industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Sistema móvil autónomo para seguimiento de objetivo basado en Lego Mindstorm

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Ángel Luis Rodríguez Carrión**
Directores: Julio José Ibarrola Lacalle
José Manuel Cano Izquierdo

Universidad Politécnica de Cartagena

Cartagena, Marzo 2015

AGRADECIMIENTOS

Este proyecto es el resultado de un intenso periodo de esfuerzo y de trabajo. Durante este tiempo, son muchas las personas que, de una u otra forma, me han ayudado a alcanzar la meta, por ello, quiero expresarles mi más sincera gratitud.

En primer lugar agradecer a Julio José Ibarrola y José Manuel Cano, mis directores, la oportunidad que me brindaron en su día de embarcarme en este interesante proyecto.

También agradezco a Pablo Martínez, su disponibilidad e interés en las ocasiones en las que he precisado su ayuda.

Finalmente, en un aspecto más personal, quisiera agradecer a mis padres el apoyo y constancia recordándome que este proyecto no se iba a acabar solo, y a mi novia, Elena, por aguantarme durante todos estos años, animarme y ayudarme tanto con este último esfuerzo. Gracias a ellos todo ha sido más sencillo.

RESUMEN

Los primeros robots operaban en entornos especialmente preparados para ellos. Cada componente de su espacio de trabajo se encontraba situado en una posición y orientación predefinidas, de modo que el robot conocía a priori y con exactitud el escenario donde se encontraba. En la actualidad muchas de las aplicaciones requieren que los robots tengan rasgos de autonomía como son la capacidad para identificar mediante sus sensores las características que tienen el entorno o la capacidad para auto-localizarse dentro del mismo.

En los últimos años se han propuesto una gran variedad de técnicas de posicionamiento de robots móviles. Su planteamiento varía considerablemente en función del medio en el cual se mueve el robot, del conocimiento que se posea tanto del entorno como de la tarea a realizar, y del conjunto de sensores disponible. En general, la posición puede determinarse a través de sensores, tanto internos como externos.

Además se aborda el diseño e implementación de un sistema de visión por computador, capaz de realizar la detección y seguimiento 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.

ABSTRACT

The first robots were designed to operate in environments specially prepared for them. Each component of their workspace was located in a predefined position and orientation, so that the robot knew a priori and accurately the scene where it was. At present, many applications require the robots to have autonomous characteristics, such as the ability to identify the environment features using its sensors, or the ability to self-localize inside the scene.

In recent years it has been proposed a variety of positioning techniques for mobile robots. The approach varies considerably depending on the medium where the robot moves, on the existing knowledge of both the environment and the task to be made, and on the set of available sensors. Usually, the position can be achieved through internal and external sensors.

Besides, the design and implementation of a computer vision system, able to detect and to track a specific object in a given environment, is addressed.

The robot has been designed according to the paradigm of behaviour-based robotics, and has been implemented under the Lego Mindstorms NXT platform.

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

ÍNDICE

CAPÍTULO 1

INTRODUCCIÓN Y OBJETIVOS

"Solo es posible avanzar cuando se mira lejos. Solo cabe progresar cuando se piensa en grande"

José Ortega y Gasset

1.1. Introducción

Uno de los campos más prometedores de la robótica móvil, es el de otorgar autonomía a vehículos terrestres con el objetivo de reducir la intervención de supervisores humanos en entornos de diferente complejidad, desde tareas rutinarias a trabajos en ambientes tóxicos o climatológicamente adversos.

Teniendo en cuenta esto, no se puede incluir dentro de esta categoría ningún tipo de máquina teleoperada ni vehículos autoguiados, ya que unos dependen del control humano y otros se limitan a seguir caminos preestablecidos o a actuar de forma repetitiva. La necesidad de movimiento no teledirigido en robótica se aprecia claramente en sistemas que desarrollan su operación en entornos remotos.

El giro de los elementos motrices de un vehículo móvil, independientemente de que sean ruedas o cadenas, puede proporcionar información suficiente como para determinar su posición. En estas situaciones se dice que la localización está basada en la odometría. Sin embargo, existen errores asociados a deslizamientos, fricciones o la propia orografía del terreno que son acumulables. De este modo, llegará un momento en el que el sistema no pueda determinar con el grado de exactitud necesario la posición del robot. Por tanto, se precisa de un sistema alternativo de localización, que sin omitir los resultados provenientes de las mediciones odométricas, corrijan la posición del vehículo y localicen el mismo con la mayor exactitud posible. Un proceso de localización, el cual, a través de la información proveniente de los sensores externos, sea capaz de determinar la posición y orientación ocupada por el robot móvil.

Por otro lado, el continuo 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 engloba el uso de las nuevas tecnologías mecánicas y electrónicas, como los robots de Lego Mindstorms NXT, con el uso de últimas tecnologías en comunicación, como el Bluetooth, y con el estudio de la detección y seguimiento de objetivos por visión artificial. En el Departamento de Ingeniería de Sistemas y Automática de la UPCT se han llevado a cabo en los últimos años varios proyectos basados en el set de robótica programable LEGO Mindstorms NXT. Entre los proyectos precursores, está el proyecto fin de carrera titulado “Estudio de las posibilidades didácticas en la ingeniería de control del LEGO Mindstorms NXT” [1]. El objetivo principal de este proyecto era el de estudiar la viabilidad del set LEGO Mindstorms NXT (Sensores, Actuadores y Unidad de Control) como medio para implementar sobre él algoritmos de control. Tras su evaluación favorable se adoptó como herramienta para proyectos posteriores.

Siguiendo con el uso de este hardware, este proyecto trabajará sobre una construcción particular de una serie de prototipos móviles con distintas configuraciones cinemáticas que consistirá fundamentalmente en la aplicación de técnicas de localización con sensorización interna sobre dicho microrobot. Posteriormente se implementará un sistema de visión para el seguimiento de un objetivo móvil en el espacio basado en OpenCV4Android.

1.2. Objetivos

El objetivo de este proyecto queda reflejado en el título: diseñar y construir un sistema móvil autónomo para seguimiento de objetivo utilizando Lego Mindstorm.

Para ello se pretende dotar de capacidades de autonomía y de seguimiento de objetivo a un robot móvil, que permita una respuesta dinámica para la localización en tiempo real mediante sensorización interna. Se implementarán varias soluciones con distintas estructuras para distintos modelos cinemáticos con el motivo de hacer una comparativa. Para el manejo de los microrobots se crearán una serie de softwares de control para facilitar el manejo de los prototipos. Se desarrollará un sistema de comunicaciones necesario para poder intercambiar información entre más de un robot/pc/smartphone simultáneamente, mediante la utilización de Bluetooth.

En concreto, la lista de tareas que se han marcado en este proyecto son:

- Conocer en profundidad todas las opciones que ofrece el robot Lego Mindstorms NXT con el firmware LejOS.
- Pruebas de velocidad de la comunicación vía Bluetooth entre las plataformas usadas.

- Análisis de los distintos sensores para microrobots.
- Estudio de las distintas configuraciones cinemáticas para microrobots.
- Diseño y construcción de plataformas factibles.
- Analizar las posibilidades de captura y reconocimiento de formas en imágenes a través de OpenCV.
- Desarrollo de aplicaciones que hagan uso de los conocimientos adquiridos.

1.3. Estructura del proyecto

El proyecto se encuentra estructurado en los siguientes capítulos:

Capítulo 1. Introducción y Objetivos

En este primer apartado se ponen de manifiesto los argumentos que respaldan este proyecto. Se expone una breve introducción al problema y se detallan los objetivos generales y específicos que se llevarán a cabo.

Capítulo 2. Estado del arte

En este capítulo introduciremos qué se entiende por un robot móvil e incidiremos en la explicación de las técnicas más empleadas en la localización, las posibilidades de realimentación y los actuales proyectos con Lego sobre localización odométrica así como en las tecnologías empleadas en este proyecto y sus principales funciones.

Capítulo 3. Robots móviles

Se detallan las diferentes configuraciones cinemáticas para microrrobots y se modelan matemáticamente las propuestas en este proyecto a estudiar.

Capítulo 4. Diseño e Implementación del sistema

Describiremos, de forma detallada, las propiedades de la estructura que compone el robot móvil diseñado, se detallan aspectos del diseño y la programación del sistema planteado. Se expondrá el diseño del robot Mindstorm, la implementación de la parte de la comunicación Bluetooth, y se explicará la aplicación Android y la programación del robot. También la construcción de parte de seguimiento de objetico que cuenta con la construcción de una torreta para rotar en 3 dimensiones y la programación de un smartphone como sistema de visión artificial.

Además, se encuentran explicados detenidamente las subrutinas desarrolladas empleadas para el control, adquisición y tratamiento de datos del robot móvil, y su relación con en el algoritmo de localización.

Capítulo 5. Observaciones y resultados obtenidos

Se presentan los resultados experimentales obtenidos durante la ejecución de los programas definidos en apartados anteriores en los diferentes prototipos para las diferentes trayectorias, así como la capacidad de detección y seguimiento de un objeto.

Capítulo 6. Conclusiones y futuros proyectos

En este apartado, se muestran las conclusiones obtenidas basadas en los resultados obtenidos, así como las posibles líneas de investigación a seguir.

Capítulo 7.Bibliografía y referencias

Se recoge la relación de recursos bibliográficos y electrónicos utilizados.

Anexos:

En ésta última 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.

CAPÍTULO 2

ESTADO DEL ARTE

"Quien no sabe lo que busca no comprende lo que haya"

Anónimo

2.1. Introducción

El problema de la localización de robots móviles consiste en contestar, desde el punto de vista del robot, a la pregunta presentada en la Figura 2.1: ¿Dónde estoy? Esto significa que el robot, valiéndose de sus sensores, debe encontrar su posición relativa al entorno donde se encuentra. El problema general de la localización consta de un cierto número de instancias que se pueden clasificar según su grado ascendente de dificultad: seguimiento de posición (Tracking), posicionamiento global (Global localization), recuperación ante transferencias intempestivas (Kinapping problem) y localización en ambientes dinámicos.

La localización del robot resulta clave en el diseño de robots verdaderamente autónomos. Si un robot no sabe dónde está, probablemente resulte difícil determinar qué hacer a continuación. Con el fin de localizarse a sí mismo, un robot tiene acceso a la información relativa y absoluta proporcionada por sus sensores internos y externos que nos permiten conocer, respectivamente, la distancia recorrida y las características del entorno del robot. Teniendo en cuenta esta información, el robot tiene que determinar su posición con la mayor precisión posible.

Esta adquisición de información se ve afectada por la imprecisión propia de los dispositivos sensoriales por lo que es preciso combinarlas de forma óptima con diferentes sistemas de localización con el fin de estimar, con la mayor precisión posible, la posición real del robot.

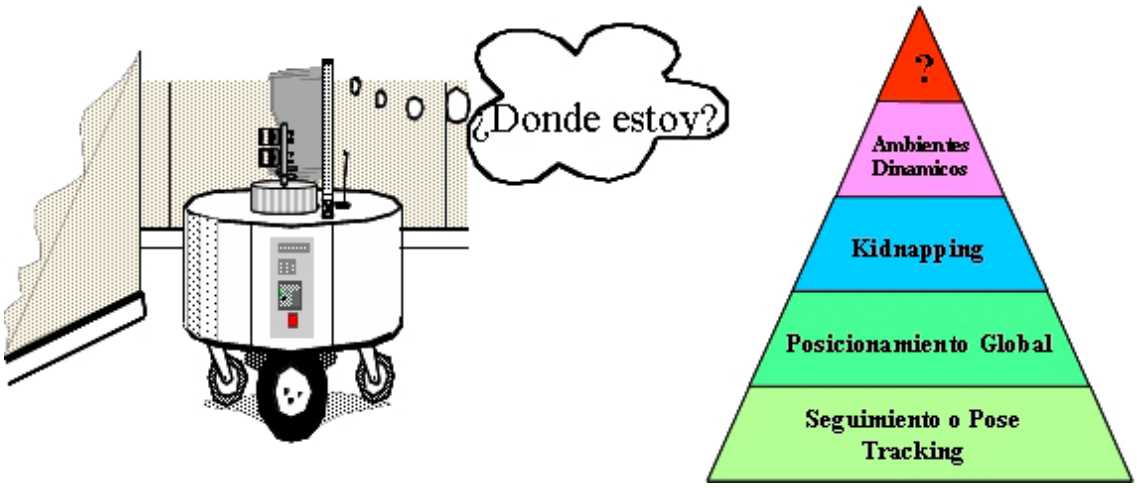


Figura 2.1: Clasificación jerárquica de la localización de robots móviles

2.2. Robots móviles autónomos

La robótica es la ciencia y la tecnología de los robots. Combina diversas disciplinas como son: la mecánica, la electrónica, la informática, la inteligencia artificial y la ingeniería de control.
El término robot se popularizó con el éxito de la obra RUR (Robots Universales Rossum), escrita por Karel Capek en 1920. En la traducción al inglés de dicha obra, la palabra checa "robota", que significa trabajos forzados, fue traducida al inglés como robot. Desde entonces, se ha avanzado mucho en investigación en la robótica, dónde se ha tornado el concepto inicial de robots estáticos y no autónomos al predominio actual de la movilidad y la autonomía.

2.3. Estructura general de un robot móvil

2.3.1. Componentes de un robot móvil

Vistos los principales tipos de robots móviles que se construyen en la actualidad, a continuación se detallan las partes constituyentes de los robots, tanto estructurales, como mecánicas y electrónicas [2].

- **Estructura**

La estructura es el esqueleto, el soporte fundamental que constituye tanto la forma como la funcionalidad del robot. Sirve de sujeción para toda la electrónica, sensores, actuadores y también es parte del aparato motriz como prolongación de los actuadores. Está formada generalmente por una mezcla de partes rígidas y flexibles, fijas y móviles y entre sus materiales destacan los plásticos, metales y aleaciones resistentes a la par que ligeras como la fibra de carbono o derivados del aluminio como se puede ver en la Figura 2.2:

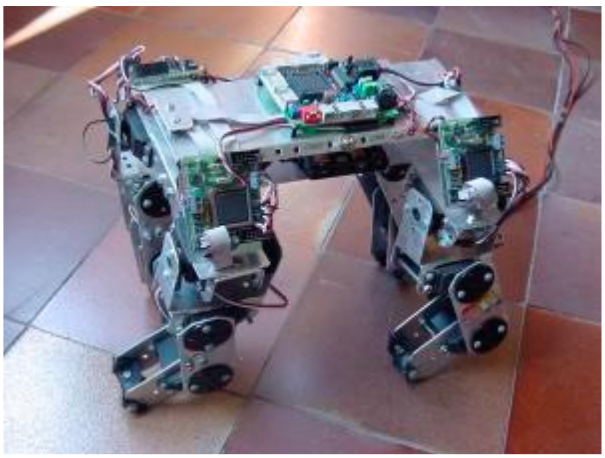


Figura 2.2: Estructura básica fabricada en aluminio para un robot cuadrúpedo.

Determina el medio en el que se va a poder desenvolver el robot, así como las actividades que será capaz de realizar. También protege partes sensibles de la electrónica de golpes, polvo, agua y otros agentes externos. Como ya se ha comentado, debe ser un compromiso entre resistencia

y ligereza, adaptándose de la mejor manera posible al tipo de tareas para las que se diseña el robot que va a poseer dicha estructura.

- **Sensores**

Muy estrechamente ligado a la autonomía de un robot está la capacidad de percibir el entorno y actuar sobre el mismo. Por medio de los sensores el robot obtiene datos sobre su entorno (sensación) y luego los procesa para su utilización (percepción). De esta forma, los sensores representan para el robot la fuente de datos sobre los cambios, tanto en su entorno (distancia a objetos, luz ambiental) como en sí mismo (nivel de la baterías, consumo de los motores, pulsos de codificadores, etc.).

En robótica móvil se usa una gran variedad de sensores que miden distintas magnitudes. En general, se puede clasificar en dos grupos: sensores internos y sensores externos. Los sensores internos son aquellos que no interaccionan con el entorno del robot. Su función es suministrar datos relacionados a las variables cinemáticas del robot y de cómo varían en función del accionamiento de los motores. Los sensores internos más característicos son: potenciómetros, tacómetros, codificadores ópticos rotativos, giroscopios y acelerómetros.

Los sensores externos son sistemas que interaccionan con el entorno para cuantificar alguna variable perteneciente al mismo. Estos elementos son los encargados de adquirir información del entorno y transmitirla a la unidad de control del robot. Una vez esta es analizada, el robot realiza la acción correspondiente a través de sus actuadores.

Los sensores constituyen el sistema de percepción del robot, es decir, facilitan la información del mundo real para que el robot la interprete. Los tipos de sensores más utilizados son los siguientes:

- Ultrasonidos*:* Son ampliamente utilizados en aplicaciones de modelado del entorno. Permiten la medida de distancias a partir del tiempo transcurrido desde la emisión de un pulso de ultrasonido hasta que éste es recibido nuevamente por el receptor. La precisión obtenida puede ser de un centímetro pero las medidas dependen de factores exteriores como la temperatura ambiente, movimientos del aire y fuentes acústicas de alta frecuencia (máquinas rotativas). Tienen un alcance entre 40 cm y 10 m.

Figura 2.3: Sensor ultrasonidos

- Infrarrojos*:* Tienen mayor precisión (±2mm) pero menor alcance que los sensores de ultrasonidos (entre 10 cm y 80 cm) por lo que son útiles, especialmente, para maniobra de aproximación y de navegación por evitación de obstáculos.

Figura 2.4: Sensor infrarojos

- Láser: Pueden ser utilizados para detectar medidas de distancia a objetos opacos, así como para la localización mediante triangulación, determinando la distancia a puntos conocidos del entorno. Poseen buena precisión y velocidad de adquisición de datos pero, en contrapartida, presentan un costo generalmente elevado.

Figura 2.5: Sensor Laser

- Visión*:* Las cámaras son ampliamente usadas en sistemas de localización y construcción de mapas del entorno. Los sistemas de visión estereoscópica, consisten en un par de cámaras que, además de conseguir la detección de contornos, también permiten obtener, de forma aproximada, la distancia a objetos a partir de la divergencia de ambas cámaras.

Figura 2.6: Cámara visión artificial

- Sensor de proximidad*:* Detecta la presencia de un objeto ya sea por rayos infrarrojos, por sonar, magnéticamente o de otro modo.

Figura 2.7: Sensores de proximidad

- Sensor de Temperatura*:* Capta la temperatura del ambiente, de un objeto o de un punto determinado.

Figura 2.8: Sensores de temperatura

- Sensores magnéticos: Captan variaciones producidas en campos magnéticos externos. Se utilizan a modo de brújulas para orientación geográfica de los robots.

Figura 2.9: Sensores magnéticos

- Sensores táctiles, piel robótica*:* Sirven para detectar la forma y el tamaño de los objetos que el robot manipula. La piel robótica se trata de un conjunto de sensores de presión montados sobre una superficie flexible.

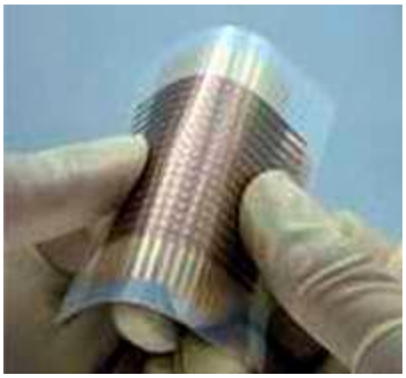


Figura 2.10: Sensor táctil flexible

- Sensores de iluminación*:* Capta la intensidad luminosa, el color de los objetos, etc. Es muy útil para la identificación de objetos. Es parte de la visión artificial y en numerosas ocasiones son cámaras.

Figura 2.11: Sensores de luz

- Sensores de velocidad, de vibración (Acelerómetro) y de inclinación*:* Se emplean para determinar la velocidad de actuación de las distintas partes móviles del propio robot o cuando se produce una vibración. También se detecta la inclinación a la que se encuentra el robot o una parte de él.

Figura 2.12: Sensores de velocidad de reluctacia variable

- Sensores de fuerza*:* Permiten controlar la presión que ejerce la mano del robot al coger un objeto.

Figura 2.13 Sensor de presión

- Sensores de sonido*:* Micrófonos que permiten captar sonidos del entorno.

Figura 2.14: Micrófonos

- Microinterruptores*:* Muy utilizados para detectar finales de carrera.

Figura 2.15: Diferentes tipos de micro interruptores

- **Actuadores**

Los actuadores son los sistemas de accionamiento que permiten el movimiento de las articulaciones del robot. Se clasifican en tres grupos, dependiendo del tipo de energía que utilicen:

- Hidráulicos*:* Se utilizan para manejar cargas pesadas a una gran velocidad. Sus movimientos pueden ser suaves y rápidos.

- Neumáticos: Son rápidos en sus respuestas, pero no soportan cargas tan pesadas como los hidráulicos.

- Eléctricos: Son los más comunes en los robots móviles. Un ejemplo son los motores eléctricos, que permiten conseguir velocidades y precisión necesarias.

Ejemplos de actuadores son motores, relés y contadores, electro válvulas, pinzas, etc.

- **Sistemas de control**

El control de un robot puede realizarse de muchas maneras, pero generalmente se realiza por medio de un ordenador industrial altamente potente, también conocido como unidad de control o controlador. El controlador se encarga de almacenar y procesar la información de los diferentes componentes del robot industrial.

La definición de un sistema de control es la combinación de componentes que actúan juntos para realizar el control de un proceso. Este control se puede hacer de forma continua, es decir en todo momento o de forma discreta, es decir cada cierto tiempo. Si el sistema es continuo, el control se realiza con elementos continuos. En cambio, cuando el sistema es discreto el control se realiza con elementos digitales, como el ordenador, por lo que hay que digitalizar los valores antes de su procesamiento y volver a convertirlos tras el procesamiento.

Existen dos tipos de sistemas, sistemas en lazo abierto y sistemas en lazo cerrado.

1) **Sistemas en lazo abierto**: Son aquellos en los que la salida no tiene influencia sobre la señal de entrada.

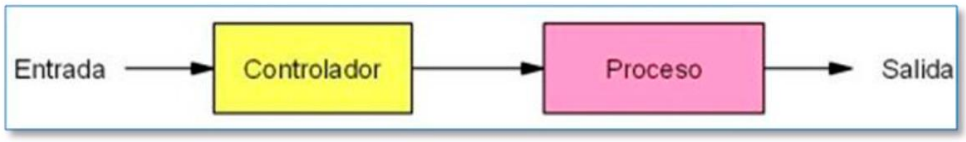


Figura 2.16: Esquema de un controlador en lazo abierto

2) **Sistemas en lazo cerrado:** Son aquellos en los que la salida influye sobre la señal de entrada.

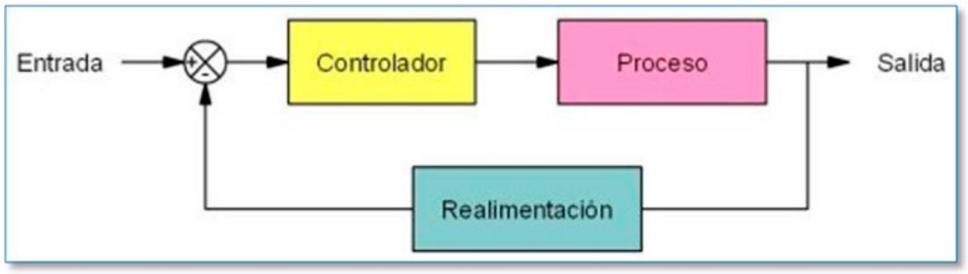


Figura 2.17: Esquema de un controlador en lazo cerrado

- **Sistemas discretos:** Son aquellos que realizan el control cada cierto tiempo. En la actualidad se utilizan sistemas digitales para el control, siendo el ordenador el más utilizado, por su fácil programación y versatilidad.

Generalmente, el control en los robots se realiza mediante sistemas discretos en lazo cerrado, realizados por computador. El ordenador procesa la información captada por los sensores y activa los actuadores en intervalos lo más cortos posibles, del orden de milisegundos.

- **Alimentación**

Parte fundamental para garantizar la autonomía del robot, dado que al ser móvil generalmente no va a poder tener energía externa, sólo contará con las reservas internas que pueda transportar, salvo en el caso de que se usen placas solares. Son muchas y muy diversas las formas en que el robot puede almacenar y transportar energía, por lo que veremos las principales:

Figura 2.18: Diferentes tipos de baterías de uso doméstico

- Baterías*:* Sin duda la forma más comúnmente utilizada como fuente de alimentación en todo tipo de robots. Son baratas, fiables y se pueden encontrar en cientos de formatos, voltajes y dimensiones.

  Desde la pila de botón más pequeña y ligera para alimentar un micro robot podemos ir a pesadas pero potentes baterías de plomo usadas en vehículos pesados que necesitan un aporte importante de energía.

  Otro aspecto importante a tener en cuenta es el formato y el conexionado de las pilas. Habitualmente encontramos las baterías recargables tanto sueltas como en packs preparados para soldar o para conectar directamente a un PCB. La elección de un formato u otro es más relevante de lo que parece.

  Por ejemplo, un robot móvil todo terreno que lleve las baterías de forma individual en un porta-pilas no sería nada extraño que con el movimiento alguna pila dejase de hacer contacto o incluso se saliera de su sitio. Y en el otro extremo, si las pilas estuvieran soldadas habría serios problemas para reemplazarlas con facilidad, por no hablar de la dependencia a la hora de cargarlas que deberá ser realizado exclusivamente en el robot, teniendo que añadir los mecanismos necesarios para ello.

  Parece por tanto que lo más adecuado es usar pilas en packs con conectores, que son las que mayor flexibilidad aportan.

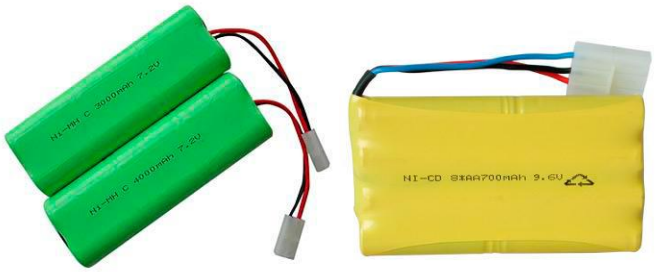


Figura 2.19: Diferentes tipos de baterías

Si el presupuesto no es un problema se pueden usar tecnologías más avanzadas como las baterías de ion de litio que últimamente están sufriendo una gran expansión gracias a tecnologías móviles de otras índoles como ordenadores portátiles y teléfonos móviles.

Figura 2.16: Amplio surtido de baterías de Ion de Litio

La diferencia de rendimiento de unos tipos de pilas respecto a otros es notable, así como su tamaño y precio, por lo tanto es importante tener claros los requerimientos del robot respecto a estos aspectos antes de decidir qué tipo de batería se va a utilizar para alimentarlo.

- Baterías inerciales**:** Este curioso mecanismo tampoco tiene una difusión amplia en el terreno de los robots móviles, pero si tiene aplicación en entornos donde la alimentación es crítica,

como centros de proceso de datos. Fue probada en vehículos durante sus albores, concretamente en autobuses urbanos de Zúrich durante los años 50 y algunas locomotoras de tren durante los 70, pero no se ha extendido su uso de forma masiva.

El principio de funcionamiento es sencillo: se acelera un rotor (habitualmente mediante electricidad) confiriéndole así una elevada energía rotacional, típicamente de varias decenas de miles de rpm. Posteriormente se decelera poco a poco, recuperando así la energía eléctrica de nuevo.

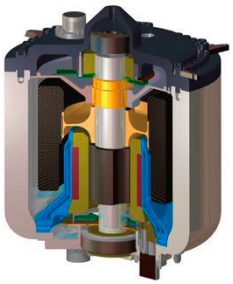


Figura 2.20: Sección de una batería inercial

La clave del aprovechamiento energético, que oscila entre el 90% y el 98%, consiste en reducir al máximo el rozamiento. Esto se consigue mediante usando un rotor de carbono (que presenta menos resistencia que el acero) estabilizado magnéticamente en un entorno controlado, generalmente un contenedor sellado tras realizarle el vacío.

- **Comunicaciones**

Dado que este es uno de los aspectos sobre los que versan las experiencias realizadas en este proyecto, se le dedicara el apartado siguiente a este tema, de forma que se traten en profundidad las características más relevantes:

- USB*:* El Universal Serial Bus (bus universal en serie) o Conductor Universal en Serie (CUS), abreviado comúnmente USB, es un puerto de comunicación de periféricos a un computador. Fue creado en 1996 con la finalidad de eliminar la necesidad de adquirir tarjetas separadas para poner en los puertos de bus ISA o PCI y mejorar las capacidades plug-and-play permitiendo a esos dispositivos ser conectados o desconectados sin necesidad de reiniciar el sistema. Sin embargo, en aplicaciones donde se necesita ancho de banda para grandes transferencias de datos los buses PCI o PCIe salen ganando e igual sucede si la aplicación requiere robustez industrial.

Se pueden clasificar en cuatro tipos según su velocidad de transferencia de datos:

1) **Baja velocidad (1.0):** Tasa de transferencia de hasta 1,5 Mbps (192KB/s). Utilizado en su mayor parte por dispositivos de interfaz humana como los teclados, ratones e incluso artículos del hogar.

2) **Velocidad completa (1.1):** Tasa de transferencia de hasta 12Mbps (1,5MB/s), según el estándar, pero se dice en fuentes independientes que habría que realizar nuevamente las mediciones.

3) **Alta velocidad (2.0):** Tasa de transferencia de hasta 480 Mbps (60MB/s) pero por lo general de hasta 125Mbps (16MB/s). Está presente casi en el 99% de los ordenadores actuales. El cable USB 2.0 dispone de cuatro líneas, una para datos, una para corriente y otra de toma de tierra.

4) **Super alta velocidad (3.0)**: Tiene una tasa de transferencia de hasta 4.8 Gbps (600MB/s). Esta especificación será diez veces más veloz que la anterior 2.0. Se han incluido cinco conectores extra, desechando el conector de fibra óptica propuesto inicialmente y es compatible con los estándares anteriores.

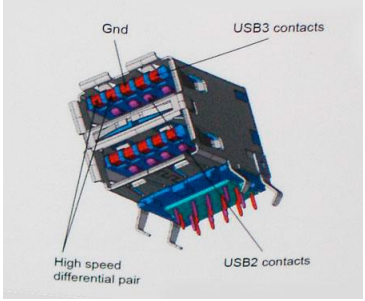


Figura 2.21: Prototipo de USB 3.0

- **Comunicaciones inalámbricas**

- WiFi: WLAN (Wireless Local Area Network, en inglés) es un sistema de comunicación de datos inalámbrico flexible, muy utilizado como alternativa a las redes LAN cableadas o como extensión de éstas. Utiliza tecnología de radiofrecuencia que permite mayor movilidad a los usuarios al minimizar las conexiones cableadas. Las WLAN van adquiriendo importancia en muchos campos, como almacenes o para manufactura, en los que se transmite la información en tiempo real a una terminal central. También son muy populares en los hogares para compartir el acceso a Internet entre varias computadoras.

Figura 2.22: Logotipo Wifi

Características:

- Movilidad: Permite transmitir información en tiempo real en cualquier lugar de la organización o empresa a cualquier usuario. Esto supone mayor productividad y posibilidades de servicio.
- Facilidad de instalación: Al no usar cables, se evitan obras para tirar cable por muros y techos, mejorando así el aspecto y la habitabilidad de los locales, y reduciendo el tiempo de instalación. También permite el acceso instantáneo a usuarios temporales de la red.

- Flexibilidad: Puede llegar donde el cable no puede, superando mayor número de obstáculos, llegando a atravesar paredes. Así, es útil en zonas donde el cableado no es posible o es muy costoso: parques naturales, reservas o zonas escarpadas.

- *Bluetooth:* Proviene de la palabra escandinava "*Blåtand"* que significa "hombre de tez oscura" pero en los tiempos que corren el significado original se ha perdido y ahora se asocia a las comunicaciones inalámbricas, un estándar global que posibilita la transmisión de voz, imágenes y en general datos entre diferentes dispositivos en un radio de corto alcance y lo que le hace más atractivo, muy bajo coste.

Figura 2.23: Logotipo Bluetooth

Los principales objetivos que este estándar quiere lograr son:

- Facilitar las comunicaciones entre equipos.
- Eliminar cables y conectores entre aquéllos.
- Facilitar el intercambio de datos entre los equipos.

Bluetooth funciona bajo radio frecuencias pudiendo atravesar diferentes obstáculos para llegar a los dispositivos que tenga a su alcance. Opera bajo la franja de frecuencias 2.4 – 2.48 GHz o como también es conocida como "Banda ISM" que significa "Industrial, Scientific and Medical" que es una banda libre para usada para investigar por los tres organismos anteriores. Pero esto tiene sus consecuencias, ya que al ser libre puede ser utilizada por cualquiera y para ello, para evitar las múltiples interferencias que se pudieran introducir (microondas, WLANs, mandos, etc.) Bluetooth utiliza una técnica denominada salto de frecuencias.

El funcionamiento es ir cambiando de frecuencia y mantenerse en cada una un "slot" de tiempo para después volver a saltar a otra diferente. Es conocido que entre salto y salto el tiempo que transcurre es muy pequeño, concretamente unos 625 microsegundos con lo que al cabo de un segundo se puede haber cambiado 1600 veces de frecuencia. Cuando coinciden más de un dispositivo bluetooth en un mismo canal de transmisión se forma lo que se llaman "piconets" que son redes donde hay un maestro que es el que gestiona la comunicación de la red y establece su reloj y unos esclavos que escuchan al maestro y sincronizan su reloj con el del maestro. Dicho alcance puede variar según la potencia a la que se transmite (a mayor potencia mayor consumo y menor autonomía para el dispositivo) y el número de repetidores que haya por el medio. Así el alcance puede estar entre los 10 y 100 metros de distancia (los repetidores provocan la introducción de distorsión que puede perjudicar los datos transmitidos).

Bluetooth puede conectar muchos tipos de aparatos sin necesidad de un solo cable, aportando una mayor libertad de movimiento. Por esta razón ya se ha convertido en una norma común mundial para la conexión inalámbrica. En el futuro, es probable que sea una norma utilizada

en millones de teléfonos móviles, PC, ordenadores portátiles y varios tipos de aparatos electrónicos, como por ejemplo:

- Domótica (activación de alarmas, subida de persianas, etc.).
- Sector automovilístico (comunicación con otros vehículos).
- Medios de pago.

En una comunicación Bluetooth se pueden alcanzar tasas de transmisión de datos de 720 kbps (1 Mbps de capacidad bruta) con un rango óptimo de 10 metros (como se ha comentado antes 100 metros con repetidores).

2.3.2. Movilidad y autonomía

La movilidad de los robots representa el grado en que éstos son capaces de moverse libremente. El primer uso práctico de los robots fue en entornos industriales con los llamados robots manipuladores. Éstos tienen la tarea de fabricar productos como los automóviles. Están programados de tal manera, que puedan repetir la misma secuencia de acciones una y otra vez, más rápido, más barato y más preciso que los seres humanos. Por lo general, consisten en un brazo móvil fijado a un punto en el suelo, con la habilidad de montar o fabricar piezas diferentes. Pueden moverse con destreza en torno a este punto fijo, sin embargo, no son capaces de moverse libremente.

Si queremos que los robots puedan realizar tareas cotidianas como la limpieza del piso, prestar ayuda a personas dependientes o que puedan ser utilizados para explorar los ambientes que son difíciles o peligrosos para los seres humanos como los radiactivos o tóxicos, trabajos submarinos o misiones a los planetas y otros lugares en el espacio, tendrán que poseer un grado de movilidad mucho mayor al de los robots manipuladores. Es decir, tienen que ser capaces de moverse libremente en el entorno que le rodea en el desempeño de sus funciones.

La autonomía de los robots depende de hasta qué punto un robot se basa en el conocimiento previo o la información del entorno para lograr su cometido. Se pueden dividir en tres tipos de autonomía:

1) **No Autónomos:** Son completamente dirigidos de forma remota por seres humanos. La inteligencia desarrollada por estos robots consiste en interpretar los comandos recibidos de los controladores humanos.

2) **Semi-Autónomos:** Pueden navegar por sí mismos pero existe cierto grado de intervención del hombre. Un ejemplo de semi- autonomía se logra mediante el ajuste del entorno dónde todo está adaptado de tal manera que el robot pueda moverse con seguridad. Otra posibilidad consiste en proporcionar el mapa del entorno dónde van a desarrollar las tareas.

3) **Autónomos:** No requieren la interacción humana para cumplir con sus tareas. Los vehículos robots totalmente autónomos son capaces de desarrollar acciones y movimientos inteligentes, sin necesidad de ninguna orientación externa.

2.4. Localización odométrica

2.4.1. Introducción

El proceso más básico para la localización de un robot móvil se basa en El modelo cinemático del sistema de propulsión. La posición de un robot tipo diferencial puede ser estimada a partir de las ecuaciones geométricas que surgen de la relación entre los componentes del sistema de propulsión y de la información de los codificadores rotativos que usualmente llevan acoplados a sus ruedas. Este procedimiento es conocido como odometría.

La palabra "odometría" se compone por las palabras griegas hodos ("viajar", "trayecto") y metron ("medida"). Los robots móviles usan la odometría para estimar, y no determinar, su posición relativa a su localización inicial. Este método proporciona una buena precisión a corto plazo, es barata de implantar, y permite tasas de muestreo muy altas. Sin embargo, la idea fundamental de la odometría es la integración de información incremental del movimiento a lo largo del tiempo, lo cual conlleva una inevitable acumulación de errores. En concreto, la acumulación de éstos en la orientación, causa grandes errores en la estimación de la posición, los cuales van aumentando proporcionalmente con la distancia recorrida por el robot.

