiShavit

Está licenciada bajo la Licencia BSD, por lo que entra en la denominación de software de código abierto (open source en inglés). Esta

licencia permite que la librería pueda ser utilizada para uso personal o comercial, sin ninguna limitación, más allá del reconocimiento de los autores.

Originariamente desarrollada sobre C, desde finales de 2009 cuenta con una interfaz mejorada, basada en C++. Si por algo ha destacado esta librería es por la gran cantidad de herramientas que pone a disposición del programador en el campo de la visión por computador, incluso en tiempo real, como se puede ver en la figura 2.2. Entre sus áreas de aplicación, destacamos:

- Sistemas de reconocimiento facial
- Reconocimiento de gesto
- Identificación de objetos
- Calibración de cámaras
- Funciones 3D y estéreo
- Aprendizaje automático (árboles de decisión, clasificadores bayesianos, redes neuronales...)

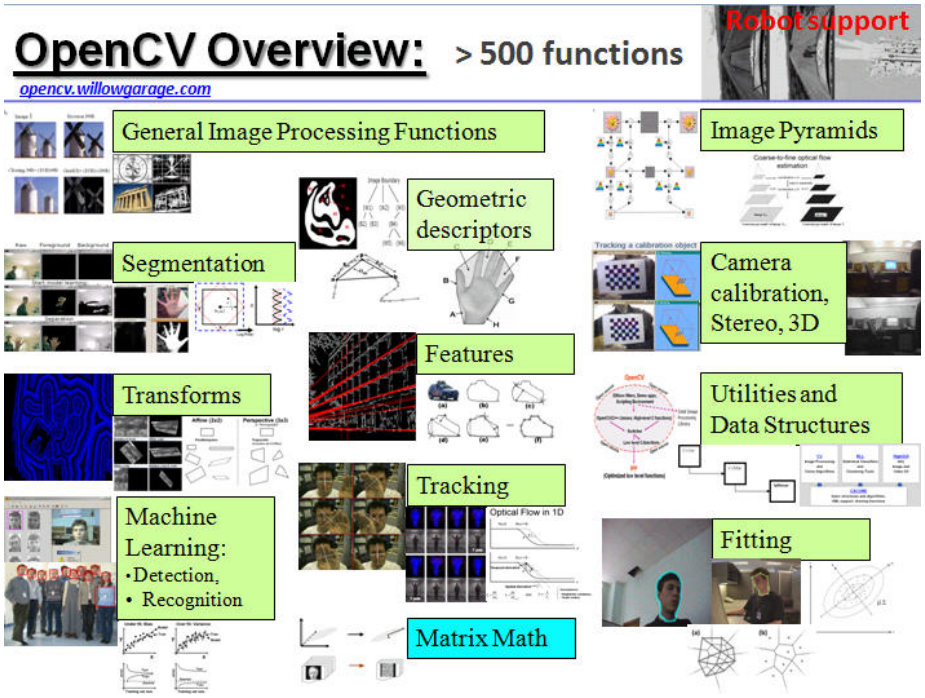


Figura 2.2: Funciones de OpenCV ©WillowGarage

Otra de las características a destacar de OpenCV es su carácter multiplataforma. Oficialmente está soportado por Windows, Linux, Mac OS, Android, iOS, Maemo, FreeBSD y OpenBSD. Esto permite su implantación en prácticamente cualquier sistema informático.

Figura 2.3: Logotipo de Willow Garage ©Willow Garage

Por último, y por hacer evidente la relación entre OpenCV y el mundo de la robótica, decir que esta librería es el paquete de visión artificial principal en ROS (Robot Operating System), un meta-sistema operativo, también mantenido por Willow Garage, especialmente pensado para el mundo de la robótica.

2.2. Librerías Qt

Qt Creator es un entorno de desarrollo integrado para el desarrollo de aplicaciones basadas en las librerías Qt (figura 2.4). Fue creado por Trolltech (ahora Qt Development Frameworks y perteneciente a Nokia), responsables también de toda la librería Qt. Al igual que sucedía con OpenCV, una gran ventaja de este software es su disponibilidad multiplataforma.

Figura 2.4: Logotipo de Qt ©Nokia

Contiene dos editores, uno de código (lenguaje C++) similar a cualquier editor con funciones avanzadas (autocompletado, marcado en colores, etc) y otro editor para diseñar interfaces gráficas (también llamado Qt Designer).

Para facilitar la integración en proyectos que impliquen utilizar ambos editores, este programa ofrece un sistema de eventos conocido como *Signals and slots*. Este paradigma de programación divide las interacciones del usuario con la interfaz en dos mecanismos de comunicación diferenciados. Por un lado cuando el usuario modifica, altera o influye en la interfaz con una acción cualquiera. A esto se le conoce como *Signal*, la cual estará conectada a una *slot*, que es, por así decirlo, la consecuencia o el cambio que se verá en la interfaz. Las posibilidades que ofrece este sistema son muchas y facilitan la creación de interfaces, ahorrando código y tiempo.

2.3. YARP

YARP (Yet Another Robot Platform) es un conjunto de protocolos y herramientas (se puede ver en la figura 2.5) para interconectar los elementos involucrados en la robótica, especialmente robótica humanoide. YARP es, al igual que el resto de herramientas utilizadas, software libre. Sus componentes se pueden dividir en tres partes:

- libYARP OS : interfaz con el Sistema Operativo
- libYARP sig : se encarga de tareas de procesado de señal
- libYARP dev : interfaz con los dispositivos

Namespace List

Here is a list of all namespaces with brief descriptions:

XmlRpc	
yarp	The main, catch-all namespace for YARP
yarp::dev	An interface for the device drivers
yarp::math	Mathematical operations, mostly a GSL wrapper
yarp::math::impl	
yarp::mjpeg	
yarp::name	Classes for constructing name servers
yarp::os	An interface to the operating system, including **Port** based communication
yarp::os::impl	The components from which ports and connections are built
yarp::sig	Signal processing
yarp::sig::draw	Very basic drawing functions, in case you don't have anything better available
yarp::sig::file	**Image** file operations
yarpcxx	C++ wrappers around an experimental C interface to YARP
yarpcxx::os	C++ wrappers around an experimental C interface to **yarp::os**

Figura 2.5: Módulos de YARP a través de Doxygen

El principal desarrollador es Paul Fitzpatrick perteneciente al MIT (Massachusetts Institute of Technology). La práctica totalidad del código está desarrollado en C++. Estas librerías han sido utilizadas en nuestro proyecto para establecer las comunicaciones entre el ordenador que ejecuta la interfaz, y los robots que sirven de plataforma.

Si por algo destaca esta librería es por su pequeño tamaño (apenas unos pocos megabytes) y su bajo consumo de memoria. Esto hace posible su uso, en nuestro caso, de manera inalámbrica con un pequeño robot humanoide de memoria limitada (HOAP-3: 512 MB).

Capítulo 3

Software desarrollado

Como resultado del trabajo de programación del proyecto, dos conjuntos de software han sido completados. El primero de ellos, RETINA, es la aplicación gráfica y, por otro lado, se ha implementado Travis como librería de C++.

3.1. RETINA

Para facilitar la integración de los algoritmos de visión artificial que se iban a desarrollar, desde el principio se pensó en hacer una interfaz gráfica que los agrupara a todos. Este es el origen de *RETINA* (figura 3.1). El nombre proviene de **Robotic Environment for Tracking Issues and Navigation Applications** cuya traducción del inglés sería: Entorno de robótica para problemas de seguimiento y aplicaciones de navegación.

Este programa está basado en las librerías OpenCV y la interfaz, basada en QT, contiene varias herramientas, cada una de las cuales engloba los algoritmos desarrollados en ese campo.

- **Sampling (Muestreo)**: Provee de unos elementos gráficos que facilitan la visualización de la cámara del robot y permiten la captura de instantáneas que son automáticamente guardadas en la galería del programa. Una vez guardadas, las imágenes están listas para su uso desde el resto de las herramientas.

- **Searching (Búsqueda o rastreo)**: Contiene los algoritmos de búsqueda y seguimiento de objetos, además de ofrecer una caja de texto donde seleccionar las imágenes con las que trabajar. El sistema verifica que la imagen pedida existe y sólo entonces permite seleccionar los algoritmos, que son:

Figura 3.1: Pantalla inicial de RETINA

- **Meanshift**: Su funcionamiento hace uso del color del objeto para su rastreo.
- **Camshift**: Es una modificación de Meanshift que añade el cambio del tamaño de la región de búsqueda en el rastreo.
- **Colormax**: También basado en la búsqueda por color, con la diferencia de que no está basado en los algoritmos anteriores. Para encontrar el color pedido, este método analiza los colores de la imagen (a través de herramientas como el histograma y el espacio de color HSV) y después señala aquellas zonas de mayor correspondencia con el color original.
- **Template matching**: La correlación normalizada es usada aquí para la comparación de imágenes (regiones de píxeles en escala de gris).
- **SURF+Centroid**: Esta famosa y moderna técnica es utilizada para hallar el centroide del objeto a buscar.
- **SURF+Homography**: Podríamos considerar este método como una extensión o modificación del anterior, en la que se ha añadido el cálculo de la homografía y la transformación de perspectiva para conseguir unos resultados más fiables.

- **Motion (Movimiento)**: Una versión de las implementaciones del flujo óptico (conocida como Lucas-Kanade piramidal, ba-

sada en Lucas-Kanade [16]) es la elegida en este proyecto para la detección de movimiento.

- **Navigation (Navegación)**: La navegación es uno de los puntos fuertes del proyecto ya que se han completado dos algoritmos novedosos para arquitecturas reactivas de navegación. Estas arquitecturas no calculan en base a mapas, sino que eligen dirección en tiempo real, en función de las entradas sensoriales:

  - **NaviOtsu**: Se inspira en una antigua técnica de umbralización [19]. Se ha pensado para entornos donde el suelo y los obstáculos tengan distinto color. Con estos supuestos, se propone que la elección del camino a seguir se haga usando una imagen binarizada (píxeles completamente blancos o negros) donde la dirección correcta comparte color con el suelo.
  - **NaviColor**: De manera más innovadora que en el caso anterior, aquí se lleva un paso más allá el cálculo de los colores. Con la imagen original y los colores del suelo, se crea una imagen en escala de grises (también llamada backprojection) donde a cada pequeña región se le asigna un color en función del parecido que tiene ésta con la distribución de colores original del suelo. Se llega así a una distinción más fina que en el caso binario anterior, y se puede elegir una dirección más adecuada.

  Estos dos algoritmos son completamente nuevos en su concepción y suponen un interesante nuevo punto de vista en la navegación reactiva basada en la visión artificial.

- **Info**: Contiene información de licencias e instrucciones de uso (figura 3.2).

Aunque no es visible desde la interfaz, el programa contiene una galería donde se almacenan las imágenes capturadas en la pestaña de muestreo (sampling). El resto de herramientas acceden a esa carpeta directamente cuando buscan una imagen (existe un campo habilitado en la interfaz para escribir el nombre de la imagen) con la que realizar los cálculos.

Por último, destacar que RETINA se ha licenciado bajo la licencia de código abierto GNU Lesser General Public License, LGPL v3, lo que permite su reutilización en cualquier otro proyecto de software libre o privativo.

