Treball Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques

Universitat de Barcelona

Control de un Robot Mediante Técnicas de Visión

Juan Carlos Calvo Tejedor

Director: Eloi Puertas i Prats

Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 10 de Juny de 2013

Agradecimientos

En primer lugar, quiero agradecer a mi familia el incansable apoyo a lo largo de todos estos años.

A Jessica, por ser la mejor compañera y apoyarme en todo momento.

A los amigos que han aportado su granito de arena.

Y por último, me gustaría agradecer a mi tutor, Eloi Puertas i Prats, por la confianza y el apoyo que ha mostrado en todo momento, y haberme dado la oportunidad de realizar el proyecto que tenía en mente.

A todos, sinceramente, gracias.

Resumen

En este proyecto se aborda el diseño e implementación de un sistema robótico dotado de visión por computador, capaz de realizar la detección y tracking de un objetivo específico en un determinado entorno.
El robot se ha diseñado según el paradigma de la robótica basada en comportamientos, e implementado bajo la plataforma Lego Mindstorms NXT.
Dado el importante incremento de las capacidades de los dispositivos móviles, el sistema de visión artificial se ha implementado mediante un dispositivo Android, utilizando la librería OpenCV4Android como soporte para el procesado de imágenes.
En cuanto a la integración de los dos dispositivos en un único sistema, se ha establecido una comunicación Bluetooth como canal de intercambio de datos. En concreto, el dispositivo móvil se encarga de proveer al robot con los datos relacionados con la posición del objeto en la imagen. De esta forma, se han extendido las capacidades del robot mediante la acoplación de la cámara como un sensor más.
Se ha realizado una comparación entre utilizar un método de detección frame a frame para realizar el tracking del objetivo, y la combinación de éste con un método especializado de tracking. Se ha comprobado que utilizando el método de tracking de Lucas-Kanade una vez detectado el objetivo se obtiene un incremento notable del rendimiento.

Abstract

In this project, a robotic system provided with computer vision is designed and implemented. The robot is capable of detecting and tracking a specific target in a certain environment.

The robot is designed according to the paradigm of behaviour-based robotics, and implemented under the Lego Mindstorms hardware.

Due to the significant increase of mobile devices capabilities, the artificial vision system has been implemented through an Android device, using the OpenCV4Android library as a support for image processing.

Regarding the integration of the two devices in one system, it has established a Bluetooth comunication as data exchange channel. Specifically, the mobile device is responsible for providing the data concerning the position of the object in the image to the robot. Thus, the capabilities of the robot have been extended by integrating the camera as another sensor of the NXT.

A comparison was made between using a detection method in each frame to perform the tracking of the target, and the combination of the detector with a specialized tracking method. It has been observed that by using the Lucas-Kanade tracking algorithm once the target have been detected, a considerable performance increase is obtained.

Resum

En aquest projecte es du a terme el disseny i la implementació d'un sistema robòtic dotat de visió per computador, capaç de realitzar la detecció y tracking d'un objectiu específic en un determinat entorn.
El robot s'ha disenyat segons el paradigma de la robòtica basada en comportaments, i s'ha implementat sota la plataforma Lego Mindstorms NXT.
Donat l'important increment de les capacitats dels dispositius móbils, el sistema de visió artificial s'ha implementat mitjançant un dispositiu Android, utilitzant la llibrería OpenCV4Android com a suport per el processat d'imatges.
Per a la integració dels dos dispositius en un únic sistema, s'ha establert una comunicació Bluetooth com a canal d'intercanvi de dades. En concret, el dispositiu móbil s'encarrega de subministrar al robot les dades relacionades amb la posició de l'objecte a la imatge. D'aquesta forma, s'han ampliat les capacitats del robot mitjançant l'acoplament de la càmera com un sensor més.
S'ha fet una comparació entre utilitzar un mètode de detecció fram a frame per a realitzar el tracking de l'objectiu, i la combinació d'aquest amb un mètode especialitzat de tracking. S'ha comprobat que utilitzant el mètode de tracking de Lucas-Kanade un cop s'ha detectat l'objectiu, s'obtè un increment notable en el rendiment.

Índice

1. Introducción

1.1. Motivación

El contínuo desarrollo de los vehículos autónomos ha otorgado una mayor importancia a los sistemas de visión por computador. Muchos modelos de coches comunes ya incluyen sensores y cámaras para la prevención de accidentes y atropellos, o incluso para asistencia al conductor en el estacionamiento. En concreto, en los sistemas autónomos son imprescindibles los métodos de detección y tracking de objetos, ya que se utilizan para detectar objetivos concretos u obstáculos, o para el reconocimiento de personas.

Paralelamente, ha surgido en los últimos años un creciente interés por la introducción de la robótica en la educación. Este fenómeno se debe en parte a la aparición de distintas plataformas robóticas de bajo coste, que pretenden acercar la robótica a un tipo de público que antes no tenía acceso a ella. Ejemplos de estas plataformas pueden ser Arduino, una placa electrónica de código abierto que permite implementar de forma sencilla el control de motores y sensores, o Lego Mindstorms, una plataforma robótica basada en piezas de Lego, que incluye un brick inteligente sencillo de programar mediante software gráfico, y un conjunto de actuadores y sensores.

La motivación principal que empuja este proyecto es la necesidad de unir la robótica y la visión por computador para crear sistemas autónomos más efectivos que puedan operar en determinados entornos realizando unas tareas concretas.

Este proyecto plantea un sistema que une la flexibilidad de una plataforma robótica con la poténcia de un dispositivo móvil. Esto representa un reto ambicioso, ya que implica un dominio mas o menos amplio de un conjunto de tecnologías diferentes trabajando en un mismo sistema: Android, OpenCV y NXT.

1.2. Definición del problema

En la Universidad de Barcelona se ha realizado en los últimos años una competición de robots Mindstorms que tiene lugar el día de la Matefest-Infofest. En ella participan alumnos de ESO y Bachillerato que programan sus robots en sesiones dirigidas por el profesor Eloi Puertas.

La competición consiste en varias rondas en las que dos robots alternan los roles de cazador y presa. El escenario en el que se desarrolla (Figura 1) consta de un fondo negro delimitado por zonas blancas que

los robots no pueden traspasar.

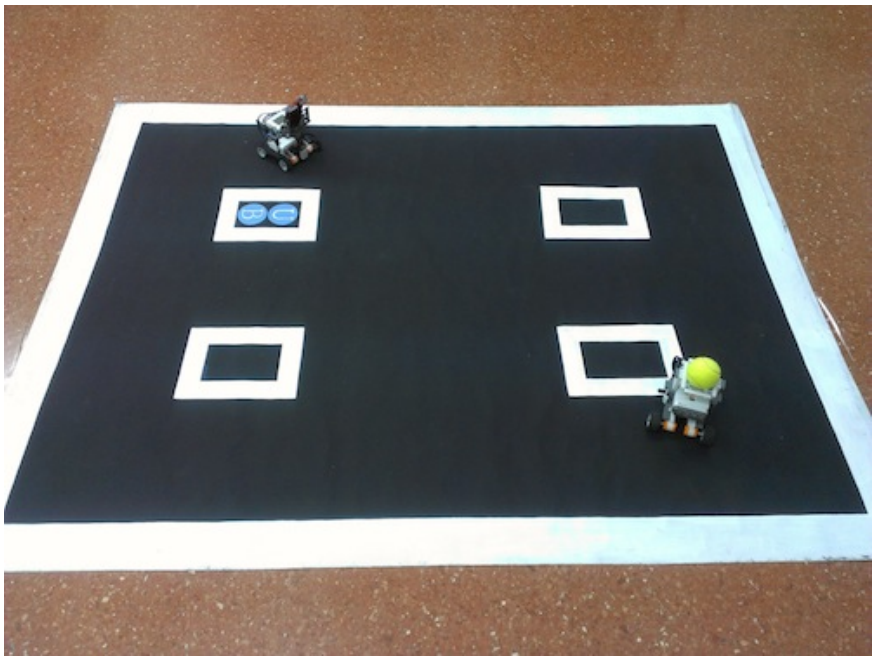


Figura 1: Escenario de la Matefest-Infofest en el que se desarrolla la competición de robots Mindstorms.

Aprovechando los robots Mindstorms y el escenario del que disponemos, se plantea incrementar las capacidades del robot mediante un sistema de visión por computador. De esta forma, el robot será capaz de detectar a su objetivo y seguirlo a través del escenario de una forma más eficaz. Típicamente se cuenta con los sensores de luz y ultrasonidos, pero éstos no aportan la suficiente información para localizar de forma precisa al objetivo.

Por lo tanto, el robot realizará la detección y el seguimiento de un objetivo mediante algoritmos de visión por computador, y deberá ser capaz de alternar distintos comportamientos para desplazarse hacia su objetivo y evitar los obstáculos cuando sea necesario.

1.3. Objetivos del proyecto

Este proyecto tiene como propósito ser una primera toma de contacto con la robótica y la visión por computador, a través de un problema de cierta complejidad que se debe resolver en un tiempo limitado.

El objetivo general es implementar un sistema robótico que sea capaz de utilizar la visión por computador para guiar su movimiento hacia un objetivo móvil en un determinado entorno. La tarea principal del sistema de visión será la detección y el seguimiento de la posición de un determiando objeto en la imagen, y proveer de estos datos al robot para que éste actúe en consecuencia.
En concreto, los objetivos específicos que se plantean y debe cumplir el sistema implementado son los siguientes:

- El robot debe ser capaz de reconocer y localizar a su objetivo
- El robot debe ser capaz de realizar un seguimiento del objetivo
- El robot debe ser capaz de evitar las zonas del escenario que no puede traspasar

1.4. Estructura del documento

La memoria se divide en 6 secciones claramente diferenciadas, más una sección para las referencias bibliográficas y una sección de anexos. A continuación se detallan los contenidos expuestos en cada apartado.

1 Introducción

En este primer apartado se ponen de manifiesto los argumentos que respaldan este proyecto. Se expone la motivación del proyecto, se define el problema y se detallan los objetivos generales y específicos que se llevarán a cabo.

2 Estado del arte

En este apartado se realiza una breve introducción a la robótica y la visión por computador. Respecto a la robótica, se introducen los conceptos de robots reactivos y robots basados en comportamientos. En cuanto a la visión por computador, se introducen los métodos más importantes de detección y tracking de objetos.

3 Análisis del problema

En esta sección se realiza un ánalisis en profundidad del problema, se expone la solución propuesta, se realiza un análisis de las tecnologías que se utilizarán para llevar a cabo el proyecto justificando su uso, y se explican las limitaciones del sistema planteado.

4 Diseño e implementación

En esta sección se detallan aspectos del diseño y la programación del sistema planteado. Se expondrá el diseño del robot Mindstorm, la implementación de la parte de visión mediante Android, la comunicación Bluetooth, y se explicará la aplicación Android y la programación del robot.

5 Observaciones y resultados obtenidos

En este apartado se realizará una exposición de los resultados obtenidos y un análisis de éstos.

6 Conclusiones y futuros proyectos

En esta sección se expondrán las conclusiones extraídas de la realización del proyecto y se comentarán posibles mejoras y proyectos futuros.

7 Referencias

En este apartado se nombrarán todas las fuentes que han servido de referencia para la realización de este proyecto.

8 Anexos

En ésta útlima sección se incluye información relacionada con la instalación y configuración del software necesario para la realización del proyecto, y con la utilización del sistema resultante de la realización del proyecto.

2. Estado del arte

2.1. Robótica

2.1.1. Introducción a la robótica

La robótica es una disciplina tecnológica que se dedica al diseño y construcción de robots. Es un campo interdisciplinar que combina la electrónica, mecánica, informática e incluso inteligencia artificial.
Los robots se clasifican según cronología y según su arquitectura.

- Por cronología: 1ª, 2ª, 3ª y 4ª generación.
- Por arquitectura: Poliarticulados, móviles, androides, zoomórficos e híbridos.

Una definición importante en el marco de este proyecto es la del concepto robot, a pesar de que no existe un consenso sobre su definición. Por lo tanto se ha tomar una de las múltiples definiciones que se pueden encontrar según la fuente. En este caso, la definición que se ha elegido es la que da la Robotics Industry Association (RIA).
"Un robot es un manipulador reprogramable y multifuncional, diseñado para mover materiales, partes, herramientas o dispositivos especializados a través de movimientos variables programados para la realización de una variedad de tareas".

Figura 2: Imagen de un robot.

Una vez introducida la robótica y el concepto de robot, se introducirá en el siguiente apartado el paradigma principal en el que se basará el robot que se va a diseñar en este proyecto: el paradigma reactivo.

2.1.2. Robots reactivos

El paradigma de la robótica reactiva emergió a finales de los años 80, fruto de la poca efectividad de los robots que combinaban la Inteligencia Artificial con la Visión por Computador, los cuales se basaban en representaciones abstractas del entorno. El problema principal era que no resultaban precisos en entornos reales. Una explicación más detallada de este punto se puede encontrar en el libro de Sergi Bermejo [6].
La robótica reactiva se basa en un principio muy simple: percibir y actuar. Esto se consigue a través de una serie de reglas que establecen una conexión directa entre la percepción y la acción (lo que podemos definir como comportamiento). Por lo tanto, un comportamiento es simplemente una transformación de una percepción en una serie de órdenes a los actuadores.
Este enfoque sencillo permite que este tipo de robots sean rápidos y puedan operar en tiempo real. Esto es debido a dos razones:

- Los comportamientos requieren algoritmos sencillos
- El sistema no necesita guardar datos en memoria

Por lo expuesto anteriormente, este tipo de robots son simples y poseen una inteligencia muy limitada, ya que no tienen la capacidad de aprender de sus percepciones previas. Por otro lado, este tipo de sistemas son más económicos debido a que no tienen necesidades de hardware tan elevadas como otros sistemas.

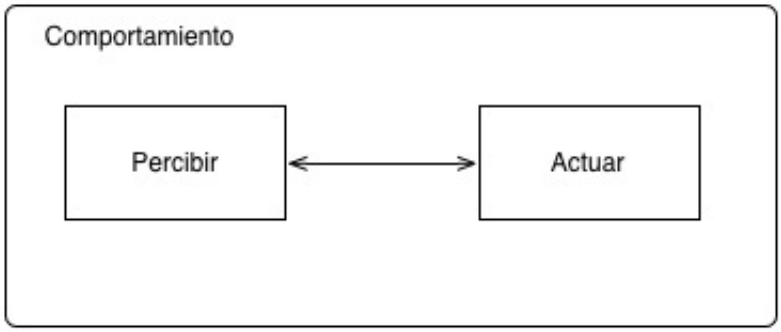


Figura 3: Organización en la arquitectura de percibir y actuar (paradigma reactivo).