A pesar de estas limitaciones, muchos investigadores coinciden en que la odometría es una parte importante del sistema de navegación de un robot, siempre que se utilice con un sistema de corrección, que actúe de forma periódica o en determinados puntos críticos del camino.

2.4.2. Tipos de errores

Los errores asociados al sistema odométrico pueden clasificarse en errores sistemáticos y errores no sistemáticos.

- **Errores sistemáticos:** Son independientes del entorno del robot e inherentes a la cinemática del mismo, por lo que siempre están presentes. Entre los más usuales se encuentran:
    - Diámetros de las ruedas diferentes, normalmente debido al desgaste.
    - El diámetro nominal de las ruedas difiere del diámetro real.
    - Mal alineamiento de las ruedas.
    - Límites del encoder: baja resolución y/o baja frecuencia de muestreo.

- **Errores no sistemáticos:** Este tipo de error depende directamente del entorno que rodea al robot. Entre los más frecuentes podemos encontrar:
    - Desplazamiento en suelos desnivelados.
    - Desplazamiento sobre objetos inesperados que se encuentren en el suelo.
    - Deslizamiento de las ruedas debido a:

- Superficie deslizante.
- Sobre-aceleración.
- Derrapes (debidos a una rotación excesivamente rápida).
- Inexistencia de punto de contacto con la superficie.

• Fuerzas externas debidas a iteraciones con cuerpos externos.

• Fuerzas internas causadas, por ejemplo, al mal estado de las ruedas guías.

Una clara distinción entre errores sistemáticos y no sistemáticos es de gran importancia a la hora de reducir los errores en la odometría. Los sistemáticos son especialmente graves, porque se acumulan constantemente, y, al cabo de un tiempo de navegación, la información ya no es válida, de forma que habrá que corregir la posición.

El mayor inconveniente de los errores no sistemáticos es que pueden aparecer inesperadamente y pueden causar errores muy grandes en la estimación de la posición.

De todos los errores, sólo aquellos debidos a imperfecciones en el diseño mecánico y sensorial de vehículo se mantienen constantes y pueden ser eliminados en el proceso de calibrado, utilizando técnicas más complejas como la navegación inercial basada en acelerómetros para medir las variaciones de velocidad, y giroscopios para la orientación, donde la precisión obtenida es muy superior, pero también conlleva un coste superior.

En esta situación, es necesario un sistema alternativo de localización, que teniendo en que cuenta los resultados odométricos obtenidos, corrija la posición del robot y la localización goce de un grado de exactitud mayor.

2.5. Visión por computador

2.5.1. 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 imágenes 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 cómo funciona el sistema visual humano, y sin embargo sigue sin conocerse el funcionamiento completo.

2.5.2. 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. [7]

A continuación se introducen brevemente algunos de los campos más importantes en los que se desarrolla la visión por computador.

- **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, como por ejemplo el robot de la Figura 2.24.

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.

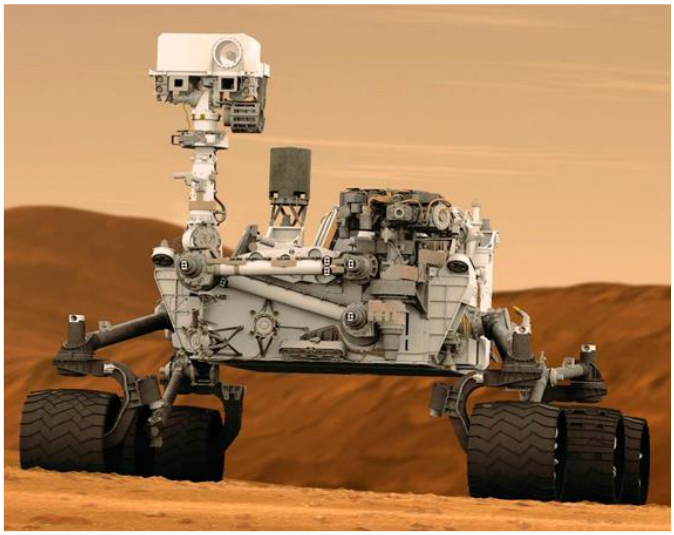


Figura 2.24: Curiosity, el vehículo autónomo explorador de la NASA diseñado para recorrer Marte recogiendo muestras para analizar la capacidad del planeta para albergar vida.

2.5.3. 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.

- **Procesamiento de imágenes**

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

Figura 2.25: Imagen resultante de aplicar un algoritmo de blending

- **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 2.26: Detección de características

- **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, todavía 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.

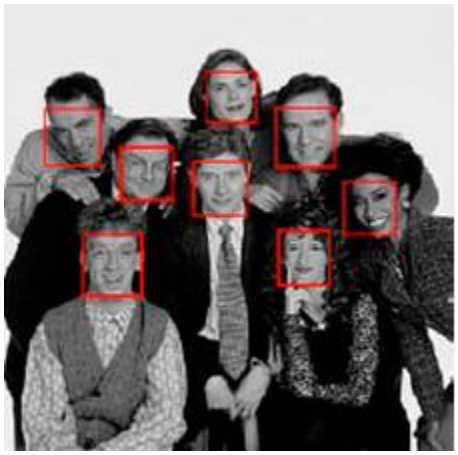


Figura 2.27: Reconocimiento facial.

2.5.4. 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.

- **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 2.28: Detección de personas en cámara de video vigilancia. La imagen superior derecha es la imagen de referencia (background).

- **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 más 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.

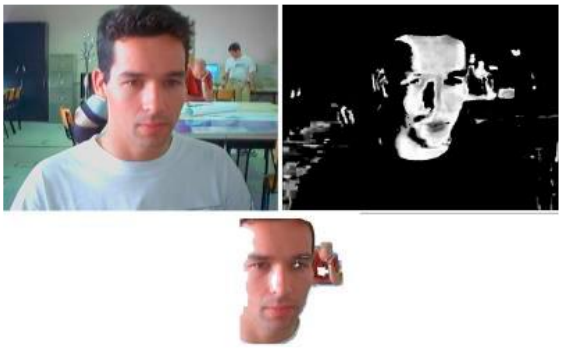


Figura 2.29: Segmentación por el color de piel en una escena con background complejo.

2.5.5. 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.

- **Lucas-Kanade**

El método de Lucas-Kanade (1981) 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, aplicación que es muy importante sobre todo en sistemas de vigilancia.

Figura 2.30: Tracking de vehículos mediante el algoritmo de Lucas-Kanade

- **Kalman Filter**

Se presenta el Kalman filter (1960) [5] como un método eficaz para el tracking de objetos. Se demuestra que mediante esta técnica se puede predecir cuál 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.

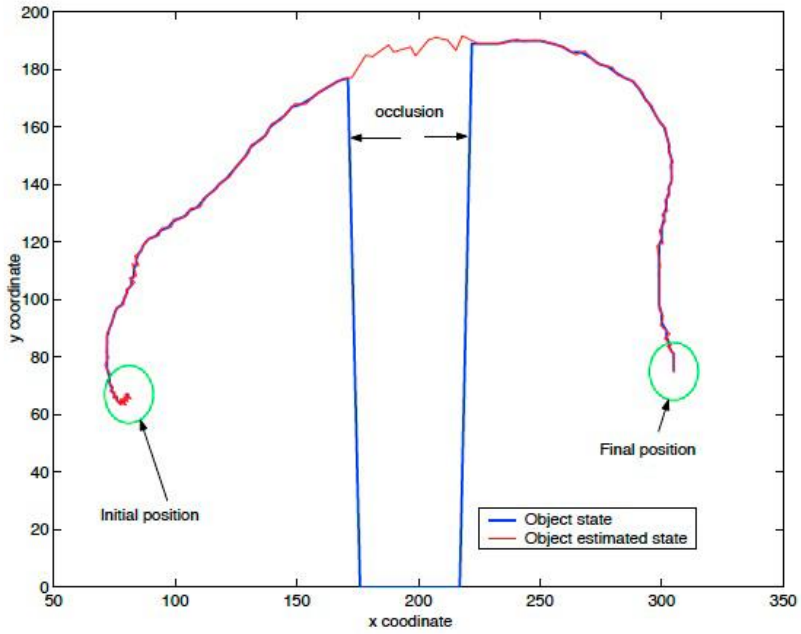


Figura 2.31: Predicción de la posición del objeto con un periodo de oclusión mediante Kalman filter

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.

2.6. Lego Mindstorm NTX 2.0

Imagina poder construir un robot completo, con sensores, motores, engranajes, reductoras, estructuras, poder programarlo y configurarlo, y todo sin soldar, taladrar, pegar o taladrar tornillos. Pues eso es LEGO-Mindstorms, una forma fácil y sencilla de aprender robótica y construir tu propio robot.[11]

Lego Mindstorms es una plataforma para el diseño y desarrollo de robots, que sigue la filosofía de la marca LEGO, armar y construir todo tipo de objetos simplemente uniendo bloques interconectables.

El bloque central es un microcontrolador, con forma de LEGO (de ahora en adelante brick). La conexión de sensores y actuadores es muy sencilla, por simple presión en cualquiera de las puertas y en cualquier posición. Las piezas de Lego tienen múltiples formas y tamaños, lo que permite construir diversas estructuras, usando los bloques. Mediante un PC, se realiza la programación del brick, usando diferentes programas y lenguajes.

Figura 2.32: Imagen del ladrillo inteligente con los sensores y motores conectados

¿POR QUÉ USAR LEGO MINDSTORMS EN ESTE PROYECTO?

En este apartado se indicarán las principales ventajas y desventajas de utilizar LEGO Mindstorms:

Ventajas

- Fácil de montar y desmontar, no es necesario usar soldadura, ni tornillos. Todo lo que se arma se puede desarmar rápidamente. Además, eso permite usar las piezas en múltiples diseños.
- Muy extendido por todo el mundo, lo que permite encontrar gran cantidad de información e ideas por Internet, diseños, soluciones, participar en foros, competiciones…

- No es un pack cerrado, es decir, se puede comprar más ampliaciones de LEGO, adquirir piezas deterioradas o perdidas, o añadir piezas echas manualmente, como por ejemplo, sensores o motores, e incluso circuitos neumáticos.
- Múltiples posibilidades y lenguajes de programación, desde el nivel más básico e intuitivo, hasta el uso de lenguajes conocidos como C o Java, e incluso la utilización de Linux…
- Que sea escalable, es decir, que a partir de un material básico haya opciones de ampliación.
- Muy indicado para entornos educativos, desde colegios a universidades, pues se puede aprender de forma fácil tanto mecánica como electrónica.
- Precio. Obviamente, comprar un robot "prefabricado", resulta más caro que construirte tu propio robot.

Desventajas

- La principal desventaja de LEGO es su estructura. Está formada por bloques de LEGO, que se unen por simple presión. Cierto que se pueden añadir elementos de refuerzo y sujeción, pero para diseños exigentes, no es recomendable. Golpes, caídas, pueden debilitar rápidamente la estructura, llegando a desarmar el robot.
- No se pueden construir estructuras circulares, pues todas las piezas y ladridos de LEGO son rectangulares.
- Relación masa-volumen. Las piezas LEGO no son útiles en diseños donde la relación masa-volumen se hace crítica. Por ejemplo, para construir un robot de SUMO, no sería eficiente, pues la estructura LEGO es demasiado liviana, y se deberían añadir pesos para hacer el robot más robusto, o el caso contrario, para construir robot pequeños, ligeros, y resistentes, las piezas LEGO son mucho peores que los materiales cómo la fibra de carbono.

2.6.1. Hardware

- **Brick NXT**

El brick es el elemento fundamental del LEGO Mindstorms NXT. Incluye la CPU, el sistema de comunicaciones, los puertos de entrada y salida y la interfaz de usuario, todo en un bloque de plástico sólido de pequeños tamaño y peso como el que aparece en la imagen siguiente.

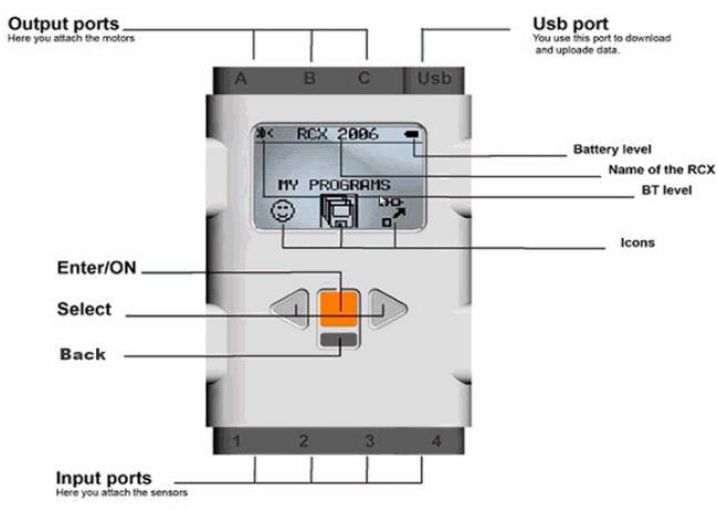


Figura 2.33: Esquema brick Lego NXT

Las especificaciones técnicas de la unidad de control son las siguientes:

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 2.34: El NXT, el cerebro del robot Mindstorms

En la siguiente figura se puede observar la arquitectura interna de la unidad de control:

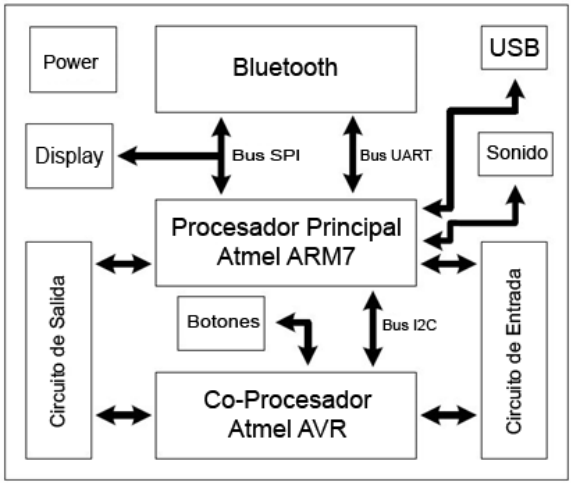


Figura 2.35: Esquema detalle bloque NXT

- **Actuadores**

En el estudio del uso del NXT parte de la premisa inicial de que los únicos actuadores disponibles son motores de corriente continua. Los que se suministran al comprar el NXT son tres motores de corriente continua y funcionamiento PWM. A pesar de su reducido tamaño, esconden en su interior un complejo sistema de reducción por tren de engranajes y un sensor de rotación de tipo tacométrico. Como se aprecia en la Figura 2.36, disponen de diversos puntos de unión con piezas de LEGO.

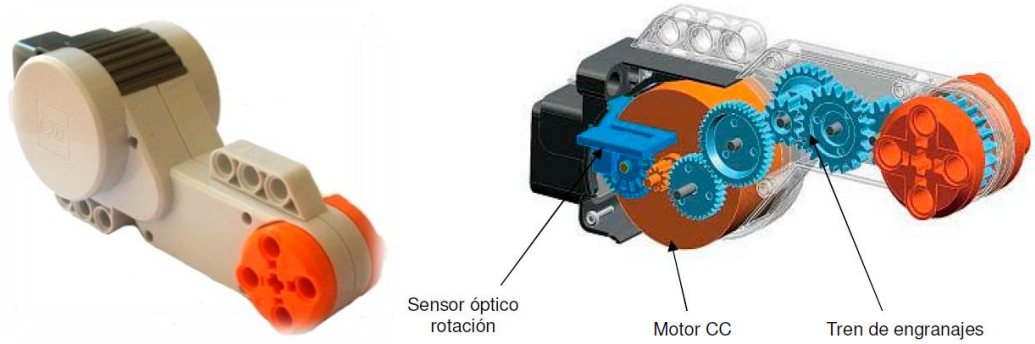


Figura 2.36: Imagen interna del motor, donde se aprecia el complicado tren de engranajes y el tacómetro

El motor alcanza sin carga una velocidad de 160 rpm, y hasta 120 rpm con una carga de 11.5 N·cm, alimentado a 9 V. Es capaz de entregar un par máximo de 25 N·cm a 60 rpm, aunque no es recomendable exceder de los 15 N·cm en periodos prolongados (puede producir fatiga al actuador).

- **Sensores**

El robot Lego puede utilizar dos tipos distintos de sensores, tanto los originales de Lego como los que han sido construidos por terceros como Hitechnic, Mindsensor y Vernier. En este proyecto nos centraremos en los originales y los de la marca Hitechnic.

1) Sensores Originales Lego:

- <u>El sensor óptico:</u> Incluye un fototransistor para la medida de iluminación, junto a un diodo LED que puede activarse o no, dependiendo de si la medición se está efectuando en un ambiente oscuro o iluminado. El rango de medida es más amplio que el espectro visible por el ojo humano, lo que puede inducir a confusiones con fuentes de luz que no se consideren en un principio, como pilotos infrarrojos. El pico de sensibilidad se obtiene para las longitudes de onda de 900 nm, mientras que en el espectro de los 400 a los 700 oscila entre un 30 y un 60% de sensibilidad. El margen de medición comprende desde los 0 hasta los 1000 lux, con una curva de respuesta bastante lineal.

Figura 2.37: sensor óptico

- El sensor acústico: Puede configurarse para devolver los valores de medida en decibelios (dB) o decibelios ajustados (dBA), estando esta última unidad de medida ponderada a los niveles audibles por el oído humano. La sensibilidad máxima se encuentra en los 90 dB (aproximadamente el mismo nivel sonoro que una calle ruidosa con mucho trabajo). La respuesta a intensidades en decibelios crecientes es aproximadamente exponencial.

Figura 2.38: sensor sonido

- El sensor de distancia: Puede por sí mismo merecer un capítulo aparte en esta documentación. Su complejo funcionamiento le hace requerir su propio microprocesador para la interpretación de la señal, y dispone de un protocolo de comunicaciones desarrollado en exclusiva, por supuesto documentado por LEGO en su web. Al contrario que el resto de sensores, no devuelve los valores en ninguna escala ni porcentaje, sino en unidades reales, bien centímetros, bien pulgadas. El alcance de medición comprende desde 3 cm hasta 255 cm (tiene una zona muerta en distancias muy cortas), y su funcionamiento por ultrasonidos a 40 KHz lo hace altamente efectivo para escenas relativamente simples y con objetos bien definidos, mientras que si aparecen elementos pequeños o en grandes cantidades, sus mediciones pueden no ser tan buenas.

Figura 2.39: sensor ultrasonidos

- Encoders: Los codificadores ópticos o encoders incrementales, constan en su forma más simple de un disco transparente con una serie de marcas opacas colocadas radialmente y equidistantes entre sí, de un sistema de iluminación en el que la luz es colimada de forma adecuada, y de un elemento foto receptor. El eje cuya posición se quiere medir va acoplado al disco transparente. Con esta disposición, a medida que el eje gire se irán generando pulsos en el receptor cada vez que la luz atraviese cada marca, y llevando una cuenta de estos pulsos es posible conocer la posición del eje y la velocidad de rotación. En la figura 2.40 se muestra el esquema funcional de un encoder.

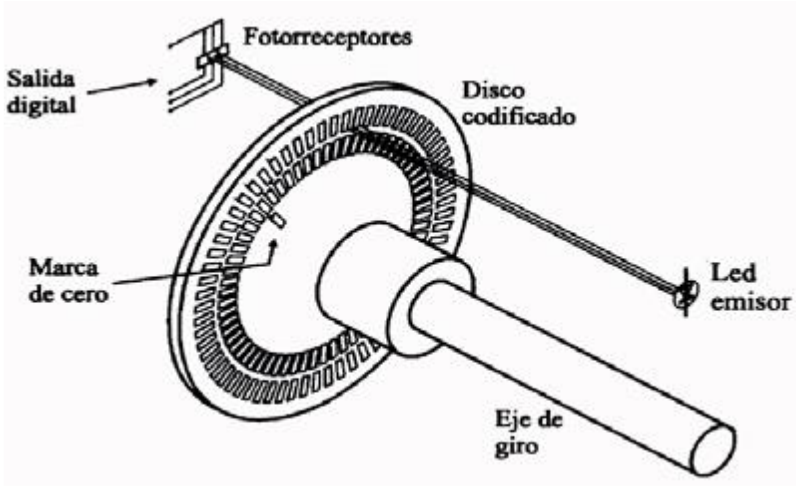


Figura 2.40: Esquema de un encoder

Por otro lado las principales limitaciones con las que cuenta son:

- Siempre es necesario un circuito contador para obtener una salida digital compatible con el puerto de entrada/salida de un microcontrolador. Otra posible forma de hacerlo se basaría en software especial según sea la aplicación específica, como por ejemplo, alguna interrupción o programación de alta velocidad, tiempo real, para obtener el tiempo de cambio entre un sector y otro.

- Los encoders pueden presentar problemas mecánicos debido a la gran precisión que se debe tener en su fabricación. La contaminación ambiental puede ser una fuente de interferencias en la transmisión óptica. Son dispositivos particularmente sensibles a golpes y vibraciones, estando su margen de temperatura de trabajo limitado por la presencia de componentes electrónicos.

En la Figura 2.41 se ve la posición del encoder en el motor:

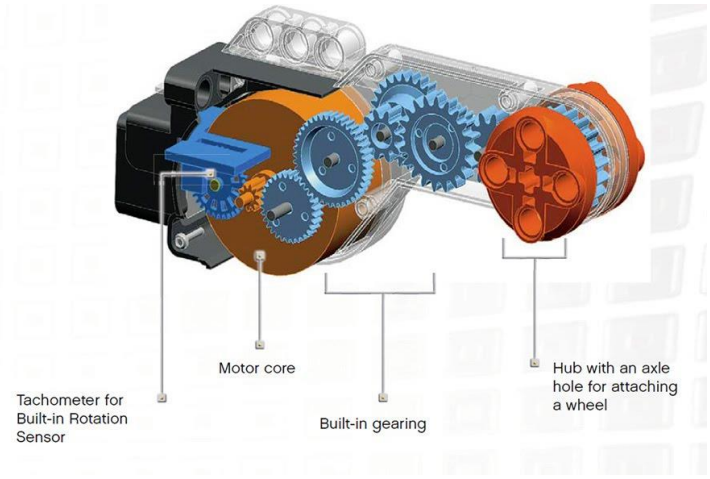


Figura 2.41: Encoder de Lego NXT

2) Sensores HiTechnic: [10]

- La brújula: Hay que tratarla como un sensor LowSpeed (se comunica mediante el protocolo I2C). Al trabajar con ella, devuelve directamente un entero de 0 a 360; el valor en grados de diferencia al norte magnético. Según las instrucciones del propio fabricante, es conveniente montarlo alejado de los motores y el ladrillo inteligente (al menos 10 cm de este último y 15 de aquellos), para evitar interferencias, y mantenerlo sujeto, firme y horizontalmente, esas recomendaciones pueden quedarse un poco escasas.

Figura 2.42: sensor brújula

- Acelerómetro: Este sensor incorpora un acelerómetro de tres ejes, distribuidos como se muestra en la Figura 2.43, de considerable precisión: las lecturas, actualizadas cada 10 milisegundos, devuelven valores entre -400 y + 400, equivalentes aproximadamente a -2g y +2g. Opera también mediante el protocolo I2C.

  La sensibilidad es muy grande comparada con el rango de actuación en que se suele mover el resto de sensores del NXT, lo que, para aplicaciones de precisión es muy útil, pero, sin embargo, puede llevar a problemas de exceso de precisión en aplicaciones más limitadas, que interpreten pequeñas variaciones que no se correspondan con cambios reales. Puede hacerse necesario el uso de un factor de escala que desprecie los cambios más pequeños.

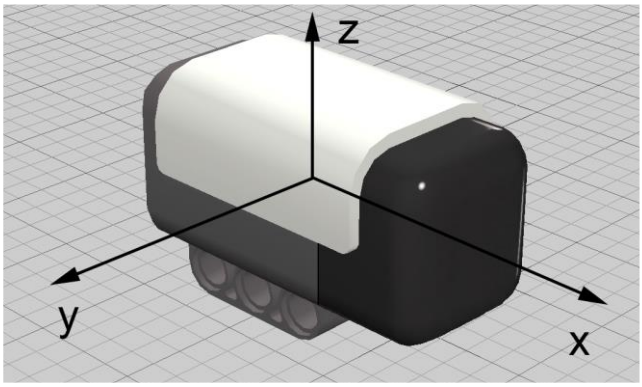


Figura 2.43: sensor acelerómetro

- Giroscópio: Este sensor devuelve el número de grados por segundo y la dirección de rotación. Permite controlar funciones en las que la medida de la rotación sea esencial.
  Contiene un sensor giroscópico de un eje que detecta la rotación y devuelve un valor que representa el número de grados por segundo de rotación [ω]. La velocidad de rotación puede ser leída hasta unas 300 veces por segundo. El eje de medida está en el plano vertical con la parte oscura del sensor mirando hacia arriba. (Mismo encapsulado que la brújula, Fig. 2.44)

Figura 2.44: Encapsulado del sensor para el NXT

- Sensor de ángulo: Mide la posición y la velocidad del eje de rotación. El sensor del ángulo permite medir 3 propiedades de rotación:
  - *Ángulo absoluto*: La posición de rotación de un eje de rotación 0-359 grados con una precisión de 1 grado
  - *Ángulo acumulado*: El número acumulado de grados de un eje ha girado
  - *Velocidad de rotación*: La velocidad de la rotación del eje en RPM (revoluciones por minuto)

Con su mecanismo de muy baja fricción, el sensor de ángulo HiTechnic es ideal para la construcción de modelos donde la medición exacta de propiedades de rotación de un eje es clave.

Figura 2.45: Sensor de ángulo

- **Alimentación**

La necesidad de una batería recargable nace inevitablemente dado el alto gasto en pilas (6 unidades AA) del NXT, aunque el consumo en si no sea desmesurado. La batería, comercializada por la propia LEGO, se compone de polímeros de ion de litio, y proporciona 1400 mAh a 7.4 V, una cantidad ligeramente inferior a la que teóricamente se obtendrá con seis pilas de 1.5 V (9 V), pero bastante aproximada a la que en realidad proporcionan dichas pilas.

La batería reemplaza la tapadera trasera del ladrillo y sobresale por debajo de este la anchura de una barra agujereada de LEGO, circunstancia a tener en cuenta a la hora del diseño, como se muestra en la Figura 2.46. El rendimiento en las pruebas es muy bueno, y en ningún momento da muestras de insuficiente voltaje o potencia. La duración es aproximadamente la misma que la de unas pilas alcalinas de buena calidad.

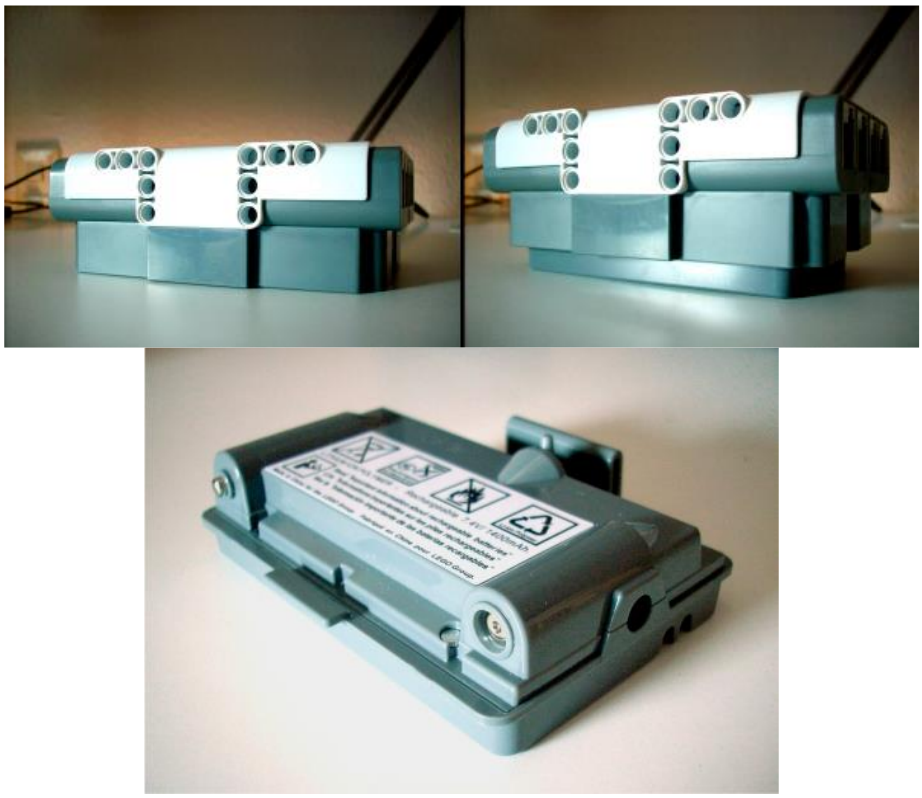


Figura 2.46: batería NXT

2.6.2. Software

La presentación comercial de Lego Mindstorms NXT incluye un software de desarrollo de tipo gráfico, basado en NI Labview, relativamente sencillo de utilizar y funcional en Windows y Mac. Sin embargo, dado que se busca una mayor profundidad al programar y un acceso a nivel más bajo, se hace necesaria la existencia de un lenguaje de programación por entrada de texto, más funcional y orientado a un uso más serio que el propuesto por LEGO (que, no obstante, es bastante completo y útil para los primeros pasos en el mundo de la programación).

- **Lenguajes y entornos integrados de desarrollo (IDEs)**

- NXT-G: Programación en modo gráfico, basada en iconos. Se trata de un software oficial de LEGO, compatible con Windows y Mac pero sin soporte para Linux. Basado en LabVIEW. Permite conexiones bluetooth y por puerto serie. Tiene algunas limitaciones en cuanto a las posibilidades de programación y en ocasiones resulta algo engorroso, sobre todo cuando se emplean muchos iconos para programas más complejos. Utiliza el firmware habitual. Permite hacer programas relativamente rápido sin tener conocimientos de programación. En contrapartida son más lentos y ocupan mayor cantidad de memoria.

  Hay 2 versiones:

  - *Educativa:* La versión 2.0 tiene Data Logging, permite monitorizar en tiempo real el estado del robot y/o sus sensores. Incluye actividades para el aprendizaje del diseño y programación de robots NXTMindstorms.
  - *Comercial:* No tiene Data Logging. Incluye instrucciones y guía de programación para los modelos básicos que se muestran en la caja (la versión 1.0 y 2.0 tienen distintos modelos).

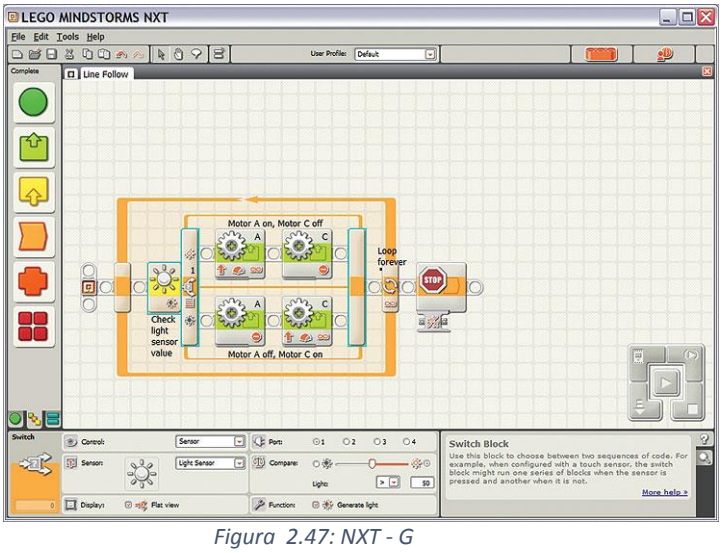


Figura 2.47: NXT - G

- RobotC: Programación en modo texto, también es un software oficial de LEGO, hasta el momento sólo con soporte para Windows. Basado en AnsiC. Permite conexiones por bluetooth y puerto serie. Las posibilidades crecen enormemente y prácticamente ya no existen más limitaciones que la propia imaginación. Los programas en modo texto ocupan mucha menos memoria que los gráficos y eso siempre es un alivio debido a las limitaciones de memoria del ladrillo NXT (aunque en la versión 2.0 la capacidad se ha duplicado). Incluye programas de ejemplo. Los programas son livianos y de rápida ejecución, sin embargo son necesarios conocimientos de programación.

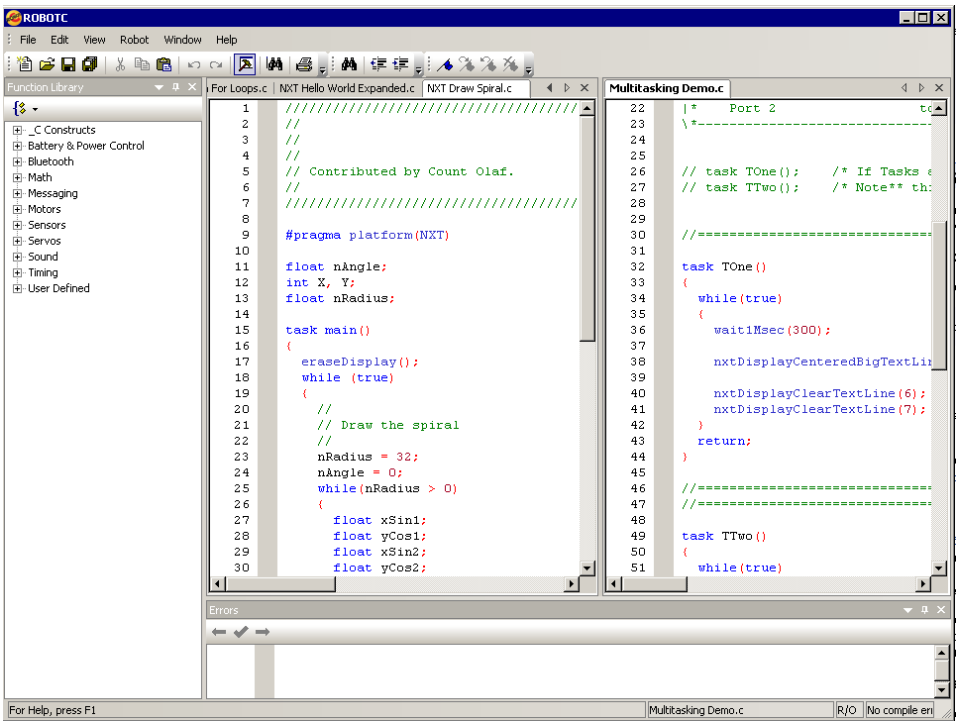


Figura 2.48: RobotC

- NXC: Programación en modo texto, se trata de un software libre desarrollado por la comunidad. Compatible con Windows, Mac y Linux descargando el compilador adecuado para cada sistema. Suele venir de la mano con NBC (ensamblador) y tiene ciertas similitudes con C. Tiene soporte para bluetooth y puerto serie.
  Importante: Los programas de RobotC y NXC no son compatibles entre sí, y no se pueden portar de una plataforma a otra, ya que hacen uso de librerías y funciones distintas.
  Hace uso del firmware estándar. Los programas son también muy ligeros y de rápida ejecución, sin embargo son necesarios conocimientos de programación.

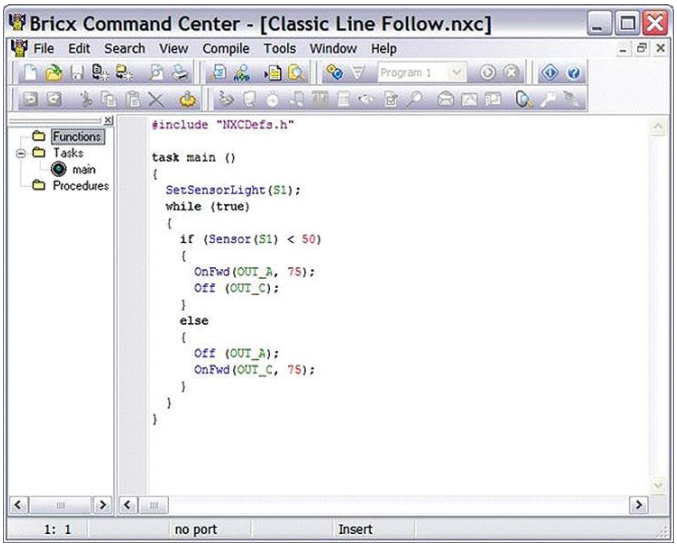


Figura 2.49: NXC

- LeJOS: Es un proyecto que persigue convertirse en una alternativa a NXT-G, proporcionando funciones basadas en Java. Como alternativa a NXT-G es capaz de ofrecernos todas las funciones existentes en NXT-G y añadirle numerosas funciones más. Al ser de desarrollo libre cualquier persona puede aportar sus conocimientos y ayudar a mejorar el proyecto. Por ello es uno de los firmwares más elegidos para primeros desarrollos y acercamientos a la plataforma Lego NXT 2.0. [12]

  Nos permite ejercer un control y seguimiento casi total de los componentes hardware que asociamos al Lego NXT 2.0, proporcionándonos funciones que nos dan la posibilidad de controlarlos al 100%. Así mismo al disponer de acceso a todo el código de la librería de LeJOS podemos realizar cambias para adaptar las funciones a nuestras necesidades específicas.

