Centro Politécnico Superior
Universidad de Zaragoza

Robótica móvil con Lego Mindstorm

Ingeniería en Informática
Proyecto Fin de Carrera

David Pellicer Martín
Noviembre 2010

Dirigido por:
Luis Montano Gella
Ana Cristina Murillo Arnal
Departamento de Informática e Ingeniería de Sistemas
C.P.S, Universidad de Zaragoza

Resumen

Los Lego Mindstorm son unos robots fabricados por Lego, que pese a lo que pueda parecer, no son un mero juguete, debido a las capacidades y posibilidades que ofrecen. Cuentan con una CPU en un sistema embebido al cual se pueden conectar servomotores, una gran cantidad de sensores, y que tiene conexión USB y Bluetooth para comunicaciones. Ademas, el hecho de utilizar piezas estándard de lego da mucha flexibilidad al tipo de modelos que se pueden construir. Estos robots ya se utilizan muchísimo en actividades didácticas a todos los niveles, desde actividades con niños en colegios, prácticas en asignaturas de universidad o incluso como base para experimentos de proyectos de investigación sobre robótica.

El objetivo de este proyecto es analizar las posibilidades de estos robots para el aprendizaje y/o investigación práctica sobre robots móviles autónomos. Además, se quiere diseñar un entorno modular y extensible que permita realizar tareas básicas de robótica móvil de manera sencilla y monitorizarlas, como una herramienta base para futuros trabajos.

En primer lugar, se realizó la elección y familiarización del entorno de programación para los Mindstorms de entre la gran cantidad de posibilidades. En nuestro caso elegimos leJOS, un *firmware* que instala una máquina virtual Java en el robot, permite programar en Java, y además ofrece una completa API tanto para el robot como para el ordenador.

Este trabajo incluye un estudio de las técnicas y algoritmos necesarios para conseguir las funcionalidades principales que se quieren desarrollar sobre los Mindstorms, además de su implementación. Se pueden agrupar en:

- Generación de movimientos: Utilizando la información de la odometría y los distintos sensores, el robot es capaz de estimar su posición y moverse hacia un objetivo siguiendo un sistema de bucle cerrado.
- Navegación reactiva y planificada: Utilizando un mapa del entorno, el robot puede encontrar una ruta óptima y navegar por ella, teniendo la posibilidad de detectar obstáculos, modificar el mapa y calcular una ruta alternativa en función de ellos.
- Utilización los sensores disponibles para realización de tareas típicas en robótica, como construcción de un modelo del entorno con el sonar o seguimiento de objetos con la cámara.

También se ha diseñado un módulo para monitorización y control de dichas actividades, que se comunique de manera remota con el robot y que permita darle ordenes, así como hacer un seguimiento de las lecturas de sus sensores y de su posición.

Para la implementación y prueba de todos estos módulos, se han diseñado y construido dos robots, uno con una cinemática basada en tracción diferencial, y otro con cinemática tipo coche, para poder comparar estos dos tipos habituales de robot móvil. Esta memoria incluye detalles de ambos modelos y de cómo se han implementado y evaluado en ellos todas las funcionalidades requeridas, utilizando distintos tipos de sensores y analizando la precisión y el tipo de tareas que se pueden resolver con cada uno.

Se debe tener en cuenta a la hora de analizar los resultados que la precisión de estos sensores no va a ser excesivamente alta, por el precio mucho más bajo en comparación con los componentes de robots más "reales". Pese a ello, se consiguen resultados bastante satisfactorios y los robots diseñados pueden realizar de manera autónoma tareas como navegar en espacios no muy amplios, y el sistema de monitorización permite que el trabajo para desarrollar nuevas funcionalidades sea mucho más cómodo.

Índice general

1

Introducción

1.1. Motivación

La robótica se ha convertido desde hace ya bastante tiempo en uno de los campos más atractivos para las personas que nos gusta la ciencia, sin embargo, no está accesible para la mayoría de la gente debido a la gran cantidad de recursos que hacen falta para poder experimentar y desarrollar en robots de verdad.

En este ámbito, Lego lanzó al mercado los robots Mindstorms, unos robots de juguete, pero con las mismas características que podría tener cualquier robot industrial o de investigación. De esta manera, con un presupuesto bajo, ganas de experimentar, y conocimientos de mecánica, informática e inteligencia artificial, es posible experimentar e imitar muchas de las funcionalidades y comportamientos autónomos de los robots reales

Algunos ejemplos de ello son robots capaces de escribir [1], resolver cubos de Rubik [2], hacer construcciones simples de lego, y por supuesto robots móviles [2]. Estos robots son muy utilizados en actividades didácticas en varios niveles, desde trabajos con niños en colegios, hasta prácticas en asignaturas de universidad sobre inteligencia artificial, mecánica y por supuesto robótica. Incluso pueden servir de base para experimentos de proyectos de investigación sobre multirobots [3].

En la asignatura "Robótica de Servicio" de las titulaciones de ingeniería del CPS, se comenzó hace poco a utilizar estos robots, y de ahí surgió la idea de realizar este proyecto, para estudiar y analizar en más detalle el alcance de estos robots y sus sensores para tareas de navegación de robots autónomos, y desarrollar una plataforma desde la cual se pueda facilitar a los alumnos la programación y monitorización de este tipo de tareas, tanto en asignaturas como en futuros proyectos. Para ello se han desarrollado librerías de calibración y ejecución de tareas básicas que se detallarán más adelante en los objetivos y medios (Sección 1.2).

[1] "Printer V04". http://www.youtube.com/watch?v=4xiXFNNqdxw
[2] "Titled Twister". http://tiltedtwister.com/

1.2. Objetivos y medios

Los objetivos generales de este proyecto, como se ha mencionado antes, son analizar el alcance de esta plataforma para tareas de robots autónomos y desarrollar un entorno que facilite el trabajo para dichas tareas. Los objetivos más detallados de este proyecto son:

- Análisis y estudio de los modelos cinemáticos mas comunes en robótica móvil.
- Elección de lenguaje y entorno de programación, sensores a utilizar, modelos cinemáticos y mecánicos para los robots y sistema de comunicaciones.
- Análisis, diseño e implementación del software necesario para generar movimiento de acuerdo con las diferentes estructuras y dinámicas.
  - Robot con tracción diferencial.
  - Robot tipo coche.
- Programación del software para las tareas de más alto nivel, que sean independientes de la cinemática elegida.
  - Localización mediante odometría y sensores.
  - Navegación, reactiva y con planificación (potencial, NF1).
  - Otras.
- Experimentación y análisis de la fiabilidad y alcance de precisión de este tipo de robots, con las diferentes estructuras y aplicaciones diseñadas.
- Diseño de un protocolo de comunicaciones con Bluetooth para el intercambio de información entre el robot y un PC.
- Programación de un interfaz visual de control y monitorización del robot desde un PC.
- Realización de la documentación necesaria para la utilización de todos los módulos implementados en otros proyectos futuros y en prácticas de asignaturas de robótica e inteligencia artificial.

Para la realización de estos objetivos, se han utilizado dos kits de Lego Mindstorm educacional, y otro de piezas suplementarias. Se han realizado pruebas en el laboratorio de robótica del CPS, así como en otros entornos de interiores. El ordenador utilizado para la programación es un portatil con procesador Intel Core 2 Duo a 2.4Ghz, sistema operativo Windows XP y lenguaje de programación Java bajo el entorno de programación Eclipse [7].

1.3. Estructura de la memoria

La presente memoria se organiza en 4 capítulos con distintos apartados, así como varios apéndices y la bibliografía.

El primer capítulo es el actual, y en él se introducen los objetivos del proyecto así como un resumen general de todo el contenido. En el segundo capítulo se hablará de los robots Lego Mindstorms, de sus características tanto técnicas cómo del sistema operativo y sensores. Se hará distinción entre los dos modelos construidos.