Este tipo de arquitecturas son indicadas en sistemas robóticos que han de llevar a cabo tareas limitadas en un determinado entorno debido a que, como ya se ha comentado, no tienen la capacidad de aprender de las percepciones previas y así adaptarse a condiciones cambiantes.
El paradigma reactivo es importante dentro de la robótica, ya que forma la base de otros tipos de arquitecturas como la híbrida, que combina el componente reactivo y el deliberativo.

2.1.3. Descomposición en comportamientos

Los robots basados en comportamientos (behaviour-based robots) se enmarcan dentro de la robótica reactiva. En este paradigma los comportamientos están organizados en capas y pueden ejecutarse secuencialmente o de forma concurrente. Por otro lado, existe una capa superior que se encarga de gestionar la prioridad de los comportamientos.
Cada comportamiento es independiente y puede estar asociado a un sensor. De esta forma ninguno tiene conocimiento de lo que están percibiendo o haciendo los demás comportamientos. Por lo tanto, a partir de un sistema robótico reactivo, solo se han de descomponer sus funcionalidades en diferentes comportamientos y establecer un sistema que coordine la ejecución de los comportamientos.
Las ventajas de programar mediante comportamientos son varias. Los comportamientos son modulares y se pueden ir añadiendo al robot, lo que pemrmite ampliar sus capacidades gradualmente. Además, se pueden diseñar comportamientos complejos a partir de la combinación de comportamientos primitivos.
Los dos comportamientos más comunes en este tipo de robots suelen ser: dirigirse hacia un punto destino (move-to-goal) y evitar algún tipo de obstáculo (avoid-obstacle). Ésta forma de dividir las acciones que debe ejecutar el robot en función del entorno permite programar de forma sencilla robots que sean capaces de seguir una línea, o realizar tracking de un objetivo para llegar hasta él como en el caso que se plantea en este proyecto.

2.2. Visión por computador

2.2.1. Introducción a la visión por computador

La visión por computador es un campo que se sitúa dentro de la inteligencia artificial y que se centra en el estudio de la información relacionada con las imágenes. Esta disciplina surgió a principios de los años 70, y su objetivo principal es entender escenas del mundo real y producir información simbólica que pueda servir para la toma de decisiones. Las investigaciones se centran en intentar desarrollar técnicas matemáticas que permitan a un computador imitar las capacidades de la visión humana. A pesar de que los humanos somos capaz de percibir y procesar imagenes 3D de nuestro entorno con aparente facilidad, éste no es un problema trivial a la hora de trasladarlo a un computador. De hecho, los psicólogos llevan décadas intentando entender como funciona el sistema visual humano, y sin embargo sigue sin conocerse el funcionamiento completo.

2.2.2. Ámbitos de aplicación

Las aplicaciones de la visión por computador son muy amplias y abarcan desde la supervisión de la calidad de productos en una línea de fabricación industrial, hasta algo tan común como una cámara de fotos, ya que la mayoría son capaces de detectar caras. Incluso hoy en día, cada vez más coches incorporan sistemas de visión artificial para aumentar su seguridad.
A continuación se introducen brevemente algunos de los campos más importantes en los que se desarrolla la visión por computador.

1-Vehículos autónomos

Ésta es una de las áreas más recientes de aplicación de la visión por computador. El fin que persigue es el de crear vehículos, ya sean sumergibles, aéreos, o de tierra, que puedan operar de forma autónoma sin la intervención de un humano. La misión de estos vehículos suele ser la producción mapas del entorno o la detección de determinados eventos.
También se enmarcan dentro de este ámbito los vehículos que solamente dan soporte al conductor o piloto mediante sus sistemas de visión artificial. Sus aplicaciones más comunes son la alerta de obstáculos

en coches o los sistemas de aterrizaje autónomos en aviones.

Los ámbitos de aplicación donde están más desarrollados son el militar, donde se usan en misiles guiados o vehículos aéreos no tripulados, y el espacial, en el que se usan vehículos autónomos con sistemas de visión para tareas de exploración espacial.

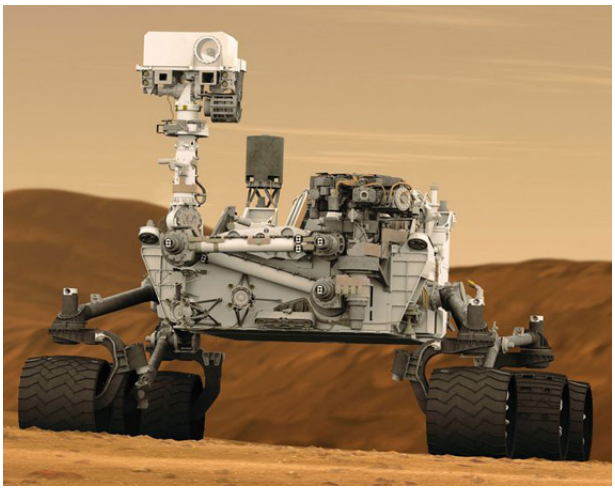


Figura 4: Curiosity, el vehículo autónomo explorador de la NASA diseñado para recorrer Marte recogiendo muestras para analizar la capacidad del planeta para albergar vida.

2-Medicina

Uno de los ámbitos de aplicación en auge es la medicina. En este ámbito, se usa la visión por computador con el fin de crear y procesar imágenes del cuerpo humano con fines preventivos y diagnósticos. Algunos tipos de imágenes clínicas en las que se aplica son las radiografías, resonancias magnéticas o ultrasonidos. Los objetivos entre otros son la identificación de tumores, detección de melanomas en la piel, evaluación del flujo sanguíneo o detección de la actividad cerebral.

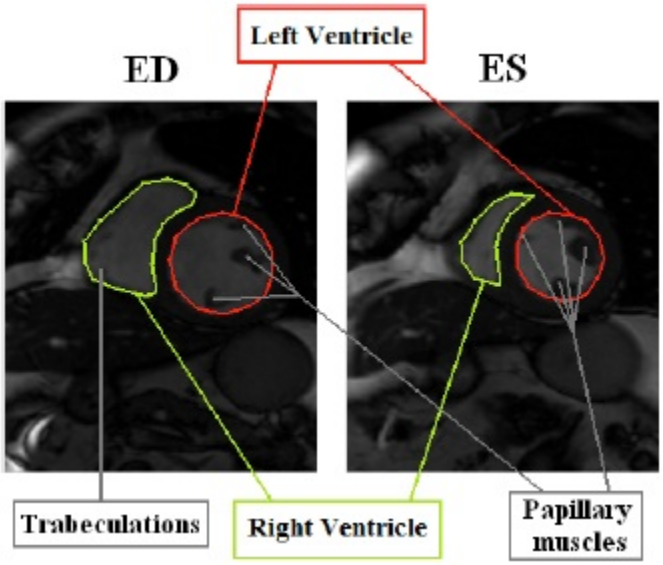


Figura 5: Detección de los ventrículos derecho e izquierdo mediante segmentación en distintos tipos de imágenes.

3-Militar

Por supuesto, otro gran ámbito de aplicación es el militar, en el cual la visión artificial es una de las técnicas más importantes para diseñar armas más inteligentes. Se aplica para analizar las imágenes de satélites militares, para guiar misiles hacia objetivos específicos (incluso en movimiento) o para detectar vehículos y soldados enemigos ocultos. Otra aplicación muy desarrollada en los últimos años en esta industria son los Vehículos Aéreos No Tripulados (UAV por siglas en inglés y más conocidos como Drones) que son capaces de controlar el vuelo por ordenador sin la intervención de un piloto.

Figura 6: El Global Hawk de las Fuerzas Aéreas de EEUU. Un Drone utilizado en misiones de vigilancia y seguridad.

2.2.3. Técnicas empleadas

Las principales técnicas que se emplean en la visión por computador son: el procesamiento de imágenes, la detección de características (feature detection), el reconocimiento de objetos y el análisis de movimiento.

A continuación se realiza una breve introducción a cada una de estas técnicas y se detallan sus principales ámbitos de aplicación.

1-Procesamiento de imágenes

Estas técnicas abarcan un espectro amplio de aplicación. Desde mejorar la calidad de las imágenes hasta comprimirlas para su almacenamiento.

Figura 7: Imagen resultante de aplicar un algoritmo de blending.

2-Feature Detection

Lo que persiguen estas técnicas es hacer una abstracción de la información de las imágenes, y decidir sobre cada punto de la imagen si es una característica de un determinado tipo o no. Algunos tipos de características que se suelen buscar pueden ser contornos o aristas. El resultado de este proceso de detección es un descriptor de la imagen.

Algunos ejemplos de feature detectors pueden ser Canny, Sobel o FAST.

Figura 8: Detección de características.

3-Reconocimiento de objetos

Uno de los problemas típicos en visión por computador es determinar si un objeto específico se encuentra en una imagen. Mediante estas técnicas se puede identificar un objeto en una imagen o en un video, usando un patrón del objeto que se conoce previamente.

Pese a que los humanos reconocemos cualquier objeto en una gran variedad de condiciones rápidamente y sin esfuerzo, todavia no hay algoritmos suficientemente efectivos para esta tarea, por lo que es un campo que todavía se encuentra en pleno desarrollo.

Algunas aplicaciones de estos métodos son por ejemplo la detección de caras o el control de calidad de productos.

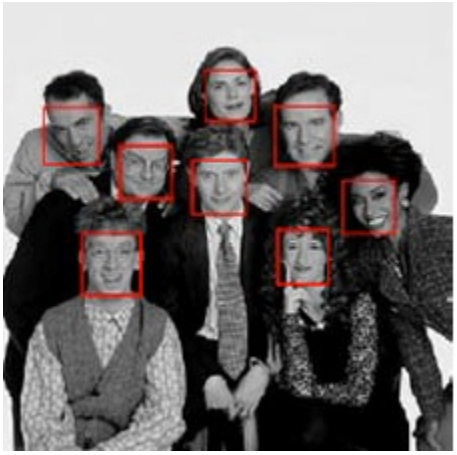


Figura 9: Reconocimiento facial.

4-Análisis de movimiento

Estas técnicas extraen información sobre el movimiento y la velocidad de los puntos de la imagen analizando frames consecutivos en secuencias de imágenes. En algunas aplicaciones como la videovigilancia, la cámara se encuentra en una posición fija y detecta el movimiento de los objetos en ella, aunque en algunos casos la cámara puede estar en movimiento.

Además de en vigilancia, estos métodos también se emplean a menudo en aplicaciones médicas o en los sistemas de navegación de vehículos autónomos.

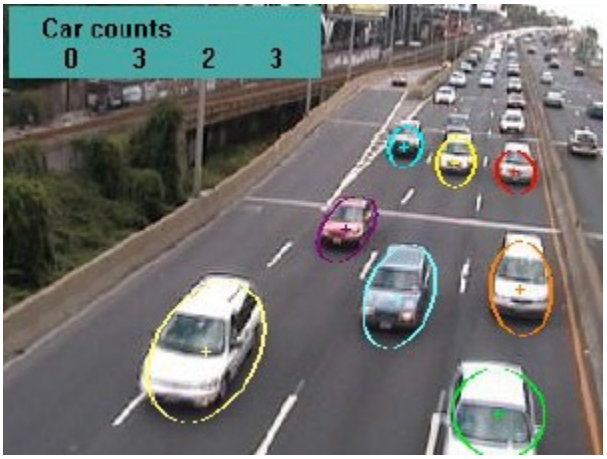


Figura 10: Monitorización del tráfico (http://www.honeywellvideo.com/).

2.2.4. Métodos de detección de objetos

En el contexto de este proyecto resultan especialmente importantes los métodos que permiten detectar objetos en una imagen. Se basan en distintas técnicas para localizar el objeto, y tienen diferentes ámbitos de aplicación según su naturaleza. Estos métodos se han introducido en el apartado anterior como Reconocimiento de objetos.

Se detallan a continuación algunos de los métodos más utilizados para el reconocimiento de objetos.

1-Background subtraction

Es un método ampliamente usado en sistemas de vigilancia, dado que permite detectar en tiempo real objetos en movimiento con cámaras estáticas. La complejidad es muy reducida ya que se basa en la diferencia entre el frame actual y un frame de referencia.

Figura 11: Detección de personas en cámara de video vigilancia. La imagen superior derecha es la imagen de referencia (background).

2-Feature detection

Los algoritmos de detección de características computan descriptores que describen información de la imagen. Pueden basarse en distintos tipos de características como vértices o puntos de interés. Algunos detectores conocidos son Canny, Sobel, FAST o SURF. Éste último [7] es muy usado para detectar en un video un objeto del cual se ha extraído previamente un descriptor, o para aplicaciones de reconstrucción 3D. SURF es invariante a la escala y a la rotación del objeto en la imagen, y más rápido que su predecesor SIFT, por lo que ha obtenido una gran popularidad.

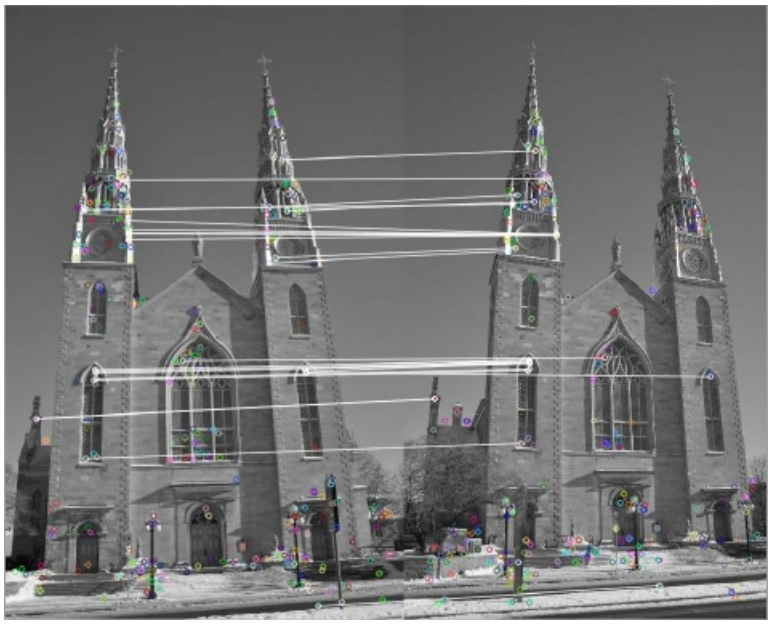


Figura 12: Detección de coincidencias entre dos imágenes mediante SURF.