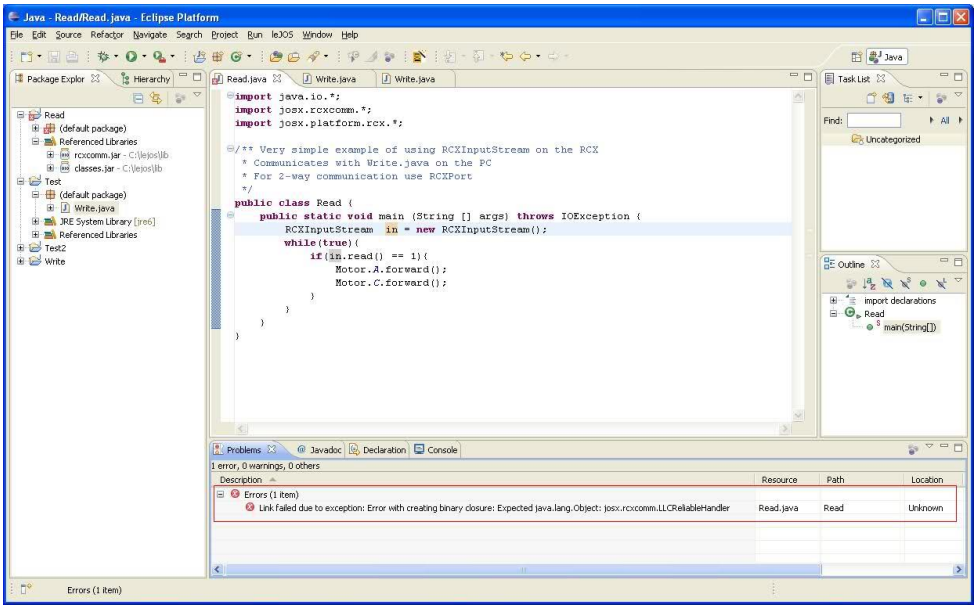


Figura 2.50: LeJOS en Eclipse

- PbLua: Programación en modo texto, basada en Lua, que es un lenguaje de programación basado en C portable, fácil de leer y de escribir que se compila directamente desde el ladrillo. Software libre. Es necesario un firmware personalizado. Se requieren conocimientos de programación y son programas que ocupan muy poca memoria y de rápida ejecución.

- LabVIEW: Es un entorno de programación gráfica empleado por millones de ingenieros y científicos para el desarrollo de sofisticados sistemas de control, medidas, pruebas mediante el uso de intuitivos iconos gráficos conectados por hilos, que recuerdan una tabla de flujo. LabVIEW permite la integración con una infinidad de dispositivos hardware y proporciona librerías que permiten el análisis avanzado y visualización de datos. La plataforma LabVIEW es escalable, trabaja en varios sistemas operativos, y desde su introducción en 1986 se ha convertido en líder industrial. LabVIEW dispone de una página de recursos relacionados con LEGO MINDSTORMS NXT, entre los que podemos encontrar el *Toolkit for LEGO MINDSTORMS NXT*.

- Matlab: La toolbox RWTH-Mindstorms NXT para MATLAB, ha sido desarrollada para el control del robot Lego Mindstorms NXT usando MATLAB a través de una conexión inalámbrica Bluetooth o vía USB. Este software es un producto gratuito de código abierto y está sujeto a la GNU General Public License (GPL). El desarrollo de esta aplicación, fue motivada por un proyecto académico de la Universidad RWTH de Aachen; en consecuencia, su diseño está orientado principalmente para fines educativos.
  Las funciones de la toolbox se basan en el protocolo de comunicación LEGO Mindstorms NXT Bluetooth para controlar el ladrillo inteligente NXT a través de una conexión inalámbrica Bluetooth o vía USB.

La lista de lenguajes que se ha expuesto es algo más extensa pero no era la finalidad de este apartado comentar todas las posibilidades. Sin embargo se va a proceder a agrupar todas las características comunes a todos los lenguajes para hacer una comparativa mucho más visual y rápida. En las siguientes tablas no se recogen todas las características que tiene cada uno de los lenguajes pero puede ser suficiente para hacerse una idea sobre que lenguaje que mejor se adapte a sus necesidades.

Características	NXT-G Retail	NXT-G Educational	RoboLab 2.9	NBC	NXC	Robot C	NI LabVIEW Toolkit	leJOS NXJ	pbLua	LEJOS OSEK
Tipo de lenguaje	Gráfico	Gráfico	Gráfico	Ensamblador	Como C	C	Gráfico	Java	Lua	ANSI C
Firmware	Estándar	Standard	Estándar(#1)	Estándar	Estándar	Estándar (#1)	Estándar	Modificado	Modificado	Modificado
IDE (¿incluido?)	Si	Si	Si	Si	Si	Si	No (#6)	plugins para Eclipse y NetBeans	No (#7)	Eclipse CDT(GCC+ ATMEL SAM-BA)
Windows	Si	Si	Si	Si	Si	Si	Si	Si	Si (#7)	Si
Mac OSX	Si	Si	Si	Si	Si	Aún no	Si	Si	Si (#7)	No
Linux	No	No	No	Si	Si	No	No	Si	Si (#7)	No (puede)
Eventos	No	No	Si	No	No	Si	No	Eventos estándar de java		Si (OSEK RTOS)
Multitarea	Si	Si	Si	Si	Si	Si	Si	Si		Si (OSEK RTOS)
Bluetooth Brick hacia PC	Si	Si	No	Si	Si	Si	Si	Si	Si	Si
Bluetooth Brick hacia Brick	Si	Si	No	Si	Si	Si	Si	Si	Si	Aún no
Bluetooth Brick hacia otro dispositivo	No	No	No	No	No	Si	No	Si	Si	No
I2C Support	(#5)	(#5)	Si	Si	Si	Si	Si	Si	Si	Si (EEUU sólo)
File System	Si	Si	Si	Si	Si	Si	Si	Si	Aún no	Sin planear
Floating Point	No	No	Si	No	No	Si	¿No?	Si	(#8)	Si
Datalog	No	No	Si	No	No	Si	¿No?	Si	No	Aún no

Figura 2.52: Comparativa de diferentes lenguajes de desarrollo para NXT

Software	Tipo de Lenguaje	Tipo de Control	Requiere Firmware del NXT	Tipo de Conexión	Fuente de Conexión	Windows	Mac OSX	Linux	Lectura de Sensores	Websites
LEGO NXT Mobile Application	Simple RC	Control Remoto	Estándar (#2)	Bluetooth	Teléfono o PDA	-	-	-	No	LEGO
FunkNXT	Simple RC	Control Remoto	Estándar	Bluetooth	Teléfono	-	-	-	Si	FunkNXT
BT RC	NXT-G	NXT a NXT remotamente	Programa ejecutándose en el NXT	Bluetooth	Otro NXT	-	-	-	Programable por el usuario	BTRC
Simple BT Remote	Simple RC	Control Remoto	Estándar	Bluetooth	Escritorio	Si	No	No	Si	Simple Windows RC
RobotC	Simple RC	Control Remoto	Estándar (#1)	USB/BT	Escritorio	Si	Aún no	No	Si	Robot C Web Site
BricxCC	Simple RC	Control Remoto	Estándar	USB/BT	Escritorio	Si		No	Si	BricxCC Web Site
OnBrick PDA	Gráfico	RC Programable	Estándar	Bluetooth	PDA	-	-	-	Si	OnBrick
OnBrick PC	Gráfico	RC Programable	Estándar	Bluetooth	Escritorio	Si	No	No	Si	OnBrick
NXT Director	Simple RC	Control Remoto Modificable	Estándar	Bluetooth	Palm PDA	-	-	-	¿No?	Director
RoboDNA	Simple RC	Control Remoto	Estándar	Bluetooth	Escritorio	Si			Si	RoboDNA
MS Robotics Studio	.NET	Programa de usuario ejecutándose en el PC	Estándar	Bluetooth	Escritorio	Si			Si	(version no commercial gratis) Download site o Microsoft Site
NI LabVIEW Toolkit	Gráfico (LabVIEW G)	Programa de usuario ejecutándose en el PC	Estándar	USB/BT	Escritorio	Si	Si		Si	LabVIEW toolkit Site
RoboLab	Gráfico	Programa de usuario ejecutándose en el PC	Estándar	USB	Escritorio	Si	Si		Si	Robolab
iCommand	Java	Programa de usuario ejecutándose en el PC	Estándar	Bluetooth	Escritorio o PDA	Si		Si	Si	iCommand
LEGO::NXT	Perl	Programa de usuario ejecutándose en el PC	Estándar	USB/BT	Escritorio	Si	Si	Si	Si	Perl
nxt-Ruby	Ruby	Programa de usuario ejecutándose en el PC	Estándar	Bluetooth	Escritorio	Si	Si	Si	Si	Ruby
NXT#	C#	Programa de usuario ejecutándose en el PC	Estándar	Bluetooth	Escritorio	Si			¿Si?	NXT#
Mindsqualls	C#	Programa de usuario ejecutándose en el PC	Estándar	Bluetooth	Escritorio	Si			Si	Mindsqualls
NXT Python	Python	Programa de usuario ejecutándose en el PC	Estándar	USB/BT	Escritorio	Si			¿Si?	Python
My Robot Me	¿Gráfico?	Programa de usuario ejecutándose en el PC	Estándar	USB/BT	Escritorio	Si	No	No	Si	Robot Me

Notas

(1) RobotC usa firmware estándar que proviene del software de LEGO.
(2) Aplicaciones de LEGO para móviles pueden enviar mensajes a los programas que están ejecución en un NXT.

Figura 2.53: Comparativa de las distintas funcionalidades.

- **LEGO Digital Designer**

El LEGO Digital Designer (o LDD), es un programa de CAD tridimensional para diseñar y reproducir montajes reales de LEGO. Una vasta paleta de piezas, que cubre desde los clásicos LEGO City hasta los Mindstorms NXT, permite construir en un entorno 3D como si de la realidad se tratase. Si bien existe una gama de, o bien programas de diseño de LEGO, o bien sets de piezas para usar en programas de diseño tradicionales, como LeoCAD o LDraw, este es el más completo y actualizado.

Entre sus opciones más útiles, está el calcular, en base a los precios de la tienda oficial LEGO, el coste de cada diseño; el comenzar con una determinada colección de piezas y trabajar con ellas; y, sobre todo, la generación automática de manuales de instrucciones. A partir del diseño final, y en base a unos algoritmos con parámetros ajustables (tales como número máximo de piezas por paso), el LDD crea rápidamente una guía de construcción paso a paso, lista para imprimir y seguir. El funcionamiento es bastante intuitivo, aunque puede dar algún problema de piezas que no lleguen a conectar justo como se desea al estar en una posición difícil, o de conexiones que en la realidad sean posibles por tolerancias de fabricación o fuerce de piezas, pero que en el entorno de LDD no puedan tener lugar. El software, funcional en Windows y Mac, se puede descargar gratuitamente de la web de LEGO.

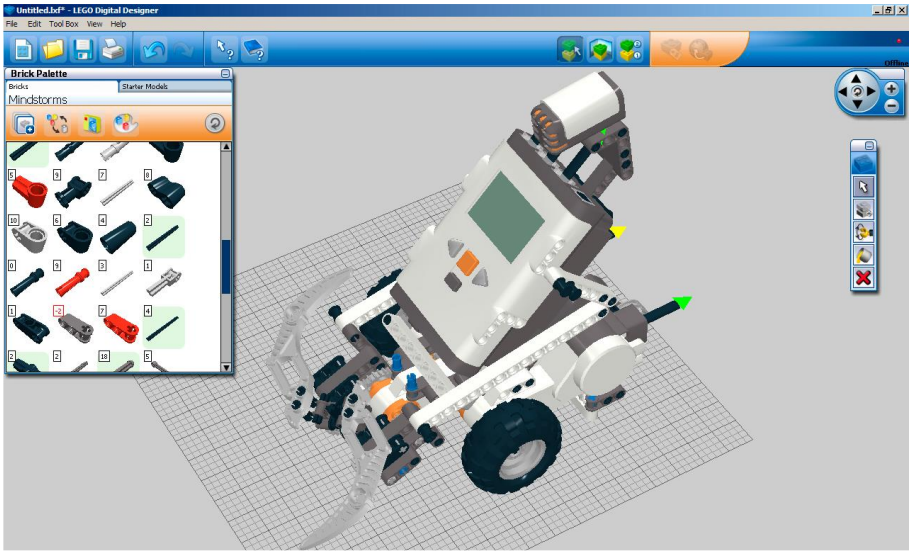


Figura 2.54: Lego Digital Designer

2.7. Proyectos relacionados

2.7.1. Introducción

Lego NXT 2.0 o Lego Mindstorms que viene a ser lo mismo, es uno de los kits robóticos más empleados para el desarrollo de prototipos robóticos. Tan solo con teclear Lego NXT en YouTube veremos que aparecen miles de video, si hacemos lo mismo en Google veremos miles de resultados de páginas sobre Lego NXT 2.0 y blogs.

Lego NXT ha logrado una amplia comunidad de desarrolladores y colaboradores, que le dan un valor añadido al producto. Es fácil encontrar información o código de proyectos desarrollados por terceros. Este tipo de información es de vital importancia para no volver a desarrollar código ya existente y accesible de forma gratuita. Por otro lado es interesante ver como otras personas han planteado el mismo proyecto y ver si partiendo de sus ideas se puedes optimizar el proceso. En general existe una gran cantidad de proyectos relacionados con los robots basados en una estructura de 4 ruedas. Esto hace que podamos buscar modelos anteriores para ver cómo han funcionado y para qué entornos pueden sernos útiles. Una vez que entramos en el mundo de la navegación autónoma vemos que existen pocos proyectos que han afrontado las dificultades que presenta este ámbito. La navegación autónoma presenta muchas dificultades complejas frente a otros proyectos. Esto se debe principalmente a que los sensores que se emplean para realizar estas tareas no son fiables al 100% y por ello hay que tener mecanismos de control para estos errores. Lego NXT es una de las pocas herramientas accesibles para programadores con un presupuesto bajo. Gracias a esta tecnología se han podido desarrollar proyectos como de los que hablamos más abajo. Estos proyectos han abarcado el problema de la navegación autónoma usando Mindstorms como la piedra angular.

Es interesante por tanto examinar estos proyectos y el enfoque que le han dado, tanto al problema como al uso el NXT 2.0. Ya que toda esta información nos va facilitar el trabajo y va a permitir que trabajemos más rápido.

2.7.2. Robot Magellan

Robo-magellan es una competición de robots enfocada a la navegación autónoma. Se trata de una iniciativa de Seattle robot society por crear un evento en el que se miden robots de bajo presupuesto.

El concurso busca distintas ideas de como abarcar la navegación autónoma, con medios reducidos. El objetivo es partir desde un punto A y llegar a un punto de destino B, intentando llegar a puntos intermedios que dan bonificaciones. Al comienzo del concurso se informan de las coordenadas y se dan 30 minutos para ultimar detalles de software y hardware. Los robots presentados en este concurso son innovadores y casi en todos los casos creados desde cero. Es muy interesante ver como distintas ideas compiten entre ellas para ver cuál de ellas es más eficiente. Un ejemplo de robot autónomo se puede ver en la Figura 2.55.

Figura 2.55: Robot autónomo participante

2.7.3. Proyecto Green Master

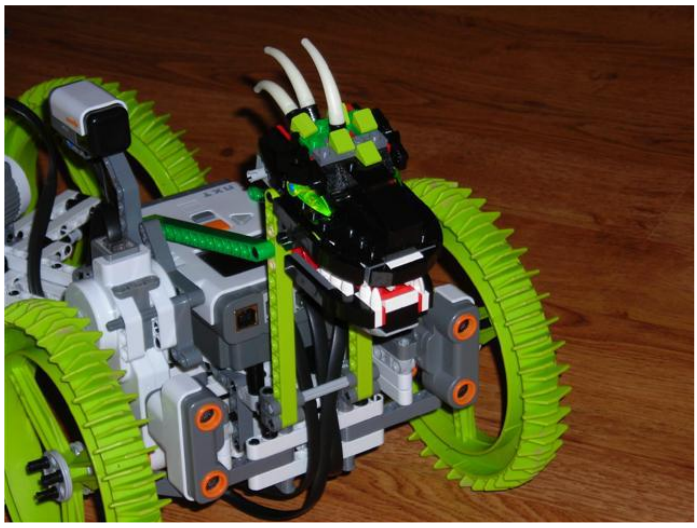


Figura 2.56: Imagen del proyecto Green Monster.

Es un proyecto de "Team Haseenplug" desarrollado en pbLua con el Lego NXT 2.0, que aborda el problema de la navegación autónoma. En su primera versión se dirigía exclusivamente mediante la brújula, buscando que el robot siguiera una coordenada en todo momento y fuese capaz de atravesar obstáculos. Para ello usan 4 ruedas de gran diámetro que le permiten moverse por el césped. En versiones posteriores se empezó a usar la tecnología GPS para empezar a dirigir el robot. En esta fase experimento con puntos fijos por los que debe pasar el robot.

Ha sido uno de los primeros proyectos en intentar combinar las distintas tecnologías (GPS, Ultrasonido, brújula y Bluetooth) y en su versión final ha sido un gran existo. Básicamente este proyecto abarca gran parte de los problemas que deberemos abarcar nosotros también. Es por un lado una guía y por otro una demostración de que el proyecto que estamos llevando a cabo es viable.

2.7.4. Proyecto RC Car

Figura 2.57: Foto propiedad de Juan Antonio Breña. RC CAR

Es la fusión entre el brick de Lego un coche eléctrico tradicional. Es un proyecto desarrollado por Juan Antonio Breña en el que se persigue la unión entre los dos componentes. En versiones posteriores se ha logrado la unión entre el brick de Lego y el coche eléctrico añadiéndoles sensores complejos como el GPS.

Este proyecto por lo tanto abarca el desafío de la navegación autónoma desde un punto de vista más avanzado. Haciendo uso de tecnología más complejas como es "Lattebox NXTe". Esto último es lo que permite el manejo de motores externos a Lego de forma fácil y eficiente. Al igual que el proyecto anterior busca la integración entre distintas tecnologías y demuestra la gran capacidad del brick de Lego.

2.7.5. GPS Cavedu

Cavedu es una iniciativa Taiwanesa de educación sobre robótica. En su página web se pueden ver numerosos tutoriales, libros o proyectos desarrollados con Lego Mindstorms. Todo este material se encuentra en taiwanés. Es difícil hablar en profundidad de esta iniciativa, ya que casi toda la información disponible sobre la misma solo está en taiwanés, dificultando la tarea de seguir de cerca su iniciativa. En su canal de YouTube se pueden encontrar numerosos videos sobre robots programados con Lego NXT 2.0.

2.8. Tecnologías empleadas

2.8.1. Introducción

Al comenzar el proyecto encontramos la necesidad de decidir cuáles eran las tecnologías que iban a hacer falta para el desarrollo final del prototipo. Una de las decisiones más importantes que se debían de tomar era cual iba a ser la unidad lógica del robot. Esta tendría que ser la encargada de administrar y coordinar todas las otras tecnologías empleadas. Aun siendo la decisión más importante de todo el proyecto resulto ser la más simple. La universidad Politécnica de Cartagena lleva años trabajo en proyectos robóticos usando Lego MindStorm NXT 2.0. Comentarios en internet respalda esta versión también y mi experiencia con ello lo ha confirmado para mí. Una vez decidida la tecnología más importante de todas tocaba decidir que otras tecnologías iban a ser necesarias para el proyecto. En este paso había ciertas tecnologías que iban a ser imprescindibles en la primera fase del proyecto. Para la navegación en general lo que necesita es saber dónde estás y que te rodea. Existen diversas tecnologías capaces de abarcar este problema y elegir las más adecuadas es importante para el correcto desarrollo del proyecto. Por ello a continuación se explica que tecnologías se han empleado y como forman parte de este proyecto.

2.8.2. Hardware

El Hardware es la parte fundamental del proyecto, sobre todo porque es el que nos impone las limitaciones más estrictas. Más adelante veremos en qué casos el hardware nos impone limitaciones importantes a la hora de lograr nuestro objetivo.

▪ Lego Mindstorm NXT 2.0

- Brújula: Se usará para el ángulo que está girado el robot.
- Acelerómetro: Integrando los valores del acelerómetro obtendremos la velocidad e integrando nuevamente hallamos la posición.
- Encoder: Mediante las revoluciones del motor podremos saber su posición.
- Sensor de ángulo: Sustito del encoder, irá conectado al motor para conocer las revoluciones.
- Actuadores: Serán los motores encargados del movimiento de los robots.

▪ Dispositivo Android

Para este proyecto se ha utilizado un dispositivo Samsung Galay S3. A continuación se muestra una tabla con un resumen de sus características técnicas más importantes:

Figura 2.55: Samsung galaxy S3

TAMAÑO	**Dimensiones**	136.6 x 70.6 x 8.6 mm
	Peso	133 g
DISPLAY	**Tipo**	Super AMOLED touchscreen capacitivo, 16M colores
	Tamaño	720 x 1280 pixels, 4.8 pulgadas
SENSORES		Brújula digital Sensor giroscópico G-Sensor Sensor de proximidad Sensor de luz ambiental
MEMORIA	**Procesador**	Procesador Exynos 4 Quad quad-core 1.4 GHz, GPU Mali 400MP
	RAM	1 GB
	Interna	16 GB
CÁMARA		8 MP, 3264x2448 pixels, autofocus, flash LED, geo-tagging, detección de rostro y sonrisa, video 1080p@30fps, cámara frontal 1.9MP 720p@30fps
CONECTIVIAD		Wi-Fi 802.11 a/b/g/n; Wi-Fi Direct; DLNA; Wi-Fi Direct Bluetooth v4.0 A2DP, EDR microUSB 2.0
CARACTERÍSTICAS	**GPRS**	Clase 12 (4+1/3+2/2+3/1+4 slots)
	Velocidad de datos	32 - 48 kbps
	OS	Android OS, v4.0.4 Ice Cream Sandwich
BATERÍA		Standard, Li-Ion 2100 mAh
	Stand-by	Hasta 590 h (2G) / Hasta 790 h (3G)

2.8.3. Software

- **LeJOS**

A diferencia de lenguajes como C o C++ java no compila en código máquina (aunque esa posibilidad existe), sino en bytecode. Esta última particularidad hace que java pueda ser ejecutada en múltiples plataformas, como pueden ser: Windows, Linux, Mac OS y Android, entre otras (En la actualidad Java puede ser ejecutada en todos sistemas operativos existentes).

La gran mejora de Java frente a lenguaje de más bajo nivel es el facilitar al usuario la programación. Para ello en Java los punteros y la memoria no son gestionados por el usuario sino que de ello se encarga la máquina virtual. Los punteros que en otros lenguajes de programación son manejados como variables especiales pasan a ser otra variable más. Esto facilite muchísimo la comprensión del código y facilita el uso de los mismo, a la vez que minimiza los errores cometidos por los programadores a la hora de su manejo. Esto conlleva a que se mejore el rendimiento ya que el usuario poco experimentado no precisa comprender un concepto tan complejo como son los punteros.

La memoria por otro lado es gestionada por el recolector de basura. Este se encarga de liberar los objetos que no están siendo usados y optimizar el uso de memoria en general. Una explicación sencilla de su funcionamiento es que todo objeto que deja de ser apuntado por un puntero puede ser borrado. En lenguajes de bajo nivel esto tendría que ser realizado por el usuario y es sumamente complicado y se pierde mucho tiempo en ello. Como se ha dicho anteriormente el código de Java es ejecutado por una máquina virtual. Esta es la encargada de pasar el código en Java a código máquina. El uso de una máquina virtual obliga a que el código sea interpretado por la máquina virtual y posteriormente procesado por la máquina donde se está ejecutando. Gracias a esta ingeniosa idea es posible ejecutar un mismo código Java en todos los sistemas operativos conocidos sin necesidad de cambiar absolutamente nada.

Figura 2.56: Logotipo LeJOS

Por último la única desventaja de Java es precisamente la máquina virtual, los código en Java necesitan más recursos para ejecutarse y son algo más lentos que en lenguajes de más bajo nivel. Hoy en día esto no es un problema ya que los equipos de más bajo nivel tienen un hardware más que decente para reducir la diferencia entre Java y C++ a algo inapreciable para casi cualquier operación.

En vista a lo descrito anteriormente Java es el candidato ideal para la creación de prototipos de gran capacidad con un coste en tiempo mucho menor que en otros lenguajes. Es perfecto al ser un lenguaje orientado a objetos y sobre todo porque existe una comunidad muy amplia en constante

crecimiento que crea mucho contenido. Es fiable y robusto y permite integrarlo con multitud de otros lenguajes, haciéndolo aún más potente. No siendo el lenguaje con mayor velocidad de ejecución si es la más extendida y uno de los que mayor versatilidad ofrece. Es robusto, fiable y ejecutable en todas las plataformas.

- ## Android

Android es un sistema operativo basado en Linux, desarrollado para el uso en terminales portables, como móviles o tabletas. Originalmente fue desarrollado por Android In hasta que en 2005 lo compro Google como futuro sistema operativo para móviles. De esta forma Google empezó a invertir en el desarrollo de Android como puerta de entrada al mercado de los móviles. [9]

El modelo de Android se basa en el de IPhone, con pequeñas aplicaciones disponibles para los terminales que realizan distintas funciones. Estas aplicaciones son desarrolladas por terceros, con fines lucrativos o sin ánimo de lucro. Este nuevo tipo de aplicaciones ha sufrido un auge considerable, ya que son fáciles de implementar y de distribuir.

La distribución de las aplicaciones en Android se hace mediante Play Store (antiguamente Market). El Play Store es una aplicación desarrollada por Google donde podemos encontrar y comprar aplicaciones de terceros o de Google mismo. Se dividen por categorías de uso, y por gratuitas y de pago.

Figura 2.57: Logotipo ANDROID

- **Funcionamiento:**

Desde el comienzo Android estuvo enfocado en la conectividad, por ello soporta de base Bluetooth, WI-FI y otras tecnologías de comunicación de datos que no sean voz.

La arquitectura de Android viene esquematizada en la siguiente Figura 2.58:

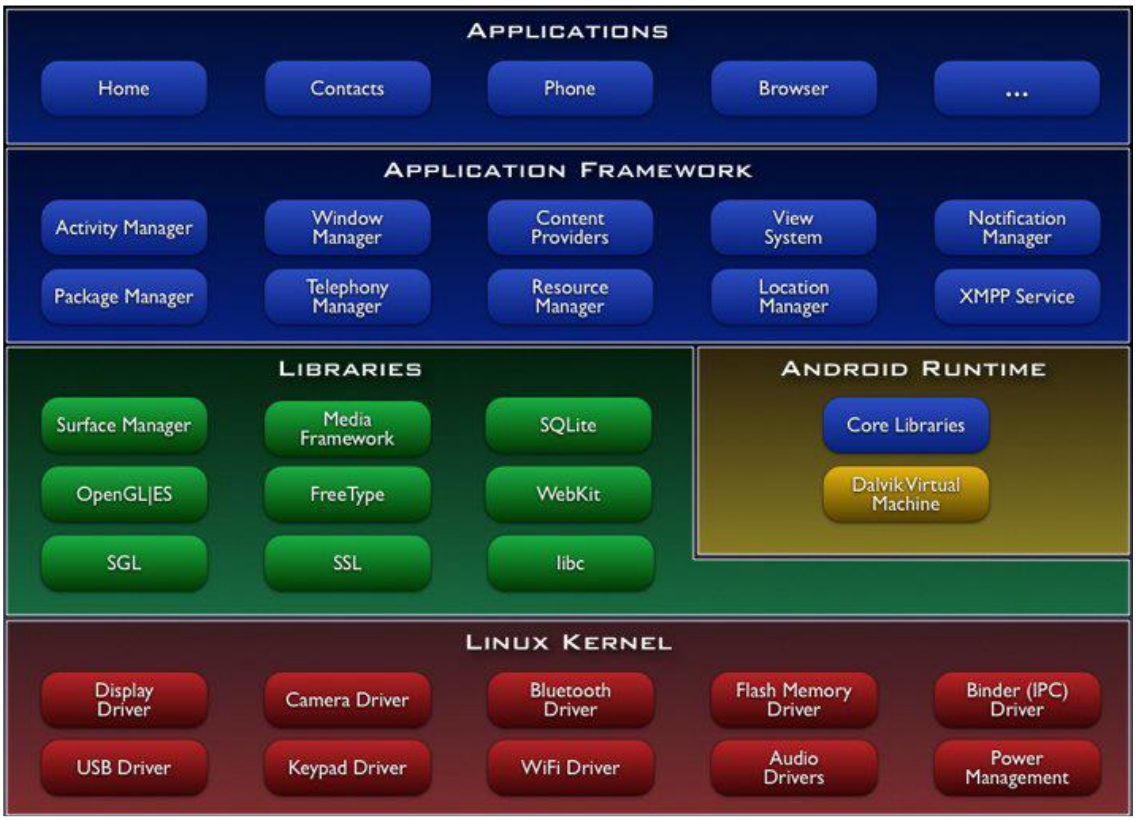


Figura 2.58: Esquema de la arquitectura de android

Esta arquitectura facilita la creación de aplicaciones así como la ejecución de las mismas. Android está basado enteramente en JAVA y por ello usa la máquina virtual de java para ejecutar las aplicaciones. En concreto usa una modificación denominada máquina virtual Dalvik. Esto permite que un dispositivo sea capaz de correr varias máquinas virtuales en paralelo permitiendo a cada proceso tener su propia máquina virtual. Los archivos ejecutables tienen como extensión ".dex" y está optimizada para dispositivos con poca memoria interna

Por último, el núcleo está basado en Linux y es la encargada de gestionar: la seguridad, la gestión de memoria, la gestión de procesos, la pila de red y el modelo de controladores. Así mismo el núcleo forma una capa abstracta entre el hardware y el software como en todos los sistemas operativos.

Conclusión:

Se ha elegido Android debido a que desarrollar aplicaciones en Android es totalmente gratuito. Solo se necesita "Android Software Development Kit" (Android SDK) o C o C++, Google App Inventor entre otras posibilidades.

Por otro lado Android está pensando directamente para desarrolladores y se pueden instalar aplicaciones sin necesidad de que estén publicadas en el Play Store. Esto facilita enormemente el desarrollo de aplicaciones para uso propio.

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. [13]

OpenCV también integra una completa librería de machine learning (MLL) centrada en análisis estadístico 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. Aun así, está más que justificada la utilización de OpenCV para llevar a cabo la parte de visión por computador de este proyecto.

Figura 2.59: Logotipo de OpenCV

IDE

- ECLIPSE: Tanto para Lego como para Android el entorno de desarrollo integrado elegido es Eclipse. Eclipse usa módulos, también llamados plug-ins. Estos permiten que se puedan usar

herramientas desarrolladas por terceros. Por lo tanto es posible personalizar Eclipse según el uso que se le quiera dar. Además nos permite programar en lenguajes que no estén soportados de base por Eclipse, también nos permite usar otro tipo de tecnologías para las que no fue desarrollado, un ejemplo sería la posibilidad de usar LaTEX. El proyecto Eclipse fue definido por su comunidad como: "*una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular*"

Figura 2.60: Logotipo Eclipse

En este proyecto no se documenta la instalación de los programas y librerías empleadas al existir gran cantidad de información en internet. En la *Bibliografía* quedan reflejados los enlaces a las páginas web consultadas.

2.8.4. 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.

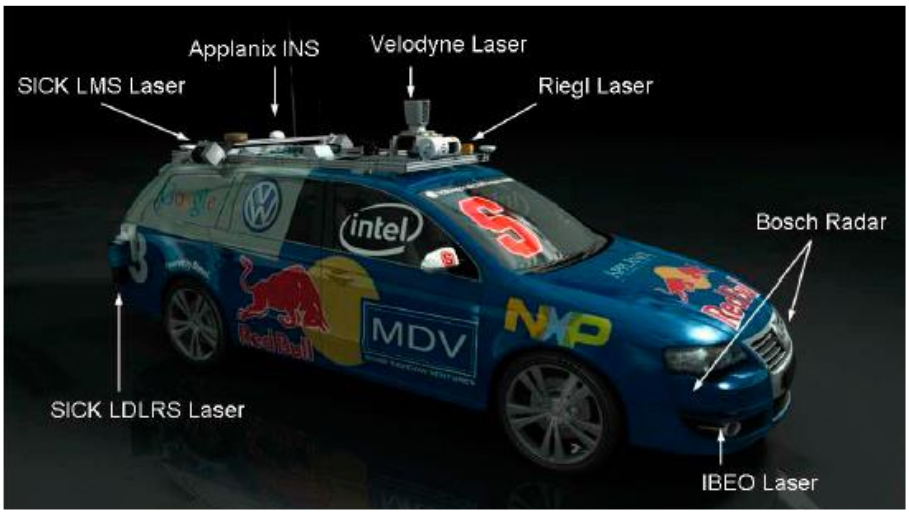


Figura 2.61: Junior, el coche desarrollado por la Stanford University para el DARPA Urban Challenge

En el caso de Lego Mindstorms, 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/wifi. 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.

2.9. Conclusiones basadas en el estado del arte

Lego Mindstorms 2.0 es una de las plataformas robóticas más prometedoras para desarrollos de bajo presupuestos. Habiendo demostrado en numerosas ocasiones que las capacidades de este brick superan con creces las de un mero juguete de Lego. Aunque originalmente fueses desarrollado como juguete para niños de unos 12 años, hoy en día es uno de los componentes más empleados en prototipos robóticos. Gracias a la aportación constante de una amplia comunidad de desarrolladores, cada día Lego NXT 2.0 tiene nuevas y prometedoras funcionalidades.

En gran parte la decisión de adoptar esta plataforma como unidad central del proyecto se debe a que en relación precio prestaciones es lo mejor que existen en el mercado. Existen otras distribuciones como se ha comentado antes, pero ninguna de ellas ofrece tantas facilidades y complementos así como una comunidad tan grande como Lego. Con este producto Lego ha abierto las puertas a un mercado nuevo, por el que casi ninguna otra empresa está apostando en la actualidad. Ofreciendo encima un servicio impecable y una tecnología que sigue mejorando a un precio asequible para el bolsillo. Añadiendo nuevos sensores, no solo Lego sino otras empresas que empiezan a dedicarse al desarrollo de productos para Lego NXT 2.0 visto el éxito que está teniendo.

La decisión final de elegir Lego NXT 2.0 se debe principalmente al amplio repertorio que existe en internet sobre esta tecnología. Permitiéndonos centrarnos en las cuestiones que más nos importan. También a la gran cantidad de gente que se dedica a desarrollar con Lego y que aporta sus conocimiento y experiencias. Esto nos permite aprender de los demás y explorar nuevas posibilidades y alternativas para los problemas que se nos plantean en este proyecto.

En definitiva Lego NXT 2.0 es el kit robótico más adecuado para casi cualquier robot de desarrollo doméstico. Es un kit que cada día tiene menos limitaciones y más funcionalidades prácticas. Una inversión a futuro para próximos desarrollos y una herramienta para investigar nuevas posibilidad así como para adentrarse en el mundo de la robótica. Un producto no solo apto para niños sino también para gente formada en el ámbito de la informática o con ganas de aprender. Es el kit robótico más completo que se puede encontrar en el mercado y el que mejores resultado da. En conclusión la herramienta perfecta para el desarrollo de este proyecto y las futuras mejoras que vayan a implementar e investigar.

CAPÍTULO 3

ROBOTS MÓVILES

"Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber."
Albert Einstein

3.1. Introducción

En este capítulo, denominado "Robots Móviles", se presenta una recopilación de información de varios autores sobre diferentes temas relacionados con los robots móviles, a fin de obtener parámetros que serán útiles en el diseño de un robot móvil con ruedas capaz de trabajar en entornos pequeños, en grupo o individualmente. Se inicia con una breve introducción relacionada a los avances, aplicaciones e importancia que tienen estos robots en la actualidad.

Posteriormente se expone el concepto de grados de libertad, centro instantáneo de rotación y se revisan los tipos de ruedas existentes. Se estudian los tipos de configuraciones diferencial, Ackerman, omnidireccional, con múltiples grados de libertad y movimiento mediante orugas. Luego, se mencionan algunos de los métodos que se emplean para que el robot pueda determinar su localización con respecto a un sistema de referencia absoluto.

Por último, se mencionan algunas de las aplicaciones que se vienen desarrollando actualmente con robots móviles pequeños.

3.2. Robots móviles con ruedas

La robótica móvil ha evolucionado conjuntamente con los avances tecnológicos recientes, lo cual ha dado lugar al desarrollo de muchas aplicaciones en áreas muy diversas. Si bien, dicha evolución ha alcanzado un nivel elevado en la actualidad, aún hay que esperar para la llegada de la revolución social de la robótica; en donde los robots llegarían a ser un elemento más en el hogar, la empresa y la sociedad en general.

Los robots hoy en día son en su mayoría excesivamente caros y su función está muy especializada, por lo que, los estudios actuales se centran en conseguir que sean más generales y económicos, de tal manera que sean accesibles a todas las personas que quieran resolver algún problema en sus tareas cotidianas. Debido a los avances tecnológicos actuales, construir un robot es

posible y se lo viene realizando en Universidades y actualmente en Compañías de juguetes. Además, construir un pequeño robot se ha convertido en el pasatiempo de muchos jóvenes, motivados por los concursos que se realizan en muchos lugares alrededor del mundo. La industria como tal no es la única que se ha beneficiado con la aparición de los robots, ya que existen áreas como: la medicina, aeronáutica y agricultura que también se han beneficiado. Una de las aplicaciones fuera de la industria, es el robot móvil todo terreno Sojourner, más conocido como Pathfinder, el cual recorrió la superficie de Marte el 5 de julio de 1997.