En el tercer capítulo se hablará de los módulos y tareas implementados. Es un capítulo más técnico en el que se explicará como se ha programado la aplicación y los algoritmos y bases teóricas utilizados para cada una de las funcionalidades del robot.

Por último, en el cuarto capítulo se analizarán los resultados y se hablará de las conclusiones sobre el proyecto y sobre el posible trabajo a realizar al fin del mismo.

2

Robots Lego Mindstorm

2.1. Introducción

Lego Mindstorms aparecen en 1998 como unos robots de juguetes fabricados por Lego en colaboración con el MIT [4]. Esta no fue la primera colaboración, unos años antes presentaron el lenguaje de programación Logo, que permitía también programar acciones para construcciones de Lego con motores, luces y sensores [1]. Pero este producto tenía muchas restricciones, ya que tenía que estar continuamente conectado al ordenador, y además por aquella época no era tan habitual como ahora tener un ordenador en casa.

Cinco años más tarde aparece el primer robot Mindstorm, que contaba con un bloque programable RCX (figura 2.1-izquierda), el cual incluía dentro un microcontrolador, al cual ya era posible cargar programas creados por el usuario, y ejecutarlos sobre una estructura con motores, ruedas y sensores. De esta manera se pasaba de tener una construcción estática, a tener un robot capaz de moverse y de interactuar con el entorno.

Fig. 2.1: Bloques RCX y NXT

En 2006, y tras sacar distintas y mejoradas versiones del bloque RCX, Lego comercializó Mindstorms NXT (figura 2.1-derecha), con un bloque de última generación que es el que se utiliza para este proyecto fin de carrera.

2.2. Características técnicas

El bloque NXT tiene las siguientes características técnicas [5]:

- Procesador ARM-7 de 32 bits (AT91SAM7S256)
- 256 KB de memoria FLASH para los programas y el firmware.
- 64 KB de RAM.
- Velocidad de funcionamiento de 48MHz.
- 3 puertos para conectar servomotores.
- 4 puertos para conectar sensores.
- Pantalla LCD.
- Bluetooth para comunicaciones entre el bloque y un ordenador, un móvil, u otro bloque.
- Puerto USB para conexión con el ordenador.

2.3. Firmware

El *firmware* que lleva el bloque NXT por defecto viene incluido en los CDs que se incluyen con el kit, y permite ser programado con el propio software visual también incluido. Este software es NXT-G, y está basado en LabView. No es una programación basada en código, sino en bloques visuales, de manera que sea sencillo para niños hacer programas simples.

Sin cambiar este *firmware*, existen opciones más potentes como el entorno de programación Bricx Command Center [6]. Este permite escribir código en diferentes lenguajes y descargarlos en el robot. Ofrece además una interfaz de intercambio de archivos entre el pc y el robot, así como la posibilidad de cambiar el *firmware*, hacer copias, etc. Estos lenguajes de programación son:

- **NBC (Next Byte Codes [8]** Es un lenguaje de código abierto con una sintaxis de tipo ensamblador para el bloque NXT.
- **NXC (Not eXactly C) [8]**. Es un lenguaje también de código abierto basado en C. Tiene la sintaxis de C, pero con grandes deficiencias como no tener instrucciones de punto flotante, o punteros. Aún así, es uno de los lenguajes más utilizados por la comunidad para crear programas para Mindstorms, debido a su gran similitud con C, el no tener que cambiar el *firmware* del robot, y el poder manejar con facilidad todos los sensores que conectemos al bloque.

Existen otros tipos de *firmware* de terceras partes que permiten programar con distintos lenguajes, por ejemplo Ada, Python, C, Matlab, Lua o Java entre otros.

Para la realización de este proyecto, se utilizó el *firmware* leJOS [9], que instala una máquina virtual Java en el bloque NXT, y permite programar en el lenguaje de alto nivel orientado a objetos Java. Se eligió este lenguaje porque ofrece programación orientada a objetos en Java, con la cual estoy muy familiarizado, es código abierto (Open Source) y por tanto no había problemas de índole legal ni de pago de licencias, y porque ofrece una API muy completa tanto para el robot como para el PC, con primitivas de sincronización y *multithreading* entre otras funciones avanzadas.

leJOS

Este firmware fue creado como un proyecto de software libre para el bloque RCX, el predecesor de NXT.

Instala una máquina virtual en el robot, así como un menú con opciones para controlar la memoria y los programas desde la propia pantalla LCD del bloque NXT. Algunas de las características de este paquete de herramientas son:

- Programación orientada a objetos en Java
- Hilos con prioridades, lo que permite programar basandose en un sistema de tiempo real.
- Vectores y matrices, incluidas multidimensionales. Esta es otra de las carencias de NXC.
- Recursividad
- Sincronización, necesaria si vamos a hacer programación multihilo basada en sistemas de tiempo real.
- Manejo de excepciones.

- Tipado similar al de Java, incluyendo float, long, String, etc.
- La mayoria de las clases incluidas en java.lang y java.io. Y en general gran parte de la API de Java está portada a esta plataforma.
- Documentación completa de la API.
- API para el robot, y para el PC, para el intercambio de información entre PC y robot.
- Control de todos los sensores, así como API para manejo del bluetooth y de los servomotores.

2.4. Robots y sensores

Como se ha explicó en la introducción, se han construido dos robots, con distintas cinemáticas y sensores. En esta sección se hablará de sus características técnicas y qué sensores utilizan.

2.4.1. Diferencial

Fig. 2.2: Robot diferencial

Este robot esta construido basándose en uno de los robots que vienen de ejemplo con el kit básico de Mindstorms (tribot). Como se puede observar, las diferencias entre el tribot 2.3 y el

Fig. 2.3: Robot tribot

robot construido 2.2 son sobre todo en el uso y distribución de los sensores: se mantienen los sensores de distancia y detección de luz reflejada, y se cambian los sensores de sonido y contacto por una cámara y una brújula.

Características

Las principales características y capacidades que se van a estudiar con este primer modelo de robot, y que se explicarán con más detalle en el capítulo 3 son:

- **Ruedas de oruga.** El modelo original de Tribot, funciona con tres ruedas, dos de ellas acopladas a un servomotor cada una, y otra libre en la parte de atrás con la única intención de equilibrar el peso. En nuestra construcción, se utilizó un sistema de tracción donde dos ruedas están acopladas cada una a un servomotor, pero hay otras dos solidarias a las motrices mediante cintas dentadas de oruga. La decisión de usar este sistema es porque el robot obtiene mayor estabilidad, además de mayor potencia al tener unas ruedas más pequeñas y una superficie más grande.
- **Cinemática diferencial.** Esta cinemática, que será explicada con detalle en el siguiente capítulo, consiste en dar movimiento independiente a cada una de las dos ruedas tractoras, de manera que combinando las velocidades de una y otra, y el sentido de giro, se consigue dar velocidad angular al robot.
- **Evitación de obstáculos con ultrasonidos.** El sensor de ultrasonidos colocado en el soporte superior del robot permitirá detectar obstáculos enfrente del robot.
- **Reconocimiento de objetos con la cámara** Esta cámara esta colocada enfrente del robot y le permitirá encontrar y seguir objetos de cierto color.

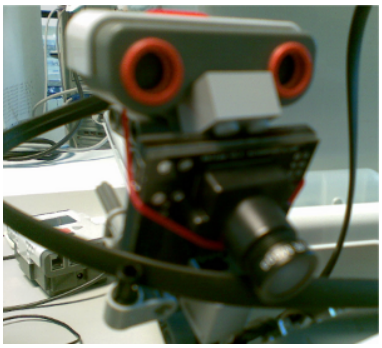

Fig. 2.4: Cámara y sensor de ultrasonidos

- **Pinzas** En combinación con la cámara, hará posible que el robot sea capaz de coger objetos simples basándose en el color y el tamaño de lo que vea.