3-Segmentación

La segmentación es un proceso de división de la imagen en varios conjuntos de píxeles, que es típicamente usado para detectar objetos o contornos. Se utiliza en aplicaciones de imagen médica, en detección de algunos tipos de objetos como por ejemplo caras o personas.

El método mas simple para conseguir la segmentación de una imagen es el thresholding, que se basa en aplicar un filtro a los valores que están por debajo o por encima de un determinado valor umbral, obteniendo al final del proceso una imagen binaria.

Figura 13: Segmentación por el color de piel en una escena con background complejo. Imagen extraída de http://w3.ualg.pt/~ftomaz/fr/fr.php. Forma parte de la tesis de PhD de Felipe Antonio Gonçalves (University of Algarve).

2.2.5. Métodos de tracking de objetos

Una vez detectado el objeto, necesitamos un método de tracking para hacer un seguimiento de la posición del objeto a través de la secuencia de frames del video. Esto se consigue estableciendo correspondencias entre dos frames consecutivos. Estos métodos se han introducido en el apartado anterior como Análisis de movimiento.

Algunos métodos que permiten el seguimiento de un objeto pueden ser el alpha-beta filter o el Kalman filter, aunque es éste último el que se va a detallar a continuación.

1-Lucas-Kanade

El método de Lucas-Kanade (1981) expuesto en [4], permite calcular una estimación del flujo óptico en una región de la imagen. Este método asume que el desplazamiento de un punto en dos frames consecutivos es pequeño y constante respecto a sus píxeles vecinos.

Es un método muy utilizado para realizar face tracking, aplicacion que es muy importante sobretodo en sistemas de vigilancia.

Figura 14: Tracking de vehículos mediante el algoritmo de Lucas-Kanade.

2-Kalman Filter

En [5] se presenta el Kalman filter (1960) como un método eficaz para el tracking de objetos. En este artículo se demuestra que mediante esta técnica se puede predecir cual será la posición del objeto en el siguiente frame, y que es capaz de hacer la predicción incluso habiendo ciertos momentos en que el objeto no se encuentra visible en la imagen.

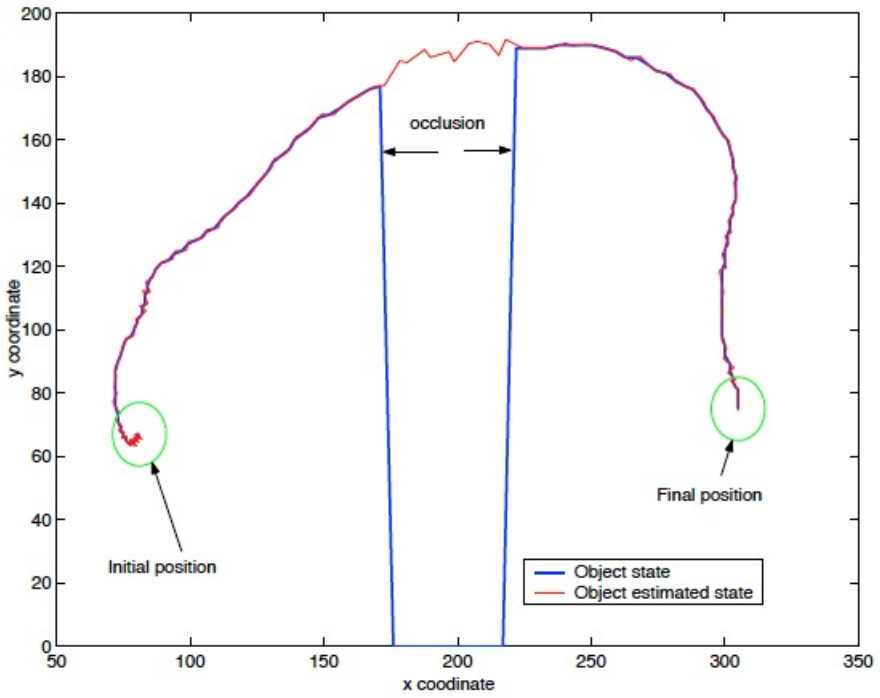


Figura 15: Predicción de la posición del objeto con un periodo de oclusión mediante Kalman filter.

Aunque estas dos técnicas (detección y tracking) pueden usarse por separado, suelen encontrarse trabajando simultáneamente en multitud de aplicaciones, ya que su combinación permite construir aplicaciones más robustas y eficientes.

Referencias

[1] Ronald C. Arkin. Behaviour-based robotics.

[2] Robin R. Murphy. Introduction to AI Robotics.

[3] Damien Grosgeorge, Caroline Petitjean, Jérome Caudron, Jeannette Fares, Jean-Nicolas Dacher. Automatic cardiac ventricle segmentation in MR images: a validation study.

[4] Simon Baker, Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. International Journal of Computer Vision.

[5] Erik Cuevas, Daniel Zaldivar, Raul Rojas (2005). Kalman filter for visión tracking.

[6] Sergi Bermejo Sánchez. Desarrollo de robots basados en el comportamiento.

[7] Herbert Bay, Tinne Tuytelaars, Luc Van Gool. SURF: Speeded Up Robust Features.

[8] Richard Szeliski. Computer Vision: Algorithms and applications

[9] http://en.wikipedia.org/wiki/Computer_vision

[10] http://en.wikipedia.org/wiki/Feature_detection_(computer_vision)

[11] http://en.wikipedia.org/wiki/Object_recognition

[12] http://en.wikipedia.org/wiki/Motion_analysis

[13] https://en.wikipedia.org/wiki/Medical_imaging

[14] http://en.wikipedia.org/wiki/Herbert_Bay

[15] http://en.wikipedia.org/wiki/Image_segmentation

[16] http://es.wikipedia.org/wiki/Curiosity

[17] http://www.freshnessmag.com/2012/08/06/nasa-curiosity-mars-rover-live-landing-event-video/

[18] http://www.northropgrumman.com/Capabilities/GlobalHawk/Pages/default.aspx

[19] http://www.mathworks.com/matlabcentral/fileexchange/24677-lucas-kanade-affine-template-tracking

[20] http://es.wikipedia.org/wiki/Robotica

3. Análisis del problema

3.1. Planteamiento del problema

Para implementar un sistema que cumpla los objetivos propuestos necesitamos un robot programable de tipo wheeled (robot dotado de ruedas), que proporcione una serie de sensores compatibles (mínimo un sensor de luz y uno de ultrasonidos). Será necesaria también una cámara para captar imágenes en tiempo real, y capacidad computacional suficiente para procesar dichas imágenes y proveer de datos al robot sobre la posición del objetivo en cada frame.

Respecto a la cámara, algunos robots tienen disponibles cámaras que pueden trabajar integradas en el propio robot como cualquier otro sensor. La limitación de esta solución se encuentra en la baja calidad de las imágenes que proporcionan estas cámaras de bajo coste, y sobretodo en la reducida capacidad de cómputo que tiene la CPU de este tipo de robots, que normalmente trabajan en un solo núcleo y a una velocidad entre 20-50 MHz, frente al procesador de un dispositivo móvil de gama baja que puede trabajar entre los 400 y 800 MHz, o uno de gama media-alta que pueden trabajar con 2-4 núcleos y a velocidades superiores a 1 GHz. Además, utilizando un dispositivo móvil se cuenta con una ventaja adicional: podemos utilizar librerías externas especializadas en visión por computador que implementan potentes algoritmos y estructuras de datos para el tratamiento de imágenes.

3.2. Solución propuesta

Se propone un conjunto formado por un robot, y un dispositivo móvil que se usará para adquirir y procesar las imágenes a través de su propia cámara. La dos partes que formarán el sistema (robot y móvil) se integrarán mediante una comunicación bluetooth para la sincronización y el intercambio de datos.

Una vez planteado el sistema, se han de elegir las tecnologías que usaremos para el desarrollo del proyecto. En este caso son tres: plataforma robótica, tipo de dispositivo móvil y librería para implementar los algoritmos de visión por computador.

Las tres plataformas que se han escogido para el desarrollo de este proyecto son: Lego Mindstorms, Android y OpenCV4Android. A continuación se introduce brevemente cada una de estas tecnologías, argu-

mentando los motivos por los que han sido elegidas para implementar la solución al problema planteado.

3.3. Metodología de trabajo

3.3.1. Entorno de programación

Como se ha introducido en el apartado anterior, se ha optado por trabajar en Android para la parte de la programación del dispositivo móvil. Por lo tanto, la opción más razonable es utilizar el IDE Eclipse, ya que facilita mucho el trabajo debido a su perfecta integración con el SDK de Android mediante el plugin ADT y su fácil configuración. Además, ya cuento con experiencia previa trabajando en este entorno y con estas herramientas para la programación en Android, y eso es un punto importante a tener en cuenta ya que no será necesario emplear tiempo extra en aprender el flujo de trabajo en un nuevo entorno.

Para la programación del robot en lenguaje NXC se puede utilizar cualquier editor de texto. Solo se necesita el compilador NBC para realizar la compilación y la descarga de los programas en el NXT. En Windows se puede utilizar el programa BricXcc, que ya integra la edición, la compilación y la descarga de los programas. En mi caso, al utilizar Mac, usaré el editor Sublime para programar, y realizaré la compilación y descarga mediante consola.

En el primer punto del anexo de este documento se puede encontrar una sencilla guía de instalación y configuración de este entorno para comenzar a programar en Android. También resulta indispensable contar con el libro de referencia [1] para aprender lo imprescindible sobre la programación en Android y comenzar a dar los primeros pasos.

3.3.2. Planificación

Un punto crítico a la hora de organizar un proyecto de cualquier tipo es la planificación. Una correcta distribución de las tareas en el tiempo es fundamental para alcanzar con garantías los objetivos propuestos.

Por lo tanto, se presenta en un documento a parte en formato pdf, un plan de trabajo, indicando la distribución de tareas a lo largo del tiempo disponible para llevar a cabo el proyecto.

3.4. Análisis de las tecnologías empleadas

3.4.1. Lego Mindstorms

El kit de Lego Mindstorms NXT ofrece una excelente plataforma robótica, dotada del hardware y software necesarios para realizar prototipos de robots de forma rápida, incluyendo un conjunto de actuadores y sensores sencillos de programar a través de su software de programación basado en bloques. El kit se compone de 4 tipos de elementos: el procesador (NXT), los sensores, los actuadores, y las piezas de Lego.

El software proporcionado por Lego está basado en LabView y permite un fácil acceso a la robótica a personas con pocos conocimientos de programación gracias a una interfaz de programación gráfica sencilla e intuitiva.

La concepción de la interfaz de sensores y motores del NXT permite implementar de forma relativamente sencilla robots de tipo reactivo (paradigma de captar, planear y actuar).

Hay varias versiones de Lego Mindstorms disponibles hasta la fecha. Se diferencian entre ellas por el precio y el número de piezas y sensores que incluyen. Para este proyecto se ha dispuesto de la versión 2.0. En la siguiente tabla se puede ver un resumen del contenido del kit.

Lego Mindstorms NXT 2.0	
	619 elementos de construcción 1 Ladrillo inteligente NXT 3 Motores 1 Sensor de ultrasonidos 2 Sensores de contacto 1 Sensor de color 7 Cables de conexión Software de programación de Lego

Figura 16: Contenido del kit Lego Mindstorms NXT.

Una vez introducido el kit, vamos a ver un resumen de sus componentes más importantes, en especial los que se necesitan para la realización de este proyecto. El primero y mas importante es el NXT, el cerebro del robot Mindstorm.

El Brick inteligente NXT

El NXT es el centro de procesamiento del robot. Es un ladrillo inteligente controlado por una CPU que permite al Mindstorms realizar diferentes operaciones. Los elementos más importantes de que dispone son: 3 puertos de salida para motores, 4 puertos de entrada para sensores, puerto USB para descargar los programas desde un PC, botones para navegar por el menú y un display. A continuación se detallan las principales especificaciones técnicas del NXT.

Microcontrolador ARM7 de 32 bits 256 kbytes FLASH, 64 kbytes de RAM Microcontrolador AVRde 8 bits 4 kbytes FLASH, 512 Byte de RAM Comunicación Bluetooth Clase 2 Puerto USB de alta velocidad (12 Mbit/s) 4 puertos de entrada 3 puertos de salida Display de 100 x 64 pixeles LCD Alimentación: 6 baterías AA	

Figura 17: El NXT, el cerebro del robot Mindstorms.

Servo Motores

Los servomotores dotan al robot de movimiento. Cada motor integra un encoder para poder leer la rotación del motor en grados, lo que permite controlarlos con mayor precisión.

Figura 18: Descripción de los servo motores del Lego Mindstorms.

Sensor de luz

Este sensor permite diferenciar entre luz y oscuridad y puede funcionar de dos modos: detectando luz ambiente o emitiendo luz y captando la cantidad de luz reflejada. Es capaz de leer la intensidad de luz en superficies pudiendo el robot utilizarlo para seguir una línea o clasificar objetos por color.

Figura 19: Descripción del sensor de luz del Lego Mindstorms.

Sensor de ultrasonidos

Gracias a este sensor el robot puede detectar objetos que se encuentren cerca de él. La lectura devuelve un valor entre 0 y 255 (por defecto cm) con una precisión de +/-3 cm. Se puede utilizar para que el robot evite obstáculos, mida distancias o detecte movimiento.

Figura 20: Descripción del sensor de ultrasonidos del Lego Mindstorms.

En cuanto a la programación del robot, está concebida para realizarse mediante el software original de Lego, que incluye a su vez el firmware del NXT. Como se ha comentado con anterioridad, su característica principal es el entorno gráfico que permite construir el programa mediante la unión secuencial de bloques, lo que permite aprender a programar el robot rápidamente a personas con conocimientos básicos de programación.

Para programadores avanzados, pueden resultar más interesantes otros lenguajes alternativos que se pueden utilizar para programar el robot. Para ello, es necesario modificar primero el firmware del NXT. Los frameworks más conocidos para programar el NXT en otros lenguajes son BrickOS, LejOS y NXC. En el marco de este proyecto se ha decidido utilizar el lenguaje NXC. A continuación se realiza una introducción a este framework y se exponen los motivos de su elección.

3.4.2. NXC

NXC (Not Exactly C) es un lenguaje simple de programación semejante a C con el que se pueden realizar programas para el robot Mindstorms. El compilador de NXC se encarga de traducir el código para que lo pueda ejecutar el NXT. Por lo tanto, lo único necesario para programar en este lenguaje es un editor de texto y el compilador.