Los robots se clasifican de la siguiente manera:

- Humanoide
- Robot móvil
- Robot industrial
- Robot inteligente
- Robot de servicios

Este trabajo de fin de grado, se centra en el estudio de los robots móviles con ruedas, en especial de los microbots, denominados así por sus reducidas dimensiones. Por lo cual siempre se estaría haciendo referencia a ellos.

Los robots móviles pequeños o microbots por lo general están destinados a realizar tareas pequeñas con rapidez y precisión, simular las actividades de humanos y animales. Su principal filosofía es la movilidad, lo cual es muy ventajoso para realizar ciertas tareas y poder trabajar en equipo con otros robots o de manera aislada. Son útiles para realizar tareas en lugares peligrosos para los humanos, aburridos, sucios o difíciles. Tal es el caso de los robots móviles que se envían a Marte, o los que se utilizan para limpiar centrales nucleares. Una de las campanas que ha hecho posible que los robots móviles sean conocidos, son los concursos de robótica, que ha despertado la curiosidad e imaginación de muchos.

Si bien los robots móviles por lo general son de reducidas dimensiones con un bajo grado de control (el control existente es para regular la velocidad de las ruedas de tracción, el grado de control no es alto comparado con el nivel de control que se emplea en los robots articulados), adquieren un alto grado de complejidad cuando tienen que realizar tareas cooperativas. Esto ha dado como resultado que se realicen investigaciones sobre distintos temas, que luego pasan a formar parte de una gran lista de información existente en Internet. Información que puede ser utilizada para realizar nuevas investigaciones o para desarrollar nuevos prototipos de robots.

El comportamiento y la forma de actuar del microbot están determinadas por el programa que se ejecuta en él. El software de los microbots se encuentra actualmente estructurado en tres niveles que son:

- Sistema operativo
- Plataforma de desarrollo
- Aplicaciones concretas

El que un microbot realice autónomamente tareas de modo eficiente depende de la construcción mecánica y de la programación como veremos en los siguientes capítulos. La precisión en su desplazamiento se encuentra determinada por su sistema mecánico, mientras que la autonomía y la inteligencia se encuentran residentes en el programa que gobierna las acciones del microbot.

La influencia de los robots ha sido tan grande que ahora podemos hablar de un mercado de robots, el cual se encuentra en una fase de crecimiento. Cada vez son más los productos robóticos que salen al mercado, existiendo varios movimientos empresariales alrededor de la tecnología robótica. El mercado de robots móviles está aún inmaduro y todavía no se ha abierto al público. Esto se debe a que aún no se ha alcanzado el grado de autonomía y fiabilidad suficientes, que son necesarios para conseguir la evolución futura de ese mercado. El crecimiento en autonomía abre la puerta a nuevas aplicaciones comerciales de los robots, como son los robots de servicio y los de entretenimiento. Sin embargo la autonomía es difícil, y salvo casos contados, el desarrollo de aplicaciones con robots sigue siendo un tema de investigación.

3.2.1. Estructura general de un robot móvil

Debido a que un robot móvil por lo general está destinado a simular el comportamiento de personas y animales con un nivel de eficiencia similar, la estructura del robot móvil se pretende asemejar a la estructura del ser vivo. Se puede apreciar la estructura del robot móvil y del ser vivo en la Figura 3.1, en donde se puede observar de manera general la estructura, tanto de un robot móvil como de un ser vivo.

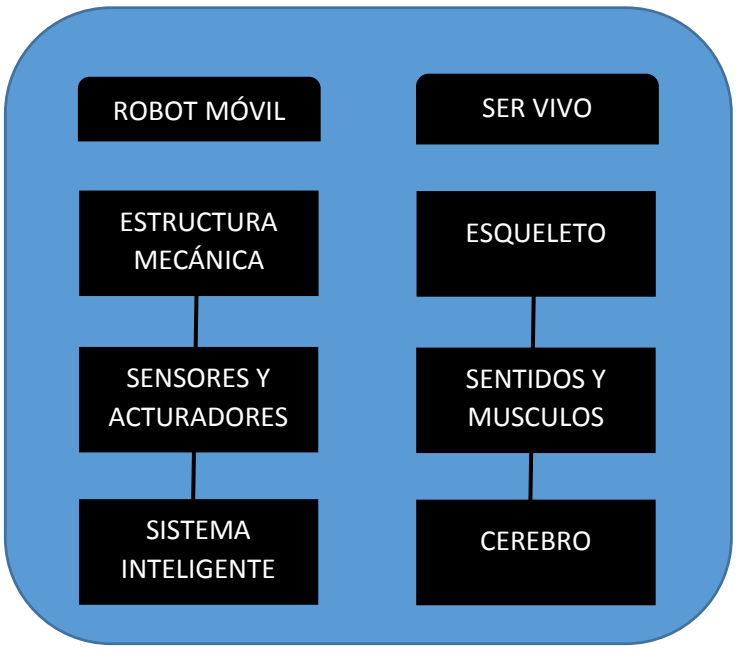


Figura 3.1: Similitudes entre un robot móvil y un ser vivo

La estructura de un robot móvil presentada en la Figura 3.1, está conformada por diferentes subestructuras, tales como:

- **Estructura mecánica:** Estructuras con ruedas, patas y orugas.
- **Actuadores:** Motores, luces, brazos, ruedas y en definitiva cualquier elemento que permita interactuar con el entorno.
- **Sensores:** Sonar, láser, cámaras, acelerómetros y cualquier elemento que nos proporcione información tanto del interior como del exterior del robot.
- **Inteligencia:** Métodos, algoritmos, etc. Estos van a permitir, a partir de la información de los sensores, interactuar con el entorno.

3.2.2. Configuraciones de robots con ruedas

Los robots con ruedas son la solución más simple y eficiente para conseguir la movilidad en terrenos suficientemente duros y libres de obstáculos, permitiendo velocidades relativamente altas. Existen diferentes tipos de configuraciones de locomoción mediante ruedas, cada una con unas características y propiedades diferentes en cuanto a eficiencia energética, dimensiones, cargas útiles y maniobrabilidad.

- **Diferencial**

El direccionamiento viene dado por la diferencia de velocidades de dos ruedas laterales, que a su vez proporcionan la tracción. Adicionalmente son necesarias una o más ruedas locas que sirven de apoyo. Es la configuración más utilizada en robots para interiores por su simplicidad y bajo coste. Sus desventajas son la dificultad de controlar los deslizamientos y las trayectorias rectas (ambas ruedas deben ir a la misma velocidad).

Figura 3.2: Configuración Diferencial

- **Tipo oruga**

Son robots con dos cadenas o pistas de deslizamiento en los laterales, como en los tanques. Tanto la tracción como el direccionamiento se consiguen mediante las cadenas que se comportan de forma análoga a dos grandes ruedas a cada lado. Desde el punto de vista cinemático, la configuración oruga se comporta como el modelo diferencial con el eje de giro situado entre las dos ruedas equivalentes.

Figura 3.3: Configuración Oruga

Esta configuración es útil para la navegación campo a través o en terrenos irregulares, en los que presenta mejor rendimiento que el modelo diferencial debido a su mejor tracción y menor deslizamiento.

Figura 3.4: Configuración Oruga tipo Tanque

- **Triciclo**

Este sistema es el del triciclo clásico. Dispone de dos ruedas de tracción y una rueda delantera (o trasera) orientable para la dirección. La rueda orientable también puede ser de tracción, en cuyo caso las dos ruedas del otro eje deben ser libres. Se trata de una configuración simple, pero puede presentar problemas de estabilidad como la pérdida de tracción cuando se desplaza por una pendiente.

Figura 3.5: Configuración Triciclo

- **Ackerman**

La configuración Ackerman es la utilizada en los vehículos de cuatro ruedas convencionales. Las dos ruedas delanteras giran para controlar la orientación y las traseras se mantienen paralelas.

Cuando se efectúa un giro, la rueda interior gira en un ángulo mayor que la exterior para evitar el deslizamiento.

El mayor problema de la configuración Ackerman es su limitación de maniobrabilidad.

El típico ejemplo de vehículo eléctrico con configuración Ackerman es un coche de radiocontrol como el de la Figura 3.6.

Figura 3.6: Configuración Ackerman

- **Omnidireccional**

Todas las configuraciones vistas hasta ahora comparten un mismo defecto: no pueden moverse en todas direcciones. Los robots omnidireccionales, en cambio, pueden moverse en cualquier dirección del plano y alcanzar cualquier posición sin necesidad de rotar antes. La base de su funcionamiento son las ruedas omnidireccionales que utilizan, cuyas ruedas locas pueden girar perpendicularmente al eje de la rueda principal a la que van unidas (Figura 3.7).

La única rueda sobre la que se aplica la acción, no obstante, es la rueda principal.

Estos robots suelen llevar tres ruedas omnidireccionales, aunque hay diseños que llevan cuatro.

Figura 3.7: Rueda omnidireccional doble de Rotacaster. Las ruedas locas (de color rojo) permiten a la rueda principal deslizarse lateralmente

Con tres ruedas, el sistema tiene suficientes actuadores para controlar los tres grados de libertad de movimiento plano (dos de traslación y uno de rotación). Aplicando distintas velocidades de giro a cada rueda, se consigue que el robot gire o que avance en la dirección deseada.

El robot omnidireccional más conocido es Robotino (Figura 3.8), que se usa con fines educativos y de investigación. Cuenta con tres ruedas omnidireccionales dispuestas con sus ejes formando ángulos de 120º.

Figura 3.8: Robot omnidireccional Robotino fabricado por Festo

3.2.3. Grados de libertad, tipos de ruedas y centro instantáneo de rotación

Los grados de libertad de un robot, así como los tipos de ruedas son aspectos que intervienen en el proceso de control y análisis de movimiento del robot. A continuación de presentan cada uno de ellos.

- **Grado de libertad (GDL)**

  Es cada uno de los movimientos de desplazamiento y rotación que puede realizar el robot.

  - Un cuerpo que se mueve en dos dimensiones tiene 3 GDL (una rotación y 2 traslaciones).
  - Un cuerpo que se mueve en tres dimensiones tiene 6 GDL (3 rotaciones y 3 traslaciones).

- **Sistema holonómico y no holonómico**

  Un sistema es holonómico si la cantidad de grados de libertad que se pueden controlar es igual a la cantidad de grados de libertad disponibles. En un sistema que es no holonómico el robot móvil no podrá desplazarse lateralmente. Esto se ilustra en la figura 3.9.

  En un sistema no holonómico, las ecuaciones diferenciales no son integrables en la posición final del robot. El saber la distancia recorrida por cada rueda, no es suficiente para poder calcular la posición final del robot. Hay que conocer como fue ejecutado el movimiento, en función del tiempo.

- **Tipos de ruedas**

  Las ruedas son el elemento que proporciona la capacidad de movilidad en un robot móvil. Se clasifican como:

- Rueda fija. Figura 3.12 (a). El movimiento se produce en la dirección de la rueda. Dónde: ω es la velocidad angular de la rueda, v la velocidad y a_x el vector unitario de dirección.

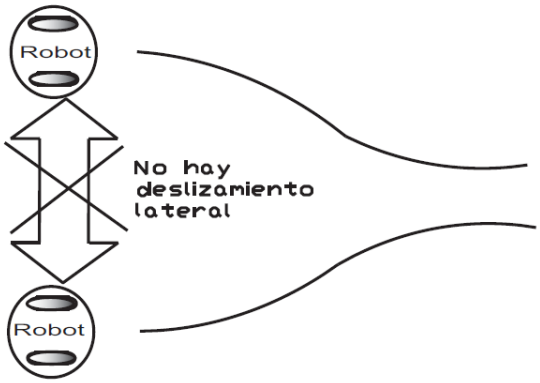


Figura 3.9: Sistema no holonómico

- Orientación centrada. Figura 3.12 (b). Además del giro t de la rueda, existe rotación alrededor del eje vertical que está dirigido al centro de la rueda. Dónde: ω es la velocidad angular de la rueda, v la velocidad y a_x el vector unitario de dirección.

- Orientación descentrada. Figura 3.13 (a). Gira sobre el eje de la rueda y rota alrededor del eje vertical situado a una distancia d desde el centro de la rueda. Dónde: ω es la velocidad angular de la rueda, v la velocidad y a_x el vector unitario de dirección.

- Rueda sueca. Figura 3.13 (b). Además de moverse en la dirección de la rueda, se mueve en dirección perpendicular a la dirección de la rueda. $v = (r * \omega)\, a_x + U a_s$. Donde: U es la velocidad de deslizamiento y as es un vector unitario en la dirección del deslizamiento, ω es la velocidad angular de la rueda y v la velocidad.

- **Centro instantáneo de rotación (ICR) o centro instantáneo de curvatura (ICC)**

  Se define como el punto por el cual cruzan los ejes de todas las ruedas; es el punto alrededor del cual el robot gira en un instante determinado. En la figura 1.5 se muestra el ICC para la configuración diferencial, Ackerman y triciclo.

3.3. Modelos cinemáticos

La elección del tipo de configuración es uno de los pasos necesarios para la construcción de un robot móvil. La configuración hace referencia a la forma en que se encuentran distribuidos los principales elementos que lo componen: plataformas, motores, ruedas, etc. Existen distintas configuraciones para robots móviles con ruedas, de las cuales tenemos: diferencial, triciclo, Ackerman, sincronizada, omnidireccional, con múltiples grados de libertad y movimiento mediante orugas.

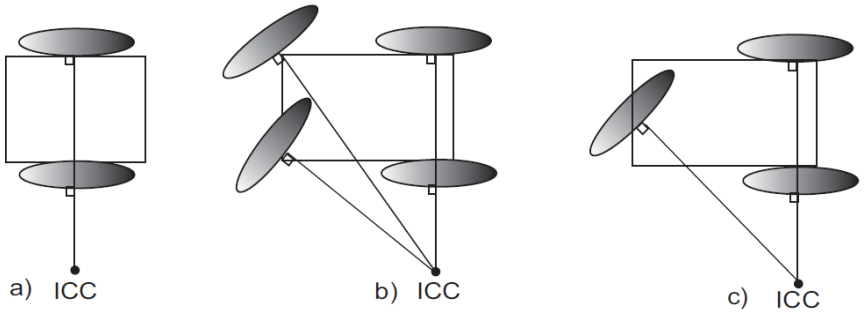


Figura 3.10: ICC en a) Configuración diferencial, b) Configuración Ackerman c) Configuración triciclo

3.3.1. Configuración diferencial

Es la más sencilla de todas. Consta de dos ruedas colocadas en el eje perpendicular a la dirección del robot. Cada rueda es controlada por un motor, de tal forma que el giro del robot queda determinado por la diferencia de velocidad de las ruedas. Así, para girar a la izquierda, hay que darle una velocidad mayor a la rueda derecha. Para girar a la derecha hay que darle una velocidad mayor a la rueda izquierda. La figura 3.4 muestra este tipo de configuración.

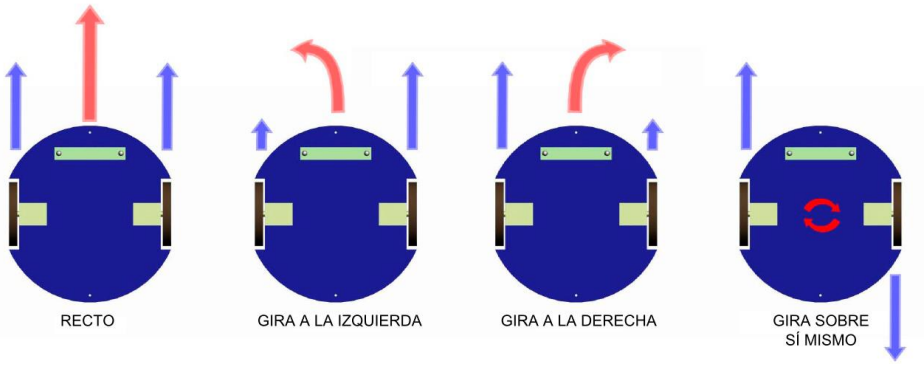


Figura 3.11: Trayectorias de un robot diferencial en función de la velocidad de

Uno de los problemas que tiene la configuración diferencial es mantener el equilibrio del robot, ya que consta de dos ruedas, por lo cual se le agregan ruedas de libre giro. Estas ruedas sirven para mantener horizontal al robot, por lo que giran libremente según el movimiento. Además, estas

ruedas se orientan hacia la dirección del robot. Dependiendo de las necesidades, se pueden agregar, una o más ruedas de libre giro. Al realizar las pruebas, las ruedas de libre giro ocasionaban ciertos problemas físicos que interrumpían el movimiento (esto queda detallado en el capítulo "Observaciones y resultados obtenidos") por lo que también se optó por sustituir la rueda por una pelota.

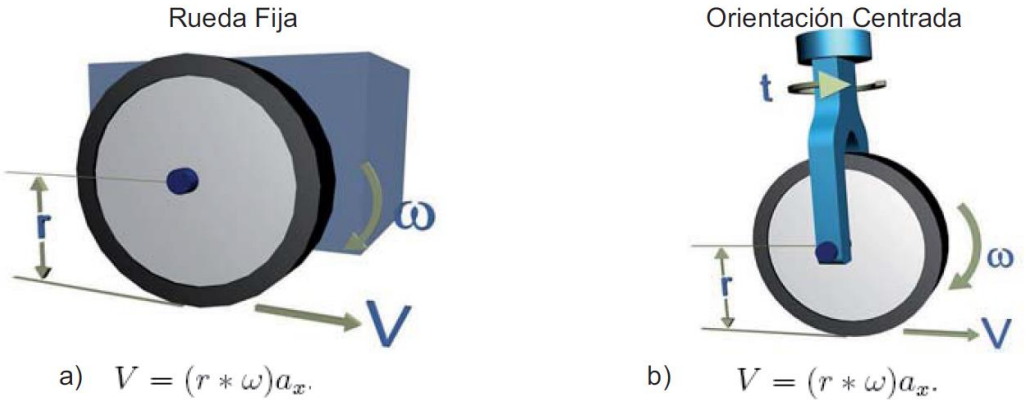


Figura 3.12: a) Rueda fija. b) Orientación centrada

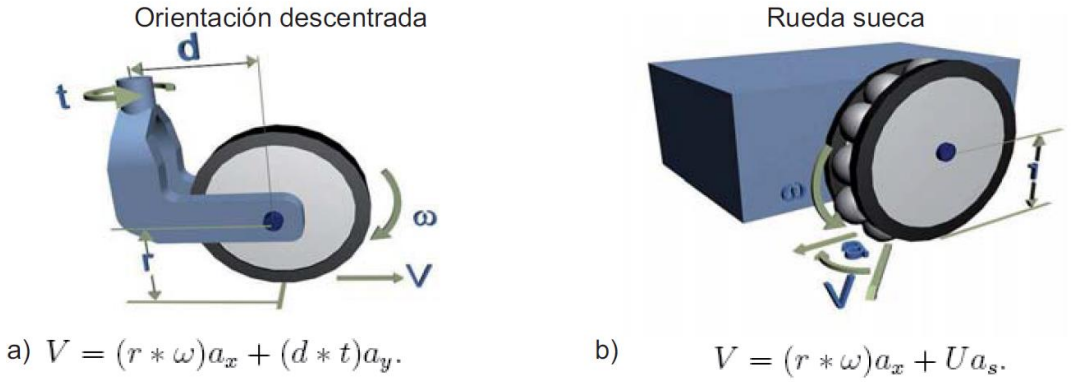


Figura 3.13: a) Orientación descentrada. b) Rueda sueca

El problema que surge debido a más de tres apoyos en el robot es la perdida de tracción y graves errores en los cálculos de odometría cuando el robot se encuentra en terrenos irregulares. La falta de tracción es provocada cuando existe más de una rueda de giro libre, esto se muestra en la Figura 3.14.

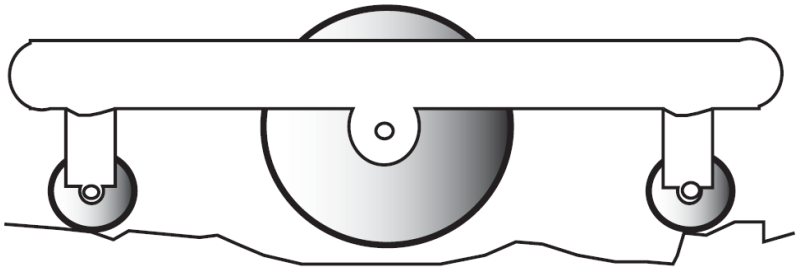


Figura 3.14: Falta de tracción debido a las ruedas de giro libre.

Este tipo de robots no son aptos para terrenos irregulares, por lo que su campo de acción se encuentra limitado a superficies planas. Debido a que la estructura diferencial es muy empleada, se calculará a continuación la posición (x, y) momentánea del robot relativo a un punto de comienzo mediante el uso de la odometría y posteriormente mediante la cinemática del robot.

Para poder obtener las ecuaciones cinemáticas es necesario establecer gráficamente todos los factores involucrados, entonces el modelo cinemático se ilustra en la Figura 3.15, donde:

Parámetros	**Significado**
p	Punto de contacto existente entre la rueda con la superficie
v	Velocidad de traslación del centro del robot
ω	Velocidad angular del robot con respecto a su centro
ICC	Centro instantáneo de rotación
r	Radio de las ruedas
$2b$	Ancho del robot
V_d	Velocidad del motor derecho
V_i	Velocidad del motor izquierdo
R	Radio de curvatura instantáneo
θ_c	Ángulo del robot respecto a las coordenadas absolutas x-y
C_m	Factor de conversión pulsos encoder/lineal
C_e	Resolución del encoder pulsos/rev
D_n	Diámetro de la rueda
n	Relación de engranajes
N	Número de pulsos del encoder

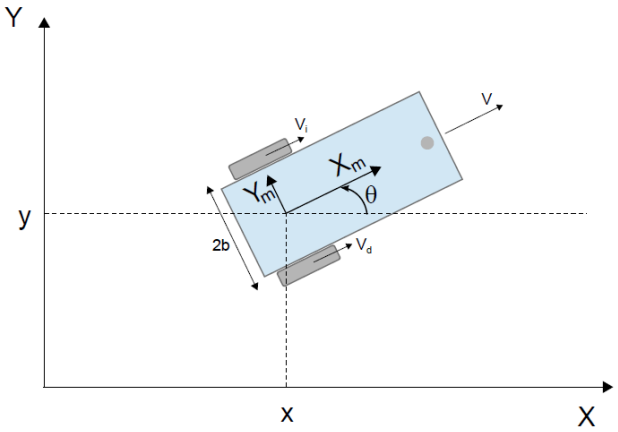


Figura 3.15: Modelo cinemático del robot

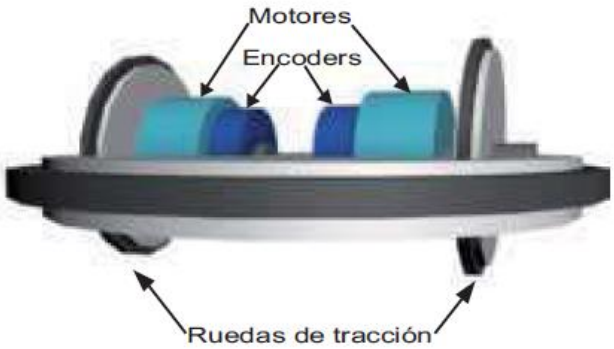


Figura 3.16: Robot con estructura diferencial

Antes de realizar el cálculo mediante odometría es necesario saber cuántas revoluciones ha dado la rueda, para lo cual se hace uso de encoders incrementales, los mismos que van colocados sobre los motores de tracción, entonces, se emplea un factor que convierte los pulsos del encoder en una descomposición lineal de la rueda para poder calcular las velocidades de cada rueda. El factor de conversión es determinado por la ecuación 3.1, en nuestro caso $n = 1$:

$$C_m = \frac{\pi D_n}{n C_e} \tag{3.1}$$

Con el factor de conversión se calcula la velocidad lineal de cada rueda como:

$$v_i = C_m \cdot N_i \qquad v_d = C_m \cdot N_d \tag{3.2}$$

La velocidad lineal de robot, v, se calcula como

$$v = \frac{v_i + v_d}{2} \tag{3.3}$$

Es decir, la velocidad lineal del robot es la media de la velocidad de las ruedas.

Cuando cada rueda gira a una velocidad diferente, el robot gira alrededor de un punto situado en algún lugar de la línea que une las ruedas, conocido como centro de curvatura instantáneo ICC (del inglés *Instantaneous Center of Curvature*).

En cada instante, las ruedas izquierda y derecha deben seguir una trayectoria que se mueva alrededor del ICC a la misma velocidad angular ω , luego se debe cumplir:

$$\omega(R + b) = v_d \qquad\qquad \omega(R - b) = v_i \tag{3.4}$$

La velocidad angular de robot se calcula como

$$\omega = \frac{v_d - v_i}{2b} \tag{3.5}$$

Y el radio de curvatura R

$$R = b \cdot \left(\frac{v_d + v_i}{v_{\ d} - v_i}\right) \tag{3.6}$$

Al analizar la ecuación 3.6, $R = \infty$ cuando $v_d = v_i$, quiere decir que el ICC se encuentra en el infinito o que el robot se movería en línea recta. Si $v_d = -v_i$ $R = 0$, indica que el ICC se encuentra en el centro del robot y por lo tanto girará alrededor de su centro.

- **Modelo cinemático directo**

Consiste en determinar la posición del robot a partir de su velocidad lineal y angular. Las ecuaciones (3.2) y (3.3) pueden expresarse de forma matricial como:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2b} & \frac{1}{2b} \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix} \tag{3.7}$$

La posición y orientación del robot viene dada por las ecuaciones

$$\dot{x} = v \cdot \cos\theta \tag{3.8}$$

$$\dot{y} = v \cdot \sin\theta \tag{3.9}$$

$$\dot{\theta} = \omega \tag{3.10}$$

O de forma matricial:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.11}$$

Siendo θ la orientación actual de robot.

Sustituyendo v y ω de (3.6) en (3.10) y ajustando se obtiene:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \frac{1}{2b} \begin{bmatrix} b\,cos(\theta) & b\,cos(\theta) \\ b\,sin(\theta) & b\,sin(\theta) \\ -1 & 1 \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix} = \begin{bmatrix} cos(\theta)\,\dfrac{v_i + v_d}{2} \\ cos(\theta)\,\dfrac{v_i + v_d}{2} \\ \dfrac{v_d - v_i}{2} \end{bmatrix} \tag{3.12}$$

- **Modelo cinemático inverso**

Este problema consiste en determinar qué acción de control hay que aplicarle a cada rueda para alcanzar una determinada posición (x, y, θ) deseada. Esto es más complejo de calcular que la cinemática directa. Se puede calcular aplicando la matriz pseudoinversa al modelo cinemático directo.

Aplicando la pseudoinversa $A^{+} = (A^{T}A)^{-1}A^{T}$, la ecuación (3.5) se obtiene:

$$\begin{bmatrix} v_i \\ v_d \end{bmatrix} = \begin{bmatrix} 1 & -b \\ 1 & b \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.13}$$

Y haciendo lo mismo con la ecuación (3.11):

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{3.14}$$

Sustituyendo la segunda en la primera se obtiene el modelo cinemático inverso:

$$\begin{bmatrix} v_i \\ v_d \end{bmatrix} = \begin{bmatrix} \dot{x}\cos\theta + \dot{y}\cos\theta - b\dot{\theta} \\ \dot{x}\cos\theta + \dot{y}\cos\theta + b\dot{\theta} \end{bmatrix} \tag{3.15}$$

3.3.2. Configuración triciclo

Los robots de triciclo tienen 2 ruedas traseras de propulsión y una delantera orientable para la dirección (Figura 3.17).

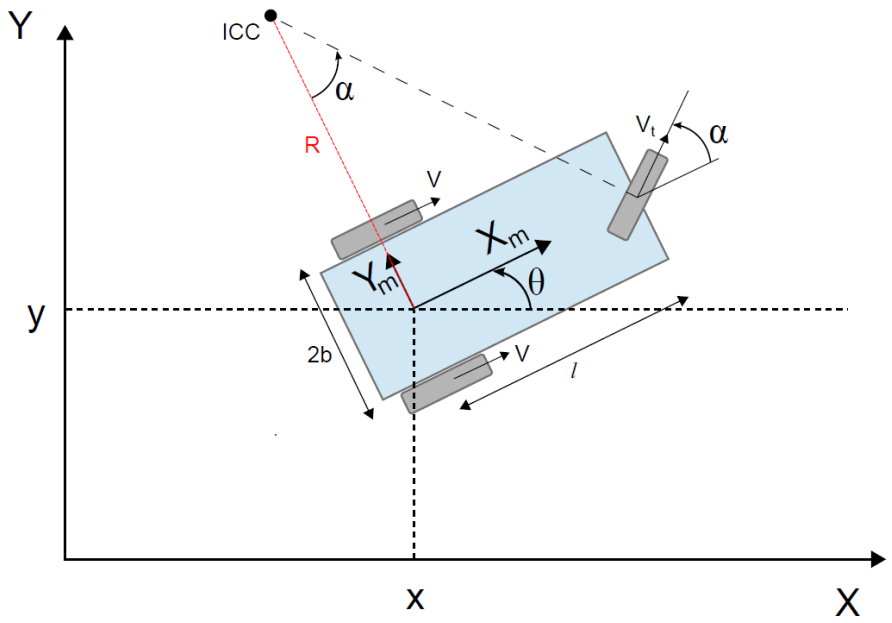


Figura 3.17: Modelo cinemático de un robot tipo triciclo

Para poder obtener las ecuaciones cinemáticas es necesario establecer gráficamente todos los factores involucrados, entonces el modelo cinemático se ilustra en la figura 3.17, donde:

Parámetros	Significado
α	Ángulo de giro de la rueda directriz
v_t	Velocidad lineal
γ	Curvatura que describe el robot
l	Distancia desde el eje trasero hasta el centro de la rueda delantera

- **Modelo cinemático directo**

La cinemática del triciclo se calcula de forma similar a la configuración diferencial. En este caso el ICC es el punto donde intersectan el eje de la rueda delantera y el de las ruedas traseras. El radio de curvatura R va desde este punto has el centro del eje trasero. Las velocidades lineal y angular son, respectivamente,

$$v = v_t \cdot \cos\alpha \tag{3.16}$$

y

$$\omega = v \cdot \gamma \tag{3.17}$$

De la figura anterior se deduce:

$$\tan\alpha = \frac{l}{R} \tag{3.18}$$

Y puesto que la curvatura es la inversa del radio, $\gamma = \frac{1}{R}$, se puede expresas en función del ángulo de la dirección como

$$\gamma = \frac{\tan\alpha}{l} \tag{3.19}$$

Desarrollando y expresando de forma matricial las ecuaciones de velocidad se tiene

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{\tan\alpha}{l} \end{bmatrix} v_t \cdot \cos\alpha \tag{3.20}$$

Sustituyendo estos valores de v y ω en la ecuación (3.11) se obtiene el modelo cinemático directo:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \frac{\tan\alpha}{l} \end{bmatrix} v_t \cdot \cos\alpha \tag{3.21}$$

- **Modelo cinemático inverso**

Se procede igual que en la configuración diferencial. Aplicando la matriz pseudoinversa a la ecuación (3.20) se obtiene

$$\begin{bmatrix} v_t \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{v^2 + \omega^2 l^2} \\ \arctan\left(\frac{l\omega}{v}\right) \end{bmatrix} \tag{3.22}$$

Y sustituyendo en ésta los valores de la ecuación (3.14) obtenemos el modelo cinemático inverso

$$\begin{bmatrix} v_t \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{\dot{x}\cos\theta + \dot{y}\sin\theta + \dot{\theta}^2 l^2} \\ \arctan\left(\frac{l\dot{\theta}}{\dot{x}\cos\theta + \dot{y}\sin\theta}\right) \end{bmatrix} \tag{3.23}$$

3.3.3. Configuración Ackerman

Es la configuración típica de los automóviles y vehículos autónomos de exteriores. Cuenta con cuatro ruedas, dos delanteras de dirección. Como ya se comentó, la característica de la dirección de Ackerman es que, al efectuar un giro, la rueda interior describe un ángulo mayor que la exterior, con el fin de evitar deslizamientos (Figura 3.18).

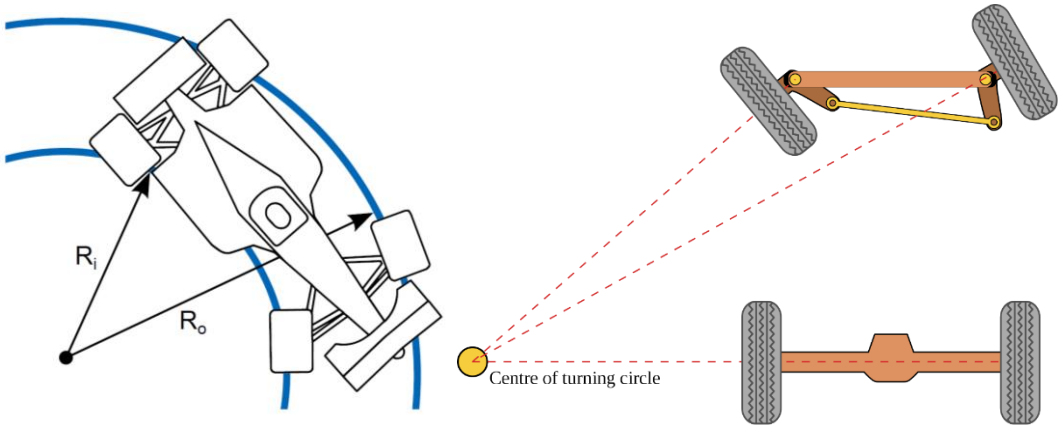


Figura 3.18: Radios interior y exterior descritos por las ruedas de un coche con configuración Ackerman

Desde el punto de vista cinemático, la configuración Ackerman es equivalente a la del triciclo una vez se conoce la rueda equivalente a las dos ruedas delanteras. En la Figura 3.19 aparece en color azul.

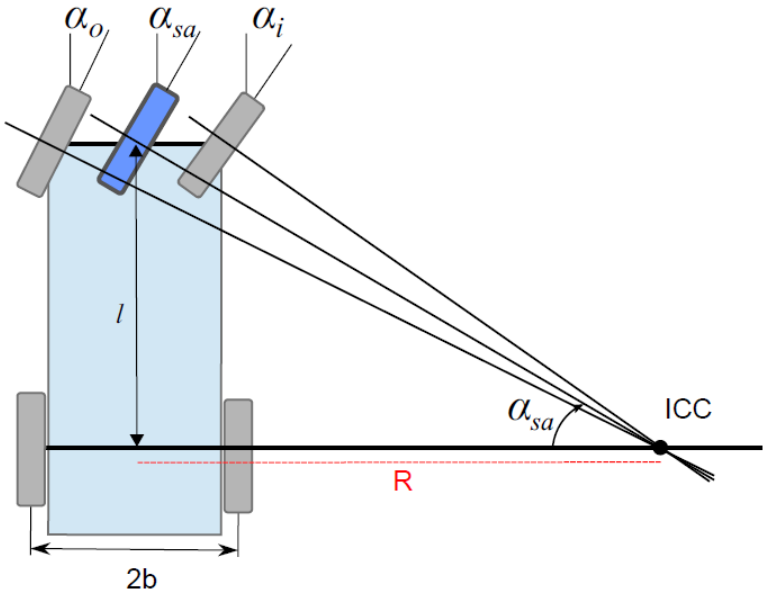


Figura 3.19: Modelo cinemático de Ackerman equivalente al triciclo

Donde:

Parámetros	Significado
α_i	Ángulo de la rueda interior (inner Wheel)
α_o	Ángulo de la rueda exterior (outer Wheel)
α_{sa}	Ángulo equivalente

La relación entre los ángulos de las ruedas interior y exterior es:

$$\cot \alpha_o - \cot \alpha_i = \frac{R+b}{l} - \frac{R-b}{l} = \frac{2b}{l} \tag{3.24}$$

Siendo la $\cot \alpha = \frac{1}{\tan \alpha}$.

El ángulo de giro de la rueda equivalente puede calcularse mediante

$$\cot \alpha_{sa} = \cot \alpha_i + \frac{b}{l} \tag{3.25}$$

o bien

$$\cot \alpha_{sa} = \cot \alpha_o - \frac{b}{l} \tag{3.26}$$

Una vez que se conocen la velocidad y orientación de la rueda equivalente, se pueden aplicar las mismas ecuaciones del modelo cinemático del triciclo.

3.3.4. Configuración omnidireccional

Este tipo de configuración esta provista de ruedas omnidireccionales Figura 3.20, lo que hace que los cálculos de odometría sean más complicados; pero el robot podrá moverse en cualquier dirección.

Figura 3.20: Robot omnidireccional. (Robot Palm)

La rueda omnidireccional, se define como una rueda estándar, a la cual se le ha dotado de una corona de rodillos. Los ejes de giro de los rodillos, son perpendiculares a la dirección normal de avance. De este modo, al aplicarle una fuerza lateral, los rodillos giran sobre sí mismo y permite que la componente de la velocidad en el eje x, no sea nulo, y por tanto, se elimina la restricción de no holomicidad. Las ruedas omnidireccionales tienen por lo general un costo elevado. El modelo cinemático de esta configuración se muestra en la figura 3.21. La configuración cinemática de este robot, se define por una estructura triangular equilátera, en cuyos vértices se encuentran colocadas tres ruedas omnidireccionales. La distancia desde el centro del triángulo equilátero a cualquiera de las ruedas es L. Todas las ruedas son no direccionales. Se representa como: ω_1, ω_2 y ω_3 a la velocidad angular de la rueda 1, rueda 2 y rueda 3. Finalmente, el modelo cinemático en el marco del robot, queda representado por medio de la matriz 3.24. Donde r es el radio de las ruedas.