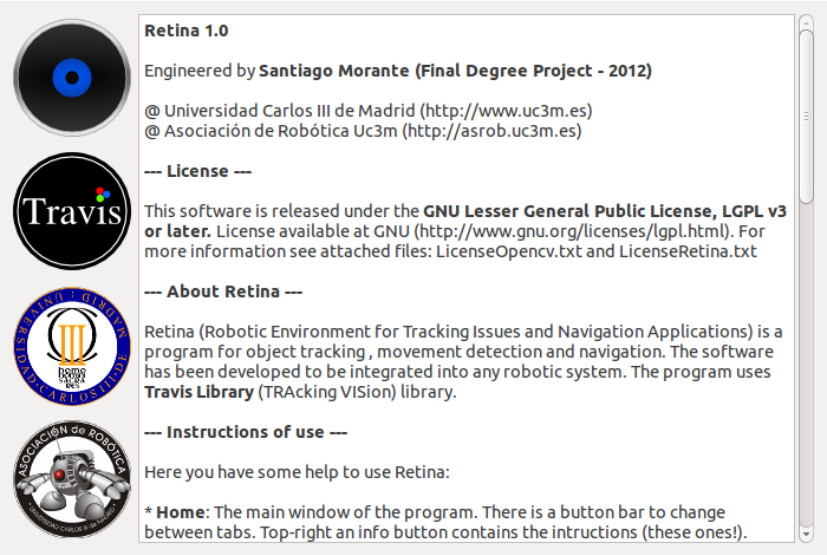


Figura 3.2: Ventana info de RETINA

3.2. Librería Travis

Paralelo al desarrollo de Retina, se hacía necesario organizar, optimizar y comentar todos los algoritmos generados durante el proyecto. Esta librería contiene todos los algoritmos que se integraron en Retina, pero sin adaptar a los requerimientos de la interfaz en Qt, conservando toda la potencia de OpenCV.

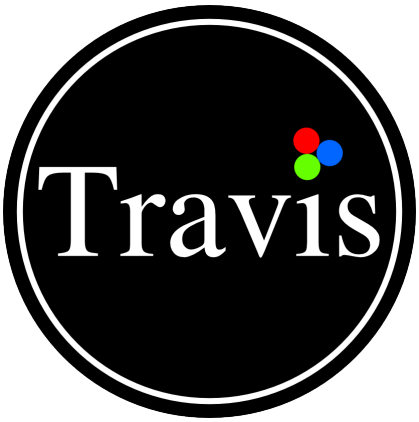


Figura 3.3: Logotipo de Travis

Tener que integrar OpenCV con Qt, hace que se pierdan algunas funcionalidades, sobre todo las relacionadas con las ventanas de información complementaria y la interacción con el ratón. Es por ello

que surge *Travis* (TRAcking in VISion, seguimiento en visión) (figura 3.3). Otra ventaja de no depender de una interfaz es la mayor rapidez en la ejecución y el menor consumo de memoria.

Travis se puede utilizar en cualquier otro proyecto ya que, al igual que Retina, está licenciado bajo la licencia de código abierto GNU Lesser General Public License, LGPL v3.

Capítulo 4

Seguimiento de objetivos

Seguimiento es el conjunto de funciones más extenso del programa y también el primero en ser explicado en el informe. Aparte de la introducción al tema, se desglosan los fundamentos teórico-matemáticos de cada uno de los algoritmos.

4.1. Acerca del seguimiento de objetivos

El seguimiento de objetivos es, posiblemente, uno de los puntos de mayor interés de la visión artificial aplicado al mundo de la robótica. Conocer la situación de, por ejemplo, una persona e interactuar con ella es imprescindible si se pretende una colaboración entre humanos y robots en el futuro.

La aplicación de los algoritmos de seguimiento no queda relegada al simple uso de seguir un objeto mostrado. La posibilidad de encontrar un objeto sin tenerlo delante en el momento de la búsqueda, permitirá, por ejemplo, que un robot encuentre un medicamento, las llaves, o incluso pueda coger un libro de una estantería. Para ello se necesitará que el robot haya visto en algún momento el objeto a buscar y lo haya almacenado en su memoria, o que puedan consultar, en tiempo real, bases de datos en la nube (internet) donde estén almacenadas imágenes de objetos, para después compararlas con el entorno que le rodea.

Centrándonos en el proyecto aquí desarrollado, se presentan varios algoritmos de reconocimiento con funciones y principios distintos. Tres de estos algoritmos (Meanshift, Camshift y Colormax) se basan en la búsqueda por color, usando como referencia una selección del usuario. Esta característica los hace indicados para rastrear elementos de colores fuertes, uniformes y que destaquen del entorno.

Otro de los métodos (Template matching) realiza una compa-

ración directa entre imágenes. Los dos últimos (SURF+Centroid y SURF+Homography) realizan sus funciones mediante la comparación de puntos claves entre los objetos.

A continuación, y empezando por los algoritmos basados en color, se procede a detallar las características de cada uno.

4.2. Meanshift

Este algoritmo fue propuesto por primera vez por Fukunaga and Hostetler [12], y más tarde adaptado por Cheng [10] para aplicarlo al análisis de imagen. La idea fundamental de este técnica, más allá de la demostración matemática que se explica en los artículos citados, consiste en encontrar máximos locales o "picos"de color en las imágenes. A estos picos se asocian todos los píxeles cercanos cuyo valor de color es parecido. De esta manera se consigue segmentar la imagen en regiones de color similares y uniformes. La ecuación matemática de Meanshift se define de la siguiente manera:

- Sea S un conjunto finito de datos de muestra.
- Sea K un kernel de una transformación lineal (eq. 4.1). Es decir el núcleo (o kernel), está formado por el subconjunto de todos los vectores del dominio que tiene por imagen al vector nulo del codominio.

$$Ker(T) = v\epsilon V | T(v) = 0\epsilon W \tag{4.1}$$

  Siendo V y W dos espacios vectoriales, T una función entre ellos y $T : V \rightarrow W$ una transformación lineal (una aplicación entre dos espacios vectoriales que preserva las operaciones de suma de vectores y producto por un escalar).

  La imagen de una transformación lineal es el conjunto de elementos en un espacio de características (W o codominio) que se corresponden con otros elementos en el otro espacio de características de la transformación (V o dominio) (eq. 4.2).

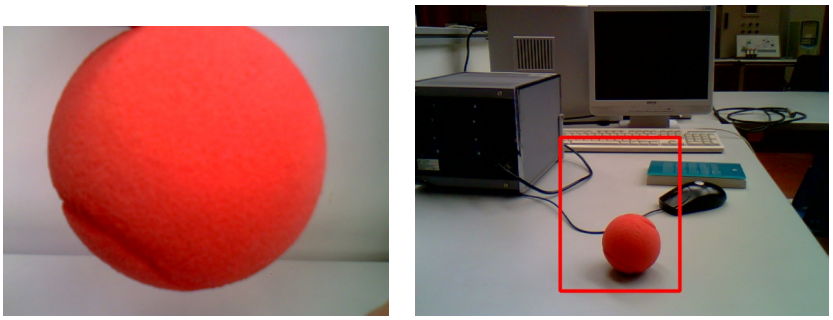
$$Im(T) = w\epsilon W | \exists v\epsilon V | T(v) = w \tag{4.2}$$

- Sea s una ventana local de píxeles alrededor de x, o posición actual.
- Y sea w : S una función de pesos.

La media ponderada de la densidad (entendida como densidad de probabilidad) en la ventana determinada por K es la ecuacion 4.3 .

$$m(x) = \frac{\Sigma_{s \epsilon S} K(s-x) w(s) s}{\Sigma_{s \epsilon S} K(s-x) w(s)} \tag{4.3}$$

La diferencia m(x)-x es lo que se conoce como Meanshift y esta estimación se repite hasta que m(x) converge [11].



(a) Referencia (b) RETINA-Meanshift

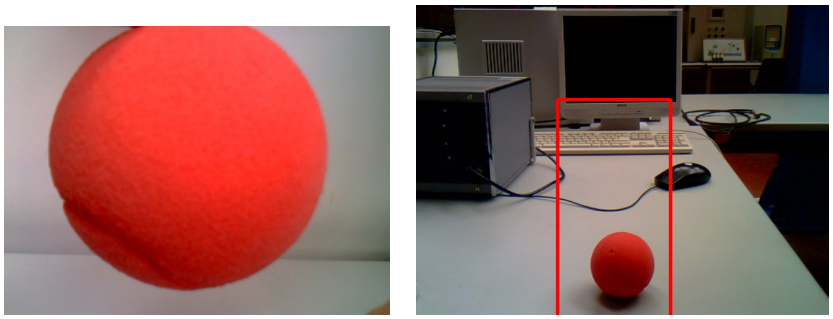
Figura 4.1: Rastreando con Meanshift

La implementación del algoritmo en OpenCV está contenida en la función del mismo nombre. Ésta analiza los fotogramas de entrada y busca el color indicado. Una vez encontrado, lo sigue en su movimiento manteniendo el tamaño de la región a buscar (figura 4.1(b)). Su eficiencia es alta para regiones con colores fuertes y uniformes, que destaquen del fondo. Un inconveniente se encuentra si el objeto desaparece de la imagen momentáneamente, pues el algoritmo pierde el rastro y es difícil que lo recupere.

4.3. Camshift

Partiendo de Meanshift, surge una adaptación de la implementación original, más enfocada al seguimiento de objetos. A esta modificación se la conoce como Camshift [1]. Aunque los algoritmos tienen los mismos fundamentos, la principal diferencia entre ambos es que Camshift utiliza distribuciones de probabilidad adaptativas, por lo que las distribuciones son recalculadas en cada fotograma. Esto conlleva que el tamaño de los segmentos varíe constantemente.

Explicado de otra forma, se tiene una imagen extraída de una cámara de vídeo. Si se selecciona una pequeña zona para rastrear, Camshift empezaría con un área de búsqueda del mismo tamaño, pero en el siguiente fotograma, recalcularía el tamaño de este área



(a) Referencia (b) RETINA-Camshift

Figura 4.2: Camshift. La distribución de probabilidad adaptativa, modifica el área de búsqueda

para englobar todos los colores parecidos. Esto hace que si el objeto se aleja o acerca de la cámara, el algoritmo recalcula el tamaño del recuadro para adaptarse a la nueva situación (figura 4.2(b)). En este mismo caso Meanshift no hubiera cambiado el área de búsqueda.

Otra ventaja que ofrece este método, pero ya a nivel de programación, es que permite indicar hacia donde ha de dirigirse el área de búsqueda, en caso de que se pierda el rastro. Esto permite determinar el centro de la imagen como punto objetivo.

4.4. Colormax

Los algoritmos vistos hasta ahora son, por así decirlo, implementaciones técnicas de funciones matemáticas complejas. Sin embargo, el algoritmo ahora presentado, ha sido desarrollado buscando unir pequeñas funciones sencillas que combinadas conforman una herramienta muy útil para buscar objetos de colores fuertes y uniformes. A continuación se presentan los pasos que sigue esta técnica.

- Primero, la imagen (figura 4.3(a)) se convierte al espacio de color HSV, donde H significa hue (color), S se refiere a saturación y por ultimo V a valor (intensidad o luz).
- Seguidamente, se calcula el histograma de la variable color, de la selección a buscar. El histograma es una representación gráfica de la frecuencia con la que se presentan ciertos valores (en nuestro caso, colores).
- Luego se calcula la *backprojection*. Esto es una imagen en escala de grises dividida en pequeñas regiones. Para cada pequeña región, se asigna un código de colores en función de la similitud

con el histograma a comparar. El color blanco significaría un alto parecido de colores, y negro se refiere a la ausencia de parecidos de color.

- Por último se busca el máximo global (región más clara) y se elige ese punto como el objeto a buscar (figura 4.3(b)). En RETINA, la precisión es seleccionable mediante una barra deslizante.

(a) Referencia (b) RETINA-Colormax

Figura 4.3: Colormax, algoritmo desarrollado con pequeñas funciones combinadas.

La ventaja que presenta este método es que si en algún momento se pierde el rastro del objeto a buscar, éste se puede recuperar. Esto se consigue porque en cada fotograma del vídeo se analizan todos los píxeles en busca del color seleccionado, por lo que si, en algún momento el objeto vuelve a aparecer en la imagen, será encontrado inmediatamente. Esto supone una gran ventaja, pues no hay que preocuparse acerca de obstáculos que impidan la visión temporal del objeto.