Las reglas léxicas de este lenguaje son muy parecidas a C. Se pueden crear constantes en tiempo de compilación, se dispone de las estructuras de control condicional e iterativas de C, etc. Además, el lenguaje NXC incluye una serie de funciones definidas para el control de los motores y sensores del robot. También se pueden crear hasta 10 tareas (threads) que se ejecuten en paralelo (pese a que el procesador tiene un solo núcleo), y dispone de elementos para la sincronización del acceso a variables compartidas como semáforos y mutex.

El motivo principal de la elección de este lenguaje para realizar el proyecto es que nos permite programar el robot trabajando a bajo nivel, y a la vez proporciona una serie de funciones integradas en su API, que facilitan el control de los elementos del robot (como los motores, sensores o display). Por lo tanto, este lenguaje proporciona comodidad y a su vez flexibilidad para poder controlar con más precisión ciertos aspectos de la programación.

3.4.3. Dispositivo Android

Android es un sistema operativo de código libre basado en Linux, adquirido por la compañía Google en 2005 y diseñado principalmente para dispositivos móviles táctiles. Es también utilizado minoritariamente en otros tipos de dispositivos como tablets o reproductores mp3.

Android permite que las aplicaciones se desarrollen en un framework de Java, lo que facilita el acceso a la creación de aplicaciones para los dispositivos de esta plataforma, dado que solo es necesario disponer del Software Development Kit (SDK) y conocimientos de programación Java.

Android comenzó a extenderse a partir de 2008, y actualmente tiene una presencia de aproximadamente el 85 % del total de dispositivos móviles que se venden en España, habiendo experimentado un crecimiento muy rápido y siendo el sistema que predomina en Europa y EEUU.

A su vez, los smartphones han experimentado una gran evolución en su capacidades, aumentando de forma drástica su potencia computacional y su memoria, incorporando un mayor número de sensores, y aumentando la sensibilidad y el tamaño de la pantalla táctil.

Por los motivos expuestos, y adicionalmente, el hecho de que ya disponía de experiencia previa en programación Android, se ha elegido trabajar sobre un dispositivo de esta plataforma. Como entorno de desarrollo se utilizará el IDE Eclipse junto con el plugin Android Development Tools (ADT) y el Software Development Kit (SDK) de Android, ya que la configuración es muy sencilla y permite trabajar de una forma cómoda y flexible.

En concreto para este proyecto se ha utilizado un dispositivo HTC One S. A continuación se muestra una tabla con un resumen de sus características técnicas más importantes.

HTC One S

TAMAÑO	130.9 x 65 x 7.8 mm
PESO	119.5 gramos
PANTALLA	4.3” AMOLED
RESOLUCIÓN	960 x 540 (HD)
SISTEMA OPERATIVO	Android 4.1 Jelly Bean
PROCESADOR	1.5 GHz de doble núcleo
MEMORIA	Almacenamiento: 16 GB RAM: 1 GB
SENSORES	Sensor giroscópico G-Sensor Brújula digital Sensor de proximidad Sensor de luz ambiental
CONECTIVIDAD	Bluetooth 4.0 Wi-Fi: IEEE 802,11b/g/n
CÁMARA	Cámara de 8 megapíxeles con enfoque automático, flash LED inteligente. Cámara frontal: VGA. Chip especial para el procesamiento de imagen.
BATERIA	1650 mAh

Figura 21: Resumen de las características principales del HTC One S.

3.4.4. OpenCV

OpenCV es una librería de visión por computador gratuita y de código abierto, desarrollada originalmente por Intel. La librería está escrita en los lenguajes C y C++ y funciona en Windows, Linux y Mac OS X.

Su principal objetivo es proporcionar una interfaz de simple uso para la construcción de forma rápida aplicaciones complejas de visión por computador; está especialmente diseñada para ser computacionalmente eficiente, con especial énfasis en aplicaciones a tiempo real, aprovechando las ventajas de la programación paralela.

La librería contiene cerca de 500 funciones que abarcan diversas áreas de la visión por computador. OpenCV también integra una completa librería de machine learning (MLL) centrada en análisis estadís-

tico para pattern recognition y clustering, que resulta muy útil para algunas tareas de computer vision.
Desde que fue lanzado en 1999, OpenCV ha sido utilizado en multitud de aplicaciones, productos e investigaciones relacionados con la visión artificial. Su publicación bajo licencia BSD permite que se puedan construir productos comerciales usando la librería sin obligación de que el producto sea open source o de aportar mejoras a la comunidad. En Julio de 2011, OpenCV comenzó a ofrecer soporte para Android con la liberación de la versión 2.3. Para este proyecto usaremos la versión 2.5 de OpenCV4Android.
Utilizar esta librería nos permitirá trabajar utilizando estructuras de datos, funciones y algoritmos implementados específicamente para el procesado de imágenes, lo que facilitará el manejo de las imágenes y su procesamiento en la aplicación. La contraparte es que se debe sacrificar una parte del tiempo en aprender a utilizar la librería antes de empezar a implementar una aplicación compleja. Aún así, está más que justificada la utilización de OpenCV para llevar a cabo la parte de visión por computador de este proyecto.

3.5. Limitaciones del sistema

En el contexto en el que se enmarca este proyecto hay multitud de sistemas muy complejos que requieren un gran presupuesto y un gran equipo humano especializado para llevarlos a cabo. Un ejemplo pueden ser los coches autónomos desarrollados por varias universidades para el DARPA Urban Challenge, que incluyen multitud de unidades de medición sofisticadas. Sin embargo, las tecnologías de que disponemos para llevar a cabo este proyecto son radicalmente diferentes, y por lo tanto debemos centrarnos en conseguir un objetivo razonable con lo que disponemos.

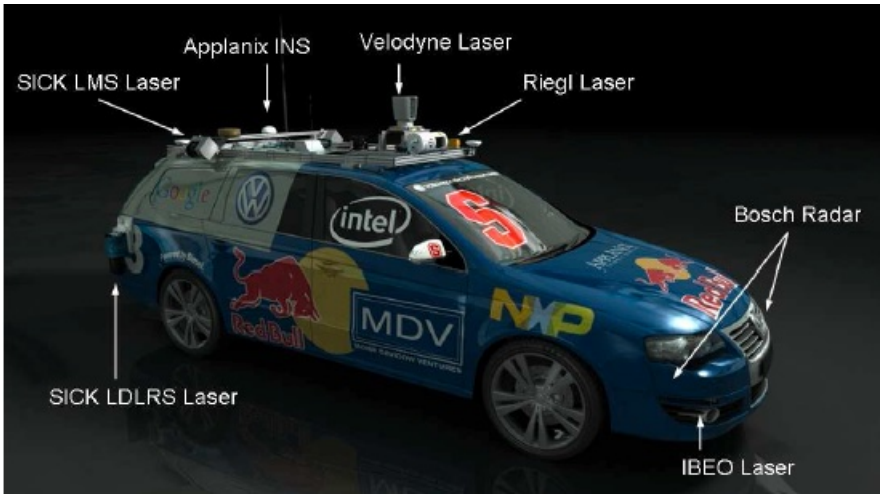


Figura 22: Junior, el coche desarrollado por la Stanford University para el DARPA Urban Challenge [7]. Cuenta con 5 sistemas de medida laser, radar y sistema de navegación.

En el caso de Lego Mindstorms, el kit es un juego para niños y, a pesar de ofrecer un hardware y software muy versátil, no es una plataforma robótica sofisticada y de alta calidad. Esto supone que las mediciones de los sensores y los movimientos de los motores serán menos precisos, lo que implica un mayor margen de error por parte del robot.

Un aspecto importante que se ha de tener en cuenta al evaluar el rendimiento del sistema será la latencia. El hecho de no tener el sistema de visión integrado en el propio robot supone una latencia extra, debido a que los datos entre el móvil y el robot deben compartirse mediante una comunicación Bluetooth. Tratándose de un sistema que ha de responder en tiempo real, este es un hándicap que hemos de considerar y tratar de forma delicada.

Otro problema de latencia se deriva del uso de un lenguaje de programación alternativo para NXT, debido a que las instrucciones se deben emular antes de que el NXT las ejecute. Esto genera una latencia media demostrada de 1.75 milisegundos. A pesar de ser aparentemente un tiempo despreciable, si lo unimos al retraso de la información transmitida por bluetooth esto puede afectar considerablemente a la capacidad de reacción del robot, y puede suponer una mayor acumulación de errores.

Por último, otro punto crítico que afecta al rendimiento del sistema es la potencia computacional. Los sistemas robóticos de gama alta suelen estar comandados por potentes PCs. En el presente, la capacidad

computacional de un dispositivo móvil no está a la altura de un PC. Ésta circunstancia obliga a tener presente el hecho de que se está programando para un dispositivo móvil, y que se ha de tener en cuenta la eficiencia del código y la exigencia de cómputo que requiere cada método a la hora de elegir el algoritmo que se usará para la detección y el tracking del objetivo.

Por los motivos expuestos anteriormente, es importante tener claras las limitaciones de nuestro sistema, y no plantear objetivos que queden fuera de alcance para poder llevar a cabo con éxito el proyecto.

Referencias

[1] Reto Meier. Professional Android 4 Application Development.

[2] http://es.wikipedia.org/wiki/Android

[3] http://www.htc.com/es/smartphones/htc-one-s/

[4] http://es.wikipedia.org/wiki/Lego_Mindstorms

[5] http://mindstorms.lego.com

[6] http://es.wikipedia.org/wiki/NQC

[7] Anna Petovskaya, Sebastian Thrun. Model Based Vehicle Tracking for Autonomous Driving in Urban Environments. Stanford University.

[8] Gary Bradski, Adrian Kaehler. Learning OpenCV. O'Reilly.

4. Diseño e implementación

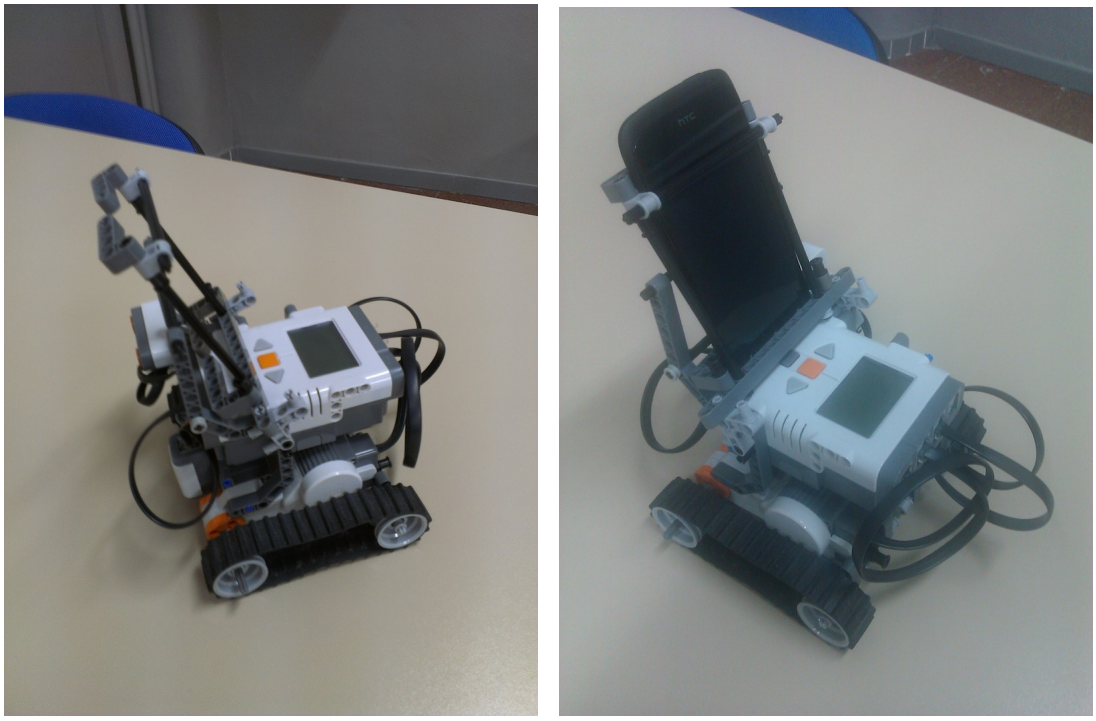
4.1. Diseño del robot Mindstorms

Para el desarrollo de este proyecto hemos optado por un diseño de tipo oruga, ya que ofrece más estabilidad y el centro de gravedad es más bajo. Esto ayudará a que el peso extra que debe cargar el robot debido al dispositivo móvil se reparta mejor.

Se ha partido de el siguiente ejemplo como punto de partida para construir el robot.

Figura 23: Ejemplo de diseño de un robot oruga.

Se ha optado por disponer horizontalmente el brick para alojar encima el móvil. Para ello, se ha construido un soporte en la parte superior del robot, lo que permitirá adquirir imágenes frontales.



(a) Robot sin el movil. Se puede ver el soporte que se ha incorporado en la parte superior.

(b) Robot con el móvil incorporado.

Figura 24: Diseño final del robot con el dispositivo móvil incorporado.

Además, se ha montado otro robot que se usará para realizar las pruebas de tracking en el escenario. Este robot es un poco diferente. Está basado en un modelo con dos ruedas, más una rueda omnidireccional que le permite realizar giros. A este robot no se le han añadido sensores ya que, su función se limitará a moverse aleatoriamente por el escenario sin necesidad de evitar las líneas blancas.

Se le ha montado una plataforma regulable en altura en la parte superior para poder colocar la pelota que el otro robot debe detectar. Se muestra a continuación una imagen del robot final:

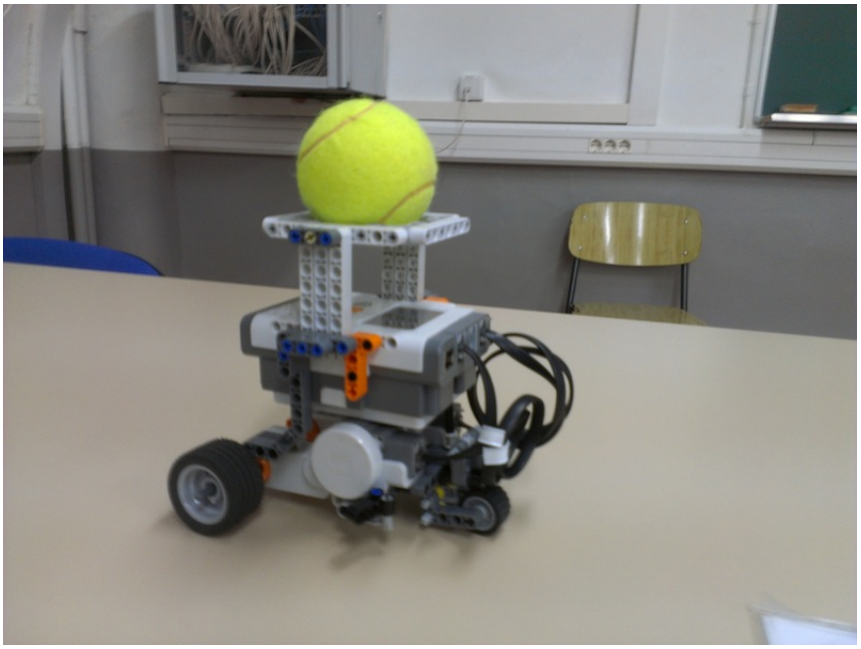


Figura 25: Robot que hará la función de objetivo a seguir.