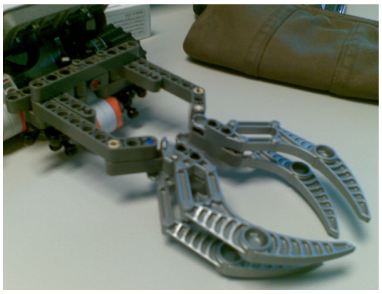

Fig. 2.5: Pinzas

- **Estimación de la posición mediante brújula y tacómetro de los motores.** Utilizando los tacómetros de los motores, que miden los grados que van girando, así como la brújula superior, este robot puede estimar su posición en el entorno.

Sensores

El robot diferencial lleva tres sensores:

- **Cámara** La cámara que se puede conectar a los robots Lego Mindstorm, no transmite imágenes completas al procesador del robot, sino que realiza un preprocesado de la imagen en el chip de la camara, y le transmite al robot que "blobs" (regiones conexas de un color predefinido) y de qué tipo, ha detectado en la imagen actual, es decir, se definen una serie de colores que queremos reconocer dando el rango correspondiente en el espacio de color típico, RGB, y el procesado del chip de la cámara devuelve ordenados de mayor a menor los blobs que ve de esos colores, incluyendo su posición en la imagen y el área del rectángulo que contiene al blob. Hay que tener en cuenta que la capacidad de diferenciar colores es muy sensible, y por ejemplo en habitaciones con poca luz, la calidad de las imágenes que consigue es muy baja, por tanto fue importante para los experimentos utilizar entornos muy bien iluminados, y con un alto contraste entre el color del suelo y el del objeto a seguir.
- **Brújula** La brújula tiene un funcionamiento muy simple, devuelve los grados a los que se encuentra desalineado del norte magnético. La precisión que da es bastante buena, pero como toda brújula tiene el problema de las interferencias, ya que cualquier cosa metálica de tamaño considerable que estuviera situada en el entorno podía hacer que las lecturas fueran erróneas. La situación en la parte más alta del robot no es arbitraria, obedece a intentar salvar las interferencias que también provocan los circuitos del bloque principal del NXT.
- **Ultrasonidos** Este sensor es capaz de medir a cuanta distancia se encuentra un objeto mediante la emisión-recepción de ultrasonidos, y la medida de el tiempo que tarda en recibir la señal que ha emitido, pero tiene el problema de que es muy direccional y por tanto solo funciona bien con objetos perpendiculares a la dirección de emisión del utrasonido. Tanto en este robot como en el de tipo coche se utilizaron las lecturas de este sensor para detectar obstáculos cercanos y poder evitarlos.

2.4.2. Tipo coche

Este robot está construido desde cero, y utiliza dos motores para el movimiento, uno para la tracción (trasera), y otro para mover el eje de giro. Este robot es más grande pero menos preciso que el diferencial, ya que la odometría (estimación de la posición propia del robot mientras se mueve) que en el caso diferencial era bastante buena, en este caso no lo es tanto ya que entran mucho más en juego rozamientos, imprecisión en el ángulo de giro o deslizamientos entre engrajanes.

Características

Aunque el software de navegación y planificación es el mismo en el caso de los dos robots, no es así la mecánica ni estimación de la posición. Las características y capacidades que se van a analizar gracias a la construcción de este otro modelo de robot son:

- **Tracción trasera** Se eligió el sistema de tracción en las ruedas traseras porque las ruedas que proporcionan el giro son las delanteras. Intentar hacer que la dirección fuera acoplada

Fig. 2.6: Robot tipo coche

a las ruedas motrices hubiera sido algo demasiado complejo de montar, y además hubiera cargado demasiado el peso en la parte delantera del robot, cuando es mucho más estable tener el peso uniformemente distribuido.

La pieza de diferencial de coche (figura 2.7) es una de las que vienen en el kit de piezas especiales y que no puede encontrarse en las cajas normales de Lego Mindstorm, equilibra la potencia entre las ruedas tractoras. Ya que si se movieran las dos solidarias, cuando el robot estuviese girando una de ellas se frenaría o patinaría, esta pieza equilibra la potencia, y hace que la rueda de más afuera en el giro gire más, y la de adentro menos, reduciendo el rozamiento y por tanto aumentando la precisión en la odometría.

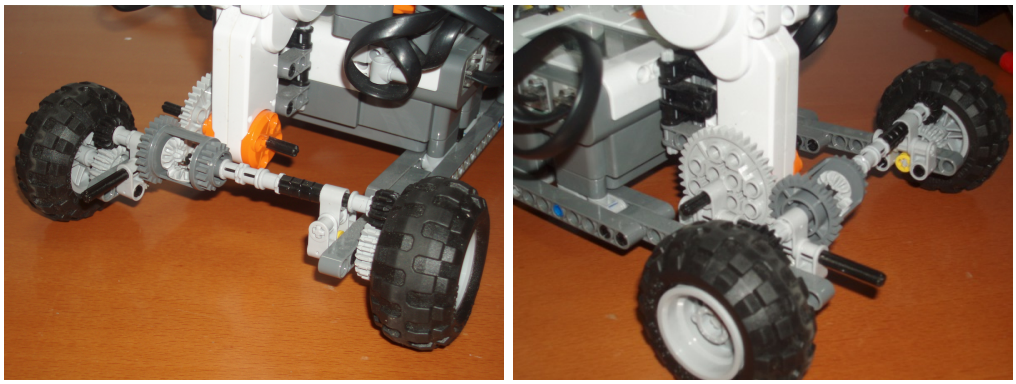

Fig. 2.7: Tracción trasera y diferencial de coche

- **Giro controlado por motor** El motor de giro, unido a unos engranajes de cadena que transforman el movimiento circular en movimiento horizontal, es el que controla qué angulo tienen las ruedas delanteras.
- **Radar** El tercer motor disponible se utiliza para rotar un eje con un sensor de ultrasonidos montado arriba, con la intención de que funcione de forma similar a un radar, y seamos

capaces de estimar un mapa básico del entorno del robot.

- **Evitación de obstáculos con ultrasonidos** Al igual que en el robot diferencial, el mismo sensor de ultrasonidos colocado en el motor, mientras se está navegando estará alineado en linea recta con el frontal del robot para poder detectar obstáculos y poder reaccionar y replanificar la ruta.
- **Estimación de la posición mediante giróscopo y odometría** Es una de las diferencias más importantes con respecto al robot diferencial, y es que se integran las estimaciones de la odometría con el giróscopo, montado sobre la parte superior del robot.

Sensores

En este modelo solo se montaron dos sensores, por la dificultad mecánica de colocar más sin hacer que el robot se hiciera demasiado pesado y tuviera problemas para moverse, ya que los servomotores no son muy potentes.

- **Ultrasonidos** Es el mismo sensor que en el módelo diferencial, pero aquí además de darle el uso de evitar obstáculos, el sensor esta montado sobre un eje movil que le permite hacer barridos para recrear en el ordenador una imagen del entorno del robot.
- **Giróscopo** Este sensor, también llamado acelerómetro, es capaz de medir la velocidad de giro desde una lectura a la siguiente, con lo cual teniendo en cuenta esa variación de tiempo es posible calcular los grados girados. Es importante calibrar el sensor al principio, así como hacer muchas lecturas de la velocidad para utilizar una media, y no quedarse con lecturas puntuales que pueden no ser del todo reales. Al igual que la brújula en el robot diferencial, no se utiliza solo este sistema de estimación, sino que se usa en unión con la odometría.

3

Robótica móvil con robots Lego Mindstorm

3.1. Arquitectura del entorno desarrollado

El software se ha construido de forma modular, de manera que se puedan reutilizar funciones de alto nivel, simplemente llamando a las funciones correspondientes de bajo nivel según el modelo cinemático y los parámetros del robot utilizado. En la figura 3.1 puede verse de manera esquemática los distintos módulos, que se explicarán a lo largo de esta sección.

Cómo ya se introdujo anteriormente 2.3, el firmware que se utilizó en el robot fue leJOS, y el lenguaje de programación Java.