Por otro lado, y desde el punto de las desventajas, Colormax se hace inadecuado para el control de servomotores en un robot. Esto es debido a que el algoritmo no tiene en cuenta la posición anterior en la que se encontraba, por lo que podría ordenar movimientos aleatorios en el motor si detecta el color en dos puntos opuestos de la pantalla con demasiada rapidez.

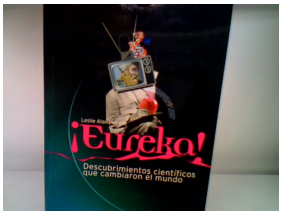
4.5. Template matching

Este método es, por así decirlo, el más sencillo de todos los algoritmos de seguimientos utilizados en el programa. Su funcionamiento se basa en la correlación normalizada (ecuación (4.4)). La correlación se utiliza como medida de similitud entre imágenes basándose

en el valor de gris de los píxeles que las forman. La normalización se usa para compensar los cambios en el valor de la intensidad (iluminación) en los píxeles. Para evitar esto se usa el valor medio de la intensidad de todos los píxeles.

$$g(x,y) = \frac{1}{n-1} \sum_{x,y} \frac{(f(x,y) - \overline{f})(h(x,y) - \overline{h})}{\sigma_f \sigma_t} \tag{4.4}$$

Donde n es el número de píxeles y $\overline{f}$ y $\overline{h}$ son las medias de los valores de la intensidad de las imágenes (referencia y fotograma). Cada σ corresponde a la desviación estándar de cada imagen.



(a) Referencia Captura

(b) Referencia Subimagen

(c) Template Matching

Figura 4.4: Etapas de búsqueda en Template Matching

Para la comparación se parte de una imagen como, por ejemplo, la foto de un objeto (figura 4.4(a)). El usuario indica mediante un recuadro, la zona de la imagen a buscar o referencia (figura 4.4(b)). Entonces, se compara esa referencia con subimágenes obtenidas de los fotogramas capturados por la cámara de vídeo. Si alguna región del fotograma supera el umbral de confianza programado, esta zona se recuadra (figura 4.4(c)).

Este método tiene serias limitaciones. La principal es que es un método no invariante a la escala o a la rotación, por lo que no

permite que el objeto se encuentre con una rotación distinta a la de la referencia ni que su tamaño sea distinto. Dicho de otra forma el objeto del vídeo tiene que estar a la misma distancia a la que se le hizo la captura de referencia y con la mismo ángulo de giro.

Es por estos inconvenientes por los que esta técnica no debe ser usada como método único de rastreo. Su función debería estar más enfocada a la confirmación del cumplimiento de la tarea.

4.6. SURF + Centroid (centroide)

SURF es un descriptor y detector de puntos de interés de rotación y escala invariante [5]. Este algoritmo es una variante, más rápida, de SIFT (Scale-invariant feature transform) [15]. Su funcionamiento se basa en la extracción de puntos invariantes de la imagen. La principal diferencia con su predecesor SIFT consiste en el uso de imágenes integrales lo que aumenta la rapidez en los cálculos.

Estas imágenes integrales o tabla de áreas sumadas (del inglés summed area table) son imágenes en las cuáles cada punto (x,y) es la suma del valor de todos los píxeles que le rodean por arriba y por la izquierda (fig. 4.5).

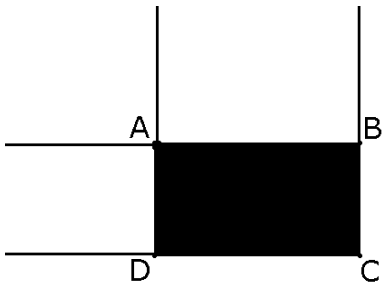


Figura 4.5: Imagen integral como suma del valor de sus vecinos de la zona superior e izquierda

SURF se centra en el reconocimiento de regiones (blob detection en inglés) y dentro de esta grupo, pertenece a los métodos basados en extremos locales o de puntos de interés.

Para empezar la detección, ambos (SURF y SIFT) utiliza una técnica denominada *Scale-Space* consistente en replicar la imagen y convolucionarla con filtros gaussianos (o paso bajo) que eliminan el ruido y hacen que la imagen parezca más "borrosa". Un ejemplo de función gaussiana se puede ver en la ecuación 4.5.

$$f(x) = ae^{\frac{-(x-b)^2}{2c^2}} \tag{4.5}$$

Donde a,b y c son constantes reales (con $a > 0$).

Después, en SIFT, se detectan los puntos característicos (aquellos que están presentes en todas las réplicas) usando una variante de la laplaciana de gausianas, llamada diferencia de gaussianas (eq. 4.6).

$$D(x, y, \sigma) = L(x, y, K_i\sigma) - L(x, y, K_j\sigma) \tag{4.6}$$

Siendo $L(x, y, k\sigma)$ la convolución de la imagen original, $I(x, y)$ con un filtro gaussiano $G(x, y, k\sigma)$ (ecuación 4.7).

$$L(x, y, K\sigma) = G(x, y, K\sigma) * I(x, y) \tag{4.7}$$

En SURF en cambio, se crea una pila de imágenes de la misma resolución y se utiliza el determinante del Hessiano o DoH (eq. 4.8) escalado por un factor σ que varía para cada imagen de la pila.

EL Hessiano o matriz Hessiana es una matriz nxn que contiene las segundas derivadas parciales y se utiliza normalmente para encontrar máximos y mínimos locales en funciones multivariable. A este determinante del Hessiano escalado también se le denomina operador de Monge-Amperé.

$$DoH^{(\sigma)}(u) = \sigma^4 det(Hg\sigma * u) = \sigma^4((D_{xx}g\sigma * u)x(D_{yy}g\sigma * u) - (D_{xy}g\sigma * u)^2) \tag{4.8}$$

Donde:

- u es la imagen
- g es el filtro gaussiano
- σ es un factor de escalado

Una vez obtenidos los puntos, se codifican usando un descriptor (un código único), que tiene en cuenta la orientación de los puntos en función de sus vecinos. Para obtener este descriptor se calculan las *Haar wavelet* de los vecinos de un punto de interés. Una *Haar wavelet* es la unión de una secuencia de funciones cuadradas. Estas funciones cuadradas, en este caso, representan el signo de la Laplaciana alrededor del punto.

El descriptor final es la suma de las *Haar wavelet* de las distintas subregiones que forman los vecinos del punto de interés.

Por último, se repiten los mismos pasos en la segunda imagen y se calculan las distancias euclídeas (similitudes) entre los puntos.

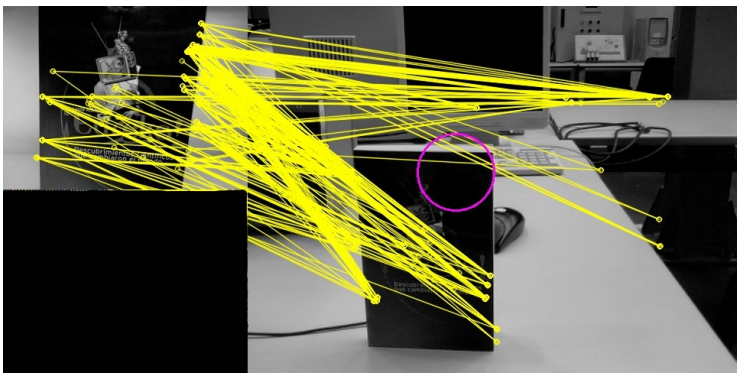


Figura 4.6: Ejemplo de rastreo usando SURF + Centroid

En nuestro programa se ha utilizado este algoritmo para conocer la posición de los puntos clave de la imagen, y una vez hallados calcular la media de sus posiciones (coordenadas). Con ello se pretende situar el centroide del objeto buscado. La implementación se resume en:

- Cargamos la imagen que servirá de referencia (el objeto a buscar).
- Convertimos esta imagen a escala de grises. Esto es debido a que la implementación de SURF en las librerías OpenCV sólo funciona con imágenes en escala de grises si se utiliza un comparador de fuerza bruta (BruteForceMacther). También se reduce de tamaño a 320x240 para asegurarnos que la referencia es más pequeña que la imagen en la que buscar.
- Calculamos los puntos clave de la referencia y sus descriptores.
- Para cada fotograma de la salida de vídeo se siguen los mismos pasos anteriores.
- Se comparar los descriptores de ambas imágenes y se calculan las coordenadas medias de todos los puntos.
- Por último se representan los resultados, el centroide (figura 4.6).

Hay que destacar la alta tasa de acierto de este método, incluso con obstrucción parcial de la vista, objetos rotados o con distinto tamaño. Su limitación proviene del propio cálculo de la media. Cuando varios puntos no apuntan al lugar deseado, el centroide se desplaza fuera del objeto.

4.7. SURF + Homography (homografía)

En este método se utiliza el mismo algoritmo SURF explicado antes, es decir, se calculan los mismos descriptores, pero en este caso se ha utilizado un comparador distinto al de fuerza bruta. El comparador usado es de tipo FLANN (fast library for approximate nearest neighbours). Esto nos ofrece varias ventajas:

- Permite el tratamiento de imágenes a color. Esto no afecta al rendimiento del algoritmo pero de cara al operador el resultado es más sencillo de interpretar.
- El comparador FLANN de OpenCV permite el acceso a la distancia euclídea obtenida de la comparación. Esto nos permite trabajar sólo con aquellos puntos "de calidad", con distancias pequeñas (lo que indica gran similitud).

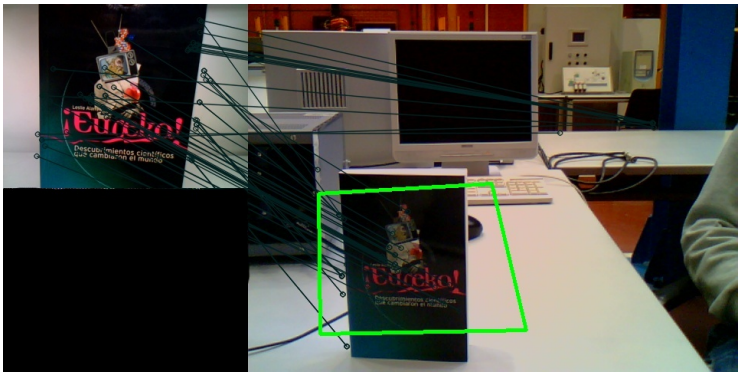


Figura 4.7: Ejemplo de rastreo usando SURF + Homography

Además de este cambio en el comparador y aprovechando las ventajas ya comentadas, se puede implementar una extensión adicional, el cálculo de la orientación a través de la homografía. La homografía (transformación geométrica entre dos figuras planas) es una matriz que representa el cambio de perspectiva. La ejecución discurre de la siguiente manera:

- Hasta la comparación de los descriptores el proceso es prácticamente igual al método anterior, a excepción de la conversión a escala de grises, que aquí no es necesaria.
- Se calcula la mínima distancia del punto de mayor calidad y se eliminan de la lista de puntos aquellos cuya distancia sea K veces mayor que la mínima. Por defecto K tiene un valor de 2,

pero en RETINA este valor puede ser modificado mediante el control deslizante de la precisión.

- Con los mejores puntos se calcula la homografía mediante RANSAC (RANdom SAmple Consensus). Este método puede estimar modelos matemáticos a partir de conjuntos de datos que contienen puntos singulares u outliers (figura 4.8).

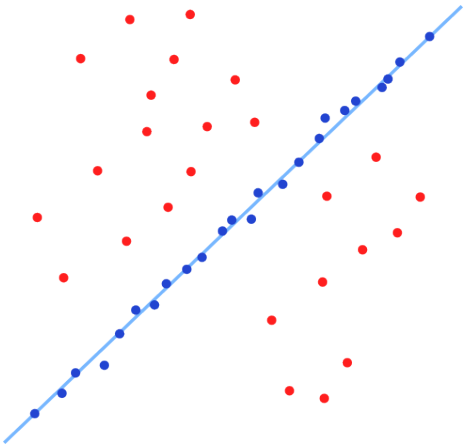


Figura 4.8: Ransac permite estimar modelos a pesar del ruido ©Wikipedia

- Por último se calcula transformación de perspectiva y se muestra en pantalla (figura 4.7).