4.2. Diseño de la aplicación Android

4.2.1. Diagrama de clases

En el siguiente diagrama de clases se muestra el diseño de la aplicación Android y la relación entre las distintas clases que la componen.

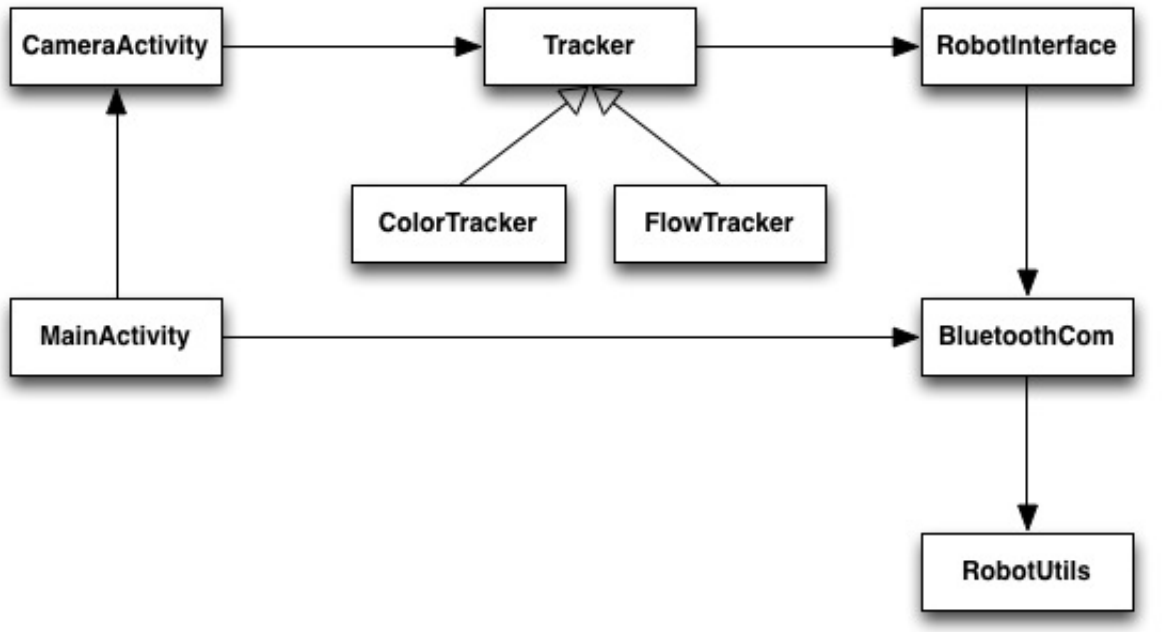


Figura 26: Diagrama de clases de la aplicación Android que procesa las imágenes captadas por la cámara y envía los datos por Bluetooth al NXT.

4.2.2. Organización del código

Se ha separado el código en tres paquetes claramente diferenciados. A continuación se detalla que clases componen cada paquete y que función realizan.

Activities

En Android las Activities son las clases que gestionan la vista de la aplicación. En este paquete hay dos activities:

- **MainActivity**: es la activity principal que se ejecuta al iniciar la aplicación. Permite iniciar la conexión Bluetooth, elegir el método de que se utilizará para realizar el tracking, e iniciar la aplicación pasando a CameraActivity.
- **CameraActivity**: Se encarga de recibir los frames de la cámara del dispositivo, procesarlos y retornarlos para que se muestren en pantalla.

Comunications

Contiene las clases necesarias para implementar la comunicación mediante Bluetooth.

- **BluetoothCom**: Esta clase proporciona los métodos necesarios para habilitar la comunicación, iniciar la conexión, y permitir el envío y la recepción de datos.
- **RobotInterface**: Esta clase hace la función de interfaz para la comunicación bluetooth con el robot. Contiene las tramas de bytes que se han de enviar al robot, y se encarga de codificar los datos en la trama.
- **RobotUtils**: Contiene constantes que son de utilidad para las dos clases anteriores.

Vision

- **Tracker**: Esta clase implementa las funciones básicas para realizar el tracking de un objeto. De esta clase extienden las demás clases para implementar un tracker especializado mediante un determinado método.
- **ColorTracker**: Esta clase implementa el tracking únicamente mediante la detección por color.
- **FlowTracker**: Esta clase implementa la detección mediante el color y el tracking mediante optical flow (Lucas-Kanade).

4.3. Procesado de imágenes en Android

Como se ha comentado anteriormente, la captación y el procesado de imágenes se realizará mediante la librería OpenCV4Android. Esta librería proporciona una interfaz para las llamadas a OpenCV desde el código Android, lo que permite trabajar desde el mismo código Java.
También es posible trabajar en un proyecto Android con OpenCV en código nativo C. Sin embargo, esta alternativa se ha probado a lo largo de este proyecto, pero además de resultar bastante más complicado configurar el proyecto Android, no se ha apreciado una mejoría en la eficiencia a la hora de procesar imágenes. Por lo tanto se ha optado por trabajar con OpenCV directamente desde el código Android.

4.3.1. Cargar la librería de OpenCV en la aplicación

Para poder utilizar OpenCV en una aplicación, además de incluir la librería en el proyecto, se ha de cargar en la aplicación. Esto se puede hacer en la activity principal de forma sencilla.

El proceso de cargar la librería es asíncrono, por lo tanto, es de utilidad tener una función callback que se ejecute cuando el proceso de carga de la librería se haya completado.

Primero, se declara la función Callback:

```
private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
    public void onManagerConnected(int status) {
        switch (status) {
        case LoaderCallbackInterface.SUCCESS:
            Log.i( TAG, "OpenCV loaded successfully" );
            initData();
            break;
        default:
            super.onManagerConnected(status);
            break;
        }
    }
};
```

Después, en el método onResume, se realiza la siguiente llamada:

```
OpenCVLoader.initAsync( OpenCVLoader.OPENCV_VERSION_2_4_3, this, mLoaderCallback );
```

4.3.2. Obtener un flujo de imágenes

Para obtener un flujo de imágenes de la cámara, la Activity que se encargará de esta tarea debe implementar la interficie CvCameraViewListener. La definición de la clase queda de la siguiente forma:

public class CameraActivity extends Activity implements CvCameraViewListener

Esta interficie obliga a implementar los siguientes métodos:

void onCameraViewStarted()

void onCameraViewStopped()

Mat onCameraFrame(Mat inputFrame)

Es la última función, onCameraFrame la que recibe la matriz correspondiente a cada frame de la cámara, y debe retornar una matriz para ser visualizada en la pantalla. En este método se puede procesar el frame y retornar la imagen resultante para que sea visualizada.

Se muestra a continuación la implementación de este método que se ha llevado a cabo en este proyecto:

```
public Mat onCameraFrame( Mat inputFrame ) {
    m = tracker.processFrame( inputFrame );
    return m;
}
```

4.3.3. Algoritmo de detección del objetivo

En un primer momento, se consideró la opción de segmentar la región del robot mediante la técnica de background subtraction, y obtener un descriptor de la región mediante el SURF detector.
Este método suponía dos problemas:

- Si el objetivo no se mueve, es imposible detectarlo mediante background subtraction, y esto supone una limitación importante.
- Sacando un descriptor desde un ángulo del objetivo, no se podrá detectar posteriormente desde todos los ángulos cuando el robot y el objetivo estén en movimiento.

Un problema adicional, es el hecho de que SURF, al estar patentado, ha sido retirado de OpenCV. A pesar de ello, se estudiaron alternativas, como por ejemplo el detector ORB presentado en [11] como una alternativa eficiente a SURF. El problema es que éste no es invariante a la escala, lo que quiere decir que no se puede detectar al objetivo desde diferentes distancias, una limitación que hace imposible su uso para este proyecto.
El método de detección que se ha implementado finalmente se basa en filtrar un determinado rango de color en la imagen para segmentar la región del objeto que se quiere detectar. Es un proceso sencillo de realizar y efectivo para el tipo de problema que se plantea en este proyecto.
A continuación, se detallan los pasos del algoritmo implementado.

1-HSV space

El espacio HSV define un modelo tridimensional de color en términos de sus componentes, y consiste en una transformación no lineal del espacio de color RGB. En lugar de un cubo como el RGB, el espacio HSV se define mediante un cono (Figura 27).

El parámetro color (Hue) se mide en grados (de 0 a 360). En cambio la saturación (Saturation) y la intensidad (Value) son números reales entre 0 y 1.

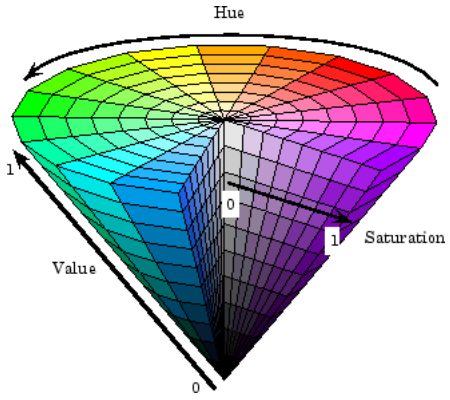
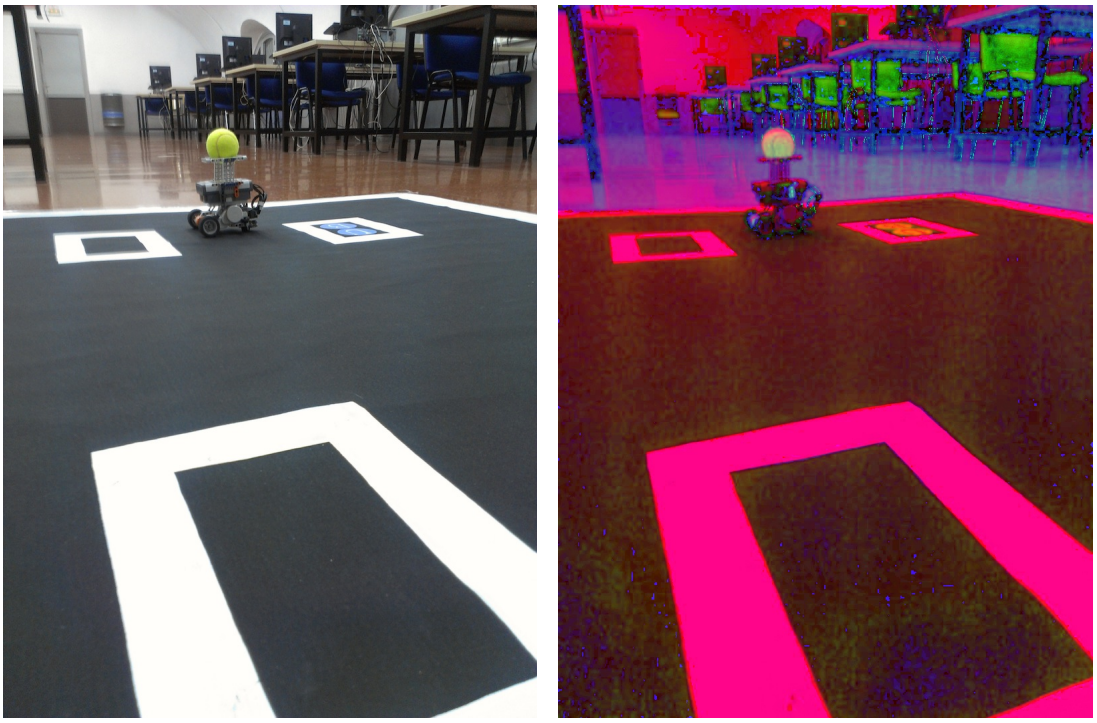


Figura 27: Cono que define los tres parámetros del modelo HSV: Hue, Saturation y Value.

Mediante la función de OpenCV cvtColor perteneciente al paquete Imgproc, se puede transformar una imagen a otro espacio de color. En este caso, para pasar de rgb a hsv se realiza la llamada con los siguientes parámetros, donde rgba es la imagen que se quiere convertir, aux la imagen destino, y el tercer parámetro indica que conversión se quiere realizar.

Imgproc.cvtColor(rgba, aux, Imgproc.COLOR_RGB2HSV_FULL);



(a) Imagen original en RGB. (b) Imagen en el espacio HSV.

Figura 28: Imagen RGB (izquierda) y su correspondiente imagen en el espacio HSV (derecha).

El último paso de esta primera fase, es un hacer un resize de la imagen para reducir su tamaño. De esta forma, a la hora de computar los siguientes pasos, el proceso será más rápido al recorrer una imagen reducida.

2-Segmentación de la región del objeto

El uso más común de la segmentación en la robótica es para identificar una región con un color particular. Este proceso se consigue mediante la aplicación de un threshold a todos los píxeles de la imagen.

La función de OpenCV inRange permite filtrar la imagen mediante dos valores umbral, uno inferior y otro superior. El resultado será una imagen en la que se han segmentado los colores que se encuentran entre los dos umbrales.

Primero se deben definir los umbrales inferior y superior para filtrar el color.

private final Scalar UPPER_THRESHOLD = new Scalar(65, 360, 360);

private final Scalar LOWER_THRESHOLD = new Scalar(55, 120, 160);

Se muestra a continuación la llamada la método inRange. El cuarto parámetro es también hsv porque se quiere que el resultado del filtrado se guarde en la misma imagen.