El módulo de movimiento consta de distintas tareas que se encargan de controlar la posición mediante los distintos sensores y la odometría. El de navegación se sostiene en el de movimiento, y gestiona el mapa del entorno en el que se moverá el robot y la evitación de obstáculos. Para ello deberá ser capaz de calcular una ruta óptima teniendo en cuenta el entorno y el objetivo o *Goal* indicado.

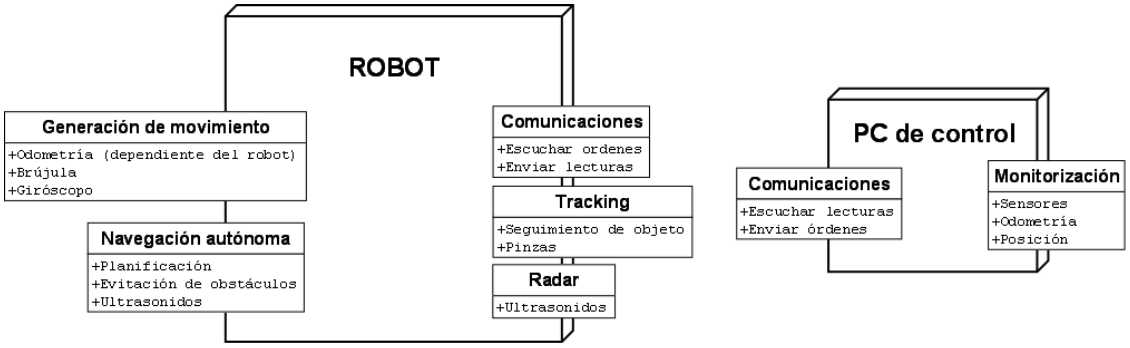

Fig. 3.1: Arquitectura del sistema

El módulo de comunicaciones permite que un PC de control pueda enviar órdenes al robot, así como obtener las lecturas de sus sensores y su posición. Sirviéndose de estos datos, la aplicación en el PC de control mostrará de manera visual el mapa del entorno con el robot a escala y los valores de posición y sensores en tiempo real. Además permitirá de forma fácil enviar ordenes de velocidad y objetivos al robot.

Por último el módulo de *tracking* sirve como experimento de visión artificial para estos robots. Implementa un algoritmo de seguimiento de un objeto de un color determinado, y tambien permite al robot cogerlo con las pinzas cuando estima que se encuentra a la distancia adecuada.

El software del robot, guiándonos en la estructura modular explicada, se ha implementado con programación orientada a objetos, que permite encapsular más fácilmente la arquitectura diseñada. En el anexo A pueden verse con más detalles las clases y tareas implementadas.

3.2. Módulos y dispositivos para generar movimiento

La generación del movimiento de ambos robots está basada en un control en bucle cerrado, donde se va variando la velocidad lineal y angular en función de la estimación de posición, que se va calculando a intervalos de tiempo establecidos.

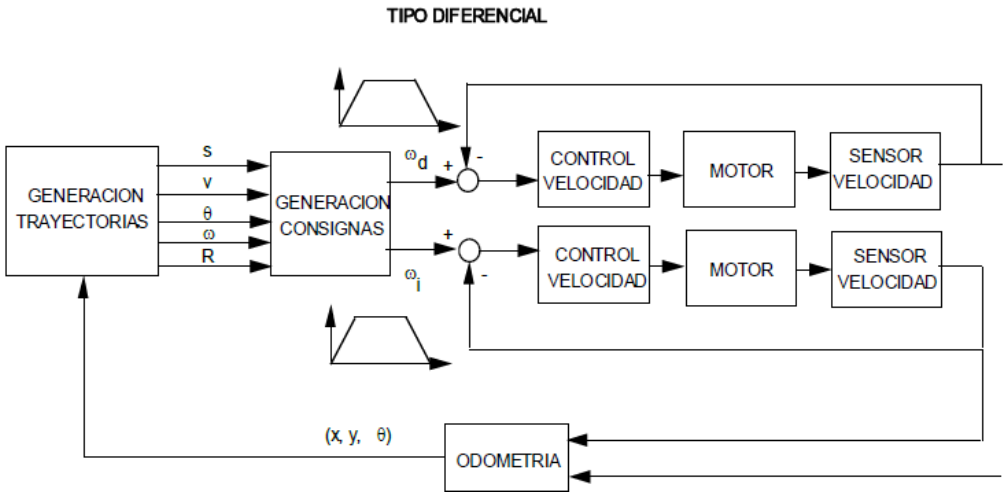

Fig. 3.2: Modelo de bucle cerrado con cinemática diferencial

Como se puede ver en la figura 3.2, en cada iteración o "ciclo" de movimiento del robot, primero genera la trayectoria a seguir, lo que permite estimar la velocidad lineal y angular necesarias (v, w) para recorrer esa trayectoria. A continuación se pueden generar las consignas, dependiendo del modelo de robot, ya que en el caso diferencial la velocidad lineal v y angular ω se deben "traducir" a velocidad de giro de cada uno de los motores (ω_R, ω_L) de las ruedas motrices, mientras que en el caso del robot tipo coche deben convertirse en la correspondiente velocidad lineal y ángulo del eje α. Cada tipo de cinemática conlleva unos cálculos diferentes para generar estas consignas y pasarlas a los motores correspondientes, pero el generador de trayectorias será común para los dos modelos (explicado a continuación).

Una vez actualizadas las velocidades del robot, antes de inciar una nueva iteración del proceso, se hará una estimación de la posición basada en la odometría del robot y las mediciones integradas de otros sensores para poder re-estimar la trayectoria desde el punto actual hasta el objetivo.

Generación de trayectorias

Para generar las trayectorias dada la posición actual y el objetivo, utilizaremos una técnica de control de movimiento [14] basado en tres constantes k_p, k_α, k_β que ajustarán el grado de curva que sigue la trayectoria. Estos valores se fueron ajustando experimentalmente hasta dar con los adecuados para este robot. En la figura 3.3 hay un esquema de la generación de trayectoria, donde puede observarse que p es la distancia entre el robot y el punto destino (*Goal*), θ es la orientación del robot, y α y β son los ángulos del triángulo entre estos dos puntos.

El algoritmo de control, nos ayuda a estimar en cada paso, la velocidad lineal y angular (v,w) q debe llevar el robot, y la distancia p que nos queda hasta llegar al objetivo. El algoritmo iterara hasta que p es menor cierto umbral, lo cual significa que ya está lo suficientemente cerca del objetivo.

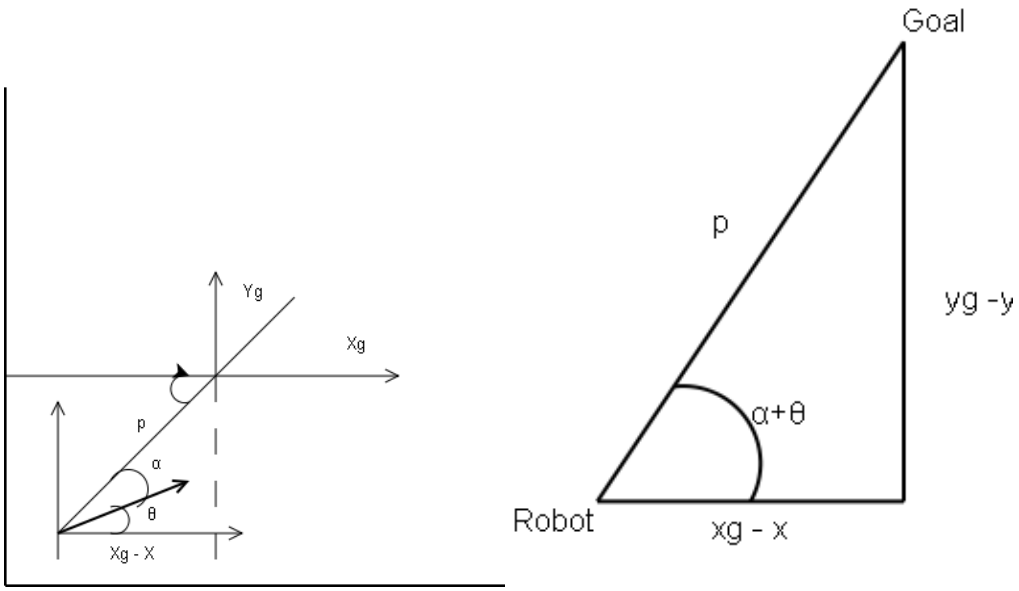

Fig. 3.3: Trayectoria entre robot y *Goal* (izquierda). Ampliación (derecha)