El resultado obtenido es visualmente superior y, además, existen más aplicaciones potenciales. Se puede pensar, por ejemplo, en la corrección o estabilización de una imagen mediante la comparación de patrones o en un robot al que se le indique una referencia relativa a un objeto y deba orientarse.

Capítulo 5

Detección de movimiento

La siguiente herramienta en ser documentada es la detección de movimiento. El flujo óptico es la base en la que se fundamenta el software. En la introducción se presentan las aplicaciones más comunes y los objetivos a alcanzar.

5.1. Introducción

La detección de movimiento está íntimamente relacionada con las aplicaciones de vigilancia y seguridad. Desde hace muchos años, existen sistemas, tanto mecánicos, como electrónicos e informáticos que permiten tener controlada una zona de personas u otros elementos no deseados. Es por ello que la literatura es amplia en este sentido y no se ha avanzado mucho últimamente en esta dirección.

En cambio otros usos si se están viendo bastante desarrollados, especialmente los que involucran sistemas de control de tráfico. Se ha encontrado un gran mercado por explotar en el uso de detectores de movimiento, que unidos a esquemas de la carretera introducidos en el software, permiten conocer el estado de las carreteras en tiempo real. Ya sea un atasco, un accidente o incluso un conductor en sentido contrario, las cámaras pueden avisar en tiempo real a los organismos pertinentes.

En nuestro caso, los objetivos no son tan ambiciosos y se pretende simplemente implementar un sistema de detección de movimiento básico en un robot. Esto le permitiría por ejemplo, seguir una pelota en movimiento, o mantener la cabeza (si es un humanoide) atenta al foco de movimiento en una habitación. Para ello se han utilizado las herramientas basadas en el *Optical Flow*.

5.2. Optical flow

Optical Flow es el patrón de movimiento causado por el movimiento entre observador y lo observado (figura 5.1). Esta técnica de detección de movimiento ha sido portada a OpenCV [6]. Para cumplir sus objetivos, este método local utiliza una ecuación diferencial que relaciona el desplazamiento del nivel de gris en un píxel entre dos fotogramas. Como se ve en la fórmula (5.1), conocida como la *ecuación de restricción del flujo óptico*, se asume que la intensidad varía suavemente con respecto al tiempo y los movimientos de la imagen son lentos.

$$\frac{\partial I}{\partial x}\frac{\partial x}{\partial t}+\frac{\partial I}{\partial y}\frac{\partial y}{\partial t}+\frac{\partial I}{\partial t}=\frac{\partial I}{\partial x}V_x+\frac{\partial I}{\partial y}V_y+\frac{\partial I}{\partial t}=I_x u+I_y v+I_t=0 \quad (5.1)$$

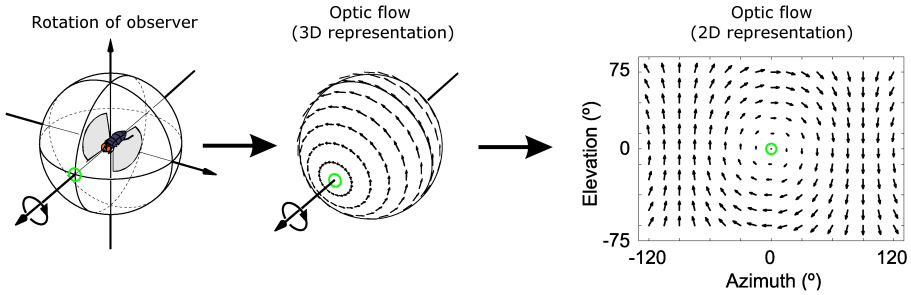


Figura 5.1: Representación de Optical Flow ©Huston SJ, Krapp HG

Para que este modelo funcione, primero se crea un mapa de puntos relevantes en la primera imagen. Estos puntos, que después serán evaluados para comprobar su desplazamiento, permiten también a través de algunos cálculos, medir la velocidad (normal) de los objetos.

Se ha de hacer especial hincapié en que la velocidad que se puede detectar es la normal. Esto es debido al *problema de la apertura* (figura 5.2).

Este problema va unido al flujo óptico y dice que no se puede saber la velocidad verdadera de una región, sino que solamente se puede conocer la componente normal de esta. Si no existe una variación (un gradiente) de intensidades en el objeto, no se pueden distinguir puntos concretos y por lo tanto no se sabe la velocidad real. En la figura 5.2 se puede comprobar como parece que el objeto negro se desplaza en diagonal hacia abajo. El problema es que, al ser uniforme, no se puede saber si se mueve en alguna otra dirección.

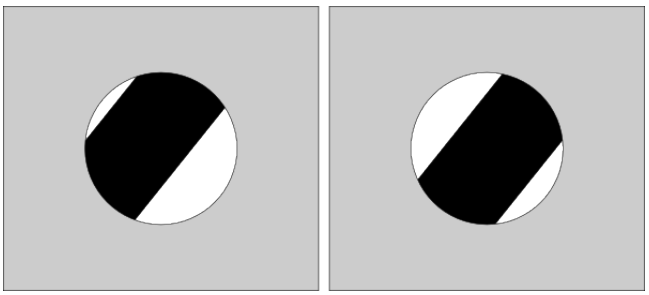


Figura 5.2: Problema de la apertura

Para solucionar este problema, cada autor ha impuesto unas restricciones adicionales al problema. En nuestro caso, utilizamos la implementación de Lucas-Kanade.

5.2.1. Lucas-Kanade

Lucas-Kanade es un método diferencial para el cálculo del flujo óptico [16]. Las restricciones adicionales que impone incluye asumir que la intensidad no varía para un mismo píxel, y que el movimiento alrededor de un píxel es constante y lento. Con ello y aplicando mínimos cuadrados, determina el desplazamiento de los puntos locales (figura 5.3).

Para optimizar el cálculo de mínimos cuadrados no lineales, Lucas-Kanade utiliza una aproximación del método Gauss-Newton (un método iterativo para encontrar el mínimo de una función). La función a minimizar es la que se ve en la ecuación 5.2 [4].

$$\arg\min_{p} \sum_{x} [I(W(x;p)) - T(x)]^2 \tag{5.2}$$

Donde:

- $I(x)$ es la imagen con la que se compara. Se tiene también que $x = (x, y)^T$, o lo que es lo mismo: x es un vector columna que contiene las coordenadas de los píxeles.
- $T(x)$ es la plantilla o modelo a comparar (en el caso del cálculo del Optical Flow, es una subregión de $I(x)$).
- $W(x;p)$ es el conjunto de deformaciones expresado de la siguiente manera: $W(x;p) = \begin{pmatrix} x + p1 \\ y + p2 \end{pmatrix}$
- El vector de parámetros $p = (p1, p2)^T$ son las traslaciones o desplazamientos.

Una vez aplicado el método de Gauss-Newton, la función óptima a minimizar es la que se ve en la ecuación 5.3.

$$\triangle p = (\sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T \left[\nabla I \frac{\partial W}{\partial p}\right])^{-1} \sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T (H(x) - I(W(x;p))) \tag{5.3}$$

Donde la iteración para calcular $\triangle p$ (variación del desplazamiento o velocidad) discurre de la siguiente manera:

- Se deforma la imagen I con W para obtener $I(W(x;p))$.
- Se calcula el error en la imagen: $(H(x) - I(W(x;p)))$.
- Se deforma el gradiente ∇I con $W(x;p)$.
- Se evalúa el Jacobiano $\frac{\partial W}{\partial p}$ en torno a $(x;p)$ (el punto original más la traslación).
- Se calcula el descenso más rápido (*steepest descent*): $\nabla I \frac{\partial W}{\partial p}$
- Se calcula la matriz Hessiana: $H = \sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T \left[\nabla I \frac{\partial W}{\partial p}\right]$
- Se calcula $\triangle p$ usando toda la ecuación 5.3.
- Se actualizan los parámetros: $p \leftarrow p + \triangle p$

Estas iteraciones suceden de manera continua hasta que p converge. Normalmente se asume que el algoritmo ha convergido cuando $\| \triangle p \| \leq \varepsilon$, siendo ε un umbral introducido por el usuario. Si asumimos:

- $v = \triangle p$
- $A = \nabla I \frac{\partial W}{\partial p}$
- $b = (H(x) - I(W(x;p)))$

La función optima se puede expresar como: $v = (A^T A)^{-1} A^T b$ lo que por un lado es una ecuación más manejable, y por otro nos permite introducir el concepto de matriz de segundo momento.

Al conjunto $A^T A$ se le conoce como matriz de segundo momento (second-moment matrix en inglés) o tensor estructural (structure tensor) de la imagen en un punto. Este tensor estructural indica la dirección predominante del gradiente de los vecinos del punto.

Uno de los problemas inherentes al método de Lucas-Kanade es su carácter local, lo que le impide ser preciso para movimiento rápidos.

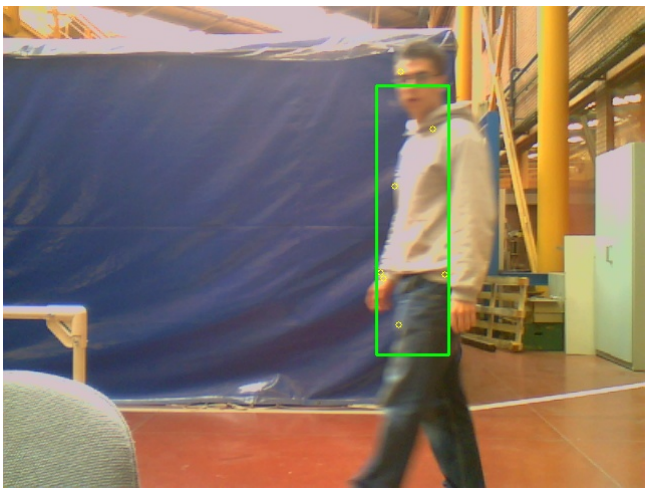


Figura 5.3: Detectando movimiento con Lucas-Kanade

Es por ello que existen alternativas o modificaciones de este algoritmo. En nuestro caso, se utilizó una versión derivada del método, conocido como Lucas-Kanade piramidal.

5.2.2. Lucas-Kanade piramidal

En este caso [7] la estimación se realiza sobre una pirámide de la imagen. Las distintas réplicas de la pirámides no son sino sucesivas reducciones de la escala de la imagen original. Puestas unas encimas de otras en orden decreciente de resolución, parecen una pirámide, de ahí el nombre (figura 5.4). Para reducir la resolución, o dicho de otra manera, para suavizar o emborronar la imagen se utiliza la convolución con filtros gaussianos.

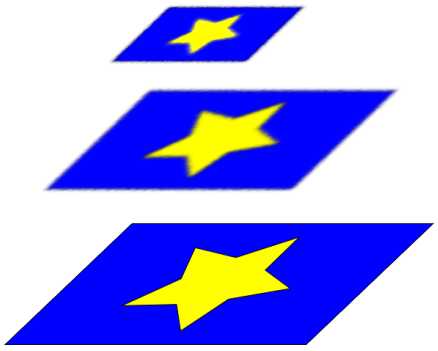


Figura 5.4: Pirámide de imágenes

Para realizar sus cálculos, se tiende a elegir esquinas y puntos

de mucha textura. Estos puntos, presentes en todas las réplicas con distintas resoluciones, permiten un algoritmo robusto a movimientos rápidos, usando las imágenes de menor resolución, pero también preciso en los detalles, con las imágenes de mayor tamaño.

Normalmente este proceso se suele asociar con algún tipo de filtro, debido a que arrastra los fallos cometidos en etapas anteriores y es más sensible al ruido.