Core.inRange(hsv, LOWER_THRESHOLD, UPPER_THRESHOLD, hsv);

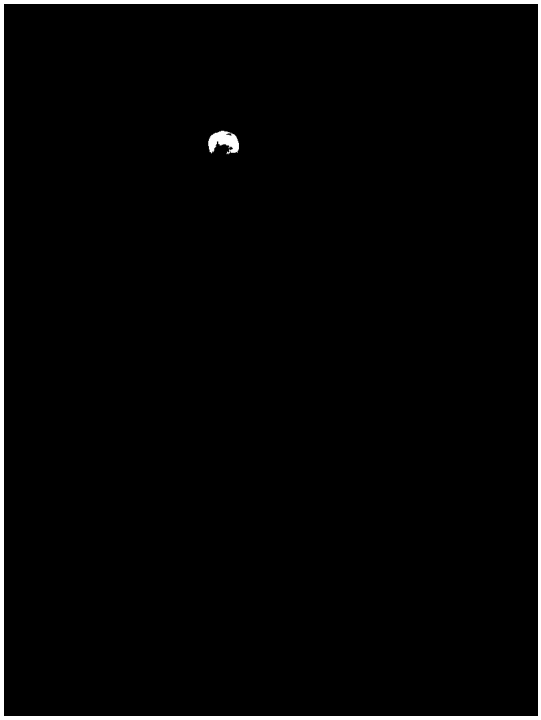


Figura 29: Imagen resultante del proceso de filtrado mediante threshold inferior y superior. Se puede ver la detección de la pelota.

Para acabar el proceso, se realiza una binarización de la imagen resultante mediante el método bitwise_or. El resultado se guardará en la matriz binary.

Core.bitwise_or(hsv, hsv, binary);

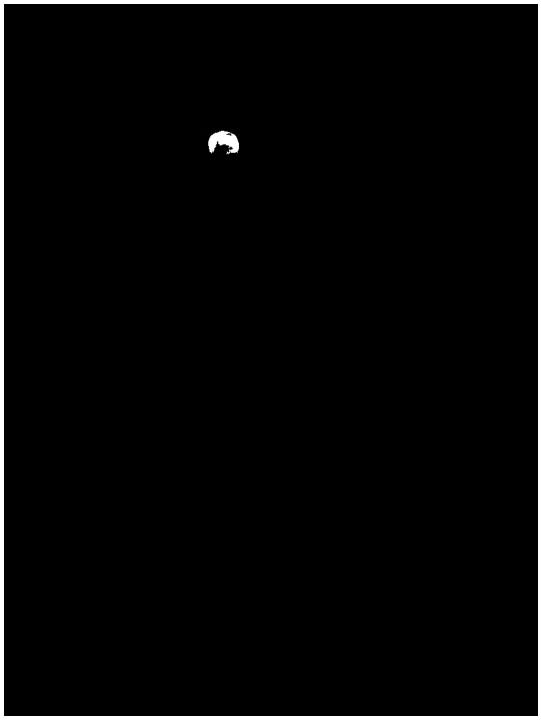


Figura 30: Imagen resultante del proceso de binarización. En blanco se puede ver la región del objeto.

Como se puede ver, no hay diferencia a nivel visual entre la imagen anterior y esta última imagen. La diferencia radica en que ésta última está compuesta por un único canal en vez de tres (hsv). Esto hará que el recorrido de la imagen para computar la posición del objeto sea un proceso mas rápido.

3-Cómputo de la posición del objeto

Una vez el objeto ha sido segmentado en la imagen, hay que computar su posición. Dado que contamos con una imagen binaria, en la cual la zona del objeto es blanca y el resto de la imagen es negra, solo hay que calcular la media en x e y de los píxeles que tienen un valor distinto de 0.

A continuación se muestra el pseudocódigo correspondiente al cómputo de la posición del objeto.

```
para toda fila en la imagen:
    para toda columna en la imagen:
        si el valor del pixel > threshold:
            x += j;
```

```
                y += i;
                countX++;
                countY++;
si countX != 0 y countY != 0:
    currentPosition.x = x / countX
    currentPosition.y = y / countY
    lost = false
sino:
    lost = true;
```

Figura 31: Imagen resultante del proceso de detección. El punto azul indica el punto en que se ha detectado la posición del objetivo.

Como se puede observar en el código anterior, en la misma rutina se puede activar un flag que indique

que el objeto no se ha encontrado en la imagen. Este caso se da cuando los contadores de x e y son 0. Ésto nos sirve para indicar al robot que no se ha detectado el objeto en la imagen, lo que provocará que cambie de estado y ejecute la rutina de búsqueda.

4-Codificación de la posición del objeto

Una vez se ha detectado la posición exacta del objetivo en la imagen, se ha de realizar una codificación para transmitir al robot una información que pueda interpretar, ya que el robot no entiende que ha de hacer si recibe las coordenadas de un punto en la imagen. Lo que el robot requiere es una indicación de si el objetivo se ha detectado en la imagen o no (para saber si debe pasar al estado de búsqueda), y un factor que se pueda aplicar directamente a la poténcia de los motores para desplazarse en dirección al objetivo.

Por lo tanto, la codificación que se ha llevado a cabo es la que se muestra en la siguiente figura:

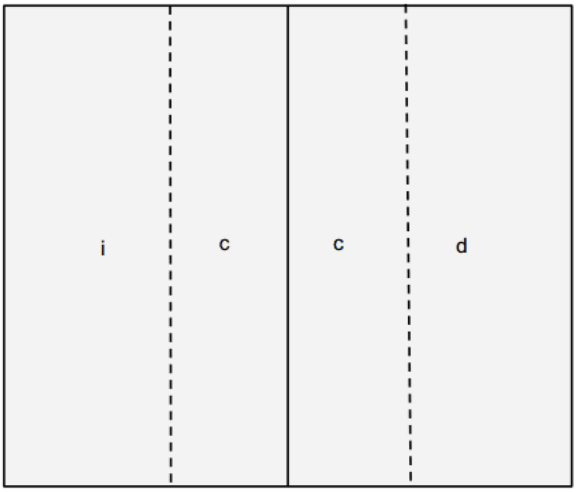


Figura 32: Codificación de la posición del objeto en la imagen.

Como se puede ver, la imagen se separa en tres regiones:

- Central: Esta región corresponde al centro de la imagen +/- una determinada tolerancia. Si el objeto se encuentra en esta región se codifica una 'c'.

- Laterales: Si el objetivo no está en el centro de la imagen, se indica en el byte de dirección una ’d’ (si está en la derecha) o una ’i’ (si está en la izquierda).

En cambio, si no se ha localizado el objetivo, en el byte de dirección de la trama se pondrá una ’l’ (lost) indicando que el objetivo no se ha detectado.

Además de la posición en la imagen, se debe indicar al robot un factor numérico que le indique cuanto ha de girar en su trayectoria para dirigirse hacia el objetivo. Este factor se ha codificado de la siguiente manera:

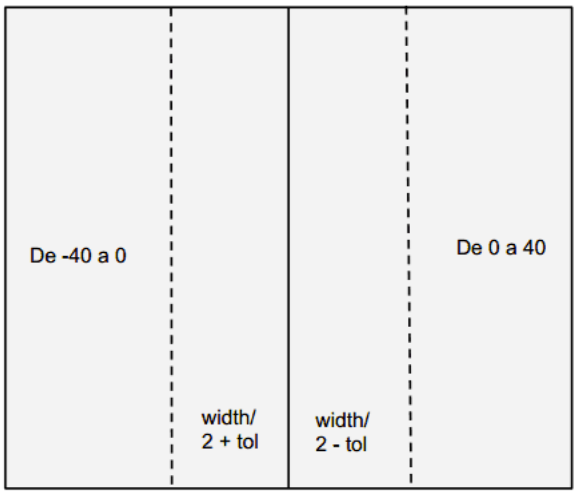


Figura 33: Codificación del factor que influye en la potencia de los motores para que el robot tome la trayectoria adecuada para dirigirse al objetivo.

Si el objeto se encuentra en la región central el factor es 0. Del centro a la izquierda disminuye proporcionalmente desde 0 a -40. Del centro a la derecha aumenta de 0 a 40.

La codificación de la información se ha realizado de esta forma porque el robot implementa un controlador proporcional para controlar la velocidad de los motores en función de la posición del objetivo en la imagen. Este punto se explica en detalle en el apartado 4.5.3. Programación del robot.

4.3.4. Algoritmo de tracking del objetivo

Mediante el algoritmo de detección del objetivo ya se podría realizar un seguimiento del mismo, detectando su posición frame a frame. Sin embargo, una vez detectado, resulta más eficiente utilizar un algoritmo de tracking para realizar el seguimiento.

El método de tracking que se ha decidido utilizar es el de Lucas-Kanade [39]. Se ha escogido este método principalmente por dos razones:

- Es un algoritmo que ejecuta rápidamente.
- OpenCV lo implementa a través de la función calcOpticalFlowPyrLK, y su uso resulta muy simple.

A continuación se expone una comparación del rendimiento de la aplicación cuando se utiliza únicamente la detección para seguir el objeto y cuando se combina con la utilización del algoritmo de tracking.

	Media de frames por segundo
Detección	7
Detección+Tracking	15

Figura 34: Comparación de la frecuencia de frames en ejecución entre el método de detección y el método de detección combinado con el de tracking.

Como se puede ver, al usar el algoritmo de tracking una vez detectado el objeto, aumenta en más del doble la velocidad de procesamiento. Obviamente, el algoritmo de tracking permite que el robot reaccione más rápido ante el movimiento del objetivo.

El problema que plantea el uso del tracking, es que la detección de la posición no es tan precisa como en la detección por color. El punto a menudo puede confundirse con otro punto de la escena, lo que provoca que el robot se dirija hacia objetivos erróneos.

En la implementación de este proyecto, esta circunstancia se ha solventado alternando periodicamente el estado del tracker. Cada 100 frames, se cambia del estado de tracking al estado de detección. De esta

forma, se refresca cada cierto tiempo la detección del objetivo evitando que el robot siga indefinidamente un posible punto erróneo.

Se muestra a continuación la función que se llama en el tracker para procesar el frame. Se puede observar como cuando el tracker está en el estado de tracking, se incrementa el contador, y al llegar a 100 se reseta y se pasa la estado de detección.

```
public Mat processFrame( Mat rgba ) {
    switch( state ) {
        case DETECTION:
            colorDetection( rgba );
            computePosition();
        case TRACKING:
            count++;
            flowTracking( rgba );
    }
    if( !lost )
        Core.circle( rgba, currentPosition, 10, new Scalar( 255, 0, 0 ), -1, 8, 0 );
    computeDirectionAndFactor();
    RI.sendProportionalMovement( direction, prop_factor );
    updatePrevPosition();
    if( count > 100 ) {
        count = 0;
        state = DETECTION;
    }
    return rgba;
}
```

Se muestra a continuación un pseudocódigo de la función que calcula la posición actual a través del método de Lucas-Kanade.

```
Video.calcOpticalFlowPyrLK( prev, gray, points[0], points[1], status, error );
si status[0] > 1: // se ha detectado el punto
    actualizar la posición del objeto
    points[0] = points[1] // actualizamos los puntos previos
```

si no:

estado = DETECTION

4.4. Comunicación Bluetooth

4.4.1. Qué es el Bluetooth

Bluetooth es un protocolo de comunicaciones peer-to-peer que opera en la frecuencia de radio de 2.4 GHz con un ancho de banda máximo de 720 kbit/s. Está especialmente diseñado para dispositivos de bajo consumo que requieren un rango corto de emisión (distancia máxima de 10 metros), y los principales objetivos que pretende conseguir son:

- Facilitar la comunicación entre equipos móviles y fijos.
- Eliminar los cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Los dispositivos que incorporan este protocolo pueden comunicarse entre ellos por radiofrecuencia cuando se encuentran dentro de su alcance. Bluetooth es ampliamente utilizado por dispositivos pertenecientes al sector de la informática y las telecomunicaciones, tales como teléfonos móviles, portátiles, impresoras, etc., ya que simplifica la búsqueda y configuración de los dispositivos debido a que éstos pueden indicar a otros los servicios que ofrecen. Estos dispositivos se clasifican como Clase 1, Clase 2 o Clase 3 en función a su potencia de transmisión, y son compatibles los de una clase con los de otra.

4.4.2. Proceso de pairing entre el móvil y el NXT

Cada dispositivo bluetooth se identifica con una dirección única de 12 dígitos hexadecimales, y se acompaña de un “friendly name” para poder referenciar los dispositivos fácilmente.

Para que dos robots NXT (o un NXT con un PC o móvil) puedan intercambiar mensajes mediante bluetooth, los dos dispositivos deben estar previamente sincronizados. En ese proceso de sincronización (llamado pairing), los dos dispositivos intercambian mensajes para compartir sus direcciones bluetooth,

sus friendly names y los perfiles que soportan. Los dispositivos confirman que soportan los mismos perfiles y que el passkey que han intercambiado coincide. Una vez que los dispositivos se han sincronizado, pueden realizar sucesivas conexiones usando sus nombres ya que recuerdan los passwords.
Pasos a seguir para realizar el pairing de un NXT con otro dispositivo:

1. Habilitar el bluetooth en el dispositivo y hacerlo visible
2. Encender el NXT
3. Activar el bluetooth en el menú del NXT
4. Iniciar la búsqueda de dispositivos en el NXT
5. Iniciar la conexión con el dispositivo
6. Aceptar el envío del passkey
7. Introducir el passkey en el dispositivo y aceptar

Una vez realizados correctamente los pasos anteriores, el NXT y el dispositivo están sincronizados y ya pueden iniciar conexiones para intercambiar datos.

4.4.3. Comunicación en Android

La API de comunicación Bluetooth de Android está basada en el protocolo RFCOMM. Mediante dicha API es posible buscar dispositivos y conectarse a ellos siempre que se encuentren dentro de rango. Mediante el Bluetooth Socket se obtiene un enlace de comunicación que permite a las aplicaciones recibir y transmitir flujos de datos.
Nótese que solo se admite comunicación cifrada entre los dispositivos, y por lo tanto deben estar previamente sincronizados tal como se ha expuesto en el apartado anterior.

1-Local Bluetooth Device Adapter

La clase BluetoothAdapter nos permite controlar el dispositivo bluetooth local del teléfono. Para tener acceso al Bluetooth Adapter hemos de llamar al método getDefaultAdapter como se muestra a continuación.

BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();

Mediante este controlador podemos activar o desactivar el bluetooth, buscar dispositivos, o leer y modificar propiedades del dispositivo local. Para tener acceso a los privilegios de lectura, tenemos que incluir el permiso BLUETOOTH en el manifiesto de nuestra aplicación. Si además queremos modificar propiedades del dispositivo, necesitamos incluir también el permiso BLUETOOTH_ADMIN.