Estas son las expresiones que relacionan las constantes k_p, k_α, y k_β con los parámetros que queremos estimar:

$$v = k_p * p \tag{3.1}$$
$$\omega = k_\alpha * \alpha + k_\beta * \beta \tag{3.2}$$
$$p = \sqrt{(x_g - x)^2 + (y_g - y)^2} \tag{3.3}$$
$$\alpha = \arctan(\frac{y_g - y}{x_g - x}) \tag{3.4}$$
$$\beta = \alpha + \theta + \theta_g \tag{3.5}$$

3.2.1. Movimiento diferencial

Una vez hemos generado la velocidad lineal y angular en función del punto de destino, tendremos que transformar esas velocidades a instrucciones al motor, y para ello tendremos que saber cual es la velocidad angular que queremos establecer en cada una de las ruedas para el caso diferencial.

Las expresiones para estimar la velocidad angular que necesitan llevar las ruedas para conseguir que el robot lleve una cierta v y w son las siguientes[15]:

$$v = \frac{r(\omega_r + \omega_l)}{2}; \omega = \frac{\omega_r - \omega_l}{2l} \tag{3.6}$$
$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2l} & -\frac{r}{2} \end{pmatrix} \begin{pmatrix} \omega_r \\ \omega_l \end{pmatrix} \tag{3.7}$$

Donde l es la distancia entre las ruedas y r su radio .

Invirtiendo estas formulas, podemos obtener cómo calcular las velocidades de los motores izquierdo y derecho a partir de las velocidades lineal y angular:

$$\begin{pmatrix} \omega_r(t) \\ \omega_l(t) \end{pmatrix} = \begin{pmatrix} \frac{1}{r} & \frac{l}{r} \\ \frac{1}{r} & -\frac{l}{r} \end{pmatrix} \begin{pmatrix} v(t) \\ \omega(t) \end{pmatrix} \tag{3.8}$$

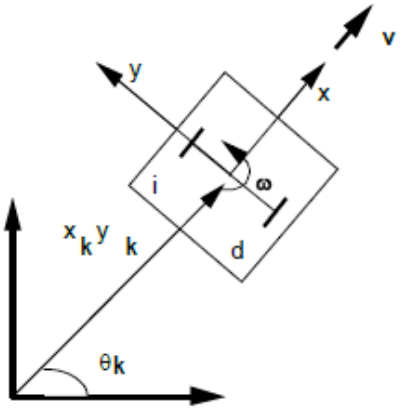

Fig. 3.4: Esquema de robot diferencial

Implementación en el robot

Se ha implementado el $interface\ Navigator$. Este $interface$ tiene una función $setSpeed(v, w)$ que debe indicar a los motores que velocidades deben llevar.

En esa función, el código es principalmente aplicar las expresiones anteriores:

```
wr = Math.toDegrees((v + (l * w)) / r);
wl = Math.toDegrees((v - (l * w)) / r);
```

Al pasarlo a grados con la función `toDegrees`se consigue que la velocidad este en grados por segundo en vez de en radianes por segundo. Esto es así porque en la API de los motores solo se puede indicar la velocidad en grados por segundo, o en potencia a los motores.

Además, antes de darle el valor, se comprueba que la velocidad que se va a imponer al motor no sea mayor que la que puede desarrollar, ya que el sistema no va a notificar ningún error y simplemente lo pondrá al máximo de su potencia con lo cual nuestras estimaciones serán erróneas porque no está consiguiendo la velocidad que le indicamos. La máxima velocidad que pueden alcanzar los servomotores es de 900 grados por segundo. Esto es problemático para mantener el radio de giro que deseamos en la trayectoria, ya que depende de la proporción entre las dos velocidades, y si le diéramos a los motores 1300 y 3000 respectivamente los dos motores irían solo a su máxima potencia, es decir, a la misma velocidad (900 grados/segundo). Por tanto y para evitar este problema, se comprueba si alguna de las dos velocidades es mayor que el máximo del motor, y en caso de serlo se reducen manteniendo la proporción entre ambas hasta que las dos estén dentro de los límites permitidos.

3.2.2. Movimiento tipo-coche

Esta sección detalla como transformar la velocidad lineal y angular objetivos a velocidades de motor para poder seguir esa trayectoria. Debido a la complejidad de la formulación exacta de este modelo cinemático, se aproximó el sistema a un triciclo. [15]

Como se puede ver en la figura 3.5, d es la distancia entre las ruedas motrices y la rueda directriz, R el radio de circunferencia que llevará el robot en esa trayectoria con respecto a las

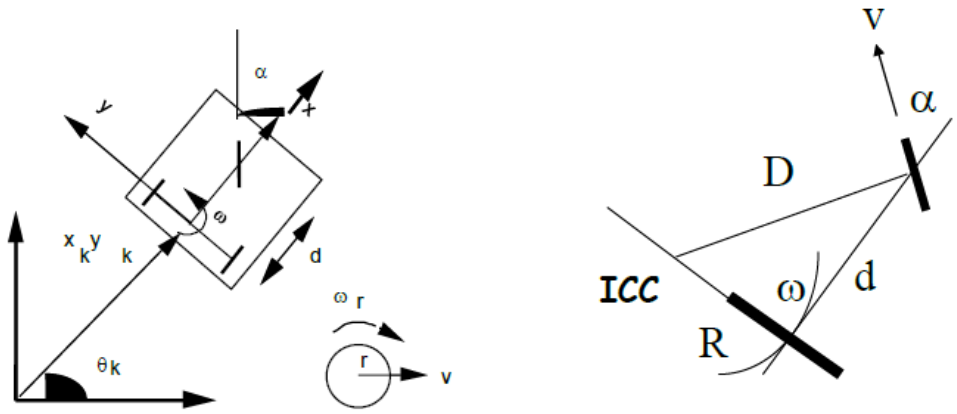

Fig. 3.5: Esquema de robot tipo coche

ruedas motrices, y D el radio con respecto a la rueda directriz, o la parte delantera del robot. Las expresiones para conseguir la velocidad lineal y angular son las siguientes:

$$R = \tan\left(\frac{\pi}{2} - \alpha\right) \tag{3.9}$$

$$v = r\omega_r \tag{3.10}$$

$$\omega = \frac{v}{D} = \frac{r\omega_r}{\sqrt{R^2 + d^2}} \tag{3.11}$$

Y una vez obtenidos los parámetros anteriores, podemos calcular α (que es el ángulo de las ruedas con respecto del eje) y la velocidad del motor:

$$\omega_r(t) = \frac{v(t)}{r} = \frac{\omega(t)\sqrt{R^2 + d^2}}{r} \tag{3.12}$$

$$\alpha(t) = \frac{\pi}{2} - \arctan\left(\frac{R}{d}\right) \tag{3.13}$$

Implementación en el robot

Para implementarlo en el robot, se hizo de forma similar al robot diferencial, un método `setSpeed(v,w)`que ejecuta las expresiones detalladas para la estimación de los parámetros de movimiento, pasando antes una comprobación de que la velocidad angular que le pedimos esta en el rango de las posibles para el robot, ya que el ángulo α, que es el que tendrá el eje de las ruedas motrices está limitado físicamente por el espacio y la construcción a un cierto ángulo.