Capítulo 6

Navegación

El último de los métodos a desarrollar ha sido un tema ampliamente tratado en la robótica, como se mostrará en la introducción. Se incluye una explicación de los fundamentos teóricos de los algoritmos usados, así como las ventajas de cada uno. Es en este tema donde se introduce el algoritmo Navicolor, diseñado específicamente para este proyecto.

6.1. Sobre las arquitecturas en navegación

La navegación ha sido, desde los principios de la robótica, uno de los temas más estudiados por todos las universidades y empresas del mundo. Desde los algoritmos planificados de los inicios, pasando posteriormente por las arquitecturas reactivas, y actualmente desarrollándose junto a los métodos híbridos (planificación de comportamientos reactivos), la cantidad de posibilidades es enorme, y simplemente resumirlo llevaría cientos de páginas.

Es por ello que simplemente definiremos algunos conceptos para entender como funciona RETINA. Por así decirlo, existen dos grandes formas tradicionales de implementar los algoritmo de navegacion: planificados y reactivos.

Los planificados se basan en el cálculo de trayectorias en función de mapas y modelos del entorno en el que se moverán los robots. Entre otros problemas de estos sistemas, destacar que no son inmunes a cambios en el entorno, y de hecho no pueden funcionar en espacios dinámicos. El recálculo de los mapas en tiempo real, suele ser inasumible desde un punto de vista computacional.

Por otro lado, las arquitecturas reactivas, no tienen ningún mapa previo, sino que utilizan la información de los sensores para ir modificando su trayectoria en tiempo real y en entornos cambiantes [9].

La ventaja frente a los algoritmos planificados es que estos permiten la inclusión de las máquinas en entornos desconocidos y/o dinámicos. Los métodos aquí diseñados, entrarían dentro de la categoría de algoritmos reactivos.

A continuación se presentan dos algoritmos originales, diseñados específicamente para este proyecto, que basándose en la distinción de colores, permiten mantener una trayectoria que evite colisiones con las paredes, circulando por un camino libre de obstáculos. Esto se consigue comparando el color de la zona inferior de la imagen (el suelo) con otras muestras obtenidas del entorno.

6.2. NaviOtsu

El primero de los algoritmos propuestos, se llama *NaviOtsu* en honor a Nobuyuki Otsu, quien en 1979 inventó un método de umbralización basado en la estadística (Método Otsu) [19]. Diferenciar entre los objetos y el entorno es lo que se conoce como segmentación y existen varios métodos para llevarlo a cabo (crecimiento de regiones, split and merge, etc), entre ellos se encuentran los de umbralización. Estos permiten decidir qué píxeles pertenecen al objeto deseado y cuales al entorno binarizando la imagen en función de un valor límite (umbral). El método de Otsu considera el histograma de la imagen como una suma de dos gausianas, una correspondiente al fondo y otra al objeto de interés. Con esta distribución, calcula cuál de todos los posibles umbrales (entre 0 y 255) maximiza el cociente entre la varianza entre las gausianas y la dispersión dentro de ellas (ecuación (6.1)).

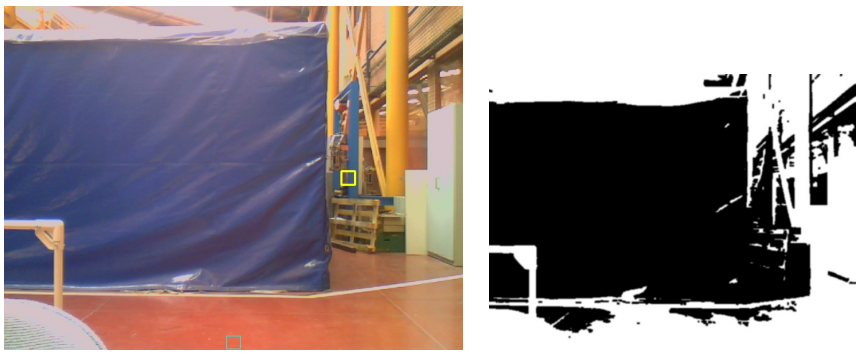
$$Q(t) = \frac{\sigma^2_{ext}}{\sigma^2_{int}} \tag{6.1}$$

Donde $Q(t)$ es el la función umbral y cada σ es una varianza. Una es la varianza externa entre las dos gausianas y la otra representa la dispersión interna. Cada σ se calcula como una suma ponderada entre el valor de cada gausiana.

En nuestro caso la umbralización del vídeo se puede considerar dinámica porque el cálculo del umbral se hace en tiempo real. Es decir, la umbralización depende de la escena que se esté visualizando en ese instante, se adapta a ella.

En la figuras 6.1(a) y 6.1(b) se puede observar la salida que genera el programa al ejecutar este algoritmo. Por un lado tenemos la pantalla que vería el operador (figura 6.1(a)). Es en esta pantalla donde se indica, mediante un recuadro de color amarillo, la dirección

con menos obstáculos. Por otro lado, podemos ver la imagen con la que el ordenador haría los cálculos de navegación (figura 6.1(b)).



(a) NaviOtsu (b) Binary Image-Otsu

Figura 6.1: Navegación a través del método NaviOtsu

Para entender como funciona, se puede decir que con la imagen binarizada (6.1(b)), se crean cuatro subimágenes: una en la zona inferior, y tres en la franja horizontal media correspondientes a izquierda, centro y derecha. El valor del recuadro inferior se compara con los de los tres superiores y se escoge el más parecido. Este se corresponde con la dirección elegida.

6.3. NaviColor

El segundo método implementado, llamado *NaviColor*, se basa en una herramienta bastante conocida, como es el calculo del histograma y la backprojection, pero aplicado de manera novedosa a la navegación. Su funcionamiento se detalla a continuación:

- Inicialmente, se tiene una imagen en el espacio de colores RGB. Este espacio es el habitual en muchos sistemas, y sus iniciales se corresponde en inglés con los colores a los que representan (R = red = rojo, G = green = verde y B = blue = azul).
- Se selecciona una pequeña región en la zona baja de la imagen original, donde se supone que se encuentra el suelo. Se extrae ese recuadro para crear una subimagen.
- Se transforma la subimagen al espacio de color HSV y se calcula el histograma de la variable H (hue o color). Este histograma representa la distribución de colores en la subimagen.
- Con la distribución de colores obtenida se crea una backprojection de la imagen original. Una bacprojection no es más que

una imagen en escala de grises donde cada pequeña región adquiere un color en función de su parecido con la distribución de colores con la que se compara (más blanco cuanto más se parece).

- Una vez obtenida la nueva imagen en escala de grises, se seleccionan cuatro pequeños recuadros (izquierda, centro, derecha y suelo) y se comparan los niveles de gris de la subimagen del suelo con los del resto de recuadros. Esta comparación se lleva a cabo comparando los histogramas con el método de la distribución de Pearson o de Chi cuadrado (eq. 6.2).

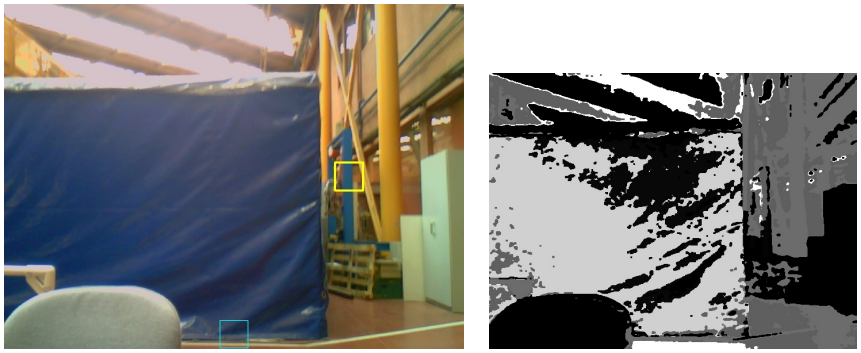
$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I) + H_2(I)} \tag{6.2}$$

  Donde:

  - H_1 y H_2 son los histogramas a comparar y d la distancia entre ellos (entendiendo distancia como similitud).
  - I es el número de píxeles de cada uno de los niveles de gris de la imagen. Normalmente este valor tiene un rango (0-255).

- Por último, aquel recuadro que ofrezca una mayor similitud será elegido como dirección a seguir.

En las figuras 6.2(a) y 6.2(b) se puede ver la implementación del método Navicolor, a través de un ejemplo de uso en la interfaz RETINA.



(a) NaviColor (b) Backprojection-Color

Figura 6.2: Navegación usando el método NaviColor

Como se puede suponer, existe una fuerte dependencia del color del suelo y su iluminación. Si el color de la zona inferior de la imagen coincide con el de las paredes, el robot no podrá distinguirlos, errando al elegir dirección. También la iluminación juega un papel importante, pues para analizar la distribución de colores se necesita una luz adecuada y suficiente. De lo contrario el histograma del color se desvirtuaría.

Asumiendo estas consideraciones, el algoritmo resulta muy útil y se ejecuta con bastante rapidez, pudiendo ser ejecutado en tiempo real sin retardos de ningún tipo. Además, sometiendo a la backprojection a una serie de filtros para eliminar el posible ruido existente, las superficies adquieren tonos uniformes, y los resultados son muy aceptables.

Capítulo 7

Resultados experimentales

Una vez analizadas las funcionalidades de los algoritmos, se querían probar en una plataforma real que permitiese poner en práctica el visual servo del que se habló al principio. Para trabajar con entornos no ideales, hay que asumir simplificaciones que permitan adaptar aquellos métodos teóricos no pensados para ambientes con ruido. Una afición personal a la robótica humanoide hizo que se eligiese un pequeño robot como banco de pruebas. Como paso previo a la demostración en el robot, se realiza a continuación una análisis estadístico de los algoritmos, a través de curvas ROC.

7.1. Análisis estadístico

Para comprobar la efectividad de los métodos se propone la elaboración de unas pruebas estadísticas basadas en una serie de experimentos y su posterior análisis. Se evaluarán algunos de los algoritmos de reconocimiento de objetos.

7.1.1. Diseño del experimento

La prueba consistirá en el reconocimiento de una serie de objetos, en base a unas capturas previas que realizará la cámara. A continuación se muestra las condiciones del experimento:

Objetos a encontrar

Los algoritmos se someterán a dos conjuntos de pruebas. El primer conjunto se hará con una objeto de color uniforme y llamativo (figura 7.1) y que no se confunda con el entorno. El segundo conjunto se realizará con una objeto con mucha textura, dibujos y colores mezclados (figura 7.2).

Las capturas previas de los objetos se harán sobre un fondo de color uniforme (preferiblemente blanco) e intentando que el objeto ocupe la mayor parte posible del espacio de la imagen.

Figura 7.1: Objeto de color llamativo

Figura 7.2: Objeto con textura

Influencia de la luz

Para evitar la influencia de la luz, la misma prueba se realizará siempre en el mismo entorno para todos los algoritmos.

Muestreo y toma de valores

Las muestras (capturas de imagen) se tomarán de manera automática por el ordenador a intervalos regulares. Como cada algoritmo tiene un tiempo de procesamiento distinto, estos intervalos se han ajustado para dar aproximadamente el mismo número capturas por minuto.

Se realizarán tres repeticiones de cada serie, asignando un umbral distinto de precisión en cada medida. La duración de cada prueba individual será de 20 segs/prueba.

Por último y con objeto de realizar un análisis posterior, se tomarán muestras negativas (sin el objeto) para cada serie en las mismas condiciones anteriores.

Umbral de éxito

Un resultado se considerará positivo o exitoso, cuando en la captura de imagen el algoritmo haya identificado correctamente el objeto mediante su señalización.

En cambio, el resultado de la captura será fallido o negativo si en la captura no se señala al objeto, si no se señala a ningún objeto o si se señala a un lugar erróneo.