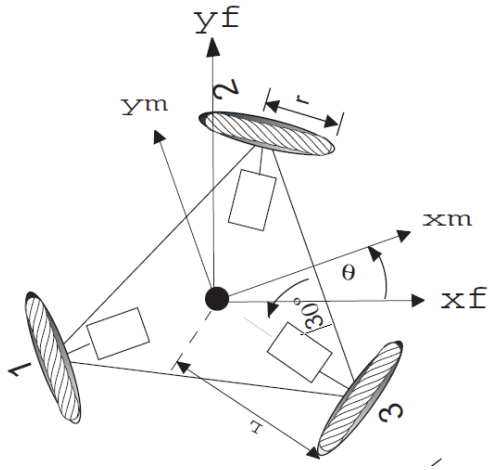


Figura 3.21: Modelo de la configuración omnidireccional.

$$\begin{bmatrix} V_x \\ V_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{\sqrt{3}}r & \frac{-1}{\sqrt{3}}r \\ \frac{2}{3}r & \frac{-1}{3}r & \frac{-1}{3}r \\ \frac{-r}{3L} & \frac{-r}{3L} & \frac{-r}{3L} \end{bmatrix} \begin{bmatrix} \omega 1 \\ \omega 2 \\ \omega 3 \end{bmatrix} \quad (3.27)$$

3.3.5. Configuración oruga

En este tipo de configuración, se sustituyen las ruedas por orugas (Figura 3.22). Esta implementación es un caso especial de la configuración diferencial.

Figura 3.22: Robot con orugas. (Robot rover)

En esta configuración, el deslizamiento en los giros es muy grande, se pierde la exactitud en los cálculos de odometría. Se emplea en lugares en donde el piso es muy irregular. Debido a que no es posible saber la posición exacta en la que se encuentra, es generalmente empleado solo en robots teleoperados, lo que constituye algo opuesto a las aplicaciones de los robots autónomos. Más específicamente son usados en búsqueda y rescate, levantamiento de bombas, y en la Industria nuclear.

Al poseer orugas, el vehículo tiene una gran área de contacto, lo que impide los hundimientos en la tierra suave. Para la maniobrabilidad, se usa direccionamiento mediante rodillos, lo que causa deslizamientos mayores y por lo tanto es más difícil determinar la posición en la que se encuentra. La exactitud del direccionamiento viene a ser más dificultoso a altas velocidades, debido al deslizamiento y a la resistencia de las huellas desiguales, causadas por las fuerzas centrifugas.

3.4. Estimación de la posición de un robot móvil

Los robots móviles se caracterizan por su capacidad de desplazarse de un lugar a otro en forma autónoma (capacidad de percibir, modelar, planificar y actuar para alcanzar unos objetivos sin la intervención, o con poca intervención de personas), ya sea en un lugar conocido parcialmente o desconocido en su totalidad. Un robot móvil es un sistema en el cual se encuentran inmersos diversos subsistemas de locomoción, control de movimientos, percepción y planificación que interaccionan entre sí. El subsistema de percepción hace que el robot sea capaz de interactuar en entornos cambiantes, así como poder reaccionar ante eventos inesperados, lo que exige la existencia de un sistema sensorial que suministre información sobre el entorno. Esta información requerida debe permitir al robot realizar tres tareas fundamentales:

1) Estimar su posición y orientación.
2) Mantener actualizado el mapa del entorno.
3) Detectar los posibles obstáculos.

El robot móvil rara vez va equipado con un único sensor para realizar todas estas tareas, sino que la práctica más habitual consiste en combinar dentro del sistema sensorial varios sensores que en mayor o menor medida se complementan. Algunas veces se emplean varios sensores redundantes con el propósito de validar la información adquirida. Para que un robot móvil pueda afrontar tareas como generar trayectorias, evitar obstáculos, etc. Se requiere que éste sea capaz de determinar su localización (posición y orientación) con respecto a un sistema de referencia absoluto.

La estimación de la posición de un robot móvil, viene dado por el tipo de entorno por el cual ha de navegar, el conocimiento que se tenga sobre el entorno, tipos de sensores que dispone, y de la tarea a realizar. La mayoría de robots móviles disponen de encoders, para detectar su movimiento, lo que les permite estimar en cada instante su posición, empleándose modelos de locomoción. Sin embargo, esta estimación no resulta muy conveniente para la mayoría de aplicaciones, ya que no es lo suficientemente precisa para ello. El motivo es que los errores, por más pequeños que sean se van acumulado durante la navegación, por lo cual se suelen usar sistemas de posicionamiento externo. Las principales técnicas utilizadas para la estimación de la posición y orientación de un robot autónomo pueden verse en la Figura 3.23.

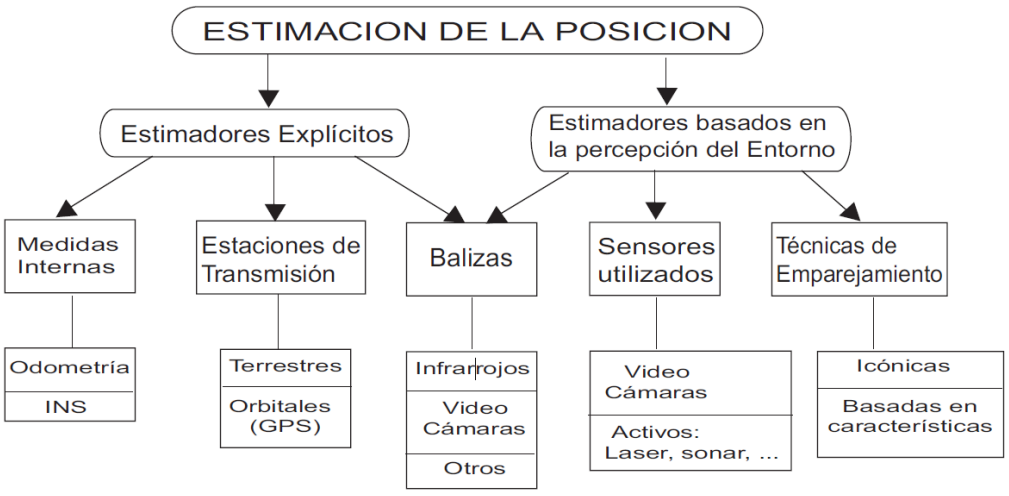


Figura 3.23: Sistema para la estimación de la posición.

Para la estimación de la posición y orientación de un robot móvil autónomo se distinguen los estimadores explícitos y los estimadores basados en la percepción del entorno. Los estimadores explícitos proporcionan directamente la posición y orientación del robot por medio de medidas, sin tener que realizar procesamiento de información para la interpretación del entorno. Los estimadores basados en la percepción del entorno emplean sensores que suministran información del exterior, por medio de la cual se puede saber la localización del vehículo mediante comparación de esta información con otros datos o modelos conocidos (pueden ser mapas, marcas naturales, objetos, etc).

El problema de la estimación está en que las medidas se encuentran asociadas a una cierta cantidad de ruido. Tanto las propias medidas como las estimaciones realizadas a partir de estas, tendrán una naturaleza aleatoria. Si no existiese el ruido, la posición y orientación del robot móvil se obtendrían simplemente al resolver el modelo matemático.

3.4.1. Estimación explícita

Se considera como estimación explicita a todos aquellos sistemas capaces de estimar la posición del robot móvil sin que se realice una interpretación del entorno.

En los sistemas de estimación explicita se distingue la estimación basada en medidas internas y la estimación basada en estaciones de transmisión. La estimación basada en medidas internas trabaja solamente con los sensores internos del robot móvil. Los sensores empleados pueden ser: giroscopios, encoders, detectores de norte, acelerómetros, tacómetros, etc. La estimación basada en estaciones de transmisión se basa en el empleo de dos unidades, la unidad montada sobre el robot móvil y la unidad o unidades externas. La unidad montada sobre el robot móvil actúa como sensor receptor, mientras que las externas actúan como emisores o transmisores de señales de referencia.

- **Estimación basada en medidas internas**

La posición y orientación de un robot móvil puede obtenerse integrando la trayectoria recorrida por este a partir de una serie de medidas internas como pueden ser: las vueltas que dan las ruedas, la velocidad, aceleración, cambios de orientación, etc. Se pueden distinguir los sistemas odométrico y los sistemas de navegación inercial.

- **Sistemas odométricos**

La odometría tiene por objeto determinar la posición y orientación del robot móvil a partir del número de pulsos obtenidos cuando giran las ruedas. Se utilizan codificadores ópticos de elevada precisión en al menos dos ruedas. La simplicidad y el bajo costo es una gran ventaja que nos ofrece el sistema odométrico. Sin embargo, es necesario una calibración constante, debido al desgaste y perdida de presión de las ruedas, desajuste de los ejes, etc. Esta técnica es vulnerable a las imperfecciones en el suelo, al deslizamiento de las ruedas y a las variaciones en la carga transportada (aunque en este caso es posible diseñar un modelo para corregir las desviaciones introducidas). La idea fundamental de la odometría es la integración de los incrementos del movimiento en el tiempo, lo que produce inevitablemente una acumulación de errores. La acumulación de errores en la orientación, causa grandes errores en la posición, los cuales se incrementan proporcionalmente con la distancia recorrida por el robot.

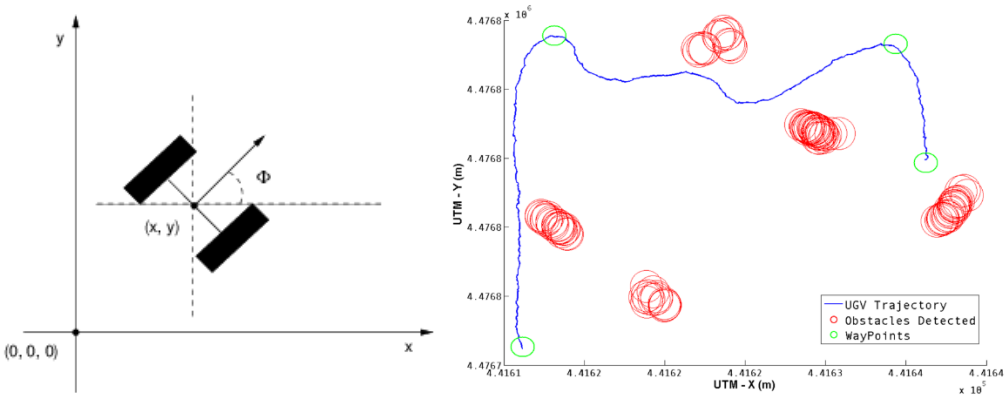


Figura 3.23: Representación de la trayectoria según odometría

A pesar de estas limitaciones, la odometría es fundamental en la robótica, ya que menos en condiciones ideales no se sabe con certitud la posición exacta del robot en un plano. Por ello el poder estimar un valor relativo fiable es de vital importancia. Para realizar los cálculos se usan las características de las ruedas y el eje. Con estos datos se puede crear un mapa relativo de la ruta que ha tomado el robot. Permitiendo crear un mapa de puntos con el recorrido hecho por el robot. Por lo tanto nos permite conocer la posición relativa de un robot en un entorno del que desconocemos las medidas exactas. Esto nos permite corregir el rumbo o como mencionado anteriormente mapear el entorno del robot. En la práctica la odometría es una herramienta complementaria a los sensores, que sin depender del correcto funcionamiento de estos nos permite estimar con gran precisión la posición del robot. Así mismo es un complemento imprescindible para poder monitorizar el correcto funcionamiento de los sensores: Al mismo tiempo la unión entre la odometría y el uso de sensores abre las puertas a alcanzar una precisión increíble a la hora de permitir que el robot se desplace sin problemas en entornos cerrados. Aunque la navegación local no haga referencia explícita a zonas cerradas, su uso es más adecuado en estos entornos ya que en exteriores se puede hacer uso de otro tipo de herramientas.

En definitiva, es un concepto que es aplicable en casi cualquier situación, ya que nos permite tener monitorizado el movimiento del robot sin necesidad de sensores complejos.

- **Sistemas de navegación inercial (INS)**

Este sistema obtiene la posición y orientación del robot móvil por medio de las medidas de aceleraciones y ángulos de orientación. Para obtener la posición, se integra la aceleración obteniéndose la velocidad, la cual se integra para finalmente obtener la posición. Como se indicó, este sistema emplea la aceleración para la obtención de la posición, para lo cual hace uso de acelerómetros que suelen estar basados en sistemas pendulares. La precisión del acelerómetro resulta crítica, ya que, los errores en la aceleración aunque sean pequeños afectan la obtención de la posición, debido a la doble integración de la aceleración. Para medir la orientación, se emplean giroscopios e acelerómetros. También es posible medir el ángulo de orientación mediante brújulas. Los sistemas de navegación inercial no son afectados por los problemas derivados de la interacción del vehículo con el suelo. En este sistema se pueden corregir los efectos de las ondulaciones e irregularidades del terreno, lo que hace que en la práctica, sean mucho más fiables y precisos que los sistemas basados en odometría. Estos sistemas son más frágiles y caros que los sistemas basados en odometría.

Los sistemas de navegación inerciales se suelen utilizar en navegación marítima, aeronaves, misiles y naves espaciales, ya que **un INS es capaz de detectar un cambio en la posición geográfica** (un pequeño desplazamiento al norte o al este), un cambio en su velocidad (módulo y dirección) y un cambio en su orientación (rotación alrededor de un eje). Como este sistema no necesita una referencia externa (sólo inicialmente), es inmune a las interferencias que podría sufrir otro sistema, como el GPS.

3.4.2. Estimación basada en el entorno

- **Estimadores basados en estaciones de transmisión**

Este tipo de sistema de posicionamiento es absoluto y su ventaja más destacada es proporcionar la posición absoluta del robot móvil en un área suficientemente grande, sin tener que estructurar el entorno, lo que hace que el robot móvil pueda moverse por escenarios muy diversos, recorriendo grandes distancias (carreteras, caminos forestales, parajes semidesérticos, etc).

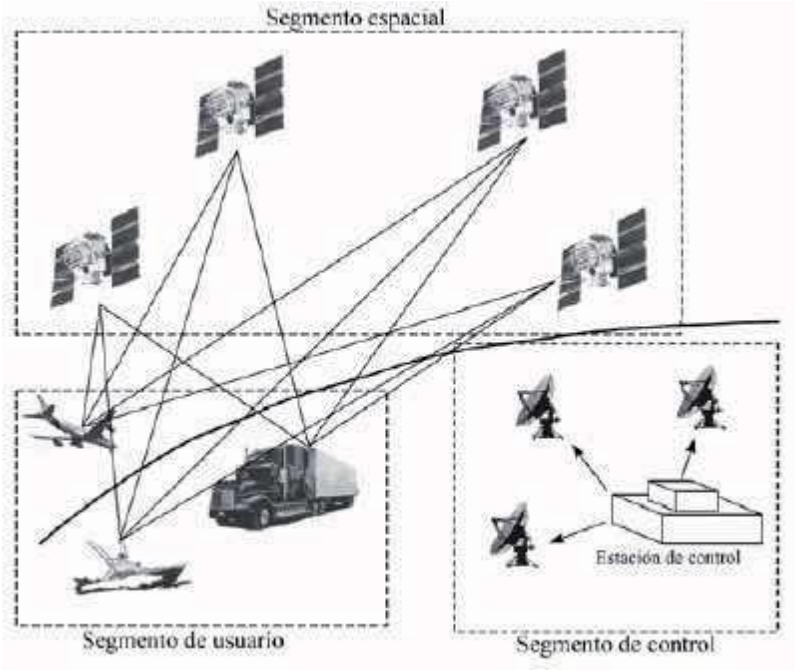


Figura 3.24: Sistemas GPS

Estos sistemas constan de un receptor, que se encuentra colocado en el robot móvil, y de estaciones de transmisión de RF. Dentro de este sistema se encuentra, los sistemas de posicionamiento mediante estaciones fijas y los sistemas de posicionamiento mediante estaciones móviles. Los sistemas de posicionamiento mediante estaciones fijas, se han empleado en la navegación de barcos, submarinos y aviones.

Utilizan señales de radio de media y alta frecuencia que son emitidas por estaciones terrestres fijas. Los sistemas de posicionamiento mediante estaciones móviles, son operados mediante satélites. Su utilización era en aplicaciones militares (misiles, aviación, submarinos) y en la navegación civil (fundamentalmente aviación). Hoy en día se emplea un sistema similar pero mucho más potente denominado GPS Figura 3.24 ("Global Positioning System" en el cual, al menos utilizando 3 satélites, el receptor calcula por triangulación la altitud, latitud y altura del vehículo de forma instantánea y continúa (tiempos entre 30 y 60ns). También puede determinar la velocidad a partir del desplazamiento en frecuencias mediante el efecto Doppler.

- **Estimación mediante balizas**

Permite determinar la posición del robot móvil en un entorno delimitado por balizas que se encuentran en posiciones conocidas. La posición del robot móvil es determinada en una forma más o menos directa en base al principio de triangulación, a partir de medidas de distancias, ángulos o de la combinación de los dos.

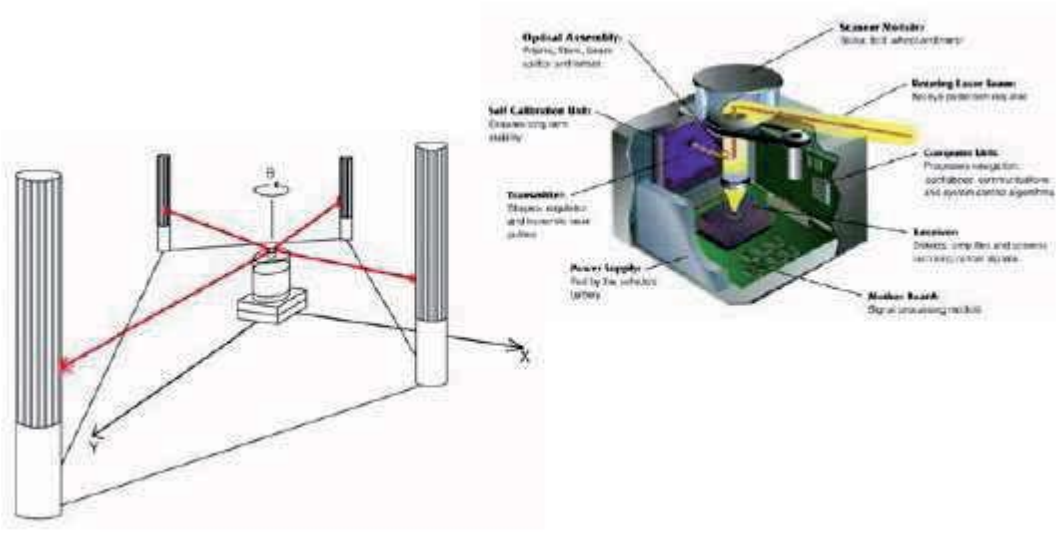


Figura 3.25: Estimación mediante balizas

La precisión y fiabilidad que este tipo de estimación proporciona, depende fundamentalmente del tipo de señal utilizada (infrarrojos, láser, radio, ultrasonidos, etc.), de las características del sensor, y del número de balizas utilizadas en la triangulación Figura 3.25. Si bien, este método está considerado como uno de los más precisos. Las principales desventajas radican en la necesidad, tanto de configurar apropiadamente el entorno de trabajo, como de garantizar que un suficiente número de estas señales quede en todo momento libre de obstrucciones y dentro del campo visual del sensor. También deben tenerse en cuenta los problemas que pueden originar las condiciones ambientales de iluminación y ruido (acústico, electromagnético, etc.). Todo ello impide la utilización de esta técnica en entornos muy dinámicos o no-estructurados.

- **Estimación mediante percepción del entorno**

Consiste en dotar al robot móvil de un sistema sensorial capaz de proporcionar suficiente información del entorno como para que este pueda de forma autónoma determinar su localización. El sistema sensorial puede operar en base a distintos tipos de sensores (cámaras CCD, sonares, escáner láser, etc) Figura 3.26 y seleccionando determinados tipos de datos u objetos a partir del conjunto de información adquirida (marcas naturales, puntos de interés, entorno completo percibido, etc.). En cualquier caso, la localización del robot móvil se determina a través del emparejamiento de los datos extraídos del entorno por el sistema sensorial, con datos previamente conocidos del entorno.

Figura 3.26: Sistema sensorial

Los sensores utilizados para la navegación de un robot móvil pueden situarse en dos grupos: activos y pasivos. Los sensores activos son aquellos que emiten algún tipo de energía al medio (luz infrarroja, ultrasonidos, luz láser, ondas de radio, etc.). Los sensores denominados pasivos se limitan a captar la energía ya existente en el medio. De este tipo son las cámaras de video CCD, las cuales perciben el entorno a través de la cantidad de luz que les llegan procedente de fuentes luminosas o bien a través de reflexiones en los objetos del entorno.

CAPÍTULO 4
DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

"Lo hicimos porque nadie nos dijo que era imposible".
Anónimo

4.1. Introducción

El objetivo de este capítulo es explicar la construcción de los prototipos e implementar el control de los mismos para que recorran una determinada trayectoria. Mediante los diferentes sensores comentados en el *Capítulo 2: Estado del Arte* conocer la posición estimada para poder comparar los resultados y comprobar su eficacia. También se construye la torre para el seguimiento de objetivo y su sistema de realimentación para el brick de Lego mediante visión artificial.

4.2. Construcción de los prototipos

4.2.1. Introducción

El Hardware es la parte fundamental del proyecto, sobre todo porque es el que nos impone las limitaciones más estrictas. Más adelante veremos en qué casos el hardware nos impone limitaciones importantes a la hora de lograr nuestro objetivo.

4.2.2. Prototipo Diferencial

La estructura del robot diferencial se ha montado utilizando elementos y piezas estándar de Lego Mindstorm NXT. Se han empleado 2 motores para la propulsión y en un principio una rueda loca. Por los problemas causados por la rueda loca al quedarse bloqueada y no girar correctamente se decidió sustituirla por una bola que mejoró el comportamiento notablemente.

Alto (cm)	Ancho (cm)	Largo (cm)	Rueda (diámetro)
10	17	18	5.6

Figura 4.1: Robot Lego Diferencial

- **Prototipo Diferencial con visión artificial**

Como construcción extra se ha decidido dotar de visión artificial al robot diferencial (Figura 4.2):

Figura 4.2: Prototipo diferencial con visión artificial

4.2.3. Prototipo Ackerman

Para esta configuración se han empleado dos motores, uno para la propulsion de las ruedas traseras y otro para controlar la dirección.Las ruedas delantes deben comportarse como es característico en la configuración Ackerman, es decir, al efectuar un giro la rueda interior debe producir un ángulo mayor que el de la rueda exterior. La forma más sencilla de conseguirlo es haciendo que las uniones de las barras del eje delantero formen un trapecio (Figura 4.3).

Figura 4.3: Aproximación al sistema de dirección de Ackerman empleando uniones en forma de trapecio

Así mismo, en el eje trasero es necesario un diferencial para permitir que las ruedas izquierda y derecha giren a distinta velocidad cuando se toma una curva. Se ha utilizado un engranaje diferencial de Lego. En la Figura 4.3 se puede apreciar el montaje en forma de trapecio del eje de dirección y el eje trasero con el diferencial incroporado, así como los dos motores para accionarlos.

Alto (cm)	Ancho (cm)	Largo (cm)	Rueda (diámetro)
11	15	20	5.6

Figura 4.4: Vista frontal y trasera del prototipo Ackerman

4.2.4. Prototipo Oruga

La configuración tipo oruga consta de 2 motores para la propulsión, no deja de ser un tipo diferencial, pero se sustenta mediante cadenas como si de un tanque se tratara. Debido a la necesidad de engranajes para poder mover el robot correctamente y el poco agarre de las cadenas en suelo liso los errores acumulados eran demasiado grandes por lo que se decidió no incluir esta configuración en el estudio.[14]

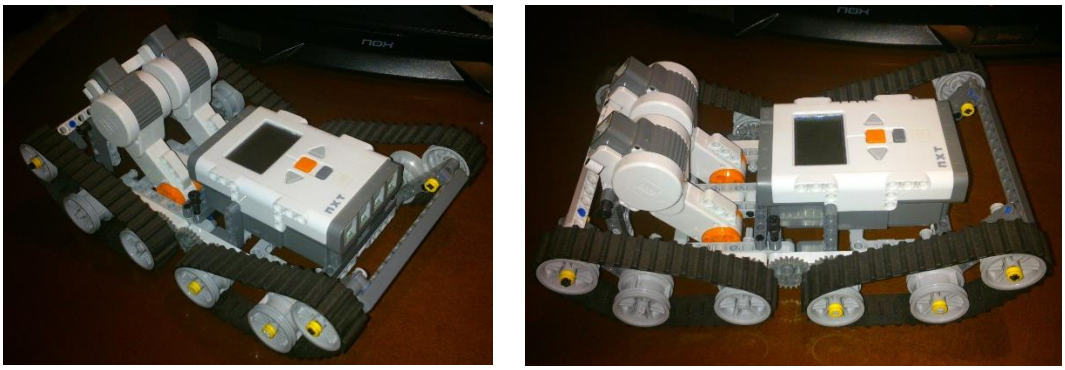


Figura 4.5: Prototipo Oruga

4.2.5. Prototipo Torreta

La torreta consta de 2 motores, una para el eje horizontal y otro para el vertical. Con un simple sistema de engranajes gira sobre sí misma y se eleva ocupando el mínimo espacio, esta construcción se muestra en la Figura 4.6:

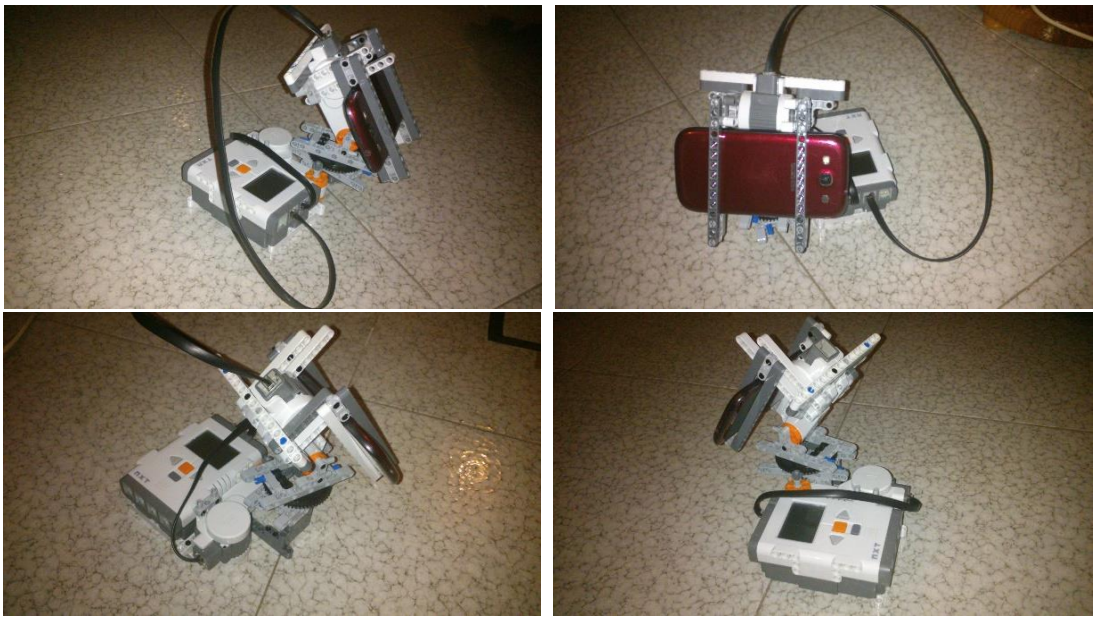


Figura 4.6: Torreta para seguimiento de objetivo

4.2.6. Robot móvil con torreta

El sistema final resulta de la unión del robot para la tracción con la torreta para el seguimiento de objetivo (Figura 4.7):

Figura 4.7: Robot móvil con torreta

4.3. Trayectorias

4.3.1. Introducción

Trayectorias complejas no ayudarían para obtener buenos resultados, pues de lo que se trata es de comprobar la eficacia de la odometría y los sensores, por lo tanto con la repetición de una trayectoria simple será suficiente. En este proyecto se optó por figuras geométricas básicas. Además se ha decidido implementar un circuito bonus: Lemniscata (una curva parecida al simbolo de infinito) que se verá deformado según cada configuración (esto queda más detallado en el *Capítulo 5: Observaciones y resultados obtenidos*).

4.3.2. Trayectorias escogidas

Las trayectorias pueden verse en la Figura 4.8, por su simplicidad han sido escogidas las figuras geométricas básicas como un cuadrado, triángulo, círculo y un circuito extra que combina varias trayectorias, la lemniscata.

En la siguiente tabla se muestra la distancia total recorrida aproximada:

	Cuadrado	**Triángulo**	**Círculo**	**Lemniscata**
Lado/Radio (m)	1	1	0.5	L=0.5, R=0.25
Distancia total (m)	12	9	9.45	9

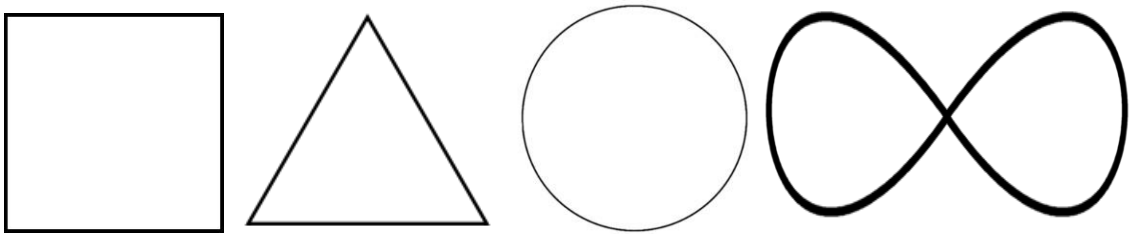


Figura 4.8: Trayectorias a realizar

4.4. Implementación del sistema de control

4.4.1. Introducción

En esta parte del proyecto están involucrados un PC y un brick de Lego. Se explica la función de cada uno y la comunicación entre ellos. El objetivo será controlar a los robots con distintas configuraciones y estimar su posición para posteriormente estudiar la eficacia de los sensores. Las pruebas consistirán en la repetición del circuito 3 veces con los diferentes sensores tomando la posición como p = (x, y, h, t) en cm, ° y ms respectivamente cada 100 ms.

4.4.2. Comunicación Bluetooth

La conexión entre maestro – esclavo es ACL (Asyncronous Connection Less), es decir un canal que proporciona un acceso asíncrono entre maestro y el esclavo (un canal por pareja). El esclavo solo puede enviar una vez que ha recibido un paquete de datos del maestro. Con este propósito el maestro puede enviar paquetes de pedida de datos a los esclavos.

En nuestro caso el robot se inicia primero como maestro y se queda esperando la petición de algún esclavo. Luego el PC se ejecuta y le pide al robot conectarse a él, si se establece conexión el robot crea un flujo de salida de datos para poder enviarle al PC los datos de la posición estimada y el PC un flujo de entrada para recibirlos. Al terminar el circuito el robot envía "-1" para hacer saber al PC que ha terminado y finalizar su ejecución.

```
LCD.drawString(waiting,0,0);
LCD.refresh();

BTConnection btc = Bluetooth.waitForConnection();

LCD.clear();
LCD.drawString(connected,0,0);
LCD.refresh();

DataOutputStream dos = btc.openDataOutputStream();
float x = opt.pp.getPose().getX();
float y = opt.pp.getPose().getY();
float h = opt.pp.getPose().getHeading();

dos.writeFloat(x);
dos.writeFloat(y);
dos.writeFloat(h);

dos.writeInt(j*100);
dos.flush();
```

Figura 4.9: Envió de datos Robot Lego

```
boolean connected = conn.connectTo("TURRET", "00:16:53:06:0c:04", NXTCommFactory.BLUETOOTH);
if (!connected)
{
    System.err.println("Failed to connect to any NXT");
    System.exit(1);
}
DataInputStream dis = new DataInputStream(conn.getInputStream());
while(in != -1)
{
    float x =  dis.readFloat();
    float y =  dis.readFloat();
    float h =  dis.readFloat();

    sbf.append(formato.format(x) + "\t"  + formato.format(y) + "\t"+ formato.format(h) + "\t" + in);
    sbf.append(System.getProperty("line.separator"));
}
```

Figura 4.10: Recepción datos PC

4.4.3. Plataforma PC

La principal función del PC es el almacenamiento de datos de los sensores para su posterior estudio. En un principio no estaba contemplado la utilización de un PC ya que el propio Lego puede crear un archivo .txt en almacenamiento interno para guardar información, pero al tener una memoria muy limita no es capaz de almacenar todos los datos por lo que la solución propuesta fue mandarlos por Bluetooth a un PC.

El PC actúa como esclavo en esta parte del proyecto. El programa que ejecuta el PC crea un fichero de texto vacío e intenta conectarse al Lego para recibir los datos de los sensores. Para conectarse necesita la MAC del dispositivo que se puede poner manualmente o se puede buscar y encontrar.

```
BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
```

Figura 4.11: Objeto BluetoothDevice

A continuación utilizando el objeto BluetoothDevice se puede declarar el objeto BluetoothSocket que define el socket que utilizará la conexión, asignándolo un identificador único de la conexión (UUID). Una vez creado el socket ya se puede proceder a establecer dicha conexión.

```
UUID uuidBt =UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
nxtBTsocket = device.createRfcommSocketToServiceRecord(uuidBt);
nxtBTsocket.connect();
```

Figura 4.12: Creación del socket y establecimiento de conexión

El socket creado es del tipo RFCOMM. Una vez que se ha conectado correctamente se define el flujo de entrada y salida de datos:

```
LegoApplication.nxtDos = new DataOutputStream(nxtBTsocket.getOutputStream());
nxtIn = new DataInputStream(nxtBTsocket.getInputStream());
```

Figura 4.13: Creación de los flujos de entrada y salida

Y cuando están definidos estos flujos, se ejecuta el hilo de recepción de paquetes que añade al archivo de texto de salida. Los datos recibidos son la posición p = (x, y, h, t) como se muestra en la Figura 4.14:

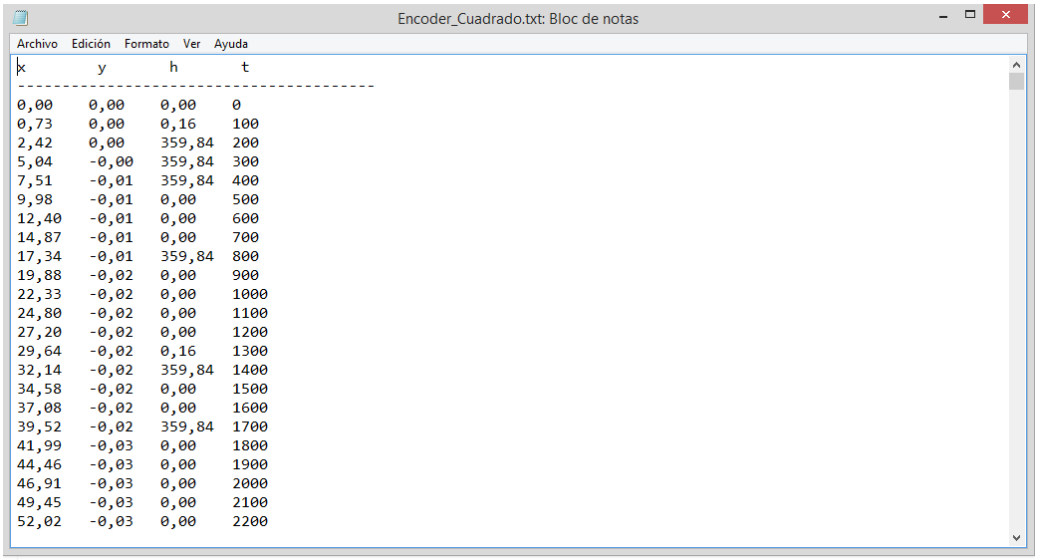


Figura 4.14: Archivo de texto de salida

4.4.4. Plataforma Lego NXT: Robot

- **Prototipo Diferencial**

En este caso de estudio, el Lego es el maestro y es el PC el que se conecta a él. Además es el encargado de calcular los datos necesarios para conocer la estimación de la posición para enviarselos al PC.

```
btc = Bluetooth.waitForConnection(0, NXTConnection.RAW);
```

Figura 4.15: Declaración del bluetooth

En el primer parametro se le indica el tiempo que debe esperar a recibir una solicitud de conexión. El valor “0” indicia que el tiempo es infinito.

El segundo parámetro es el realmente importante, pues se indica el modo de conexión que se requiere. Si se llama a la función sin ningún parámetro, esta los preestablecerá por defecto, y la conexión dará error, ya que el modo por defecto no permite la conexión otros dispositivos que no sean de Lego.

Los tres modos que se pueden seleccionar son:

- NXTConnection.RAW: Este modo es el que se utiliza para conexiones que no sean con otros dispositivos Lego NXT, como PC, Smartphone. Y es el que se ha seleccionado en este proyecto para los prototipos.
- NXTConnection.LCP: Este modo se utiliza para conexiones a través del protocolo LCP. Este protocolo creado por Lego, se utiliza para mandar comandos al robot, sin la necesidad de estar ejecutando un programa en él. Estos comandos pueden ser comandos directos, que permiten manejar el robot, o comandos del sistema. Este modo se ha utilizado para la torreta.
- NXTConnection.PACKET: Este es el modo predeterminado, y es el mejor si lo que se quiere es conectarse a otro dispositivo Lego NXT.