<uses-permission android:name="android.permission.BLUETOOTH"/>

<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>

Para comprobar si el dispositivo está habilitado podemos usar el método isEnabled. Una vez activado podemos acceder a propiedades de nuestro dispositivo, como el nombre o la dirección.

```
if (bluetooth.isEnabled()) {
    String address = bluetoothAdapter.getAddress();
    String name = bluetoothAdapter.getName();
}
```

Para activar el Bluetooth Adapter se puede iniciar una Preference Activity de sistema de la siguiente manera:

startActivityForResult(new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE), 0);

De esta forma se le mostrará al usuario un diálogo para que confirme si desea activar el bluetooth. También podemos activarlo directamente siempre y cuando hayamos incluido el permiso de administrador sobre el bluetooth en el manifiesto.

```
if (!bluetooth.isEnabled()) {
    bluetooth.enable();
}
```

Para obtener información mas detallada del estado actual del adaptador, se puede usar el método getState, el cual retorna una de las siguientes constantes:

STATE_TURNING_ON

STATE_ON

STATE_TURNING_OFF

STATE_OFF

2-Obteniendo el dispositivo remoto

Para establecer la conexión con un dispositivo se deben cumplir las siguientes condiciones:

- El dispositivo debe estar visible.
- El dispositivo remoto debe aceptar conexiones a través de Bluetooth Server Socket.
- Los dispositivos local y remoto deben estar sincronizados.

Si se cumplen esta serie de requisitos, se puede empezar a buscar un dispositivo. El objeto Bluetooth Device se utiliza para representar dispositivos remotos, a los cuales se pueden realizar peticiones para obtener sus propiedades e iniciar una conexión.

Antes de realizar la conexión al dispositivo, es recomendable comprobar que éste se encuentra visible y que los dos están sincronizados. Se puede realizar una búsqueda de los dispositivos visibles o, en el caso de que conozcamos la dirección del dispositivo con el que vamos a realizar la conexión, es posible especificarla en el método getRemoteDevice.

BluetoothDevice device = bluetooth.getRemoteDevice("01:23:77:35:2F:AA");

Si se quiere obtener una lista con los dispositivos que están actualmente sincronizados, se puede obtener a través del BluetoothAdapter.

Set<BluetoothDevice> bondedDevices = bluetooth.getBondedDevices();

3-Estableciendo conexión

Para iniciar un canal de comunicación con un dispositivo remoto, hay que crear un objeto Bluetooth Socket mediante el Bluetooth Device que hemos obtenido en el paso anterior.

Según el libro de referencia [2], para crear una nueva conexión, se ha de llamar primeroo al método createRfcommSocketToServiceRecord a través del Bluetooth Device de la siguiente manera.

String uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB"));

BluetoothSocket clientSocket = device.createRfcommSocketToServiceRecord(uuid);

Pese a esto, a la hora de implementar mi aplicación, este paso causaba problemas y no permitía crear una conexión correcta con el NXT. La solución la encontré gracias a la página www.stackoverflow.com, y consiste en las siguientes dos líneas que sustituyen a las dos anteriores.

Method m = device.getClass().getMethod("createRfcommSocket", new Class[] { int.class });

clientSocket = (BluetoothSocket)m.invoke(device, Integer.valueOf(1));

Una vez se ha obtenido el socket, ya se puede iniciar la conexión mediante la llamada al método connect.

clientSocket.connect()

4-Transmisión de datos

Una vez se ha establecido la conexión, es posible transmitir y recibir datos usando los sockets de los dos dispositivos. La transferencia de datos sobre Bluetooth Sockets se gestiona a través de los objetos de Java InputStream y OutputStream, los cuales se obtienen mediante el socket con los métodos getInputStream y getOutputStream.

Se expone a continuación un ejemplo de la declaración de un paquete de bytes que contiene el mensaje "init".

```
private byte[] init = new byte[] {
    (byte)0x09, (byte)0x00, // Bytes length (both not included)
    (byte)0x80, // Direct comand, No response expected
    (byte)0x09, // Comand type
    (byte)0x05, // Inbox number
    (byte)0x05, // Message size (including null termination)
    (byte)'i', (byte)'n', (byte)'i', (byte)'t', // Message
    (byte)0x00 // Null termination
};
```

Una vez declarado el paquete y configurado con todos los datos necesarios, obtenemos el stream de salida y enviamos los datos a través de él.

OutputStream out = clientSocket.getOutputStream();

clientSocket.write(init);

4.4.4. Protocolo de comunicación del NXT

Ya se ha expuesto como realizar la conexión, abrir un flujo de datos y enviar paquetes de bytes a otro dispositivo desde un dispositivo Android. En lo que concierne a este proyecto, lo interesante ahora es

conocer el funcionamiento del protocolo Bluetooth en el NXT para poder recibir estos datos correctamente. Primero se detallarán los aspectos importantes del protocolo del NXT, y después se expondrán ejemplos de recepción de los datos Bluetooth.

1-Vista general del protocolo

La longitud máxima que permite el NXT en los paquetes enviados por bluetooth es de 64 bytes. Todos los mensajes deben incluir dos bytes al inicio para indicar la longitud de la trama. Estos dos bytes no se tienen en cuenta para indicar la longitud del paquete. En la figura 35 se muestra la estructura de una trama de datos del NXT.

Length, LSB	Length, MSB	Command Type	Command	Bytes 4 to N-1: Message	Byte N: End

Figura 35: PAG. 22. Trama de datos para enviar comandos bluetooth al NXT.

En la siguiente figura se detalla la función de cada byte en la trama.

Byte 0	LSB (Less Significant Byte). Dígito menos significativo para indicar la longitud del paquete
Byte 1	MSB(Most Significant Byte). Dígito mas significativo para indicar la longitud del paquete
Byte 2	Tipo de comando 0x00: Comando directo, se requiere respuesta 0x01: Comando de sistema, se requiere respuesta 0x02: Comando de respuesta 0x80: Comando directo, no se espera respuesta 0x81: Comando de sistema, no se espera respuesta
Byte 3	Comando. Byte que indica que se ha de hacer con los datos Open, Read, Write, Delete data, Direct comunication with NXT
Byte N-1	Estos bytes aportan información adicional
Byte N	0x00 que indica el final de paquete

Figura 36: Especificación de la función de cada byte de la trama y los valores que pueden tomar según lo que deban indicar.

Especialmente interesante resulta la posibilidad de dar ordenes directas al robot mediante comandos directos sin necesidad de que éste ejecute ningún programa en el robot. De esta forma es posible controlar

remotamente el movimiento del robot. Un ejemplo simple de comando directo es el que nos devuelve como respuesta el nivel de batería del NXT.

Byte 0	0x00 o 0x80
Byte 1	0x0B

Figura 37: Trama de bytes correspondiente a un comando directo para solicitar el nivel de batería del NXT.

Byte 0	0x02
Byte 1	0x0B
Byte 2	Status
Bytes 3-4	Voltage en mV

Figura 38: Trama de bytes correspondiente a la respuesta del NXT indicando su nivel de batería.

2-Comunicación Master-Slave

La comunicación entre robots NXT se basa en un protocolo master-slave, donde el dispositivo master es el que inicia la conexión, y puede tener conectados hasta 3 NXT slaves en las líneas 1, 2 y 3. Los dispositivos slaves ven siempre al dispositivo master en la línea 0.

En el ámbito de este proyecto, se ha decidido que el dispositivo móvil ejerza de master en la comunicación, y el NXT de slave. Por lo tanto será el móvil el que iniciará la conexión y el robot esperará a que la conexión esté establecida.

3-Esperando la conexión

En el programa del NXT es muy fácil comprobar la conexión o esperar a que ésta se produzca. Únicamente hemos de comprobar una variable de status del bluetooth, y hacer una espera mediante la instrucción until hasta que la condición se cumpla. En este caso, la variable BT_CONN es el numero correspondiente al canal donde se espera que la conexión se produzca.

until(BluetoothStatus(BT_CONN) == NO_ERR);

El programa se quedará bloqueado en este punto hasta que la conexión esté establecida, después seguirá con el flujo normal de ejecución.

4-Recepción de datos

La recepción de datos en el NXT se realiza de una forma muy sencilla. Se dispone de dos funciones en NXC que permiten leer datos recibidos por bluetooth.

ReceiveRemoteString(INBOX, true, in_message);

ReceiveRemoteNumber(INBOX, true, in_number);

Mediante las siguientes funciones se puede responder a los mensajes recibidos.

SendResponseString(OUTBOX, out_message);

SendResponseNumber(OUTBOX, out_number);

4.4.5. La cámara como un sensor más del NXT

Debido a que en este proyecto estamos trabajando sobre un robot puramente reactivo, la visión más simple y eficaz para nuestro sistema es que el móvil (y en consecuencia la cámara) sea como un sensor más para el NXT. El dispositivo móvil se encargará de ejecutar los algoritmos pertinentes sobre las imágenes captadas a través de la cámara, computar la posición del objeto en la imagen, y enviar una trama con los datos al robot para que éste actúe en consecuencia.

Por lo tanto, el dispositivo móvil únicamente proporcionará al robot la información de la posición del target en cada frame, y en caso de no haber coincidencia, le indicará que el target no se ha encontrado en el frame actual.

En el lado del robot necesitaremos una rutina que compruebe constantemente los datos recibidos por bluetooth y actualice los datos referentes a la posición del target para que el robot pueda responder en tiempo real a su movimiento. Se ha implementado un thread (task en NXC), que se encargue de esta tarea, de la misma forma que se encargan sus correspondientes threads de comprobar los sensores de luz y ultrasonido.

La trama de datos que envía el móvil es muy simple. El primer byte indica la situación del objeto en la imagen (izquierda, derecha, centro o perdido). El segundo y tercer byte indican de forma numérica

una posición relativa del objeto en la imagen, siendo este factor 0 si el objeto se encuentra centrado, y aumentando hasta 40 desde el centro hasta los extremos de la imagen. En la siguiente figura se puede ver una tabla explicativa.

Byte 0	Indica en que parte de la imagen se encuentra el objetivo, o si no se ha encontrado Valores: 'i' (izquierda), 'd' (derecha), 'c' (centro), 'l' (lost)
Byte 1	Dígito más significativo de la posición relativa del objeto en la imagen El rango de valores es {0, 40}: Por lo tanto este byte toma valores en el rango {0, 4}
Byte 2	Dígito menos significativo de la posición relativa del objeto en la imagen Toma valores en el rango {0, 9}

Figura 39: Especificación de la función y los valores que puede tomar cada byte de la trama que envía el dispositivo móvil al NXT con los datos de la cámara.

En el programa del robot debemos recibir estos datos y decodificarlos para almacenarlos en las variables que el programa necesita para funcionar. Se muestra a continuación la tarea (thread) que se encarga de leer los datos recibidos por bluetooth y actualizar las variables necesarias para el control de la trayectoria del robot.

```
task check_camera_sensor():
    while( true ):
        ReceiveRemoteString( INBOX, true, in_message );
        if( StrLen( in_message) > 0 ):
            if( mensaje[0] == 'l' ): lost = true;
            else: lost = false;
            if( state != AVOIDING ):
                dirToAvoid = 'x';
                direction = in_message[0];
                in_message[0] = '0';
                factor = StrToNum( in_message );
                if( direction == 'i' ): factor = factor * (-1);
                else: factor = factor * 1;
                updateMotorsPower();
```

4.5. Programación del robot

4.5.1. Autómata de estados del robot

A continuación se muestra el diagrama de estados del robot. En él se detallan las condiciones que se han de dar para la transición entre los distintos estados.

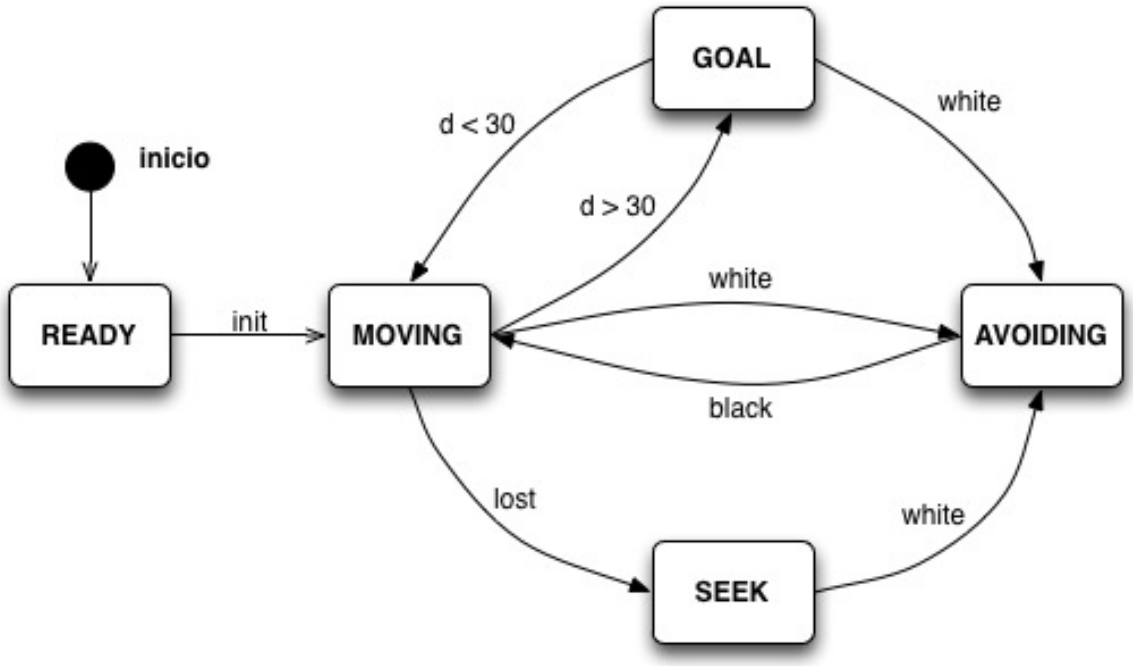


Figura 40: Autómata de estados del robot NXT para realizar el tracking de un objetivo.

Para el correcto funcionamiento del robot, se debe dar prioridad a algunos estados frente a otros. El estado de más prioridad es el de evitar obstáculos (AVOIDING). Si durante la ejecución de cualquier estado se detecta una línea blanca, se ha de cambiar al estado AVOIDING, ya que esquivar un obstáculo es más importante que buscar al objetivo o seguirlo.

4.5.2. Programación paralela en el NXT

El procesador NXT permite la programación de varios flujos de ejecución paralelos. A pesar de que el procesador es de un único núcleo, el hecho de dividir la adquisición de datos de los sensores y la actuación sobre los motores permite que el robot reaccione de forma más rápida. Esta pequeña diferencia resulta importante en un sistema que debe responder en tiempo real, y en el caso de este proyecto es más importante ya que, al tener cierta latencia en el sistema debido a la comunicación entre la cámara y el NXT, se ha de aprovechar toda oportunidad de compensar esa pérdida de rendimiento.