7.1.2. Evaluación de resultados: Curvas ROC

El análisis de los resultados se llevará a cabo mediante las curvas ROC. Las curvas ROC (acrónimo de Receiver Operating Characteristic, o Característica Operativa del Receptor) son un método gráfico de evaluación de un clasificador binario (acierto/fallo) en función del umbral de clasificación.

Para cada uno de los umbrales y con las capturas positivas (con objeto) y las negativas (sin objetos), se elaborará una matriz con los verdaderos positivos, los falsos positivos, los verdaderos negativos y los falsos negativos de la manera que se ilustra en la figura 7.3.

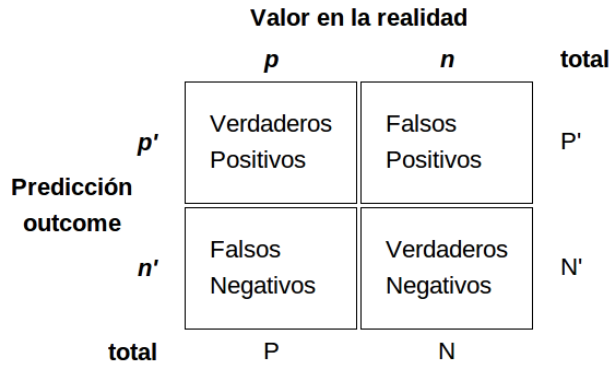


Figura 7.3: Matriz de confusión

Con estos resultados se obtienen dos parámetros: el ratio de falsos positivos (ecuación 7.1) y el ratio de verdaderos positivos (ecuación 7.2).

$$RFP = \frac{FP}{FP + VN} \tag{7.1}$$

$$RVP = \frac{VP}{VP + FN} \tag{7.2}$$

Donde:

- RFP es el Ratio de Falsos Positivos
- FP son los Falsos Positivos
- VN son los Verdaderos Negativos
- RVP es el Ratio de Verdaderos Positivos
- VP son los verdaderos positivos
- FN son los falsos negatvos

Con estos datos se elabora una gráfica (ejemplo en figura 7.4) donde el eje y corresponde al **RVP** y el eje x al **RFP**. Después se representan los resultados de las ecuaciones para cada umbral y uniendo estos puntos se forma una recta cuyo análisis simplificado indica que cuanta más área se encuentre debajo de la curva, mejor es el clasificador.

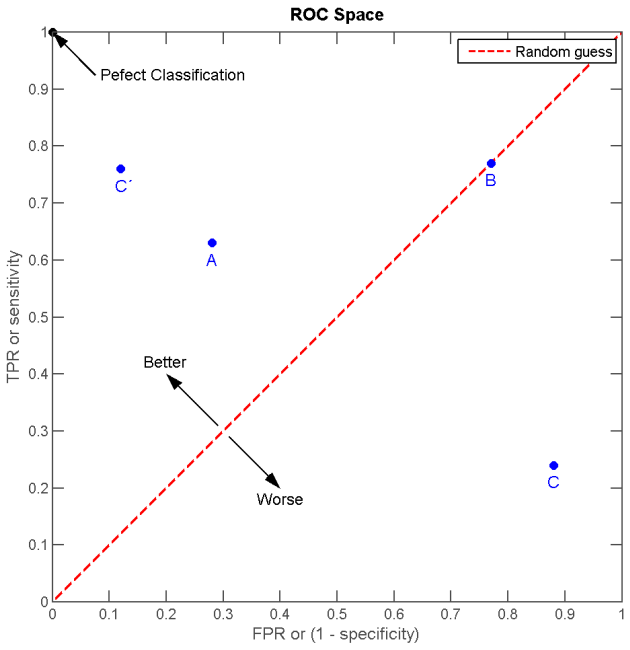


Figura 7.4: Curvas ROC: Ejemplo

7.1.3. Resultados e interpretación

A continuación se muestran los resultados obtenidos y la interpretación que se extrae de ellos.

Incompatibilidades y consideraciones previas

Debido al diseño propuesto del experimento y al uso de las curvas ROC como método de análisis, existen varias incompatibilidades con los algoritmos:

- Meanshift: Este algoritmo, por la manera en la que fue creado, no contiene un sistema o variable para poder variar la precisión, por lo que no es posible medir su respuesta a distintos umbrales.
- Camshift: Al ser un método derivado de Meanshift, adolece del mismo problema que éste.
- Template Matching: Esta función fue concebida originalmente para detectar patrones en elementos estáticos, y fue en este proyecto donde se adaptó para trabajar en entornos dinámicos. Por este motivo, los resultados aquí mostrados se puede considerar sesgados o intencionados, ya que el número de aciertos o fallos que obtenga el algoritmo dependerán directamente del número de veces que el objeto se encuentre en la misma posición que la captura original.

Por otro lado y de manera previa a la exposición de resultados, indicar estos serán, de manera general, peores que los obtenidos normalmente en este tipo de análisis.

El motivo está en el funcionamiento de las curvas ROC. Estas están pensadas para evaluar clasificadores binarios, y en este proyecto se usan así, salvo por una particularidad: la localización. El obstáculo adicional para nuestros algoritmos es tener que situar además correctamente el objeto en el espacio, no sólo indicar que está presente. Por ello existe una cierta desviación hacia resultados negativos en el análisis.

Colormax

Colormax fue puesto a prueba en el experimento ya explicado, con los siguientes resultados:

Para la pieza a color (figura 7.5) este método ofrece buenos resultados, especialmente cuando aumentamos la precisión, ya que de esta forma aumenta el número de verdaderos negativos, al ignorar puntos que se parezcan ligeramente al original.

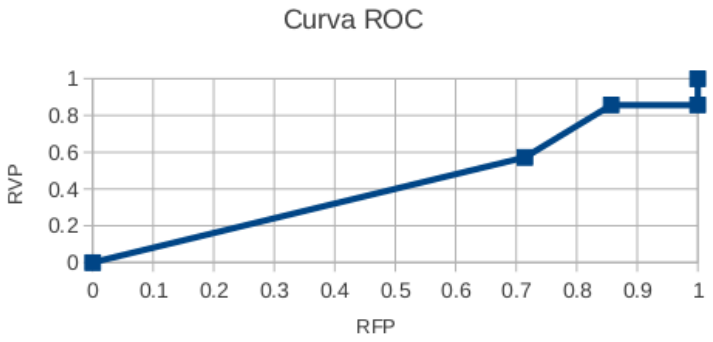


Figura 7.5: Curvas ROC: Colormax-Pieza color

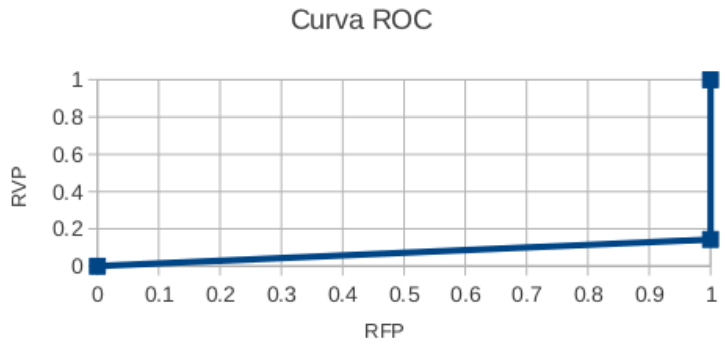


Figura 7.6: Curvas ROC: Colormax-Pieza textura

En cambio, y como se puede ver en la figura 7.6, el resultado para una pieza de mucha textura es pésimo. Esto era de esperar, ya que el algoritmo no puede centrarse en un color y hace una mezcla de los presentes en la imagen, no consiguiendo así situar el objeto de ninguna manera.

Template Matching

Para este algoritmo y debido a su construcción interna, sólo se evaluó para una de las imágenes, ya que el resultado sería similar.

Como se puede observar (figura 7.7) el resultado es similar a lanzar una moneda al aire. Además, y como ya se ha comentado, las curvas están altamente influenciadas por la distancia a la que se encuentra la pieza.

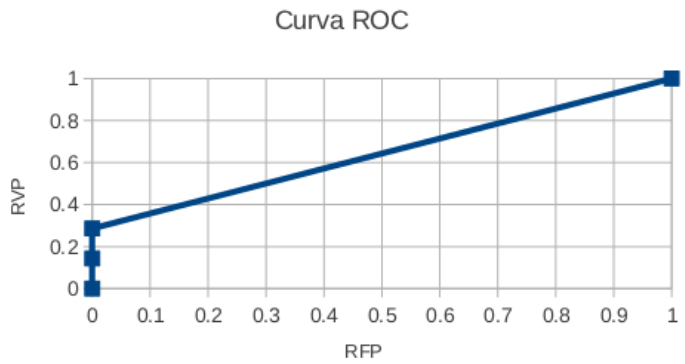


Figura 7.7: Curvas ROC: Template Matching

Algoritmos basados en SURF

Los métodos basados en la extracción de características usados en este proyecto tienen la misma base, y solamente varían en la interpretación final. Por ello sólo se evaluarán los resultados de uno de ellos, siendo estos extensibles al otro.

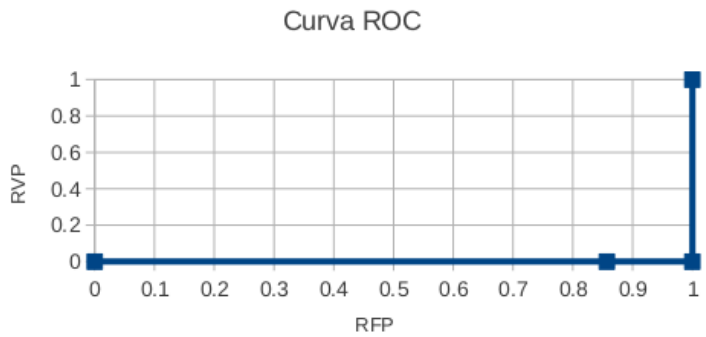


Figura 7.8: Curvas ROC: SURF-Pieza color

Como era de esperar, los aciertos al intentar buscar una pieza que sólo destaca por su color son nulos. En la figura 7.8 se puede comprobar que la extracción de características no tiene cabida en una sistema de detección por color.

En cambio si la pieza contiene cierta textura, o al menos la suficiente para extraer alguna característica, la eficiente de SURF mejora (figura 7.9). Al aumentar la exigencia del filtro, se eliminan muchos falsos positivos, pero a su vez disminuyen los verdaderos positivos, haciendo más difícil situar el objeto en el lugar correcto.

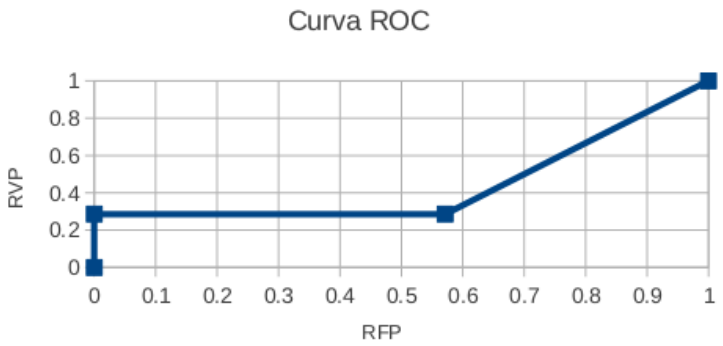
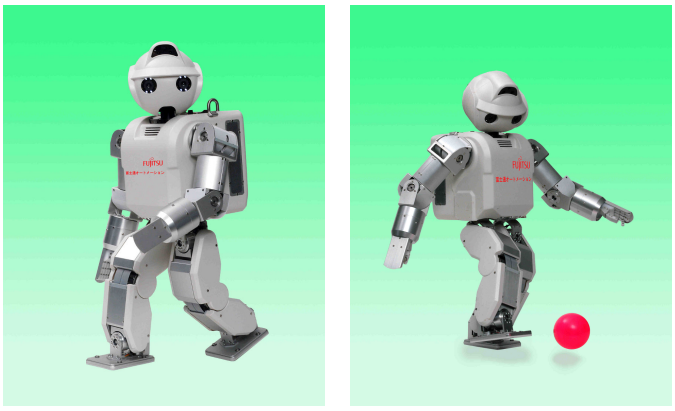


Figura 7.9: Curvas ROC: SURF-Pieza textura

7.2. El robot HOAP-3

Para la demostración se eligió el robot humanoide HOAP-3. Diseñado por Fujitsu, este pequeño robot de 60 cm y 8.8 Kg. de peso (figura 7.10(a)), resulta perfecto para la investigación, pues está pensado como un sistema de código abierto. De hecho, porta en su interior el sistema operativo libre RT-Linux.