- **Implementación del modelo matemático en el robot**

  Para facilitar el control del robot, la lectura de los sensores y obtener la estimación de la posición se ha implementado una serie de clases que nos facilitaran la tarea, en la Figura 4.16 se muestra el esquema de clases:

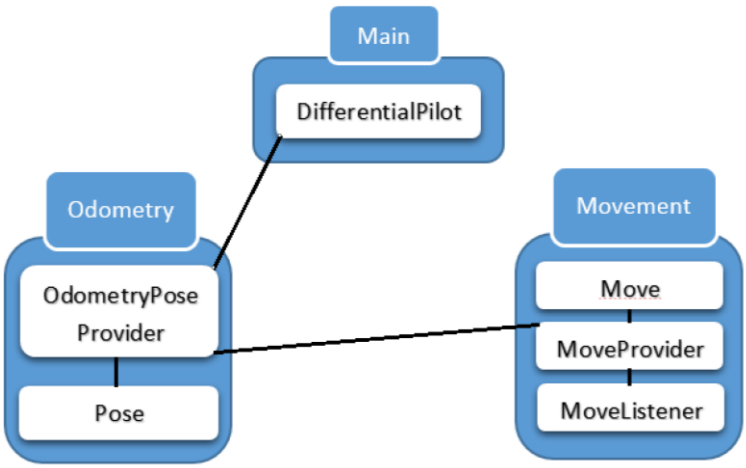


Figura 4.16: diagrama de clases prototipos

Se ha separado el código en 3 paquetes claramente diferenciados. A continuación se detalla que clases componen cada paquete y que función realizan.

- **Main Activity**
- DifferentialPilot: Es la actividad principal que se ejecuta al iniciar la aplicación. Define el pilot y le manda las órdenes de movimiento.

- **Odometry**
  Contiene las clases necesarias para estimar la posición y localizar al robot.

- OdometryPoseProvider: Mantiene la trayectoria del robot y contiene la información de la posición.
- Pose: Esta clase es la encargada de definir y actualizar la posición del robot.

- **Movement**
- Move: Define los distintos tipos de movimiento (*travel, rotate, forward,* etc).
- MoveProvider: Esta clase devuelve los movimientos del robot en tiempo real.
- MoveListener: Actualiza y proporciona los movimientos del robot a partir del motor.

A continuación se explican las principales clases con un ejempo de funcionamiento:

- **DifferencalPilot:** Esta clase es una abstracción software del mecanismo de un piloto de un robot. Contiene los metodos para controlar el movimiento del robot: hacia adelante o marcha atrás en linea recta o en camino curvo, rotar, etc. Esta clase funciona controlando 2 motores independientes el uno del otro. Para crear un piloto se le pasan los parametros necesarios como son el diametro de las ruedas, y la distancia entre ejes. Un ejemplo de como se crearía este piloto es:

```
DifferentialPilot pilot = new DifferentialPilot(5.7f,17.0f, Motor.A, Motor.C, true);  // parameters in cm
pilot.setRobotSpeed(30);  // cm per second
pilot.travel(50);         // cm
pilot.rotate(-90);        // degree clockwise
```

Figura 4.17: Creación DifferenciaPilot

Donde DifferencialPilot inicializa las variables necesarias para la cinemática directa del robot, como son:

Parámetro	Definición
_WheelDiameter	Diámetro de la rueda
_TurnRatio	Radio de giro mínimo
_DegPerDistance	Conversión grados-distancia
_trackWidth	Distancia entre el centro de cada rueda

```
public DifferentialPilot(final double leftWheelDiameter,
        final double rightWheelDiameter, final double trackWidth,
        final RegulatedMotor leftMotor, final RegulatedMotor rightMotor,
        final boolean reverse)
{
  _left = leftMotor;
 _left.addListener(this);
  _leftWheelDiameter = (float)leftWheelDiameter;
  _leftTurnRatio = (float)(trackWidth / leftWheelDiameter);
  _leftDegPerDistance = (float) (360 / (Math.PI * leftWheelDiameter));
  // right
  _right = rightMotor;
  _right.addListener(this);
  _rightWheelDiameter = (float)rightWheelDiameter;
  _rightTurnRatio = (float)(trackWidth / rightWheelDiameter);
  _rightDegPerDistance = (float)(360 / (Math.PI * rightWheelDiameter));
  // both
  _trackWidth = (float)trackWidth;
  _parity = (byte) (reverse ? -1 : 1);
  setTravelSpeed(.8f * getMaxTravelSpeed());
  setRotateSpeed(.8f * getMaxRotateSpeed());
  setAcceleration((int)(_robotTravelSpeed * 4));
}
```

Figura 4.18: Clase DifferencialPilot

- **OdometryPoseProvider:** Calcula la trayectoria del robot y la posición con la ayuda de la clase *Pose*. Añade un "Listener" a DifferencialPilot. A continuación se detalla un ejemplo para su mejor comprensión:

```
private DifferentialPilot pilot;
private OdometryPoseProvider pp;

public static void main( String[] args) throws SensorSelectorException
{
    pilot = new DifferentialPilot(5.6f, 17.0f, Motor.A, Motor.C);
    pp = new OdometryPoseProvider(pilot);
    pilot.addMoveListener(pp);

    pilot.setAcceleration(60);
    pilot.setTravelSpeed(20);
    pilot.setRotateSpeed(70);

    pilot.travel(100, true);
    while(opt.pilot.isMoving())
    {
        LCD.clear();

        float x = pp.getPose().getX();
        float y = pp.getPose().getY();
        float h = pp.getPose().getHeading();
    }
}
```

Figura 4.19: Añadir Listener de odometría al piloto

En este ejemplo se crea un piloto para la configuración diferencial mediante la clase *DifferentialPilot* al que se le pasan los parámetros del robot. A este pilot se le añade un listener de movimiento para que la clase *OdometryPoseProvider* estime la posición.

El método "*Travel*" realiza la siguiente función:

```
public void travel(final double distance, final boolean immediateReturn)
{
  _type = Move.MoveType.TRAVEL;
  _distance = distance;
  _angle = 0;
  if (distance == Double.POSITIVE_INFINITY)
  {
    forward();
    return;
  }
  if ((distance == Double.NEGATIVE_INFINITY))
  {
    backward();
    return;
  }
  movementStart(immediateReturn);
  setSpeed(Math.round(_robotTravelSpeed * _leftDegPerDistance), Math.round(_robotTravelSpeed * _rightDegPerDistance));
  _left.rotate((int) (_parity * distance * _leftDegPerDistance), true);
  _right.rotate((int) (_parity * distance * _rightDegPerDistance),
          immediateReturn);
  if (!immediateReturn) waitComplete();
}
```

Figura 4.20: Función "Travel"

Si la distancia es positiva llama a "*forward()", sino a "backward()".*

```
public void forward()
{
  _type = Move.MoveType.TRAVEL;
  _angle = 0;
  _distance = Double.POSITIVE_INFINITY;
  movementStart(false);
  setSpeed(Math.round(_robotTravelSpeed * _leftDegPerDistance), Math.round(_robotTravelSpeed * _rightDegPerDistance));
  if (_parity == 1)
  {
    fwd();
  } else
  {
    bak();
  }
}
```

Figura 4.21: función "forward()"

En forward se comprueba un flag de paridad por si los motores están al reves y se llama a *"fwd()" o "bak()":*

```
/**
 * Left motor..
 */
protected final RegulatedMotor _left;
/**
 * Right motor.
 */
protected final RegulatedMotor _right;
```

```
private void fwd()
{
  _left.forward();
  _right.forward();
}
```

Figura 4.22: Funciones para mover el motor

Donde *RegulatedMotor* es una clase de Lego para controlar los motores.

Pasamos ahora a la odometría, para obtener los datos de la estimación de la posición usamos el método *"getPose"* de la clase *OdometryPoseProvider:*

```
public synchronized Pose getPose()
{
  if (!current )
  {
    updatePose(mp.getMovement());
  }
  return new Pose(x, y, heading);
}
```

Figura 4.23: Obtener la posición

Si la posición no es la misma que la actual la actualizamos:

```
private synchronized void  updatePose(Move event)
{
  float angle = event.getAngleTurned() - angle0;
  float distance = event.getDistanceTraveled() - distance0;
  double dx = 0, dy = 0;
  double headingRad = (Math.toRadians(heading));

  if (event.getMoveType() == Move.MoveType.TRAVEL    || Math.abs(angle)<0.2f)
  {
    dx = (distance) * (float) Math.cos(headingRad);
    dy = (distance) * (float) Math.sin(headingRad);
  }
  else if(event.getMoveType() == Move.MoveType.ARC)
  {
    double turnRad = Math.toRadians(angle);
    double radius = distance / turnRad;
    dy = radius * (Math.cos(headingRad) - Math.cos(headingRad + turnRad));
    dx = radius * (Math.sin(headingRad + turnRad) - Math.sin(headingRad));
  }
  x += dx;
  y += dy;
  heading = normalize(heading + angle); // keep angle between -180 and 180
  angle0 = event.getAngleTurned();
  distance0 = event.getDistanceTraveled();
  current = !event.isMoving();
}
```

Figura 4.24: Cálculo de la posición

Se calcula la nueva posición respecto a la anterior y se suman las diferencias obteniendo así la nueva posición p = (x, y , h).

En el caso de usar la brújula el ángulo del robot ya no será calculado mediante odometría, se usará la libreria de sensores de lego para obtener los datos:

```
CompassHTSensor compassht = new CompassHTSensor(SensorPort.S1);
float h = compassht.getCompassHeading();
```

Figura 4.25: Obtención datos de los sensores

Análogamente para el acelerómetro:

```
Accelerometer accel = SensorSelector.createAccelerometer(SensorPort.S1);
float x = accel.getX();
```

Figura 4.26: Obtención datos Acelerómetro

▪ Robot Ackerman

Análogamente al prototipo diferencial, para controlar el prototipo Ackerman será necesario una serie de clases que controlen todos los elementos del robot. Todas las clases son las mismas que para el modelo diferencial cambiando las ecuaciones del modelo.

La eficacia de esta configuración depende de los siguientes factores físicos:

- La superficie del terreno.
- La precisión del sistema de giro.
- La capacidad del robot de conducir en línea recta (si el robot intentar ir en línea recta pero va haciendo una pequeña curva, la eficacia descenderá significativamente).

Y los parámetros:

Parámetro	Definición
driveWheelDiameter	Diámetro de las ruedas
driveMotor	Motor de propulsión
steeringMotor	Motor de giro
minTurnRadius	Radio mínimo de giro

A continuación se explican las principales clases:

```
public SteeringPilot(double driveWheelDiameter, lejos.robotics.RegulatedMotor driveMotor,
        lejos.robotics.RegulatedMotor steeringMotor, double minTurnRadius,
        int leftTurnTacho, int rightTurnTacho) {
    this.driveMotor = driveMotor;
    this.steeringMotor = steeringMotor;
    this.driveMotor.addListener(this);
    this.driveWheelDiameter = driveWheelDiameter;
    this.minTurnRadius = minTurnRadius;
    this.minLeft = leftTurnTacho;
    this.minRight = rightTurnTacho;

    this.isMoving = false;
}
```

Figura 4.27: Creación SteeringPilot

Definimos un robot Ackerman creando una instancia de la clase *SteeringPilot* (Figura 4.28) pasándole los parámetros físicos del robot. Una vez definida una instancia de nuestro robot debemos calibrarlo (Figura 4.26), para ello giramos las ruedas al a la derecha y a la izquierda hasta que encuentren resistencia y no puedan moverse, en ese momento guardamos el valor del encoder. Esos valores son los límites de giro del robot. El centro será la media de ambos:

```
public void calibrateSteering() {
    steeringMotor.setSpeed(100);
    steeringMotor.setStallThreshold(10, 100);

    steeringMotor.forward();
    while(!steeringMotor.isStalled()) Thread.yield();
    int r = steeringMotor.getTachoCount();

    steeringMotor.backward();
    try {
        Thread.sleep(200);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    while(!steeringMotor.isStalled()) Thread.yield();
    int l = steeringMotor.getTachoCount();

    int center = (l + r) / 2;
    r -= center;
    l -= center;

    minRight = r;
    minLeft = l;

    steeringMotor.rotateTo(center);
    steeringMotor.resetTachoCount();
    steeringMotor.setStallThreshold(50,1000);
}
```

Figura 4.28: Calibración robot Ackerman

Para movernos trazando una curva se usará el método *travelArc (Figura 4.29):*

```
public void travelArc(double turnRadius, double distance, boolean immediateReturn) throws IllegalArgumentException {

    double diff = this.getMinRadius() - Math.abs(turnRadius);
    if(diff > 0.1) throw new IllegalArgumentException("Turn radius can't be less than " + this.getMinRadius());

    // 1. Check if moving. If so, call stop.
    if(isMoving) stop();

    // 2. Change wheel steering:
    double actualRadius = steer(turnRadius);

    // 3 Create new Move object:
    double angle = Move.convertDistanceToAngle((float)distance, (float)actualRadius);
    moveEvent = new Move((float)distance, (float)angle, true);

    if((distance == Double.NEGATIVE_INFINITY) | (distance == Double.POSITIVE_INFINITY)) {
        driveMotor.backward();
    }

    // 4. Start moving
    // Convert Float infinity to Integer maximum value.
    int tachos = (int)((distance * 360) / (driveWheelDiameter * Math.PI));
    driveMotor.rotate(tachos, immediateReturn);
}
```

Figura 4.29: método travelArc

Como parámetros se le pasa el radio de giro y la distancia. Primero se comprueba que el giro sea posible y que el robot esté parado, si no se detendrá la acción actual para que no exista conflicto. Mediante la función *Steer* (Figura 4.30) definimos el ángulo de giro:

```
private double steer(double radius) {
    if(radius == Double.POSITIVE_INFINITY) {
        this.steeringMotor.rotateTo(0);
        return Double.POSITIVE_INFINITY;
    } else if(radius > 0) {
        this.steeringMotor.rotateTo(minLeft);
        return getMinRadius();
    } else { // if(radius <= 0)
        this.steeringMotor.rotateTo(minRight);
        return -getMinRadius();
    }
}
```

Figura 4.30: Función Steer para giro

Una vez que las ruedas conocen el radio de giro se calcula el ángulo que debe recorrer el robot y se añade un objeto *Move* para, mediante odometría, estimar el movimiento del robot.

Por último se le comunica al motor de propulsión cuanto debe girar para recorrer esa distancia.

4.4.5. Plataforma Lego NXT: Torreta

LEGO NXT pueden recibir comandos a través de Bluetooth (soportan el perfil de puerto serie) o comunicación USB. Este protocolo de comunicación se llama comandos directos NXT o protocolo de comunicación Lego NXT (LCP). Este protocolo de comunicación NXT no pasa por los Mindstorms (software oficial de LEGO) o cualquier otro software de terceros. Se utiliza para controlar NXT desde un dispositivo remoto (smartphone, PC, otros robots...) sin necesidad de programar el brick.

El envío de comandos a Lego NXT a través de Bluetooth es bastante simple. La idea es transferir una trama de bytes hacia el robot para su inmediata ejecución.

Los comandos se dividen en dos grupos:

1) **Comandos del sistema:** Comandos administrativos que controlan la administración del brick o el protocolo. Ejemplo de comandos administrativos puede ser acceder al sistema de archivos de la configuración del brick o de la comunicación;
2) **Comandos directos:** Controlan el movimiento del robot, la lectura de sensores… Algunos de los comandos pueden solicitar una respuesta del robot mientras que otros comandos son sólo un conjunto de órdenes.

En el siguiente apartado se detalla la comunicación entre el smartphone y el lego NXT.

4.4.6. Plataforma Android

- **Diagrama de clases**

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

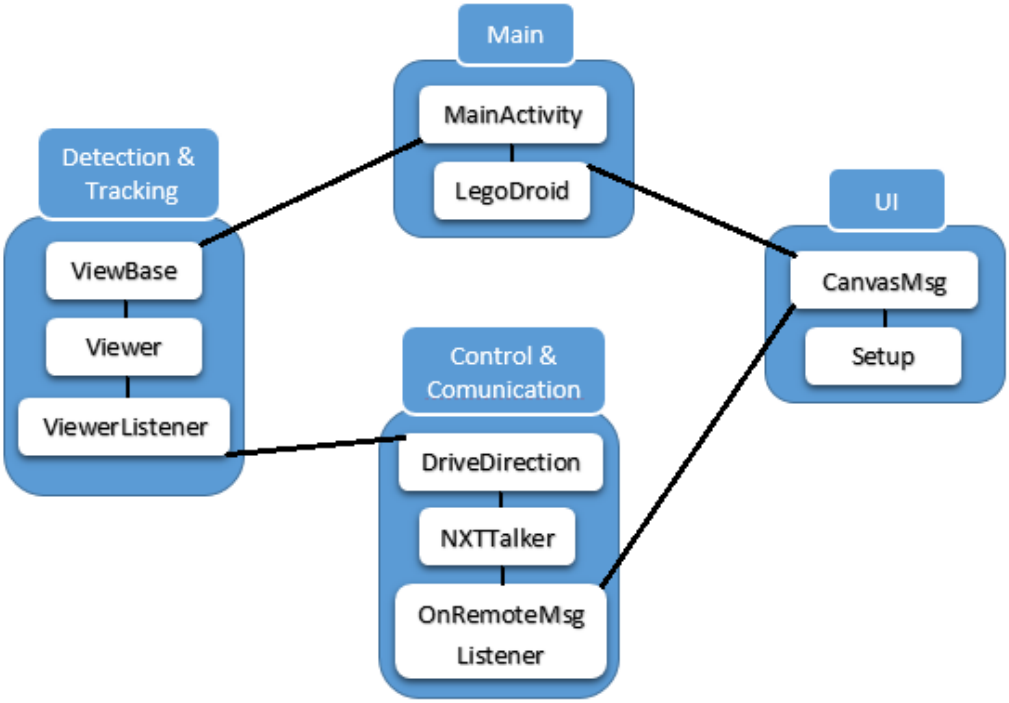


Figura 4.31: Diagrama de clases

- **Organización del código**

Se ha separado el código en 4 paquetes claramente diferenciados. A continuación se detalla que clases componen cada paquete y que función realizan.

- **Main Activities**

En Android las activities son las clases que gestionan la vista de la aplicación. En este paquete hay 2activities:

- MainActivity: Es la actividad principal que se ejecuta al iniciar la aplicación. Carga las librerías externas (OpenCV) e inicializa las variables, después pasa a la actividad *LegoDroid*.
- LegoDroid: Contiene el menú principal donde se realiza la conexión con el lego NXT. Además se encarga de recibir los frames de la cámara del dispositivo y mostrarlos en pantalla.

- **Control & Comunication**
  Contiene las clases necesarias para implementar el control del robot y la comunicación mediante Bluetooth.
- DriveDirection: Calcula los parámetros que deben ser enviados al robot en función de la información de la visión artificial.
- NXTTalker: Esta clase es la encargada de enviar los comandos al robot.
- OnRemoteMsgListener: Establece la comunicación entre las distintas clases para el envío de información.

- **Detection & Tracking**
- ViewBase: Establece la cámara como sistema de visión.
- Viewer: Esta clase implementa las funciones básicas para realizar la detección de un objeto por color o reconocimiento facial.
- ViewerListener: Comunica a *DriveDirection* lo que computa la visión para su procesado.

- **User Interface**
- CanvasMsg: Esta clase es la encargada de mostrar por pantalla la información, como la posición del objeto, el centro de la pantalla, etc.
- Setup: Proporciona todas las opciones de búsqueda, selector de color, tamaño de pantalla, etc.

Figura 4.31: Menú principal de la aplciación

Procesado de imágenes en Android

Como se ha comentado anteriormente, la captación y el procesado de imágenes se realizarán 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.

- ## 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 interfaz SurfaceView. La definición de la clase queda de la siguiente forma:

public class LegoDroid extends Activity implements SurfaceView

Esta interfaz 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.

- ## Algoritmos de detección del objeto

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 4.32). 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.

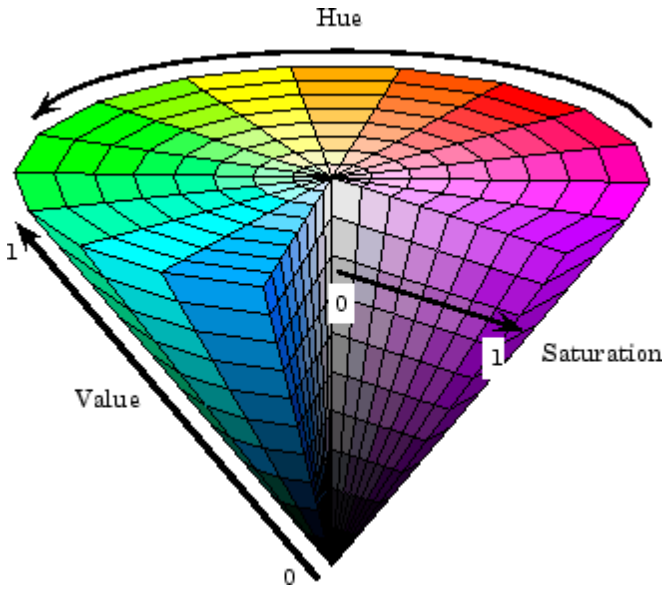


Figura 4.32: Cono que define los tres parámetros del modelo HSV: Hue, Saturation y value

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.

Figura 4.33: Imagen RGB y su correspondiente HSV

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 al 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 );
```

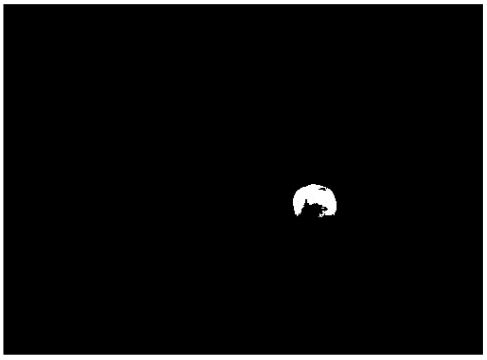


Figura 4.34: Imagen resultante del proceso de filtrado mediante threshold inferior y superior. Se puede ver

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 );
```

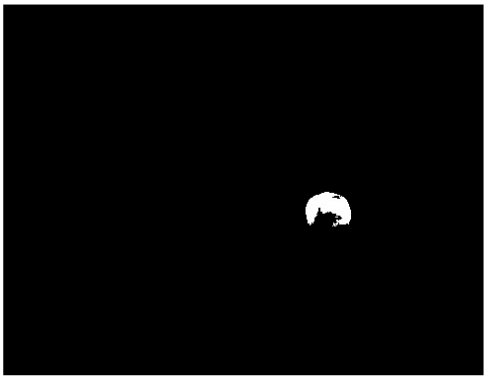


Figura 4.35: Imagen resultante del proceso de binarización. En blanco se puede ver la región del objeto.

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 más rápido.

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, utilizando la función *findcontours* proporcionada por OpenCV y aproximando el área a un rectángulo cogiendo los valores de x e y máximo y mínimos, solucionando así el problema de luminosidad (Figura 4.36):

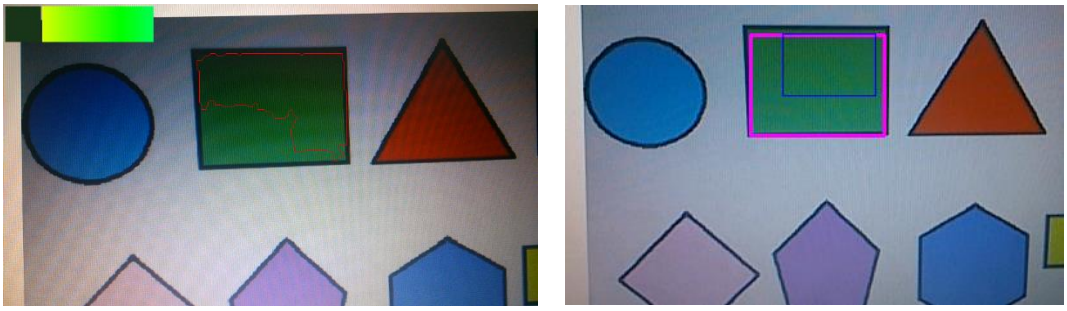


Figura 4.36: Aproximación del área del objeto a un rectángulo

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 potencia 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:

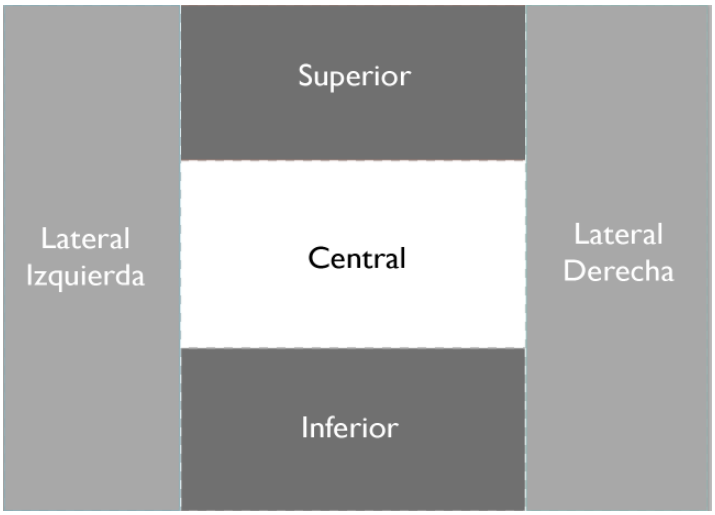


Figura 4.37: Regiones de visión

Como se puede ver, la imagen se separa en 3 regiones:

1) **Lateral izquierda/derecha:** Si el objetivo no está en el centro de la imagen, se le envía al robot que gire en plano horizontal en la dirección correspondiente.
2) **Superior/Inferior:** En esta región el robot se moverá en el plano vertical para localizar el objetivo.
3) **Central:** Esta región corresponde la centro de la imagen, si existe una intersección entre el contorno del objeto y el rectángulo central el robot mantendrá la posición.

En cambio, si el objeto no está en ninguna de estas regiones ordenará al robot que se ponga en modo búsqueda.

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 objeto en la imagen.