R se ha calculado de la siguiente manera:

$$R = \sqrt{\frac{v^2}{\omega^2} - d^2} \tag{3.14}$$

Una vez conocemos todos los parámetros, debemos girar el motor de la dirección lo suficiente como para dejar α en los grados indicados. Un esquema de la situación puede verse en la figura 3.6

De los parámetros de la figura 3.6 solo nos falta calcular a. Esa a es lo que se ha desplazado el eje en horizontal con respecto a su posición inicial, teniendo en cuenta que los diámetros de los

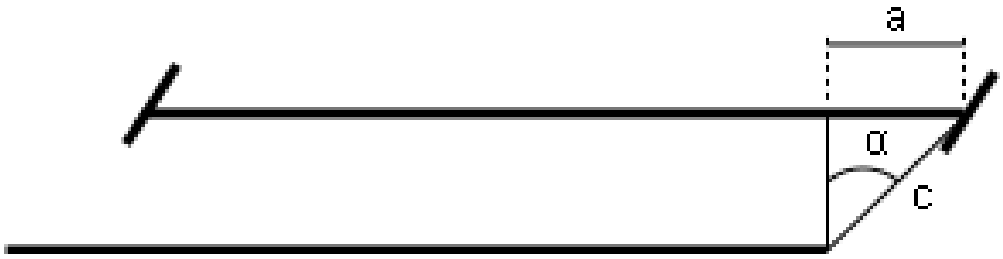

Fig. 3.6: Dirección del robot tipo coche

engranajes son de 10mm y 25mm respectivamente, se deduce que la relación de tracción entre esas dos ruedas dentadas es de $\frac{10}{25} = 0{,}4$, de ahí se puede deducir que la distancia de una vuelta completa del motor sería de $2\pi r 0{,}4$, con lo cual se puede definir la expresión para estimar a:

$$a = \frac{grados\_girados * 2 * \pi * r * factor}{360} \tag{3.15}$$

Ahora que conocemos a, c y α que es el ángulo que queremos conseguir, podemos calcular los grados que tenemos que girar el motor.

$$\sin\alpha = \frac{a}{c} = \frac{grados\_girados * 2 * \pi * r * factor}{360 * c} \tag{3.16}$$

$$grados\_girados = \frac{sin\alpha * 360 * c}{2 * \pi * r * factor} \tag{3.17}$$

Una vez conocemos los grados que debe girar el motor, la función de la API Motor `rotateTo(tachos)` hace que el motor gire hasta los grados que le digamos. De esta manera se consigue colocar el eje con el ángulo α deseado.

3.2.3. Estimación de la posición

Como ya se explicó en las características de los robots, la estimación de la posición se hace integrando las estimaciones de la odometría y con las medidas de los demás sensores.

En los dos modelos el sistema era similar: dos tareas en paralelo, una encargada de estimar la odometría según las medidas de los tacómetros de los motores, y otra del sensor elegido para complementar esta estimación (brújula en el diferencial, giróscopo en el coche).

La odometría se basa en el cálculo inverso de las velocidades llevadas durante un espacio de tiempo, en función de los giros del motor. De esta manera la odometría utiliza las lecturas del tacómetro (cuenta de los grados que va girando el motor) y estimar la velocidad real que está llevando el robot en ese periodo. A partir de ahí se estima la posición actual del robot respecto a la posición inicial, utilizando esta con la velocidad real estimada en lugar de con la teórica que el sistema pretendía llevar, ya que la diferencia entre lo que dice el control y lo conseguido realmente puede diferir bastante, por falta de baterías, por frenado debido al rozamiento, etc ...

Con la velocidad lineal y angular calculada a partir de la odometría, se realimenta el bucle cerrado para que varíe las velocidades para ajustarse mejor a la trayectoria prevista.

El periodo de la tarea de odometría es muy pequeño, para tener constantemente lecturas actualizadas de lo que está ocurriendo, sin embargo tanto el giróscopo como la brújula, dan más precisión cuanto más espaciadas son sus lecturas, por tanto se mantiene sólo la estimación de la odometría hasta que se reciben nuevas lecturas del sensor correspondiente, y entonces se corrige la posición según dicha medida, ya que los sensores tienen más precisión.

3.2.4. Resultados y conclusiones

Los resultados de la estimación de la posición están dentro de lo esperado, teniendo en cuenta la precisión de los sensores y motores utilizados. Al cabo de varios metros recorridos el error empieza a ser considerable. En recorridos de unos 8 metros incluida la ida y la vuelta a la casilla de inicio, tiene un error de entre unos 300 y 400 mm. Lo que más error produce son las trayectorias curvas, mayor velocidad angular es necesario peor, ya que los rozamientos hacen que las cálculos hechos con las lecturas de los tacómetros de los motores introduzcan demasiado error cuando el robot está girando, así como en las lecturas del giróscopo. En el caso de la brújula, aunque la precisión que da es bastante alta, la alta sensibilidad a interferencias hace habitual el tener lecturas con varios grados de error, por lo que no consigue corregir la posición tanto como se debería.

Se hicieron distintas pruebas, y estos fueron los resultados medios de las ejecuciones de los experimentos:

- **Robot diferencial**
  - **Trayecto de 4 metros en linea recta:** En 10 pruebas realizadas, el error medio en distancia fue de 15 cm.
  - **Giro sobre si mismo de 360º** En 5 pruebas realizadas, el error medio en grados era de 3º.
  - **Trayecto de 3 metros con giro:** En 10 pruebas realizadas, el error medio en distancia fue de 24 cm.
- **Robot tipo coche**
  - **Trayecto de 4 metros en linea recta:** En 10 pruebas realizadas, el error medio en distancia fue de 22 cm, y además en vez de llevar una trayectoria de 0º grados, podía llegar a errores de 5º.
  - **Circunferencia con radio mínimo de giro del robot:** El radio mínimo que podía girar el robot, debido a sus restricciones mecánicas, es de 38 cm. Al hacer una vuelta completa sin cambiar el radio de giro, el error medio es de 4 cm, prácticamente inapreciable.
  - **Trayecto de 3 metros con giro:** En 10 pruebas realizadas, el error medio en distancia fue de 27 cm.

Hemos observado en los experimentos que el robot coche no es capaz de mantener una linea recta durante mucho tiempo, ya que no es capaz de mantener las ruedas directrices en posición fija. Esto hace que pese a ir corrigiendo eso con la ayuda del giróscopo se produzca error. El comportamiento del robot diferencial en ese sentido es mucho mejor ya que si que es capaz de mantener una linea recta.

La conclusión es que estos robots no son capaces de ser autónomos durante grandes distancias, debido a que el error que van acumulando es demasiado grande conforme va pasando el tiempo. Quizá con un recorrido basado en balizas, que notificaran al robot de su posición real cada cierto tiempo si que se tendría un comportamiento más preciso.

3.3. Módulos y sensores para navegación autónoma

Para que un robot sea autónomo, debe ser capaz de navegar por su entorno de trabajo con la menor asistencia posible, es decir, sin que un humano vaya guiando sus pasos. En esta sección estudiaremos las características que permiten que los robots Mindstorm naveguen de forma autónoma, tales como un mapa del entorno para poder planificar movimientos a largo plazo evitando los obstáculos o la modificación en ruta del camino establecido por encontrar algún obstáculo que no era conocido en el cálculo inicial de la trayectoria.

El robot puede llegar a un objetivo con planificación o sin ella, todo dependerá de si le hemos cargado un mapa del entorno o no. El mapa se carga mediante la aplicación de control en el PC y se transmite por bluetooth. Una vez el robot tiene el mapa cuando se le indique una posición (x y) a la que ir, discretizará el mapa en celdas del tamaño indicado en el archivo de descripción del mapa, marcará cual es su casilla *Goal*, y estimara la mejor trayectoria para llegar a ella según el método de planificación NF1 [10].