(a) (b)

Figura 7.10: HOAP-3 en distintas posturas ©Fujitsu

En cuanto a su anatomía, posee 28 grados de libertad (6 por pie, 5 por brazo, 1 en la cintura, 1 por mano y 3 en el cuello) lo que permite una gran cantidad de movimientos. También viene equipado con un procesador Pentium M a 1.1 Ghz, un módulo Wifi, leds, dos cámaras, altavoces y micrófonos. etc. Para la navegación y la interacción, el HOAP-3 tiene sensores de presión, de aceleración y de distancia.

7.3. Descripción de la demostración

Entre los objetivos principales de la demostración, se pretende que el robot identifique objetos y almacene una captura de ellos. Después, y usando los algoritmos diseñados en este proyecto, el HOAP-3 tendría que ser capaz de rastrear los objetos que se le indiquen, en base a su color. Como confirmación de que ha encontrado el objeto, el robot debería ser capaz de girar la cabeza en la dirección del movimiento del objeto.

En cuanto a los algoritmos de detección de movimiento, se pretende que el robot pueda capturar movimiento a través de sus cámaras y seguir el foco de movimiento.

Todo el proceso, desde la captura inicial de los objetos por la cámaras del robot, hasta la ejecución de los algoritmos, tiene que poder ser gestionado y controlado por RETINA, la interfaz desarrollada en este proyecto.

Con estas metas como guías, se obtendría un ejemplo completo que unido a la interfaz creada, completaría un sistema amigable en el que probar distintos algoritmos de visión artificial.

7.3.1. Documentación de la demostración

La pruebas se llevaron a cabo en una de las naves disponibles en el edificio Agustín de Betancourt del campus de Léganes de la Universidad Carlos III de Madrid (figura 7.11).

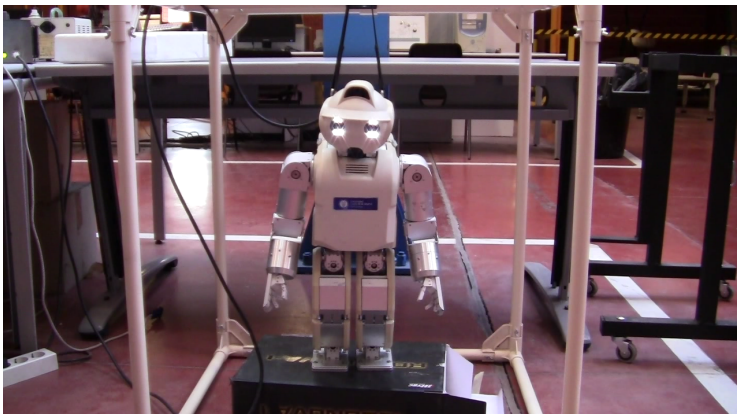


Figura 7.11: HOAP-3 preparado para la demostración

Para documentar la demostración se instaló una cámara fija que grababa al robot en todo momento. Por otro lado, un software capturaba la pantalla del ordenador donde se estaba ejecutando RETINA. Esta versión de la interfaz es una modificación de la original

para adaptarla al sistema de comunicaciones del HOAP-3 (basado en YARP) y para asegurar el control de los servos del motor.

La primera demostración consistió en ejecutar el sistema de detección de movimiento. Para ello una persona agitaría una pelota delante del robot. Este respondería girando la cabeza hacia el lado en el que se encontraría el objeto (figura 7.12).

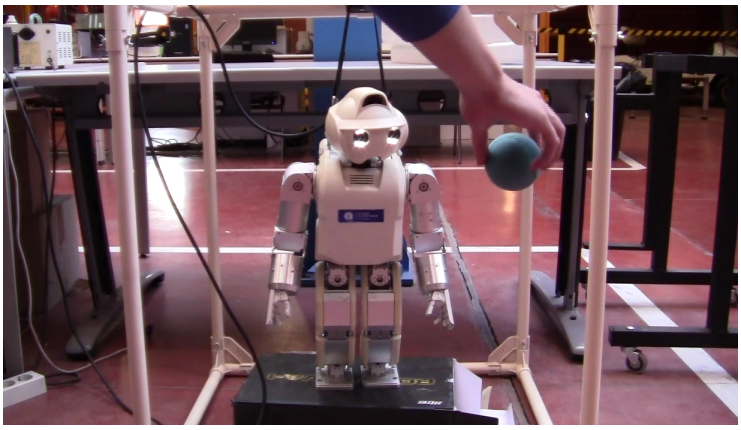


Figura 7.12: Demostración: detección de movimiento

La interfaz adaptada muestra el punto de vista del robot (figura 7.13). Desde esta misma interfaz, y usando las cámaras del robot, se tomaron las capturas iniciales de los objetos usados en la prueba.

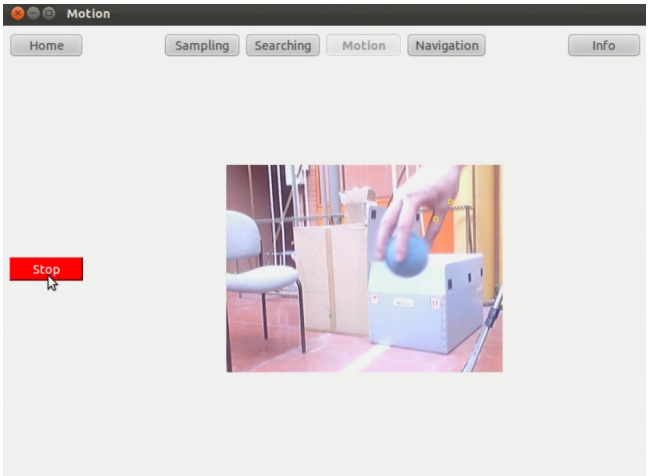


Figura 7.13: RETINA versión HOAP-3: Detectando Movimiento

En cuanto a los algoritmos de seguimiento, se probaron dos de ellos: Camshift y Meanshift. Template matching fue descartado debido a los bajos resultados de rendimiento que ofreció en las es-

tadísticas, y Colormax ofrece un resultado aceptable para confirmar la presencia del objeto deseado, pero no uno consistente para ejecutar una trayectoria de movimiento. Para los comandos de tipo visual servo, interesan algoritmos que tengan en cuenta el resultado de los fotogramas anteriores, para así poder realizar trayectorias suaves y no movimientos rápidos y aleatorios que podrían comprometer la estructura y la electrónica del robot.

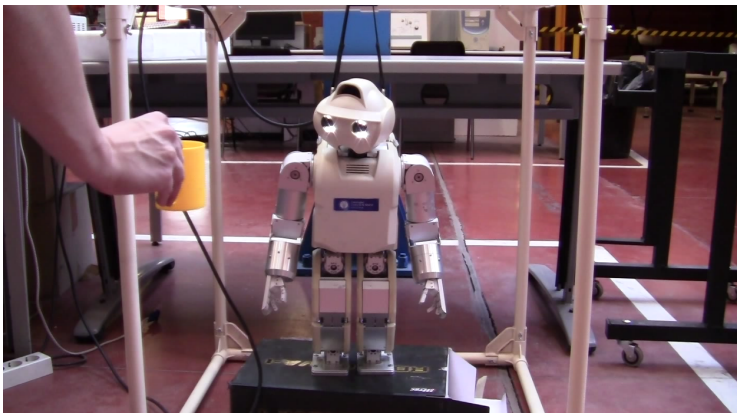


Figura 7.14: Demostración: rastreo por color I

En las pruebas de los algoritmos de búsqueda por color, y al igual que en el caso anterior con la detección de movimiento, desde la interfaz RETINA se puede observar cómo el algoritmo identifica un objeto y su rastreo posterior (figura 7.15).

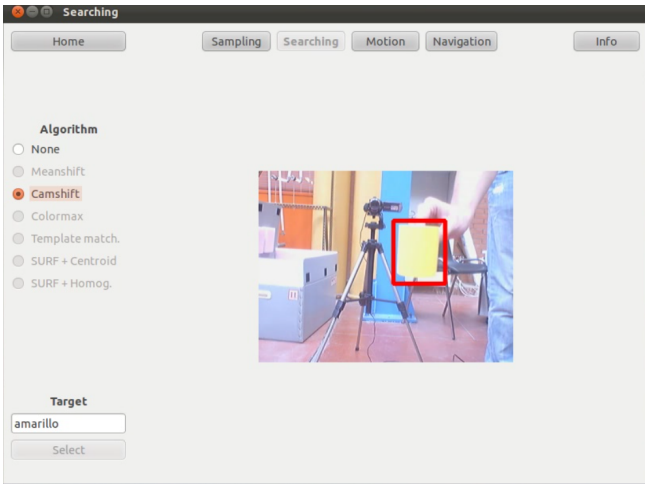


Figura 7.15: RETINA versión HOAP-3: Camshift

Como se puede comprobar en las figuras 7.14 y 7.16, los objetos

utilizados para la demostración varían en color y forma. Esto es así para demostrar la independencia de estas variables en la prueba.

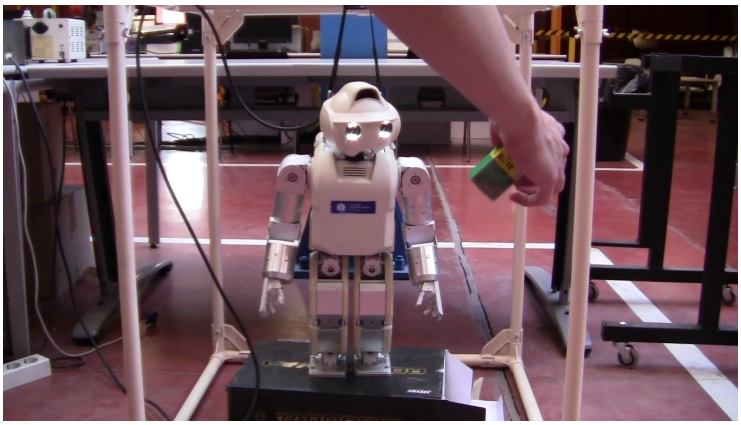


Figura 7.16: Demostración: rastreo por color II

Como conclusiones de esta demostración, podemos decir que el resultado general fue positivo. Los únicos inconvenientes fueron los derivados de la iluminación (quizá excesiva) de la nave, lo que dificultaba al programa la distinción de objetos claros del fondo.

La resolución máxima (320x240) y la frecuencia de fotogramas que puede ofrecer el robot está muy limitada, lo que también fue un factor limitante en el proceso. Es por ello que en las figuras 7.15 y 7.13 se aprecia un espacio vacío entre la imágen del vídeo y los botones. RETINA está adaptada para trabajar a resoluciones de 640x480.