- ## 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 );
```

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.

- **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 4.38 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 4.38: 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 más 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 4.39: 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 órdenes 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.

- **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.

CAPÍTULO 5

OBSERVACIONES Y RESULTADOS OBTENIDOS

"No entiendes realmente algo a menos que seas capaz de explicárselo a tu abuela."
Albert Einstein

5.1. Introducción

En este capítulo se comentan los problemas surgidos durante la realización de las pruebas y los resultados de las mismas. Se han llevado a cabo decenas de pruebas de las que solo comentaremos de las que mejores resultados se obtuvieron. Una vez tengamos todos los datos se hará una comparativa de los sensores para los distintos modelos cinemáticos. Además se comprobará la eficacia del sistema de seguimiento de objetivo mediante visión artificial.

5.2. Estimación de la posición

5.2.1. Acelerómetro

Se nos pide comúnmente si es posible utilizar las mediciones del acelerómetro para estimar la velocidad y la posición. La respuesta corta es "sí y no". Todo depende de cuánta precisión se necesita. En general, la estimación de la velocidad y la posición basada en acelerómetro de bajo coste (menos de 1000€) son muy pobres y normalmente inutilizables. Esto no se debe a los propios acelerómetros son pobres, sino porque la orientación del sensor debe ser conocida con un alto grado de precisión para que las mediciones de gravedad se puede distinguir de la aceleración física del sensor. Incluso los pequeños errores en la estimación de la orientación producirán extremadamente altos errores en la aceleración medida, que se traducen en errores aún mayores en las estimaciones de velocidad y posición. Sin giroscopios de alta precisión o sin añadir referencias externas como GPS, una precisa navegación por estimación no es posible. Sin embargo, no todas las aplicaciones requieren una gran cantidad de precisión, y, a veces exactitud absoluta no es tan importante como la capacidad de medir desviaciones de corto plazo en la velocidad y la posición.

Un acelerómetro es un dispositivo que mide la aceleración de un elemento. No está necesariamente relacionado con el cambio de velocidad, sino que a veces se asocia con el fenómeno de peso experimentado por una masa de referencia conocida por el dispositivo de medida. Generalmente, el acelerómetro mide la aceleración mediante la medida de cuánto presiona la masa sobre algo cuando una fuerza actúa sobre ella. La fuente de error principal de un acelerómetro es el

"bias" (m/s^2) (offset de la señal de salida sobre el valor real), el cual difiere de cada acelerómetro concreto. Un error de bias constante de valor ε provoca un error en la posición que al ser doblemente integrado supone un crecimiento cuadrático a lo largo del tiempo. El error acumulado a lo largo del tiempo en la posición estimada es: $s(t) = \varepsilon \cdot \frac{t^2}{2}$, siendo t el tiempo en el que se considera. Es posible estimar el valor del error bias mediante una medida del valor medio de la salida del acelerómetro a largo plazo cuando no se experimenta ninguna aceleración. Con ello, se puede recalibrar la medida de aceleración periódicamente, de forma que el error se reduzca. Los errores de bias no corregidos son los que limitan, principalmente, el rendimiento del sistema inercial de localización.

La Figura 5.1. resume los errores de aceleración, velocidad y posición que se puede esperar teniendo en cuenta diferentes errores en la estimación de la orientación del sensor.

Angle Error (degrees)	Acceleration Error (m/s/s)	Velocity Error (m/s) @ 10 seconds	Position Error (m) @ 10 seconds	Position Error (m) @ 1 minute	Position Error (m) @ 10 minutes	Position Error (m) @ 1 hour
0.1	1.71	0.2	1.7	62	6157	221.6 E 3
0.5	8.55	0.9	8.6	308	30.8 E 3	1.1 E 6
1.0	17.11	1.7	17.1	615	61.6 E 3	2.2 E 6
1.5	25.66	2.6	25.6	924	92.4 E 3	3.3 E 6
2.0	34.22	3.4	34.2	1232	123.2 E 3	4.4 E 6

Figura 5.1: Se espera exactitud de las estimaciones de velocidad y de posición

Como se muestra, un error de ángulo de un grado hará que la velocidad estimada varie por 1,7 m / s después de 10 segundos, y la posición por 17,1 metros en la misma cantidad de tiempo. Después de un minuto, un grado de error del ángulo hará que la estimación de la posición de desvie por casi un kilómetro completo. Después de diez minutos, la posición será de 62 kilómetros.

En resumen, es posible utilizar los acelerómetros para estimar la velocidad y la posición, pero la exactitud será muy pobre, así que en la mayoría de los casos, las estimaciones no son útiles sin la utilización de sensores con un elevado precio más una fusión sensorial y/o usos de filtros como el de Kalman. Por estos motivos se ha decido no utilizar el acelerómetro en esteproyecto.

5.2.2. Prototipo Diferencial

A continuación se muestran las trayectorias realizadas por el prototipo diferencial con el uso de encoders y de una brújula.

- **Encoder**

  - **Cuadrado**

Como se observa en la Figura 5.2, la posición real del robot va desviándose de la trayectoria debido fundamentalmente al deslizamiento con el suelo, mientras que la odometría al seguir contando revoluciones considera que está en la posición correcta.

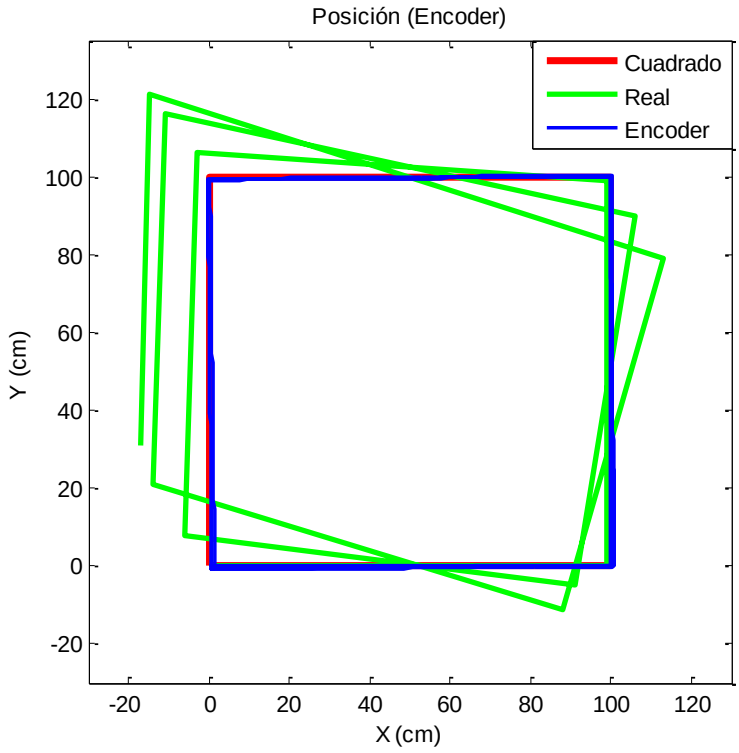


Figura 5.2: Trayectoria Cuadrada con Encoder

Centrándonos en una de las esquinas vemos como la trayectoria estimada por la odometría va cometiendo un pequeño error mientras que la real va acumulando un error más grande (Figura 5.3):

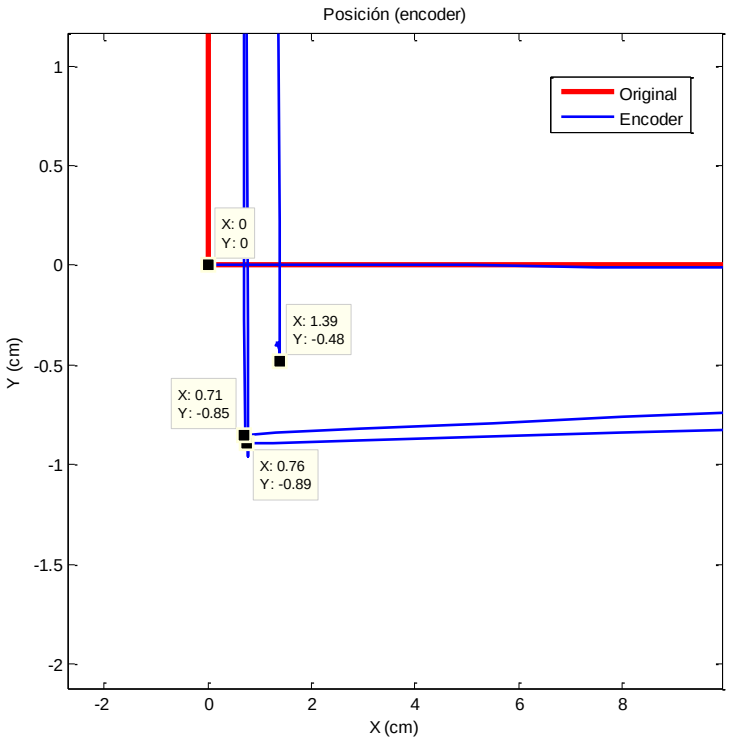


Figura 5.3: Detalle inicio trayectoria cuadrado

Observando la gráfica de Ángulo vs Tiempo se ve la orientación del robot en cada vuelta de la trayectoria (Figura 5.4):

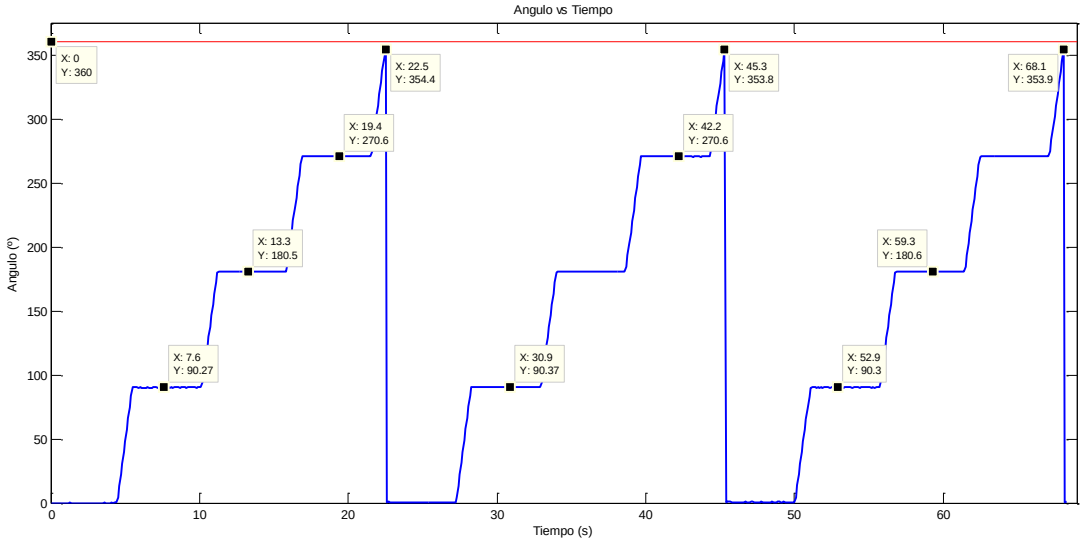


Figura 5.4: Cuadrado - Ángulo vs Tiempo

Se puede apreciar como nunca llega a girar los 360 grados por lo que irá cometiendo un pequeño error.

Una vez tenemos todos los datos vemos la posición final y el error relativo medio cometido:

Vuelta	Estimado			Real		
	x	y	h	x	y	h
Final (cm)	1.39	-0.48	353.8	-17	31	20.5
Er (%)	1.2	0.9	1.7	11	14	6

- **Triángulo**

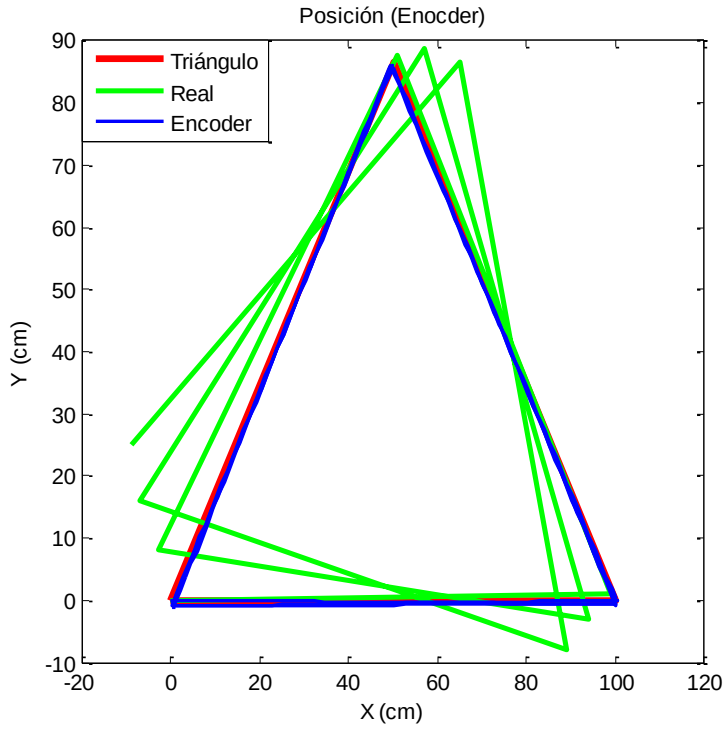


Figura 5.5: trayectoria triángulo

Observando la gráfica, la estimación de la posición para la trayectoria triángular parece levemente mejor que la trayectoria cuadrada, para ver con más detalle el recorrido nos centramos en la esquina inferior izquierda (Figura 5.6):

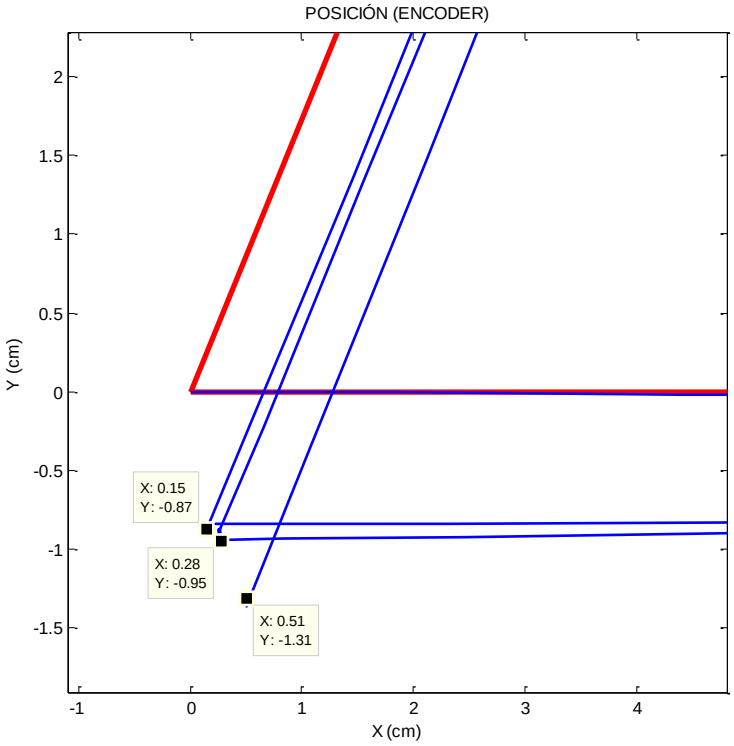


Figura 5.6: Detalle trayectoria triángulo

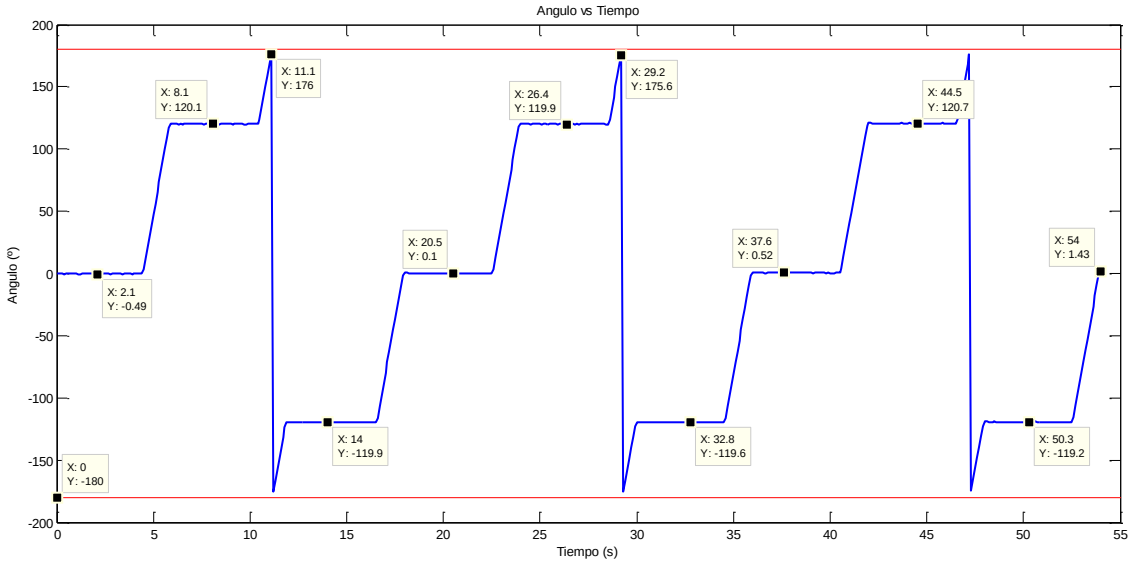


Figura 5.7: Triángulo - Ángulo vs tiempo

Una vez tenemos todos los datos vemos la posición final y el error relativo medio cometido:

Vuelta	**Estimado**			**Real**		
	x	y	h	x	y	h
Final (cm)	0.51	1.31	1.43	-9	25	16
Er (%)	0.5	0.7	1.9	9	10.2	4.5

- **Círculo**

Quizás la trayectoria circular sea la más facil para un robot diferencial ya que solo se necesita que giren sus motores a diferentes velocidades constantes durante todo el recorrido:

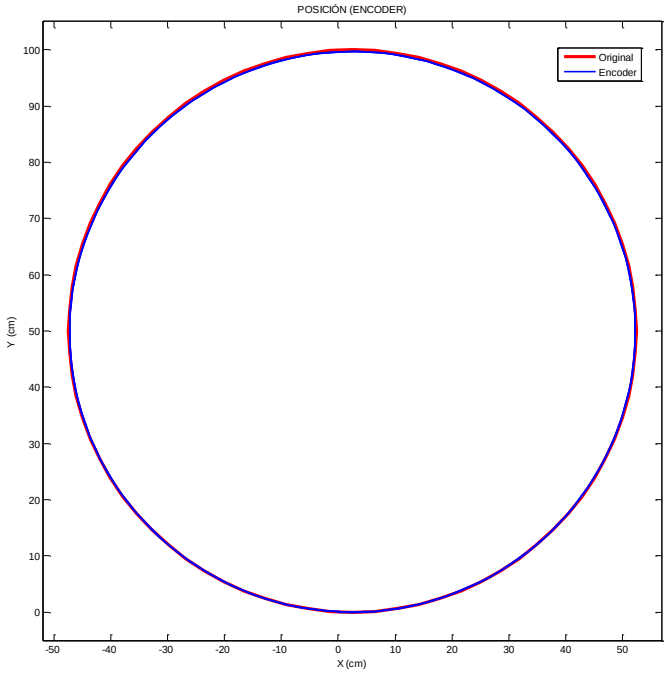


Figura 5.8: Trayectoria circular

Para ver el error de la estimación vemos detalladamente una parte de la curva:

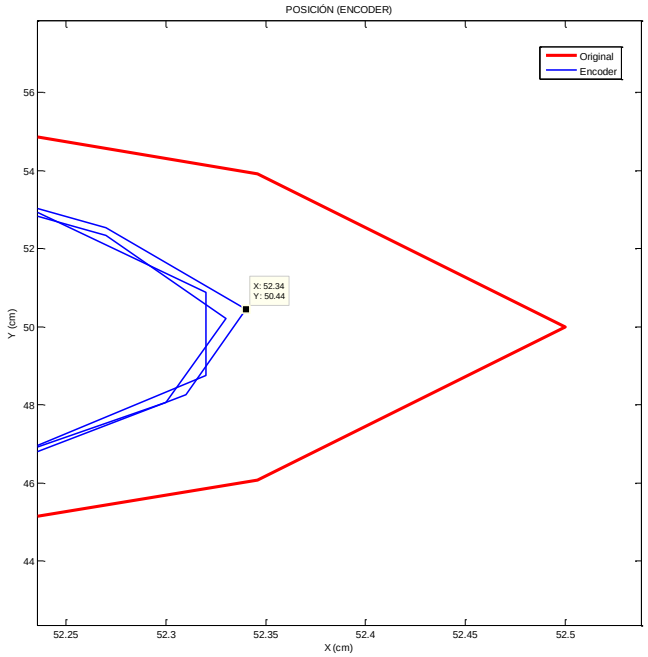


Figura 5.9: Detalle trayectoria circular

El ángulo estimado va de 0 a 360 y en la posición final toma un valor de -2º.

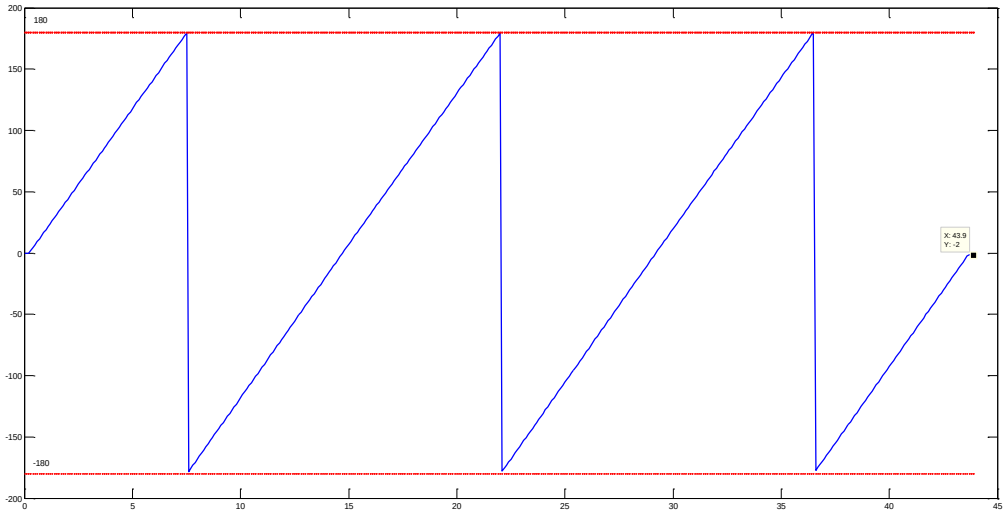


Figura 5.10: Círculo - ángulo vs tiempo

Una vez tenemos todos los datos vemos la posición final y el error relativo medio cometido:

Vuelta	**Estimado**			**Real**		
	x	y	h	x	y	h
Final (cm)	-4.3	0.35	-2	-24	8	13
Er (%)	0.7	0.6	0.7	0.8	0.7	0.9

- **Lemniscata**

Para realizar la lemniscata se ha deformado la trayectoria original y se ha reconvertido en 2 semicircunferencias de radio 25 cm y 2 rectas de 71 cm ya que el robot solo consta de 2 movimientos, en línea recta o sobre si mismo.

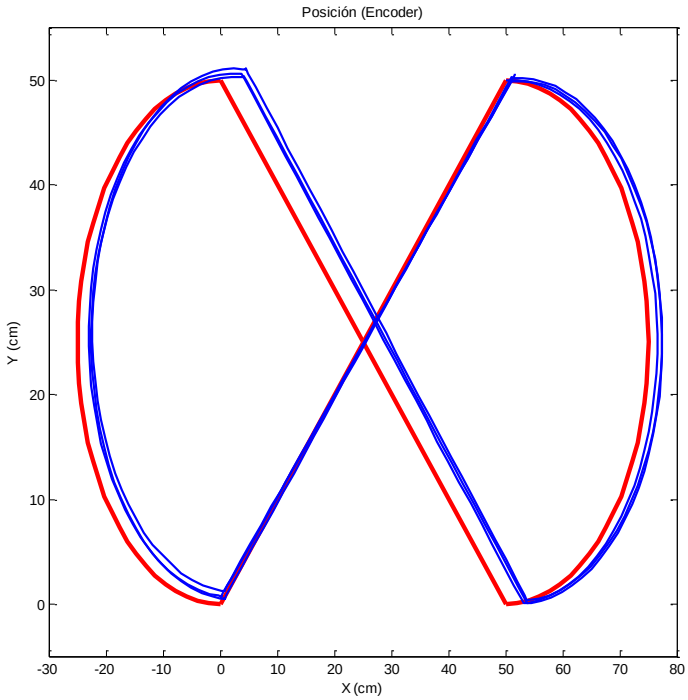


Figura 5.11: Trayectoria lemniscata

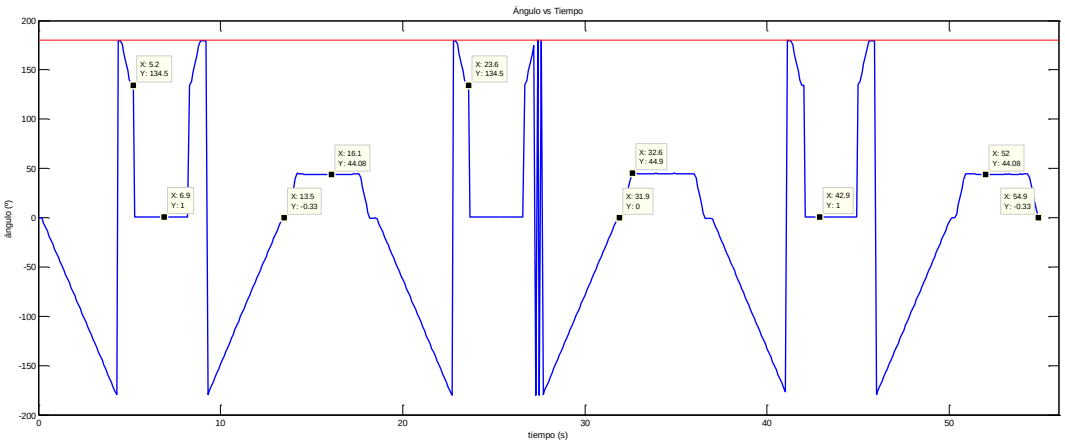


Figura 5.12: Lemniscata - ángulo vs tiempo

Una vez tenemos todos los datos vemos la posición final y el error relativo medio cometido:

Vuelta	**Estimado**			**Real**		
	x	y	h	x	y	h
Final (cm)	0.76	0.6	-0.33	9	14	5
Er (%)	8	5	0.2	22	17	9.2

▪ Brújula

A continuación se exponen los resultados obtenidos mediante el uso de una brújula. La propia empresa Hitechnic nos avisa de colocar a una distancia de seguridad (unos 15 cm) el sensor para no tener interferencias electromagnéticas (principalmente los motores y el propio brick) y dar resultados erróneos. Aún habiendo cumplido este requisito el sensor ha proporcionado una medida distorsionada durante las pruebas que no he podido resolver como se en la Figura 5.15. Como consecuencia la trayectoria real se vio más distorsionada por el giro excesivo del robot.

• Cuadrado

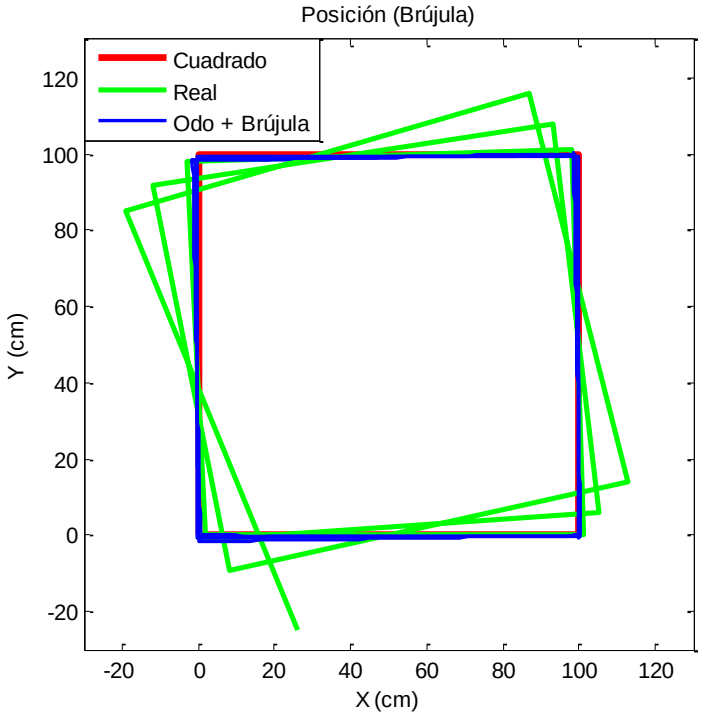


Figura 5.13: trayectoria cuadrado

Centrándonos en la esquina inferior vemos el resultado final en cada vuelta (Figura 5.14):

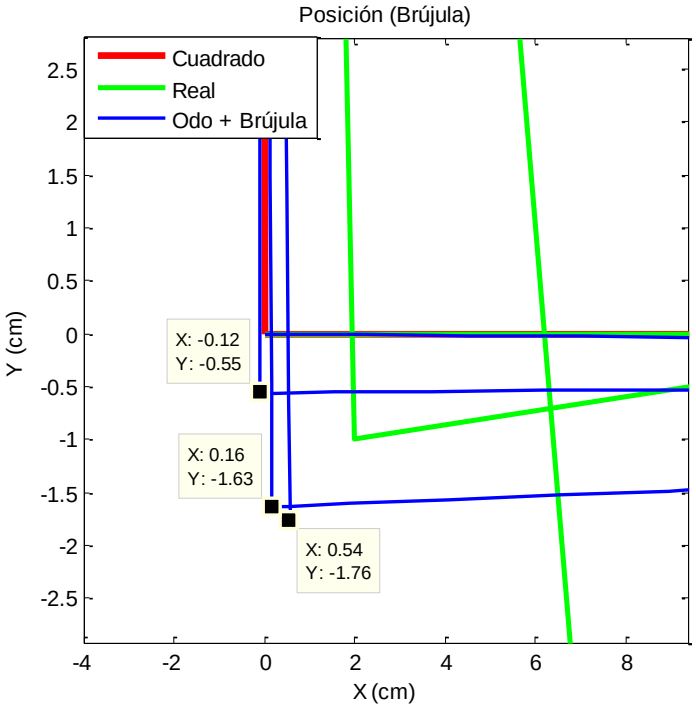


Figura 5.14: Detalle trayectoria cuadrado

Como se ha cometando, la medición de la brújula presenta un ruido continuo (figura 5.15):

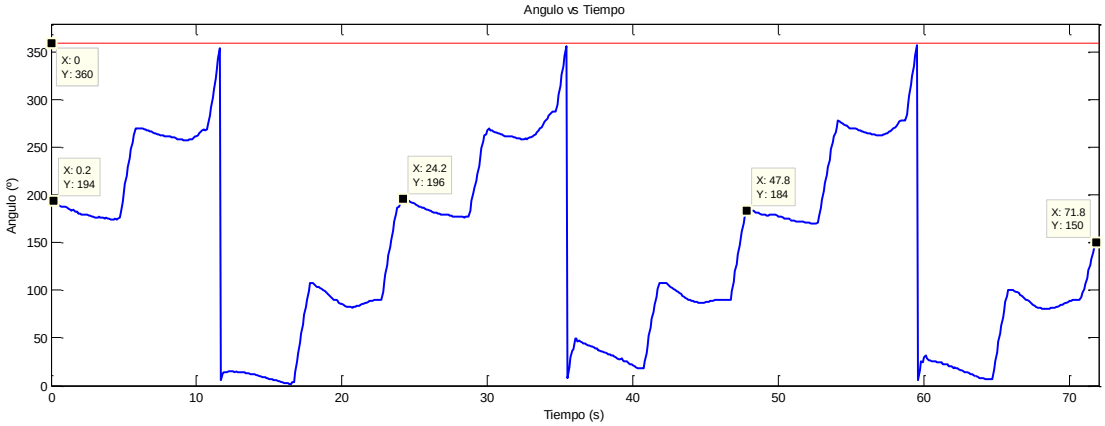


Figura 5.15: brújula - ángulo vs tiempo

Una vez tenemos todos los datos vemos la posición final y el error relativo medio cometido:

Vuelta	Estimado			Real		
	x	y	h	x	y	h
Final (cm)	0.54	-1.76	150	25	-26	221
Er (%)	0.8	1.2	1.6	12	15	16.6

- **Triángulo**

Al igual que en la trayectoria cuadrada, por un sobregiro del robot la trayectoria real acumula un mayor error sumado a los problemas físicos ya comentados (Figura 5.16):

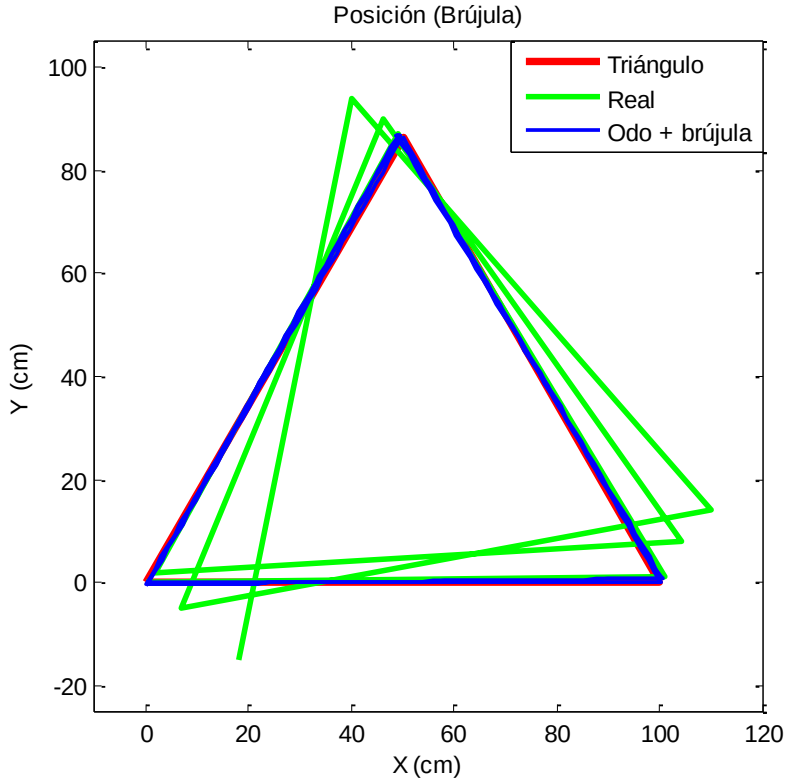


Figura 5.16: trayectoria triángulo

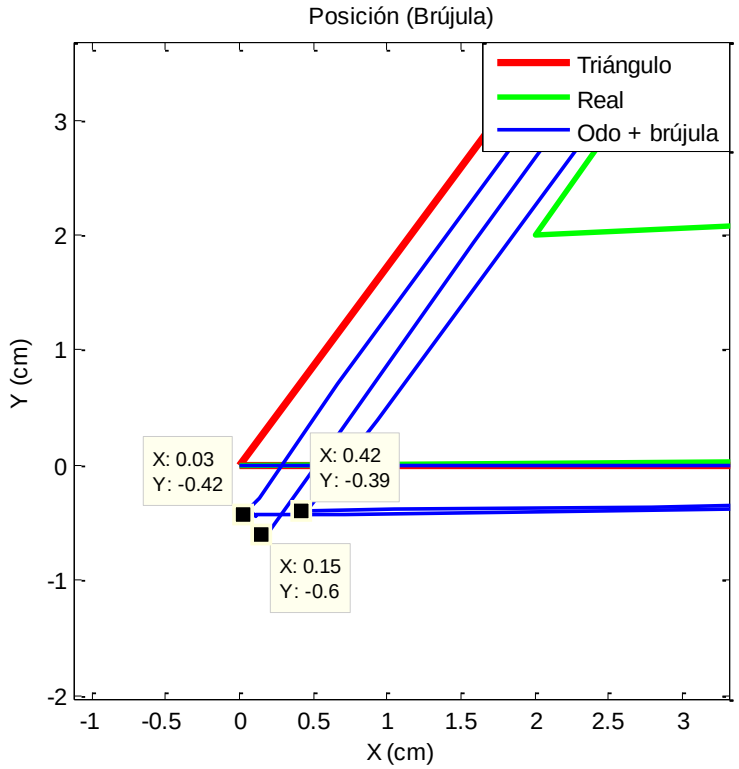


Figura 5.17: detalle trayectoria triángulo

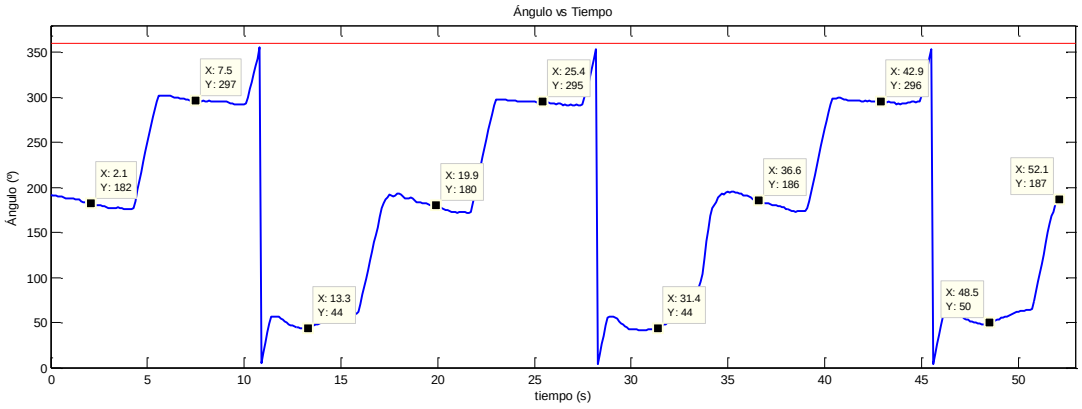


Figura 5.18: triángulo - ángulo vs tiempo

Una vez tenemos todos los datos vemos la posición final y el error relativo medio cometido:

Vuelta	**Estimado**			**Real**		
	x	y	h	x	y	h
Final (cm)	0.15	-0.6	187	18	-15	213
Er (%)	0.6	0.8	2	14	11	9

- **Círculo**

Para poder diferencias todas las trayectorias se ha decido marcar con un punto la posición real del robot ya que es igual que la estimada.

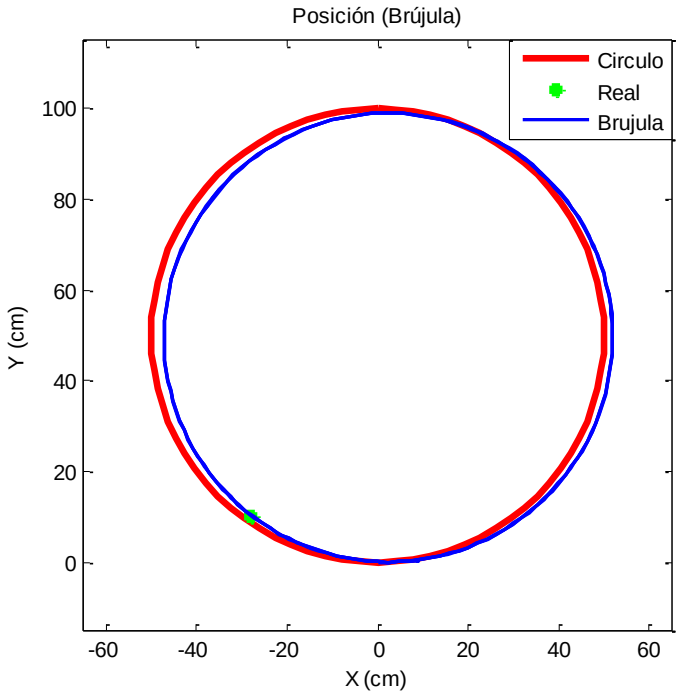


Figura 5.19: trayectoria círculo

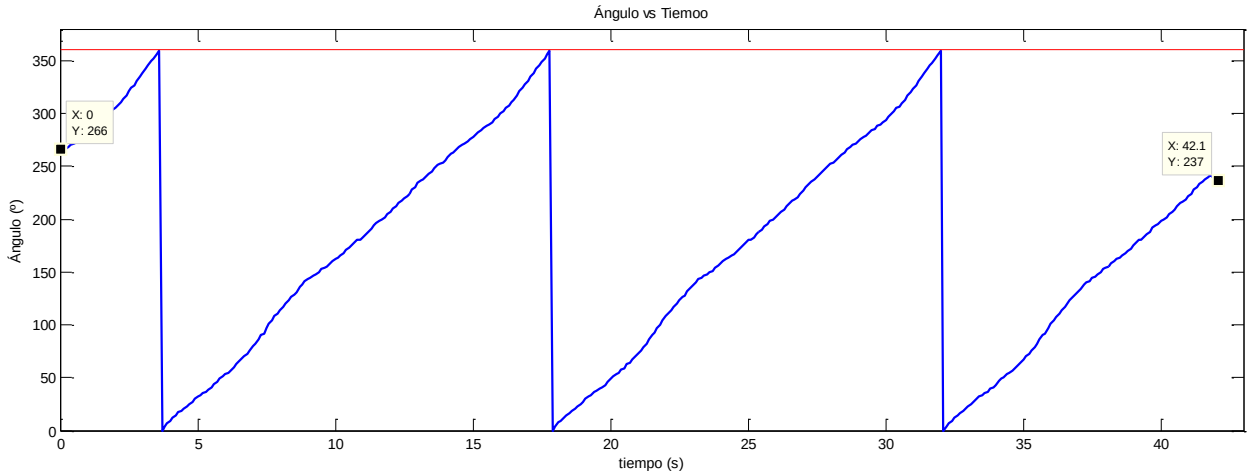


Figura 5.20: círculo - ángulo vs tiempo

Una vez tenemos todos los datos vemos la posición final y el error relativo medio cometido:

Vuelta	Estimado			Real		
	x	y	h	x	y	h
Final (cm)	-8.7	0.31	237	-28	10	205
Er (%)	3.2	0.5	0.1	3.6	0.8	0.2

- **Lemniscata**

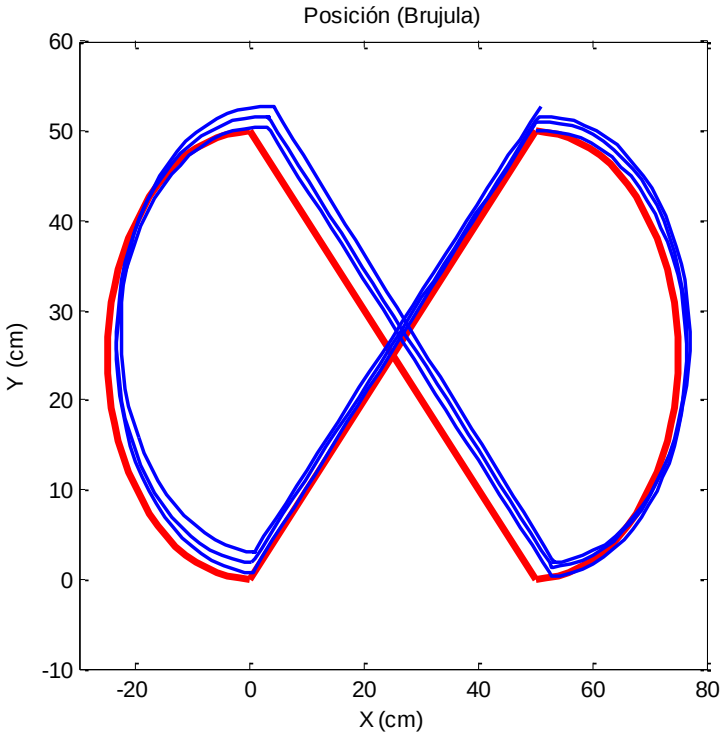


Figura 5.21: trayectoria lemniscata

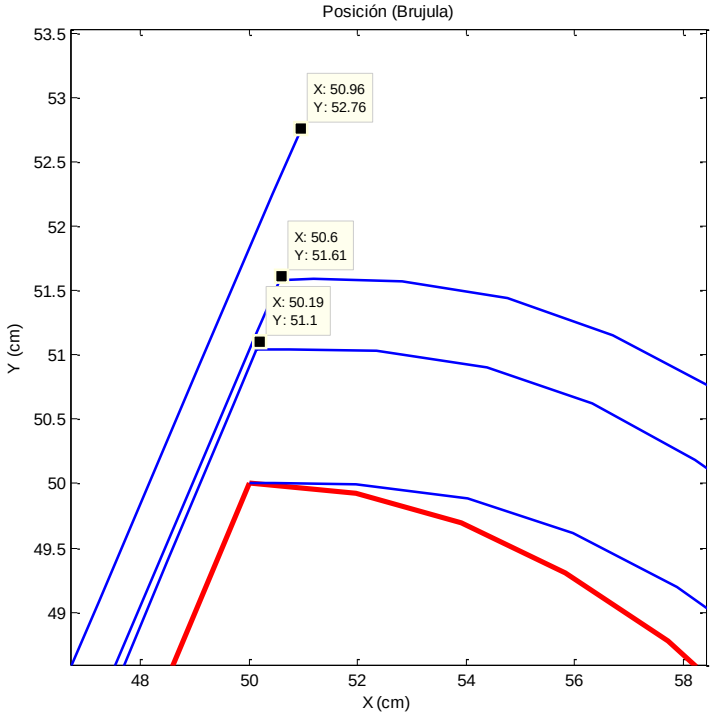


Figura 5.22: detalle lemniscata

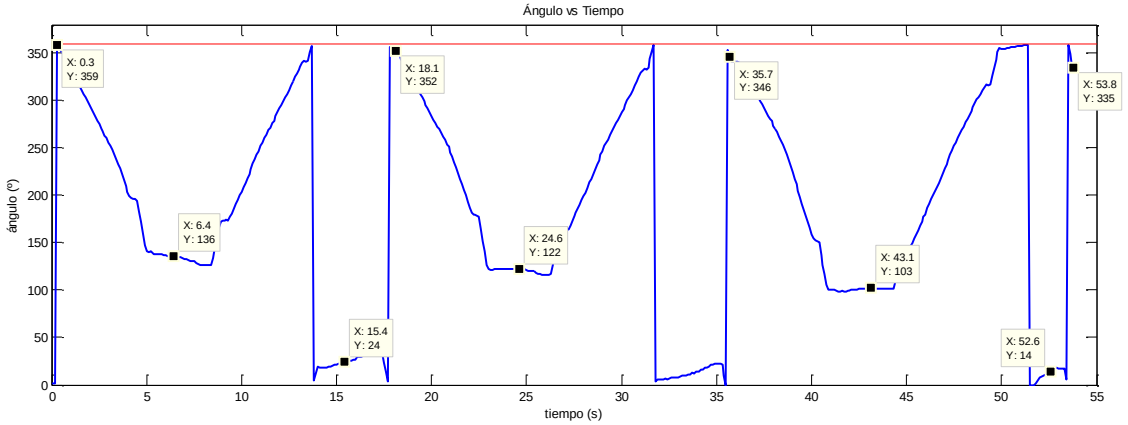


Figura 5.23: lemniscata - ángulo vs tiempo

Una vez tenemos todos los datos vemos la posición final y el error relativo medio cometido:

Vuelta	**Estimado**			**Real**		
	x	y	h	x	y	h
Final (cm)	1	2.76	335	12	17	307
Er (%)	10	6	0.5	23	19	11

5.2.3. Prototipo Ackerman

- **Encoder**

  - **Cuadrado**

Para simplificar la representación de la trayectoria real se han rectificado las curvas uniendo el punto de inicio y fin de la misma ya que la información perdida no difiere el resultado final.

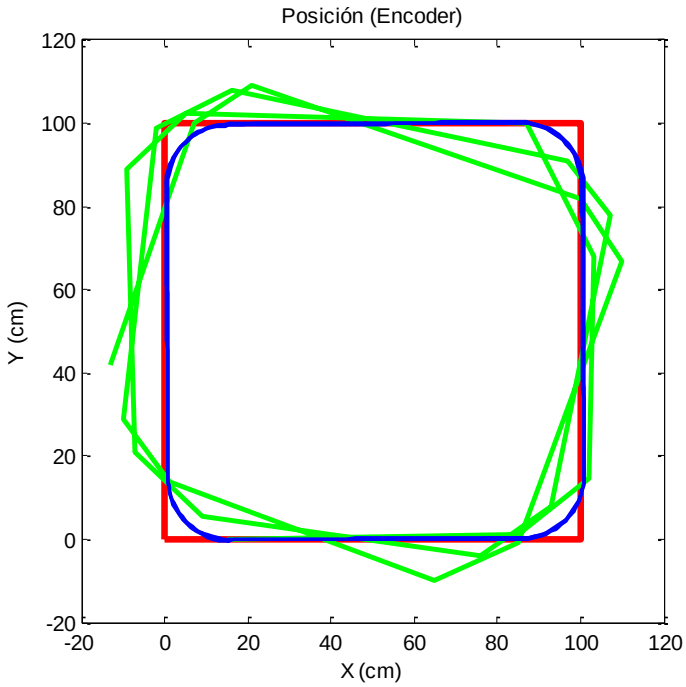


Figura 5.24: Trayectoria cuadrado

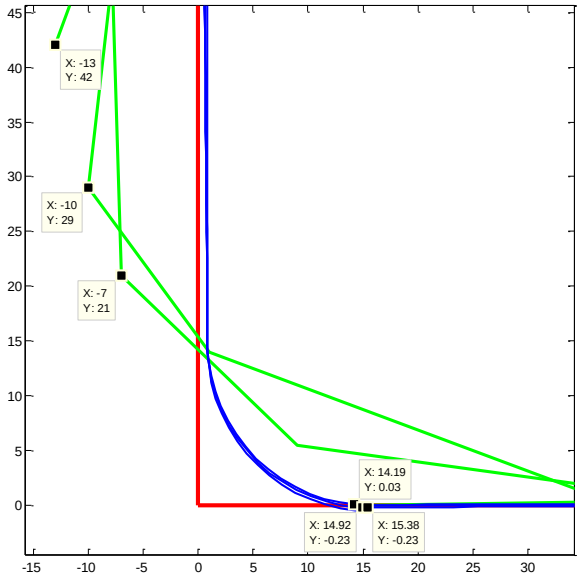


Figura 5.25: detalle cuadrado

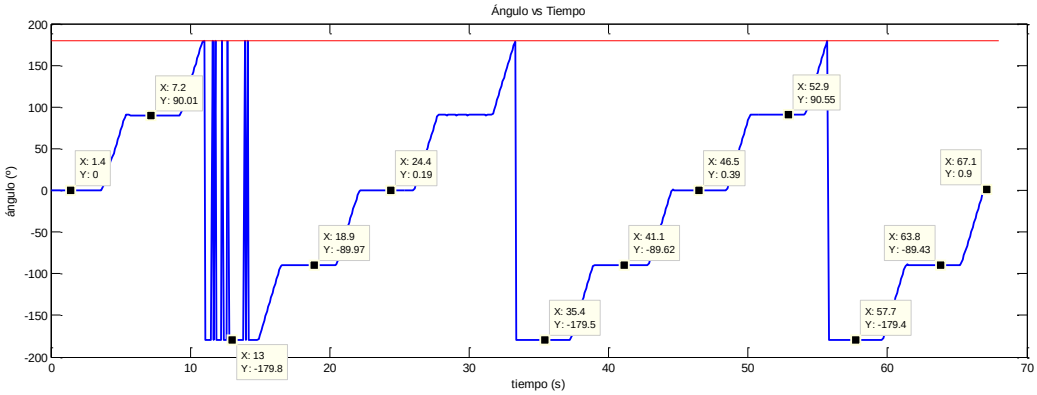


Figura 5.26: cuadrado - ángulo vs tiempo

Vuelta	Estimado			Real		
	x	y	h	x	y	h
Final (cm)	0.38	-0.23	0.9	-13	42	55
Er (%)	0.28	0.17	0.5	11	17	18

- **Triángulo**

Debido al ángulo de giro mínimo de este prototipo y por no reducir la distancia recorrida se ha decidido bordear por el exterior la trayectoria.

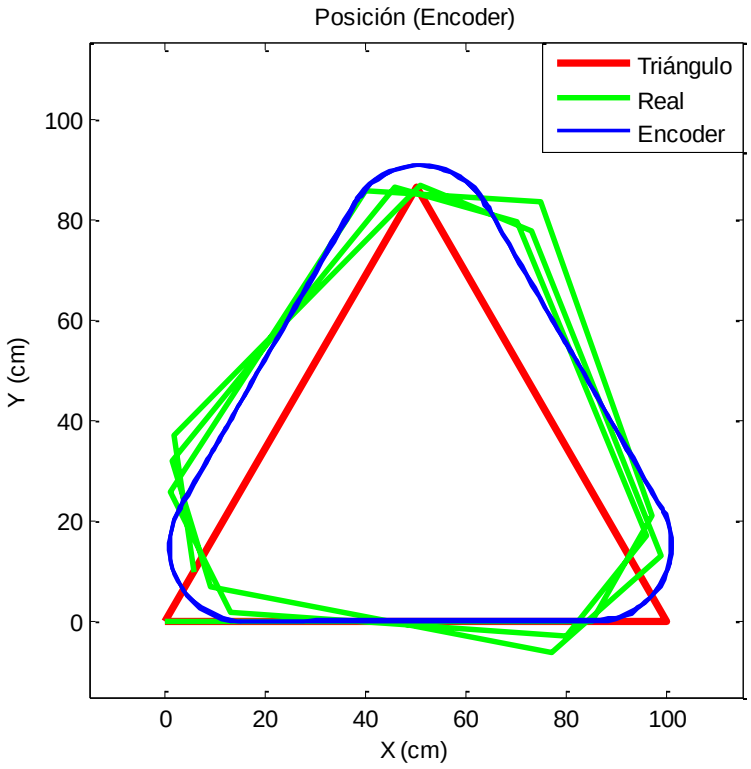


Figura 5.27: trayectoria triángulo

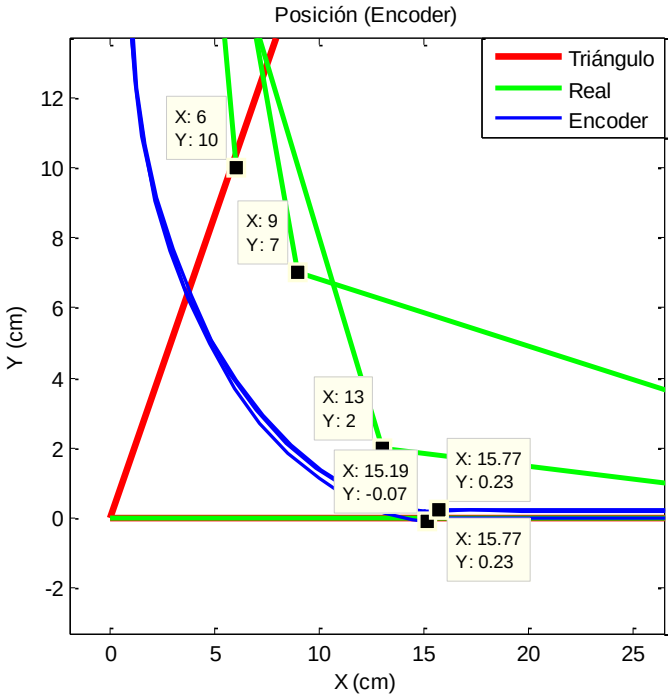


Figura 5.28: detalle trayectoria triangular

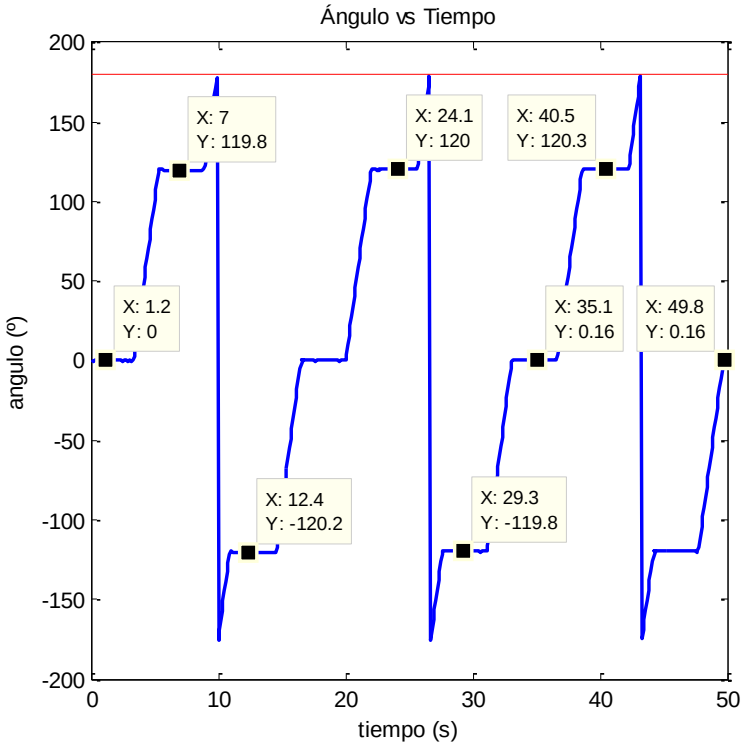


Figura 5.29: triangulo - ángulo vs tiempo

Vuelta	Estimado			Real		
	x	y	h	x	y	h
Final (cm)	0.77	0.23	0.16	6	10	21
Er (%)	0.5	0.2	0.1	5	8	9

- **Círculo**

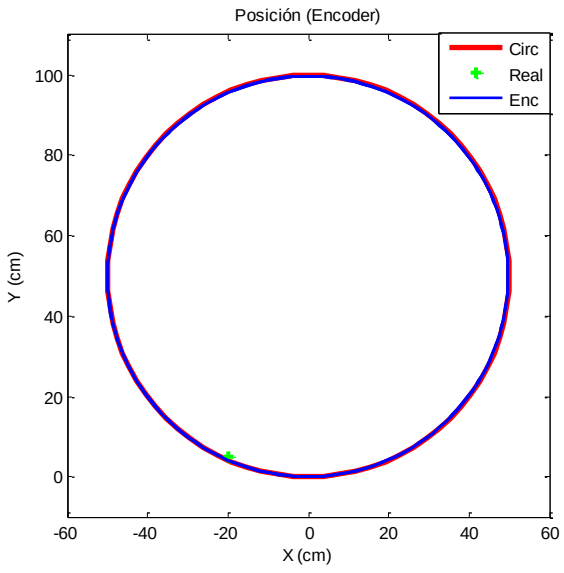


Figura 5.30: trayectoria circulo

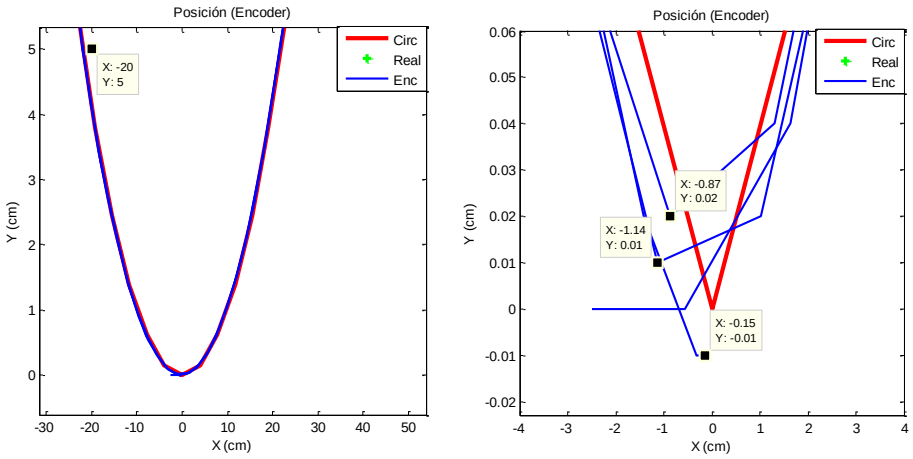


Figura 5.31: detalle trayectoria circular

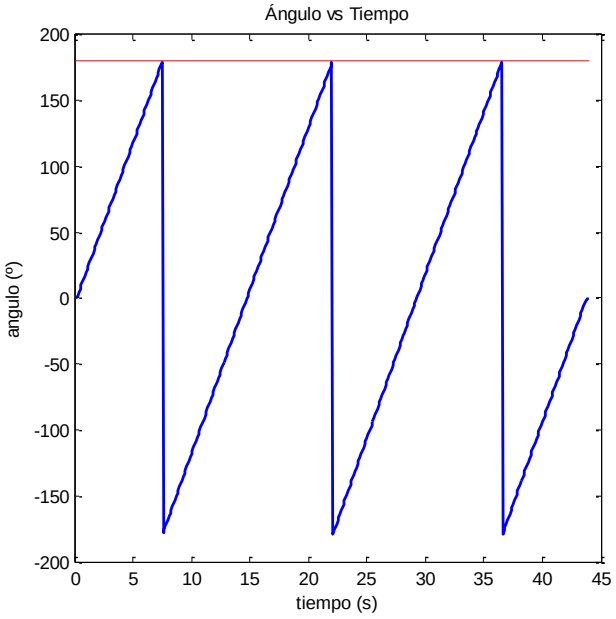


Figura 5.32: círculo - ángulo vs tiempo

Vuelta	Estimado			Real		
	x	y	h	x	y	h
Final (cm)	-0.15	0	0	-20	5	14
Er (%)	0	0	0	0.2	0.09	0.1

- **Lemniscata**

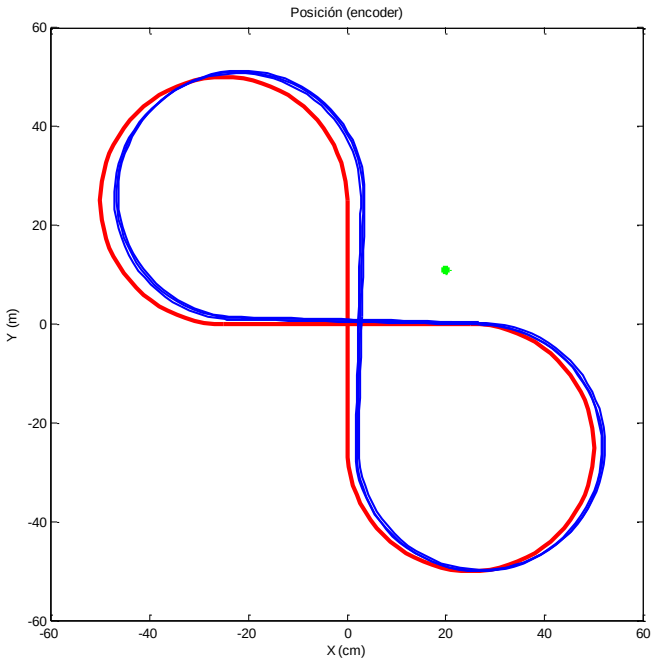


Figura 5.33: trayectoria lemniscata

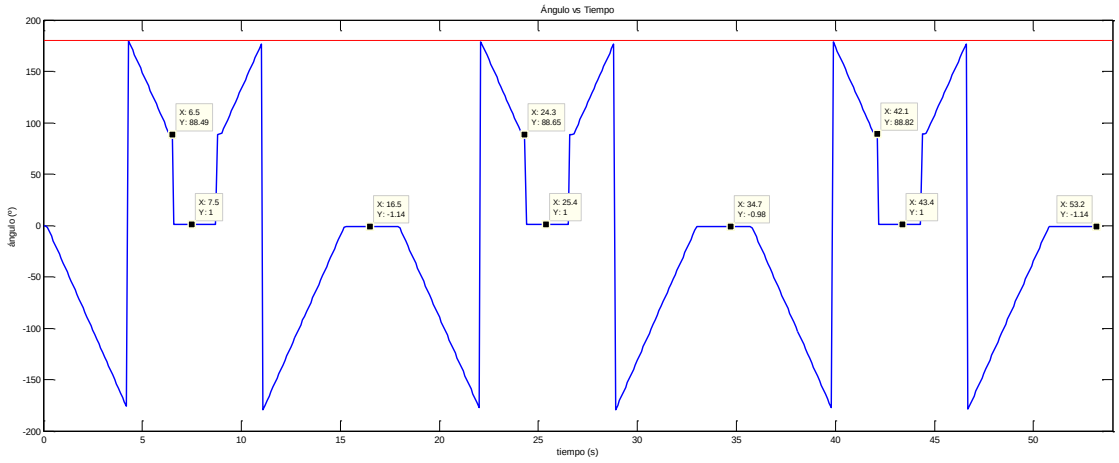