Una vez tenga la ruta definida, hará trayectorias simples como las explicadas en el punto anterior entre su posición y el centro de la casilla siguiente de su ruta. Cuando esta a cierta distancia (definida también) del centro de la casilla, se marcará como objetivo el centro de la siguiente casilla de su ruta, de esta manera seguirá la ruta más corta hasta llegar a su destino.

Tanto la evitación de obstáculos que se puede producir durante el trayecto si algun elemento obstaculiza su ruta inicial, como el algoritmo de cálculo de dicha ruta óptima y su implementación serán explicados en los siguientes puntos de esta sección.

3.3.1. Algoritmo NF1

El algoritmo NF1 es un algoritmo de planificación y creación de rutas sobre un mapa. Es también llamado algoritmo de "gota de aceite".

El algoritmo consiste en recursivamente, y empezando por la casilla *Goal*, rellenar con pesos de las casillas adyacentes a las últimas procesadas, aumentando el peso conforme nos alejamos del "goal", de manera que las adyacentes al *Goal* tendran peso 1, las adyacentes (y no visitadas) de las de peso 1 tendrán peso 2, y así sucesivamente hasta asignarle un peso a la celda incial, donde se encuentra el robot actualmente. En la figura 3.7 puede verse un ejemplo. Puede observarse que solo tiene en cuenta las casillas adyacentes a las que es posible llegar "físicamente", es decir, si hay barreras por en medio u obstáculos los tendrá en cuenta y no la considera adyacente.

Dependiendo del algoritmo que se siga después para recorrer el mapa, una vez puestos los pesos, se tendrá una ruta u otra. Por ejemplo en la figura 3.8 puede verse una ruta en la que el robot no es capaz de moverse diagonalmente entre celdas, y donde la primera opción es moverse en X, y después en Y, de manera que siempre busca la casilla con peso menor en horizontal, y después en vertical. Y también otra ruta en la que el robot sí es capaz de moverse diagonalmente.

3.3.2. Evitación reactiva de obstáculos

La evitación reactiva de obstáculos consiste en detectar de alguna manera los obstáculos que impidan seguir el camino, y que no estaban previstos en la planificación, ni marcados en el mapa, y reaccionar con tiempo suficiente para evitarlo y llegar al destino por otro camino.

En los dos modelos de robot construidos, se ha colocado el sensor de ultrasonidos en la parte alta del robot, y alineado hacia el frente, de manera que sea capaz de detectar obstáculos que estén de frente. Una vez se detecta un obstáculo, se marca esa casilla como ocupada en el mapa, y se vuelve a recalcular la ruta mediante el algoritmo NF1 explicado anteriormente. Además, se notifica

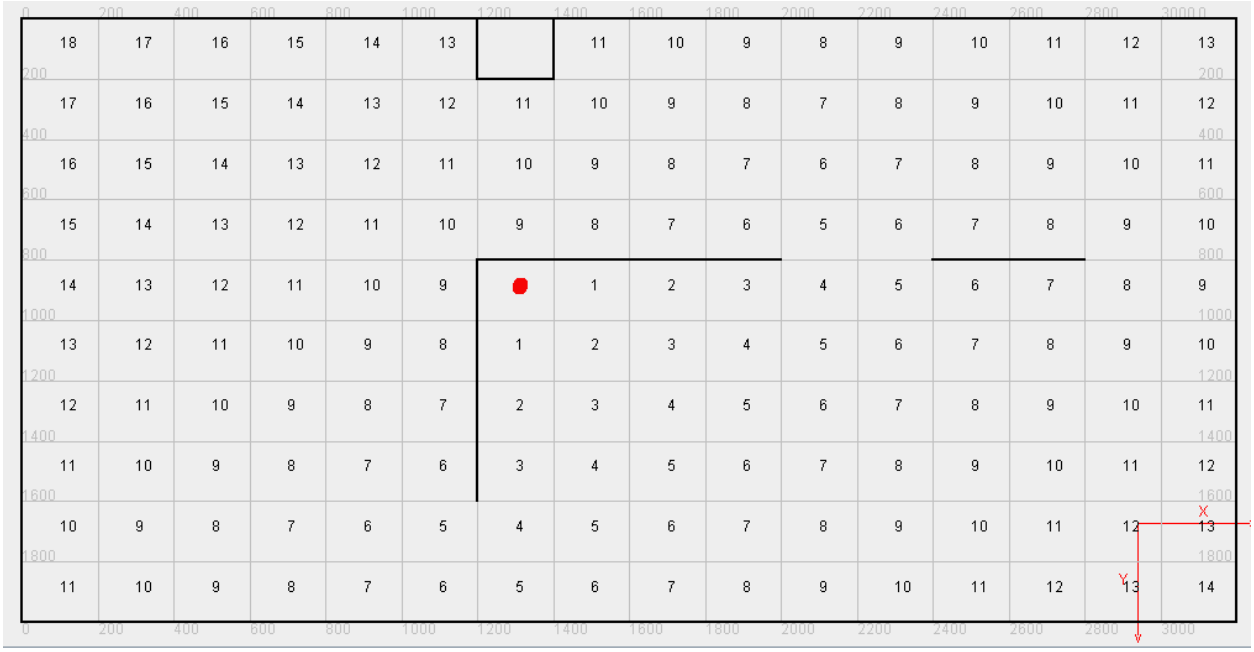

Fig. 3.7: Ejemplo de mapa rellenado con pesos mediante algoritmo NF1. El origen esta en la esquina superior izquierda y el objetivo o *goal* está marcado con un punto rojo. Las lineas rectas negras representan paredes o limites del mapa.

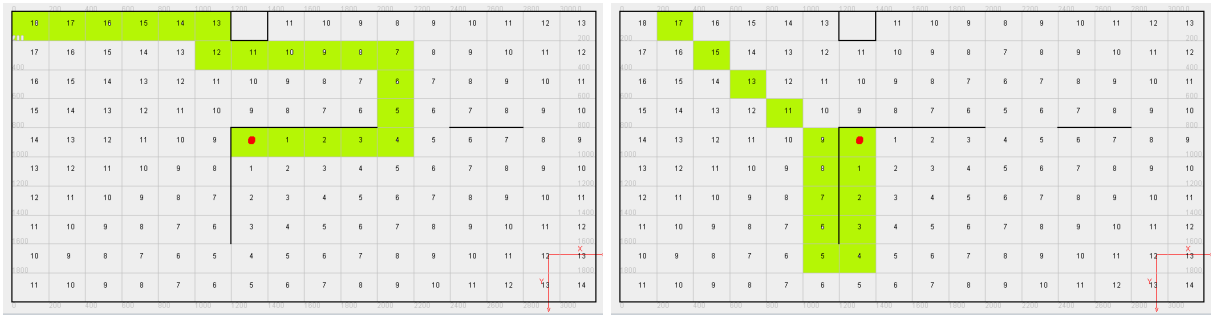

Fig. 3.8: Distintas rutas dentro del mismo mapa

por bluetooth al ordenador que controla y monitoriza el robot que se ha detectado el obstáculo, y la casilla que debe eliminarse para poder ver en pantalla la nueva ruta.

3.3.3. Implementación en el robot

Para la implementación de estas tareas de navegación, se creó la clase `NavigationMap`, que extiende `LineMap`, una clase de mapas de la API de leJOS. Este mapa consiste en las dimensiones del rectángulo del mapa, y va añadiendo lineas. Además en la implementación creada se incluye el concepto de celdas, que se extienden por todo el rectángulo. Funciones añadidas permiten conocer si una celda es adyacente a otra teniendo en cuenta las paredes añadidas al mapa.

Este mismo mapa está implementado en el código de la aplicación de monitorización, para tener una monitorización completa de la planificación que está llevando el robot.

Las operaciones de cálculo de rutas se hacen en el propio robot. Se analizó la posibilidad de que los cálculos se hicieran en el PC por su capacidad de cómputo más alta, pero se comprobó que las

operaciones no eran tan complicadas y el cálculo es instantáneo incluso realizándolo en el robot, por lo que la latencia de comunicaciones a través de bluetooth hubiera sido superior al tiempo que le cuesta calcular una ruta y rellenar el mapa de pesos con el algoritmo NF1.