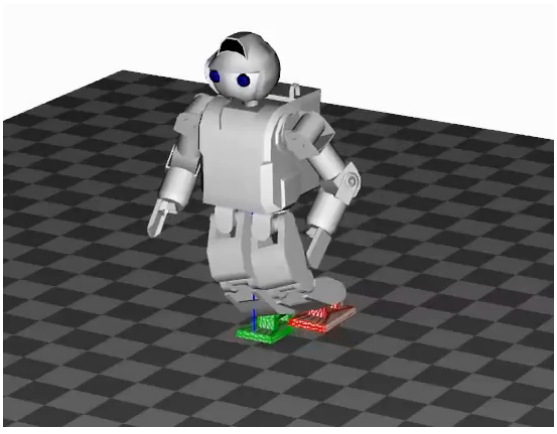


Figura 7.17: Robot dando un paso hacia adelante ©UCMERCED

Por otro lado, los algoritmos de navegación no fueron probados en el humanoide, debido a la propia estructura de éstos. Según han

sido diseñados, su función es dotar a robots móviles (con ruedas) de un método de navegación reactiva rápida para circular por entornos con paredes, obstáculos, etc. Sin embargo, el HOAP-3 es un humanoide y su movimiento es claramente distinto al esperado por estos algoritmos.

Un robot móvil, suponiendo uno común con ruedas y una cámara al frente, puede circular sin variar la altura de la cámara. En su recorrido, y a pesar de que haya curvas o cambios de velocidad, la perspectiva que obtiene es siempre la misma, pudiendo tomar como referencia el suelo constantemente.

En un robot humanoide, sin embargo, el movimiento se realiza dando pequeños pasos. Estos pasos implican una serie de movimientos que varían el ángulo y la altura de la cámara, perturbando la medida (ejemplo en la figura 7.17). Es por estos motivos que los algoritmos de navegación no están incluidos en esta demostración.

Conclusiones

Se presentan a continuación las conclusiones extraídas de la realización de este proyecto, así como una aproximación a las posibles extensiones que se podrían construir utilizando este trabajo como base.

7.4. Conclusión

Una vez finalizado el proyecto y mirando con perspectiva, podemos extraer varias conclusiones acerca del trabajo y de la visión artificial.

Con respecto a la visión artificial, destacar que se ha avanzado mucho últimamente, y el potencial de sus aplicaciones aumenta con cada nueva característica que se desarrolla. De hecho, la visión por computador es ya un elemento imprescindible en la robótica (y especialmente en la humanoide). No se concibe actualmente diseñar un robot, sin incluir algún elemento que permita la captura de imágenes.

Centrándonos en los algoritmos aquí expuestos, podemos concluir que cada algoritmo aborda con bastante solvencia su función, consiguiendo unos resultados aceptables (teniendo en cuentas las consideraciones previas de cada uno de ellos). Si bien también es cierto, que ninguno puede abarcar todas las situaciones (búsqueda por color y forma de manera conjunta). En cuanto a la interfaz gráfica se ha conseguido un entorno amigable y multiplataforma, que permite incluir algoritmos que hagan uso de la visión artificial de los robots.

De todo el trabajo desarrollado, destacar especialmente la parte de navegación, y en especial Navicolor. Este algoritmo es original de este proyecto, y es una característica innovadora en la navegación reactiva. Mediante el análisis de colores del entorno, permite la circulación de robots móviles sin colisionar con obstáculos.

Por otra parte, destacar que gracias a las magnificas librerías disponibles en la comunidad del software libre, el desarrollo de aplicaciones, incluso aquellas con interfaz gráfica, se hace muy rápido y

sencillo. El uso de un lenguaje común (C++ en las librerías aquí utilizadas) facilita la integración de todos los componentes necesarios en el mismo programa. Esto ayuda además en la reducción del consumo de memoria, que habría aumentado si hubiese habido que usar traductores o pasarelas entre lenguajes. Para hacer una contribución a la comunidad científica y a la del software libre, todo el código generado en este proyecto será liberado. Con esto se espera poder ayudar a las personas que quieran ver y usar el código.

Las conclusiones se resumen en que se han usado una serie de algoritmos efectivos en sus objetivos, se han integrado dentro de una interfaz atractiva y funcional, e incluso se ha conseguido aportar una nueva herramienta al campo de la navegación reactiva. Todo ello con un análisis estadístico asociado e implementado en un robot humanoide que ha servido para hacer demostraciones de uso.

7.5. Desarrollos futuros

Un posible desarrollo o trabajo derivado de este proyecto, sería la adición de un complemento 3D. En estos momentos, y gracias a algunas novedades como la cámara Kinect de Microsoft Corporation, se ha desarrollado un gran interés en la visión tridimensional. Como no podía ser de otra manera, el mundo de la robótica se ha puesto en cabeza en esta línea de investigación, pues los beneficios son abrumadores. La manipulación de objetos y la navegación en robots móviles son los principales beneficiados de esta tecnología. Además, la comunidad del software libre ha liderado este movimiento a través de la liberación de varias librerías a tal efecto. La principal y más conocida es OpenKinect / libfreenect, la cual ofrece una integración muy completa al mundo 3D. Es por ello que la inclusión de un módulo 3D en Retina sería una paso adelante en la mejora de los algoritmos actuales.

Otra posible línea de investigación sería la mejora, a nivel interno, de los algoritmos aquí usados. Algunas funciones tienen limitaciones que podrían ser cubiertas implementando nuevos métodos, o combinaciones de varios. Por poner un ejemplo, se podría crear un algoritmo de fuerza bruta que partiese de template matching y analizase posibles rotaciones o cambios de escala. El reto en este caso es hacerlo eficiente y manejable por un ordenador convencional. También en este apartado sería posible pensar en unas posibles etapas previas a los algoritmos. Algo parecido a un filtro que adaptase la imagen a cada algoritmo de manera adecuada.

De manera más informal, es posible incluir RETINA en algu-

na plataforma móvil en forma de aplicación (conocidas ahora como *apps*). Gracias a los kits de desarrollo (SDK en inglés) proporcionados por los principales fabricantes de sistemas operativos móviles (Apple y Google en este momento), la creación de aplicaciones está en auge y el desarrollo de éstas es rápido para los programadores acostumbrado a los lenguajes usados (Java en Android, Objective-C en iOS).

La dificultad en este caso, estriba en el alto consumo de memoria de Retina y el gasto de batería que supondría, retos estos que se tendrían que solventar con imaginación y esfuerzo.

De esta manera se cubren los trabajos derivados de este proyecto que podrían surgir y se anima a futuros estudiantes e investigadores en su continuación.

Bibliografía

[1] J.G. Allen, R.Y.D. Xu, and J.S. Jin. Object tracking using camshift algorithm and multiple quantized feature spaces. In *Proceedings of the Pan-Sydney area workshop on Visual information processing*, pages 3–7. Australian Computer Society, Inc., 2004.

[2] R.O. Ambrose, H. Aldridge, R.S. Askew, R.R. Burridge, W. Bluethmann, M. Diftler, C. Lovchik, D. Magruder, and F. Rehnmark. Robonaut: Nasa's space humanoid. *Intelligent Systems and their Applications, IEEE*, 15(4):57–63, 2000.

[3] J.M. Armingol, A. de la Escalera, C. Hilario, J.M. Collado, J.P. Carrasco, M.J. Flores, J.M. Pastor, and F.J. Rodriguez. Ivvi: Intelligent vehicle based on visual information. *Robotics and Autonomous Systems*, 55(12):904–916, 2007.

[4] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

[5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, pages 404–417, 2006.

[6] S.S. Beauchemin and J.L. Barron. The computation of optical flow. *ACM Computing Surveys (CSUR)*, 27(3):433–466, 1995.

[7] J.Y. Bouguet et al. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. *Intel Corporation, Microprocessor Research Labs, OpenCV Documents*, 3, 1999.

[8] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, 2008.

[9] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, 1986.

[10] Y. Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.

[11] R.T. Collins. Mean-shift blob tracking through scale space. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–234. IEEE, 2003.

[12] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, 1975.

[13] M. González-Fierro, A. Jardón, S. Martínez de la Casa, M.F. Stoelen, J.G. Víctores, and C. Balaguer. Educational initiatives related with the ceabot contest.

[14] S. Hutchinson, G.D. Hager, and P.I. Corke. A tutorial on visual servo control. *Robotics and Automation, IEEE Transactions on*, 12(5):651–670, 1996.

[15] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[16] B.D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In *International joint conference on artificial intelligence*, volume 3, pages 674–679. Citeseer, 1981.

[17] Larry Matthies, Mark Maimone, Andrew Johnson, Yang Cheng, Reg Willson, Carlos Villalpando, Steve Goldberg, Andres Huertas, Andrew Stein, and Anelia Angelova. Computer vision on mars. *International Journal of Computer Vision*, 2007.

[18] N.J. Nilsson. *Principles of artificial intelligence*. Springer Verlag, 1982.

[19] N. Otsu et al. A threshold selection method from gray-level histograms. *IEEE Transactions on systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[20] M.A. Salichs, R. Barber, A.M. Khamis, M. Malfaz, J.F. Gorostiza, R. Pacheco, R. Rivas, A. Corrales, E. Delgado, and D. García. Maggie: A robotic platform for human-robot social interaction. In *Robotics, Automation and Mechatronics, 2006 IEEE Conference on*, pages 1–7. Ieee, 2006.

[21] R. Szeliski. *Computer vision: Algorithms and applications*. Springer-Verlag New York Inc, 2010.

[22] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *The 2005 DARPA Grand Challenge*, pages 1–43, 2007.

[23] J. G. Víctores, A Jardón, M. F. Stoelen, S. Martínez, and C. Balaguer. Asibot assistive robot with vision in a domestic environment. *Robocity2030 7th Workshop. Móstoles. Spain. Oct, 2010. Visión en Robótica. ISBN: 84-693-6777-3. Universidad Rey Juan Carlos. pp.61-74. 2010.*

Anexo: Documentación de la librería Travis

Se muestra a continuación la documentación del código de la librería Travis, realizada con un generador de documentación automático. En nuestro caso, el documentador elegido es Doxygen.

Chapter 1

Module Documentation

1.1 Travis

Functions

- void **OptFlowDetec** ()
- void **NaviOtsu** ()
- void **NaviColor** ()
- void **SearchColor** (int option, int accuracy, Rect selection, IplImage ∗selectedImage)
- void **TemplateMatching** (int accuracy, Rect selection, IplImage ∗selectedImage)
- void **SurfCentroid** (int accuracy, IplImage ∗selectedImage)
- void **SurfHomography** (int accuracy, IplImage ∗selectedImage)

1.1.1 Detailed Description

Travis (TRAcking VISion library)

Author: Santiago Morante

1.1.2 Function Documentation

1.1.2.1 void NaviColor ( )

Navigation by checking color floor and finding similar on 3 directions.

1.1.2.2 void NaviOtsu ( )

Reactive navigation by dinamic Otsu thresholding method.

1.1.2.3 void OptFlowDetec ( )

This function detects moving areas on live video.

1.1.2.4 void SearchColor (int *option*, int *accuracy*, Rect *selection*, IplImage * *selectedImage*)

This function contains 3 methods to follow objects by color.

Parameters

option	is the algorithm selector (1 = Camshift,2 = Meanshift,3 = Colormax).
accuracy	is the required precission (1 = Low, 2 = Medium, 3 = High).
selection	is the rectangle that contains colors to find.
selectedImage	is the image to select color to track.

1.1.2.5 void SurfCentroid (int *accuracy*, IplImage * *selectedImage*)

Object recognition by using SURF algorithm. It also indicates the most probably centroid.

Parameters

accuracy	is the required precission (1 = Low, 2 = Medium, 3 = High).
selectedImage	is the template image to be found.

1.1.2.6 void SurfHomography (int *accuracy*, IplImage * *selectedImage*)

Object recognition by using SURF algorithm. It also calculates the homography.

Parameters

accuracy	is the required precission (1 = Low, 2 = Medium, 3 = High).
selectedImage	is the template image to be found.

1.1.2.7 void TemplateMatching (int *accuracy*, Rect *selection*, IplImage * *selectedImage*)

It finds similar images using a template.

Parameters

accuracy	is the required precission (1 = Low, 2 = Medium, 3 = High).
selection	is the rectangle that contains small image to find.
selectedImage	is the image to select small image to track.