Figura 5.34: lemniscata - ángulo vs tiempo

Vuelta	**Estimado**			**Real**		
	x	y	h	x	y	h
Final (cm)	1.62	0.36	-0.82	20	11	14
Er (%)	6	3	1	16	9	7

5.3. Comparación de resultados

A continuación se resumen en una tabla los errores relativos en cada prueba realizada. Por causas desconocidas durante la realización de las pruebas la brújula presento un ruido que desvió su medida unos grados, con la consecuencia de unos peores resultados.

- **Diferencial**

Error (%)	Cuadrado			Triángulo			Círculo			Lemniscata		
Encoder	1.2	0.9	1.7	0.5	0.7	1.9	0.7	0.6	0.7	8	5	0.2
Brújula	0.8	1.2	1.6	0.6	0.8	2	3.2	0.5	0.1	10	6	0.5

- **Diferencial con torreta**

Error (%)	Cuadrado		
Encoder	1.9	0.8	1.2

- **Ackerman**

Error (%)	Cuadrado			Triángulo			Círculo			Lemniscata		
Encoder	0.28	0.17	0.5	0.5	0.2	0.1	0	0	0	6	3	1

Como se puede observar no existe una gran diferencia entre los resultados obtenidos con los diferentes sensores, si bien es cierto que el modelo cinemático Ackerman ha proporcionado resultados más precisos. Cabe pensar que el uso de una brújula aporta mejores resultados al no depender de los parámetros físicos del robot, pero por el problema comentado en este proyecto la diferencia ha sido mínima.

Las principales causas de los errores en la estimación de este proyecto han sido:

- Deslizamiento de las ruedas.
- Modificación de los parámetros del robot durante la ejecución (por el peso añadido de la torreta las ruedas se deformaban).

5.4. Seguimiento de Objetivo

5.4.1. Seguimiento de objetivo por color

Realizando un sencillo experiento se comprueba la eficacia de la visión artificial:

En la siguiente imagen (Figura 5.35) se ilustran diferentes figuras geométricas con distintos tamaños y colores:

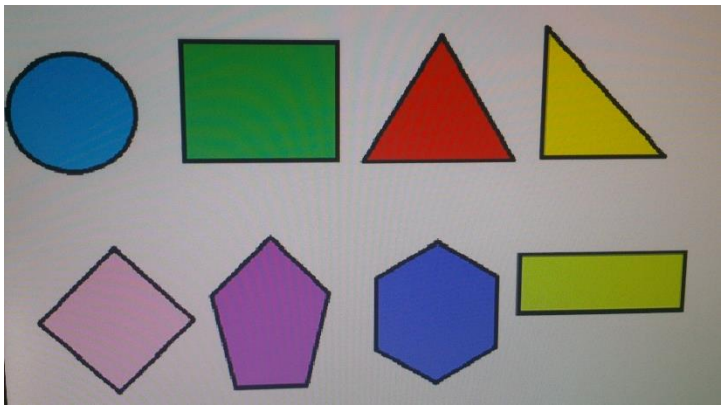
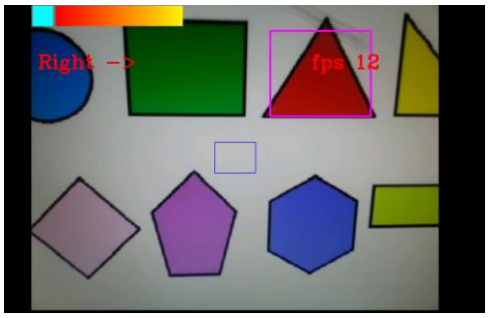
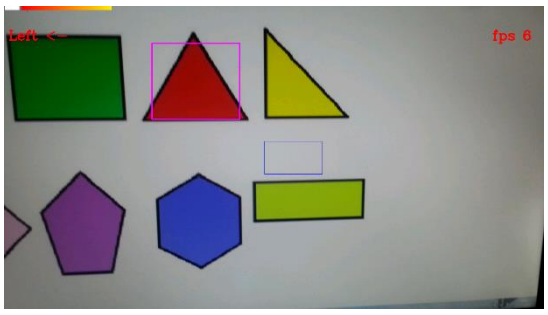
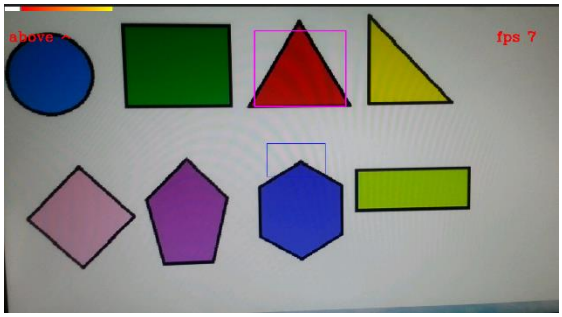


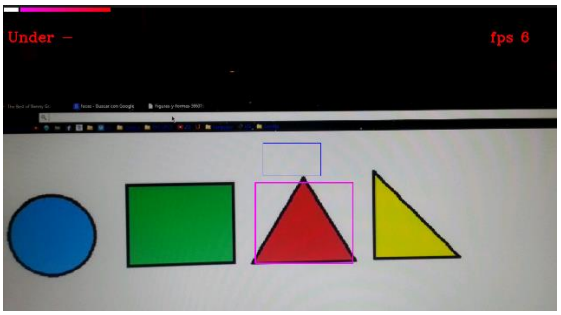
Figura 5.35: figuras geométricas para reconocimiento

En el smartphone se selecciona el color a seguir pinchando sobre él en la pantalla y nos dice en qué lugar está el objeto respecto al centro, en el momento que exista una intersección entre los rectángulos central y contorno del objeto se considera detectado (Figura 3.36):









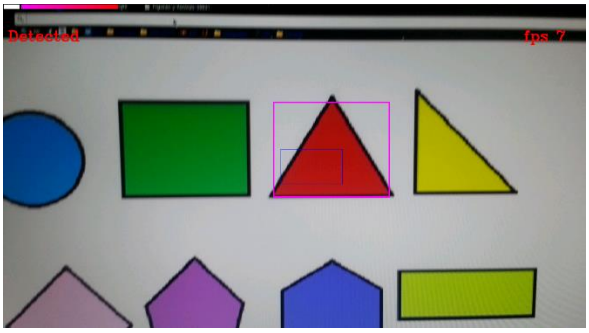


Figura 5.36: Localización objeto con color

Una vez el objeto esté en pantalla, el smartphone envía los comandos al brick de lego para su seguimiento, sino pasará al modo búsqueda.

El objetivo de este proyecto es el seguimiento de un objeto móvil en el espacio:

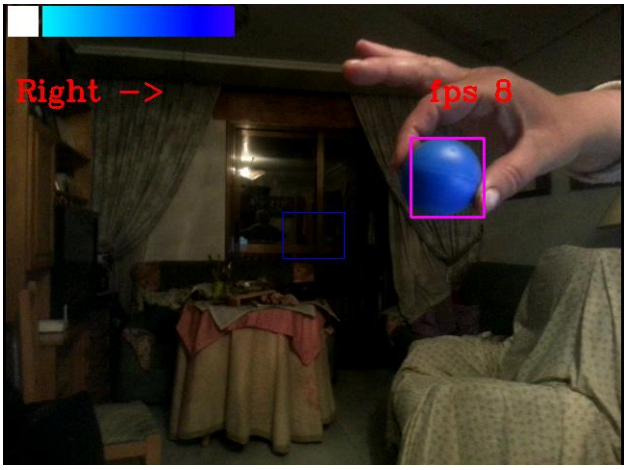


Figura 5.36 Detección y seguimiento de un objeto móvil

Como se observa en la Figura 5.36, el sistema de visión nos comunica donde se encuentra el objeto, por lo que mediante un simple control proporcional podemos ordenar a los motores que giren hacia la derecha en este caso. Un control PID podría realizarse si se calculase la distancia del objeto al centro de la pantalla en el eje X y en el eje Y para que fuera decelerando conforme se acercara al objetivo, o para una mayor aceleración si el objetivo se aleja a mayor velocidad.

Como prueba final se ha relizado el experimento de la trayectoria cuadrada con el prototipo diferencial con la torreta incorporada:

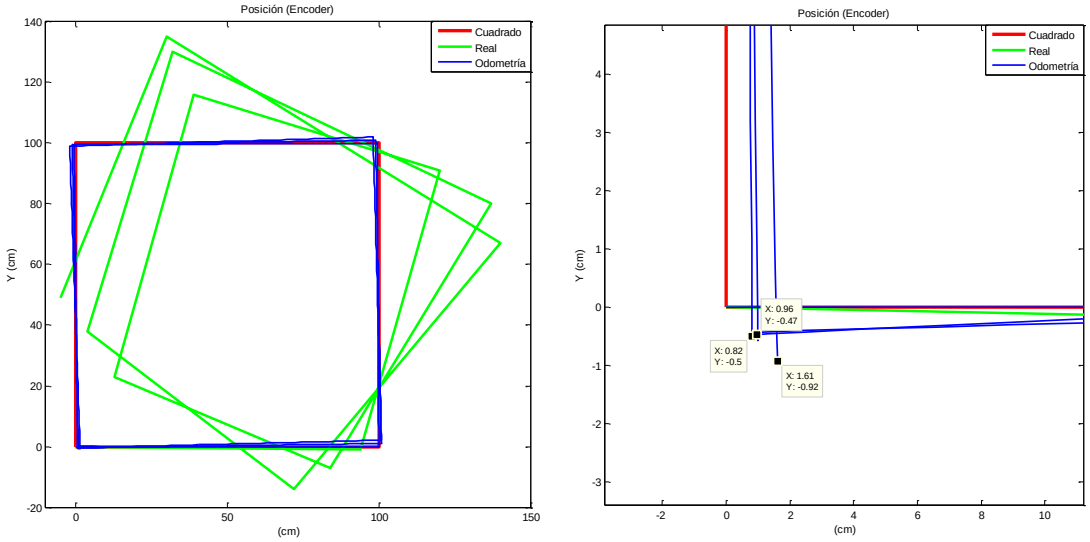


Figura 5.37: Trayectoria cuadrada

Debido al peso extra por la torreta la capacidad de giro del robot se ve reducida, además de la deformación de las ruedas que conllevan a una acumulación de error continua. Sin embargo la odometría, aun proporcionando peores resultados que el prototipo sin torreta, se mantiene cerca de la trayectoria real.

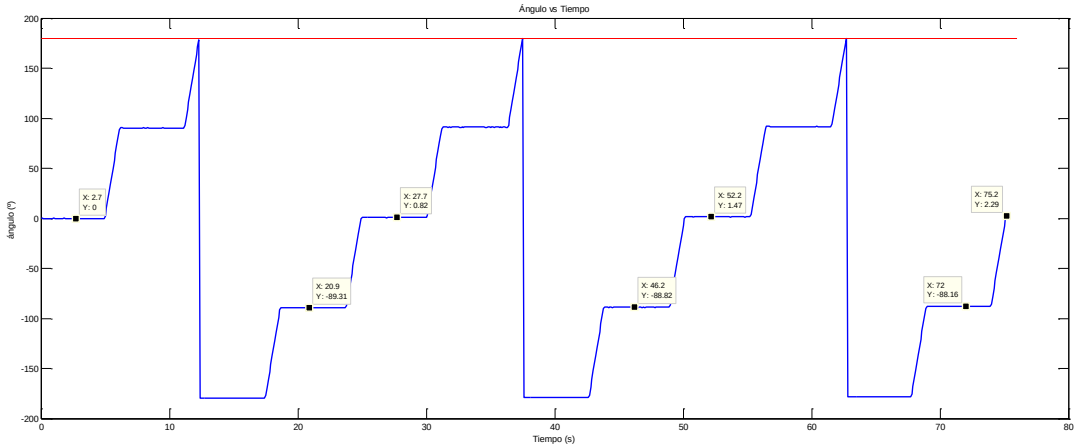


Figura 5.38: cuadrado - ángulo vs tiempo

Vuelta	Estimado			Real		
	x	y	h	x	y	h
Final (cm)	1.61	-0.92	2.29	-5	49	25
Er (%)	1.9	0.8	1.2	27	33	25

5.4.2. Seguimiento de objetivo por reconocimiento facial

La librería de OpenCV cuenta con una librería de reconocimiento facial. Por la facilidad de su uso se ha decidido mostrar los resultados aunque en un principio no estaba planteado para este proyecto su utilización (Figura 5. 39):

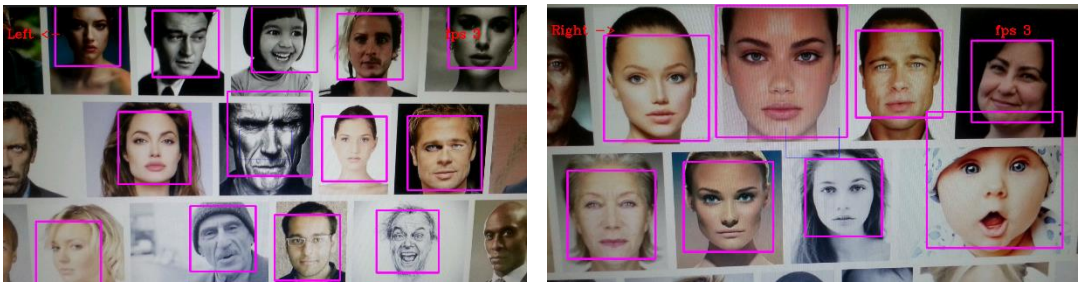


Figura 5.39: Reconocimiento facial

Además permite proporcionarle un rostro para su exclusivo reconocimiento.

El único problema con el método escogido 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 una detección bastante rápida, sin importar la distancia a la que se encuentre el objetivo.

Por otro lado, la inclusión del método de tracking ha aportado un incremento importante en el rendimiento. Al principio, el objeto 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.

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 una serie de videos en los que se pueden valorar distintos aspectos del sistema. Estos vídeos se pueden encontrar en el *Anexo B: Videos de los experimentos realizados*.

CAPÍTULO 6

CONCLUSIONES Y FUTUROS PROYECTOS

"La mejor forma de predecir el futuro es inventarlo"
Alan Kay

6.1. Conclusiones

Se han concluido las tareas propuestas en el capítulo de objetivos:

- Conocer en profundidad todas las opciones que ofrece el robot Lego Mindstorms NXT con el firmware LejOS.
- Pruebas de velocidad de la comunicación vía Bluetooth entre las plataformas usadas.
- Analizados los distintos sensores para microrobots.
- Estudio de las distintas configuraciones cinemáticas para microrobots.
- Diseñadas 4 plataformas totalmente funcionales.
- Analizadas las posibilidades de captura y reconocimiento de formas en imágenes a través de OpenCV.
- Desarrollada aplicación que hace uso de los conocimientos adquiridos.

Para la consecución del objetivo principal de este proyecto "sistema móvil autónomo para seguimiento de objetivo basado en Lego Mindstorm" y una vez acabado podemos resaltar dos tareas principales por el tiempo invertido:

1. Comparación de distintos sensores de Lego NXT con distintos modelos cinemáticos para la estimación de la posición.
2. Implementación de un sistema de visión artificial para el seguimiento de objetivo de bajo coste para microrobots.

Para el control del robot se ha descartado como medio de programación el software proporcionado por el fabricante. En su lugar, ha sido fundamental el uso del firmware libre *LeJOS*, que al ser un lenguaje orientado a objetos y proveer de una ejecución multi-hijo nos permite un control total del robot.

La elección de un smartphone Android utilizando la librería OpenCV ha sido la otra clave de este proyecto, respecto al software por la compatibilidad de OpenCV con Android y su fácil implementación, y al hardware por su reducido tamaño y peso.

6.2. Futuros proyectos

Debido a que este proyecto es multidiscipinar, es muy difícil abarcar en profundidad todos los ámbitos posibles que se han ido planteando a lo largo de la realización del mismo. Aunque se pueden destacar tres:

1) Control de un robot móvil
2) Localización
3) Sistema de visión artificial

- **ROS (ROBOT OPERATING SYSTEM)**

ROS es uno de los paquetes de software disponibles para el NXT. Al igual que otras distribuciones como LeJOS permite realizar más tareas que las que nos permiten el software de base, pero a diferencias de LeJOS es más complejo aunque con mayor potencial. ROS nació y sigue siendo un proyecto open source con licencias BSD. La licencia BSD es parecida a OpenSSL pero más permisiva. Así esta licencia permite el uso de software no libre. ROS permite también la conversión directa de código escrito en NXT-G a ROS, también ofrece una monitorización en tiempo real de los distintos parámetros. Es posible observar los movimientos que realiza el robot en un entorno 3D permitiendo corregir fallos.

Otra de las herramientas que trae ROS es un algoritmo de mapeo del entorno. Creando un mapa basado en puntos de los objetos que el sensor de ultrasonidos detecta. Posiblemente la parte más interesante de ROS sea la monitorización de las clases y sus relaciones. Así se puede ver cómo interactúan los diversos componentes y como se comunican. Dándonos la posibilidad de ver el flujo de datos que es producido por cada acción realizada.

Por lo tanto ROS es una de las plataformas más prometedoras en conjunto con Lego NXT, aunque también acepta multitud de otros tipos de robots. Es por ello que ROS está teniendo una aceptación muy grande en el entorno de la robótica y su comunidad está creciendo a velocidad increíbles.

Figura 6.1: Robot Operating System

- **SLAM**

El algoritmo de SLAM es un algoritmo que hace uso del filtro de kalman. Este filtro es un complejo algoritmo que debe ser adaptado para cada caso particular donde se quiera usar. Permite calcular cual es la posición más probable para un objeto (en nuestro caso un robot) basándose en ciertos parámetros. Este filtro permite depurar muchísimo los mapas creados por los robots en movimiento, pasando de una serie de puntos unidos por líneas rectas a un mapa mucho más suave y con curvas. En el caso de SLAM se une el filtro para la corrección de la posición y para mapear el trayecto que está realizando el robot al mismo tiempo.

Figura 6.2: Mapa proporcionado por SLAM

Por lo tanto es un algoritmo basado en la probabilidad de que un objeto se encuentre en un sitio concreto basándose en ciertos parámetros.

- **Kinect**

Es una tecnología desarrollada por Microsoft. Kinect permite al usuario controlar e interactuar con la consola sin necesidad de contacto físico o uso de un mando tradicional. Este dispositivo permite interactuar con la consola mediante gestos, o el uso de objetos. Es capaz de reconocer profundidad, una tecnología que aun a día de hoy es muy puntera. Gracias a sus 3 cámaras es capaz de reconocer movimiento, colores y profundidad. Esto hace que sea el sensor idóneo para mapear el entorno o usarlo como ojo para un robot. En el ámbito de la robótica, Kinect, proporciona la posibilidad de crear un sensor que permite al robot moverse por entonos cerrados con gran precisión. Al poder calcular la profundidad o distancia a la que se encuentran los objetos nos permite optimizar enormemente el cálculo del recorrido más idóneo para el robot.

Sin lugar a dudas está tecnología va a traer grandes avances en el ámbito de la robótica.

Kinect está formado por 3 pilares básicos:

1) Captura de movimiento.
2) Reconocimiento de voz
3) Motor de movimiento

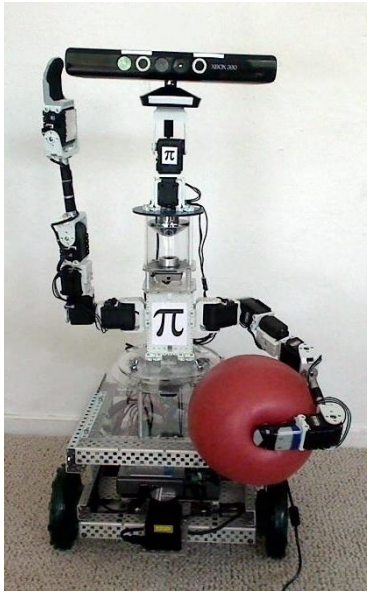


Figura 6.3: Kinect usado en un robot como visión

La captura de movimiento es el eje principal de Kinect, con el que es capaz de reconocer el movimiento y traducirlo a instrucciones para el ordenador. En este apartado está también la capacidad de reconocer colores y profundidad de la que hemos hablado antes.

Gracias a la captura de movimiento no solo se pueden interpretar gestos sino que también el entorno que rodea a la cámara, posibilitando así desarrollos no enfocados a videojuegos. También permite desarrollos en el ámbito de la robótica, como pueden ser el uso de Kinect como unidad visual del robot.

El reconocimiento de voz es importante porque ciertas funciones son fácilmente manejables desde comandos de voz. Esto permite que nos comuniquemos con la máquina mediante señales sonoras previamente definidas.

Por último y después de analizar los hogares de los posibles consumidores Microsoft se dio cuenta que era importante integrar un motor que permitiese a Kinect regular hacía donde estaba mirando. Esto es importante para permitir la mejor calibración posible y un mejor funcionamiento del dispositivo que si se calibrase a mano.

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.

CAPÍTULO 7
BIBLIOGRAFÍA

[1] PFC. "Estudio de las posibilidades didácticas en la ingeniería de control del LEGO Mindstorms NXT", año 2008, de Guillermo Nieves Molina. UPCT.

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

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

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

[8] "Simulación de vehículos eléctricos ligeros". Jaime Jiménez Cuesta. UPV.

[9] Aplicación de Android: http://developer.android.com/

[10] Brújula: www.hitechnic.com/

[11] Lego Mindstorm: www.mindstorms.lego.com/en-us/support/files/firmware.aspx

[12] leJOS NXJ Tutorial: http://lejos.sourceforge.net/

[13] Visión artificial: http://opencv.org/

[14] Sariel.pl. http://sariel.pl. sariel's custom LEGO Technic creations.

[15] Manual instalación LeJOS:

http://www.lejos.org/nxt/nxj/tutorial/Preliminaries/GettingStartedWindows.htm

[16] Manual instalación Eclipse para Android:

https://developer.android.com/sdk/installing/installing-adt.html

[17] Manual instalación OpenCV en Eclipse:

http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html#manual-opencv4android-sdk-setup

ANEXO A

CÓDIGOS DEL PROYECTO

Los códigos de este proyecto quedan adjuntos a este documento.

ANEXO B

VIDEOS EXPERIMENTOS

Los videos de los experimentos tanto de la estimación de la posición como de la visión por computador de este proyecto quedan adjuntos a este documento.