4.5.3. Proportional controller

Una vez se ha obtenido el mensaje proveniente de la cámara y se han decodificado los datos (apartado 4.4.5), se cuenta con un flag que indica si el objeto se ha detectado o no, y un factor en el rango {-40, 40} que indica cuanto ha de girar el robot para dirigirse al objetivo.

Para transmitir esta información a los motores y que esto se traduzca en un giro en la dirección del objetivo, se implementa el controlador proporcional. A continuación se muestra la rutina que actualiza la potencia de ambos motores.

```
sub updateMotorsPower() {
    if( factor == 0 ) {
        left_power = HIGH_POWER;
        right_power = HIGH_POWER;
    } else {
        left_power = MID_POWER + factor;
        right_power = MID_POWER - factor;
    }
}
```

Como se puede ver en el código, si el factor es cero (el objetivo está centrado en la imagen) se aplica la misma potencia a los dos motores para que el robot se desplace en línea recta.

Si no, se aplica la fórmula del controlador proporcional. Si el factor es negativo, el motor derecho recibe una potencia mayor y el robot gira a la izquierda. En caso contrario, el motor izquierdo recibe más potencia y el robot gira a la derecha.

4.5.4. Pseudocódigo del controlador

A continuación se muestra el pseudocódigo del controlador del robot:

```
task controller() {
    while( true ) {
        actualizar pantalla
        switch( state ) {
            case MOVING:
                si white: state = AVOIDING
                si no si lost: state = SEEK
                si no: proportional_forward
            case AVOIDING:
                parar motores
                si !lost:
                    si dirToAvoid == 'x':
                        dirToAvoid = direction
                    backward 400 ms
                    si direccion == 'i': rotar izquierda 1 seg.
                    si no: rotar derecha 1 seg.
                        rotar derecha 1 seg.
                        activar flag dirToAvoid
                si no:
                    backward 400 ms
                    rotar izquierda 1 seg.
                si black:
                    si !lost: state = MOVING
                    si no : state = SEEK
            case SEEK:
                si white: state = AVOIDING
                si no si !lost: state = MOVING
                si no:
                    si dirToAvoid:
```

```
                    rotar derecha
                si no:
                    si dirección == 'i':
                        factor = 40
                    si no: factor = -40;
                    actualizar potencia motores
                    proportional_forward
            case GOAL:
                si white: state = AVOIDING
                si no:
                    factor = 0
                    actualizar potencia motores
                    proportional_forward
}
```

Referencias

[1] Dave Astolfo, Mario Ferrari, Giulio Ferrari. Building robots with Lego Mindstorms NXT.

[2] Reto Meier. Professional Android 4 Application Development.

[3] http://www.robotc.net/wiki/Setting_up_Bluetooth_for_NXT

[4] Daniele Benedettelli. Programming LEGO NXT Robots using NXC.

[5] http://opencv.org/

[6] http://opencv.willowgarage.com/wiki/

[7] http://stackoverflow.com/

[8] http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html

[9] Documentación Oficial-Appendix 1. Lego Mindstorms NXT Comunication Protocol.

[10] Documentación Oficial-Appendix 2. Lego Mindstorms NXT Direct Commands.

[11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski. ORB: An efficient alternative to SIFT or SURF.

[12] http://es.wikipedia.org/wiki/Modelo_de_color_HSV

5. Observaciones y resultados obtenidos

Por la naturaleza de este proyecto, no se pueden mostrar resultados cuantitativos. Por lo tanto, se realizará una valoración cualitativa de las observaciones que se han apreciado en las diversas pruebas realizadas con el robot sobre el terreno.

Para ello, se han grabado cuatro videos en los que se pueden valorar distintos aspectos del sistema. Estos vídeos se pueden encontrar en la carpeta videos del fichero comprimido anexo.

A continuación se comentan algunas observaciones acerca de las distintas situaciones del robot en los videos.

OpticalFlow1

Este primer vídeo se llevó a cabo únicamente con la pelota, moviéndola por el escenario para evaluar el comportamiento del controlador proporcional. Como se puede ver, el robot varía su trayectoria de una forma bastante contínua en función de la posición del objetivo. También se puede ver la transición de estado cuando pierde el objetivo, pasando al comportamiento de búsqueda, en el cual gira sobre sí mismo hasta detectarlo de nuevo. Por último, se puede concluir que la rutina de evitar obstáculos resuelve correctamente el problema, y que resulta más difícil detectar la pelota cuando está demasiado cerca.

OpticalFlow (dificult avoid)

En este vídeo se puede ver claramente que resulta difícil definir el comportamiento de evitar obstáculos para este tipo de escenario. Esto es debido a que la línea exterior que delimita el escenario y las zonas blancas interiores no se pueden tratar de la misma forma, ya que la linea exterior se ha de evitar, pero las zonas interiores a menudo se deben sortear para llegar al otro lado y alcanzar así el objetivo.

Por otro lado, se puede ver que si los dos robots se están moviendo, resulta difícil hacer la detección, ya que se detecta al objetivo pero se pierde inmediatamente y el robot vuelve a ejecutar la rutina de búsqueda.

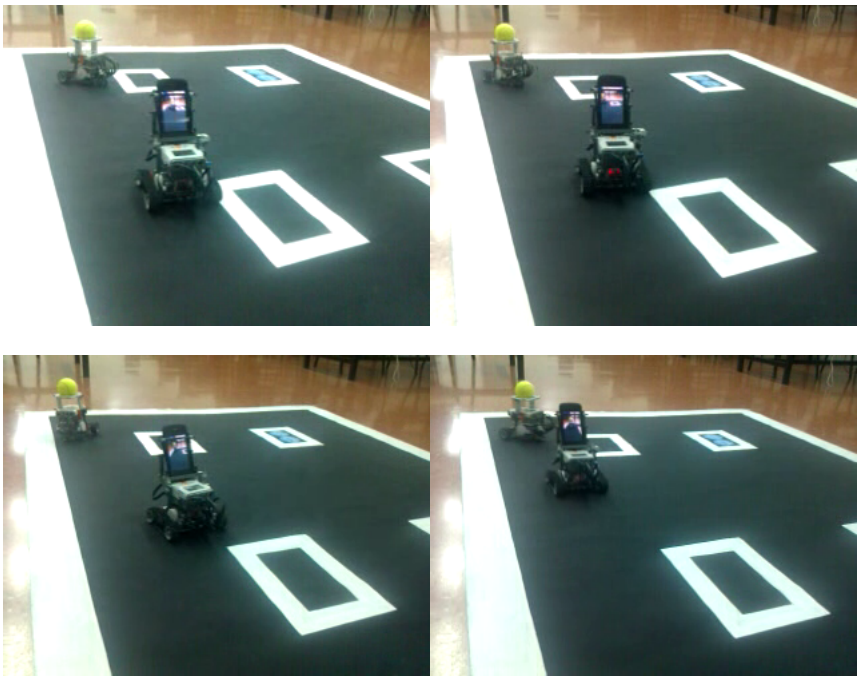


Figura 41: Cuatro frames extraídos del vídeo en los que se puede apreciar la variación de la trayectoria del robot en función del objetivo.

Optical (caza rápido)

En este vídeo se puede apreciar que al no tener que evitar ningún obstáculo, y al mantenerse el objetivo por un tiempo en la misma posición, resulta muy fácil y rápido para el robot llegar hasta su posición y no dejarlo ir.

OpticalFlow (ultimo)

En este último vídeo se puede ver que al principio resulta difícil detectar al objetivo, ya que se encuentra un poco alejado, y puede que influya la variación de luz, ya que no siempre resulta difícil detectar elobjetivo estando alejado. Una vez está más cerca, el robot lo detecta y modifica su trayectoria para dirigirser hacia él.

6. Conclusiones y futuros proyectos

La eficacia del método de detección ha sido la esperada. El único problema con este método es la variación de los resultados dependiendo de las condiciones de luz. Aun así, en el entorno de trabajo, el robot se ha comportado bien, realizando un detección bastante rápida, sin importar la distancia a la que se encuentre el objetivo (uno de los motivos por los que se descartó el uso del detector ORB).
Por otro lado, la inclusión del método de tracking ha aportado un incremento importante en el rendimiento. Al principio, el punto a veces se confundía en la fase de tracking con otros objetos de la escena, lo que provocaba que el robot tomase trayectorias erróneas. Para evitar esto, se ha introducido una actualización periódica de la detección para no perder el objetivo, lo que ha conseguido paliar esta situación.
Concluyendo, por los motivos expuestos anteriormente, se puede decir que se han alcanzado los objetivos que se plantearon al inicio del proyecto y por lo tanto, que se ha concluido con éxito.

Una vez extraídas las conclusiones, se pasa a comentar posibles ampliaciones y trabajos futuros en la línea de este proyecto.
Debido a que este proyecto es multidiscipinar, es muy dificil abarcar en profundidad todos los ámbitos posibles que se han ido planteando a lo largo de la realización del mismo.
Uno de los aspectos que no se han podido abordar, y no por ello es menos interesante, es el control de la cinemática del robot. Controlar la posición y la orientación del robot para que se desplace a puntos concretos con precisión es algo que podría ser muy útil en este tipo de problemas.
Otra mejora que se propone es el uso del Kalman Filter para realizar el tracking, ya que este método es más inmune al ruido, y esto podría ayudar a que la trayectoria del robot fuese más contínua.
Por último, una mejora que considero importante, consistiría en detectar también a través de las imágenes las línias que delimitan el escenario. Esto se podría realizar mediante la transformada de Hough, y podría ayudar a anticipar la conducta de evitar obstáculos, y no depender únicamente del sensor de luz para la detección de las líneas blancas, ya que como se ha podido ver en el vídeo OpticalFlow (dificul avoid), a veces le resulta difícil al robot sortear el obstáculo.
En conclusión, puedo decir que finalizado este proyecto, tengo más ganas de profundizar en el campo

de la visión por computador y de la robótica que al inicio. Los resultados obtenidos a lo largo de este proyecto han resultado motivadores, y la ventana de posibilidades que se abre cada vez que se aprende algo nuevo es muy amplia.

Por ello, me gustaría ligar mi futuro académico y profesional a estos dos campos, y profundizar en técnicas más complejas que se puedan emplear en otro tipo de problemas.

7. Anexos

7.1. Instalación y configuración del software y las librerías necesarias

7.1.1. Instalar el IDE Eclipse

1. Descargar e instalar Eclipse. En este proyecto se ha utilizado la versión Helios para desarrolladores Java (http://www.eclipse.org/downloads/packages/release/helios/sr2).

También es posible descargar el bundle ADT + Android SDK (http://developer.android.com/sdk/index.html), que está preparado para comenzar a programar en Android inmediatamente. Con este paquete no es necesario instalar Eclipse, los plugins ADT y CDT, ni el Android SDK. En la siguiente página se encuentra disponible la descarga, y los pasos de instalación se encuentran en el apartado Setting Up the ADT Bundle. Si se descarga este paquete se pueden saltar los pasos 2 y 3.

7.1.2. Instalar plugin ADT

El proceso de instalación de plugins en Eclipse es muy sencillo.

1. Seleccionamos Help -> Install New Software...

2. En la ventana que aparece, pulsar el botón Add. Introducir en el siguiente diálogo un nombre para el plugin y la dirección url (https://dl-ssl.google.com/android/eclipse/).

3. Pulsar OK y esperar a que aparezcan los paquetes disponibles. Seleccionar los que se van a instalar y pulsar Next. Aceptar las condiciones y pulsar Next de nuevo. Una vez acabado el proceso pulsar Finalizar.

7.1.3. Instalar Android SDK

Descargar el SDK de Android en la siguiente página, y descomprimir el contenido en la carpeta de trabajo.
http://developer.android.com/sdk/index.html

7.1.4. Instalar OpenCV4Android SDK

1. Descargar el SDK de OpenCV (http://sourceforge.net/projects/opencvlibrary/files/opencv-android/)
2. Descomprimir el contenido en la carpeta de trabajo.
3. Para importar la librería OpenCV a Eclipse, hacer click con el botón derecho del ratón en el Package Explorer y seleccionar Import... -> General -> Existing Projects into Workspace. Seleccionar la ruta donde se ha descomprimido el SDK de OpenCV y pulsar Finish.

7.1.5. Instalar OpenCV Manager

Para poder ejecutar aplicaciones que utilizan OpenCV necesitamos instalar en el dispositivo, o en el simulador el OpenCV Manager.
En el caso de un dispositivo móvil, desde la Play Store se puede instalar gratuitamente la aplicación OpenCV Manager.
Si no, con el emulador conectado. Si hemos declarado la carpeta platform-tools como variable de entorno, y copiamos el apk en nuestra carpeta root, ejecutar la siguiente instrucción:
adb install OpenCV_2.4.3.2_Manager_2.4_x86.apk
Si no, entrar en la ruta /plattform-tools (dentro de la carpeta del android sdk), y ejecutar la instrucción incluyendo la ruta del apk.

7.1.6. Añadir la librería de OpenCV en un proyecto

Para poder incluir la librería en un proyecto, primero se ha de importar a Eclipse.

1. En el explorador de proyectos, hacer click en el botón derecho del ratón y seleccionar importar.
2. Seleccionar la ubicación de la carpeta donde se ha guardado el SDK de OpenCV4Android.
3. Pulsar Finish.
4. Click derecho en el proyecto en el que se va a incluir OpenCV, y seleccionar la opción Propiedades.
5. Seleccionar la segunda pestaña (Android), y en el cuadro inferior seleccionar Add.

6. Seleccionar la librería y aceptar.

De esta forma estarán disponibles las funciones de OpenCV en el proyecto Android. Para usarlas se deben declarar los imports correspondientes en el fichero Java.

7.2. Guía de usuario de la aplicación

Los dispositivos deben estar previamente sincronizados por bluetooth. Además se debe descargar el programa en el brick NXT, y tener instalada la aplicación Android en el móvil.
A continuación se detallan los pasos a seguir cuando se cumplen los requisitos anteriores.

1. Ejecutar el programa del robot en el brick NXT.
2. Ejecutar el programa en el móvil.
3. En el móvil pulsar conectar, y esperar a que aparezca el texto Desconectar. Esto significa que se ha establecido la conexión entre el móvil y el robot.
4. En el movil, elegir el método de tracking.

En el móvil, pulsar start.

Referencias

[1] http://opencv.org/

[2] http://developer.android.com/