El algoritmo está programado de forma recursiva, y se llama cada vez que indicamos desde el PC un nuevo destino al que el robot debe ir, o que el robot encuentra un obstáculo y debe recalcular su ruta.

Al igual que el resto de módulos, la navegación es una tarea que se ejecuta en un hilo diferente de ejecución al resto, de manera que el propio sistema de tiempo real del robot se encarga de distribuir el tiempo de CPU entre las diferentes tareas.

Al utilizar múltiples hilos de ejecución dentro de un sistema de tiempo real, es muy importante que las variables compartidas estén protegidas y que sean de acceso exclusivo. Este control de acceso en exclusión se realiza con la directiva de Java `syncronized`. Si no se hiciera esto existe riesgo de bloqueos entre las tareas o de información incorrecta en las variables compartidas debido a que estén leyendo y escribiendo varios hilos a la vez en las mismas variables.

3.3.4. Resultados y conclusiones

Tanto la planificación como la evitación de obstáculos funcionan de forma óptima, teniendo en cuenta las dificultades de cálculo de la posición explicadas en el apartado de cinemática.

El algoritmo de planificación asegura que siempre se encuentra una ruta óptima, en caso de existirla, y mediante la evitación de obstáculos en linea recta se es capaz de obtener una ruta alternativa sorteando los posibles imprevistos. En la figura 3.3.4 puede verse el resultado correcto de una planificación con obstáculos prefijados en el mapa. El camino pintado de verde es la ruta óptima cálculada, y puede verse la trayectoria real del robot superpuesta en todo el recorrido.

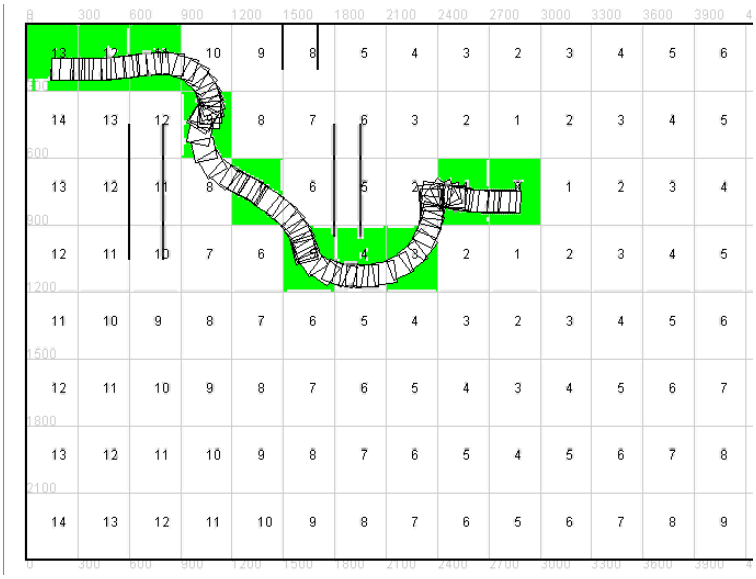

Fig. 3.9: Trayecto del robot diferencial con obstáculos conocidos en un mapa

En el caso del robot coche, hubo que ampliar la distancia a la cual el *Goal* se da por válido, y se pasa al siguiente, ya que al tener muy limitado el radio de giro, si se ponía una restricción demasiado grande el robot siempre estaba dando vueltas a la celda ya que nunca conseguía acercarse lo suficiente. Este umbral puede verse en la figura 3.10, comprobando la última posición del robot, y donde está la X que marca el *Goal* realmente.

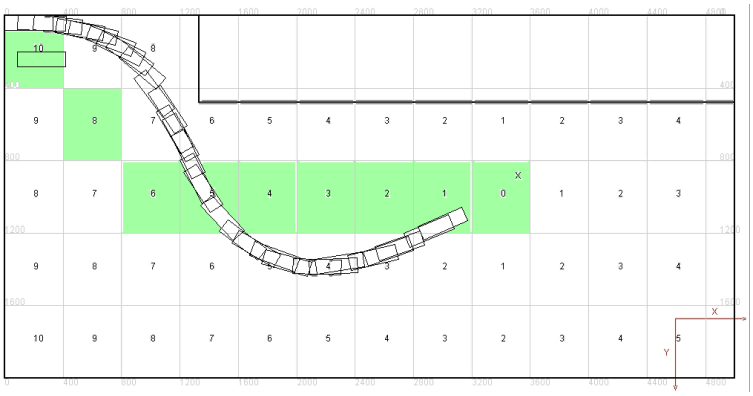

Fig. 3.10: Trayecto de robot coche con obstáculos prefijados

Los vídeos de las figuras 3.10 y 3.3.4 pueden verse en la carpeta videos:

- Robot diferencial navegando: *diferencial_navegando.mp4* (vídeo del robot) y *diferencial_navegando_captura.avi* (captura de lo que se monitoriza en la aplicación).
- Robot tipo coche navegando: *coche_navegando.mov* (vídeo del robot) y *coche_navegando_captura.avi* (captura de lo que se monitoriza en la aplicación).
- Robot diferencial evitando obstáculos: *diferencial_evitando.mp4*. (video del robot) y *diferencial_evitando_captura.avi*.

El mayor problema que se observó en la navegación con evitación reactiva de obstáculos, aparte del provocado por la estimación poco precisa de la posición, es que solo es capaz de ver obstáculos que estén en perpendicular con el sensor de ultrasonidos. Esto es así porque el ultrasonido lanza un haz que rebota, y teniendo en cuenta el tiempo que le cuesta volver al sensor se calcula la distancia a la que está, pero en un objeto que este colocado diagonalmente, el haz rebotará en otra dirección, no volviendo al sensor y por tanto siendo invisible para el. Por tanto aunque se ve que el sistema funciona, un sensor más avanzado, capaz de detectar objetos colocados en mas direcciones, dotaría de mucha más robustez al sistema.

3.4. Otros módulos

Ademas de los modulos anteriores que controlan las tareas principales de movimiento y navegación autónoma del robot, también se han desarrollado otros módulos para inicializar, controlar y monitorizar las tareas del robot desde el PC (comunicaciones y monitorización), para detectar objetos de color y seguirlos hasta atraparlos ("tracking" visual) y para construir modelos sencillos del entorno a partir de las medidas del sensor de ultrasonidos.

3.4.1. Comunicaciones

La comunicación entre robot y PC se establece por bluetooth, mediante las librerias PCComm incluidas en la API de leJOS para PC, y la librería nxt.comm de la API para el robot. Cuando se inicia la aplicación en el robot, tras inicializar sus variables, tareas y sensores, se queda escuchando por bluetooth hasta que el PC le llame y se establezca la conexión. El PC por su parte espera a que el usuario pulse en el botón de conectar para establecer la conexión.

Una vez establecida la conexión, se crean dos objetos en cada uno de los puntos, un objeto `DataOutputStream`y otro `DataInputStream`en cada uno de los extremos. Estos objetos permiten escribir, (en Output) y recibir (en Input). Hay dos hilos de comunicaciones también, uno encargado de escribir y enviar cuando hay algo que lo precise, como posición, sensores, mensajes de error, ordenes, y otro hilo encargado de escuchar, en el caso del robot cuando escucha una orden ejecuta los métodos necesarios para cumplirla, y en el caso del ordenador, mediante el patrón Observer se notifica a la ventana principal de la aplicación que debe dibujar algo, o notificar algo en el Log.

En este modulo, hemos desarrollado un pequeño protocolo para que la aplicacion y el robot sepan el tipo de información que les llega en cada trama y de cuantos bloques consiste. La lista completa de posibles mensajes entre ordenador y PC, así como la forma de sus tramas está completamente detallada en el anexo C.

3.4.2. Aplicación de monitorización y control

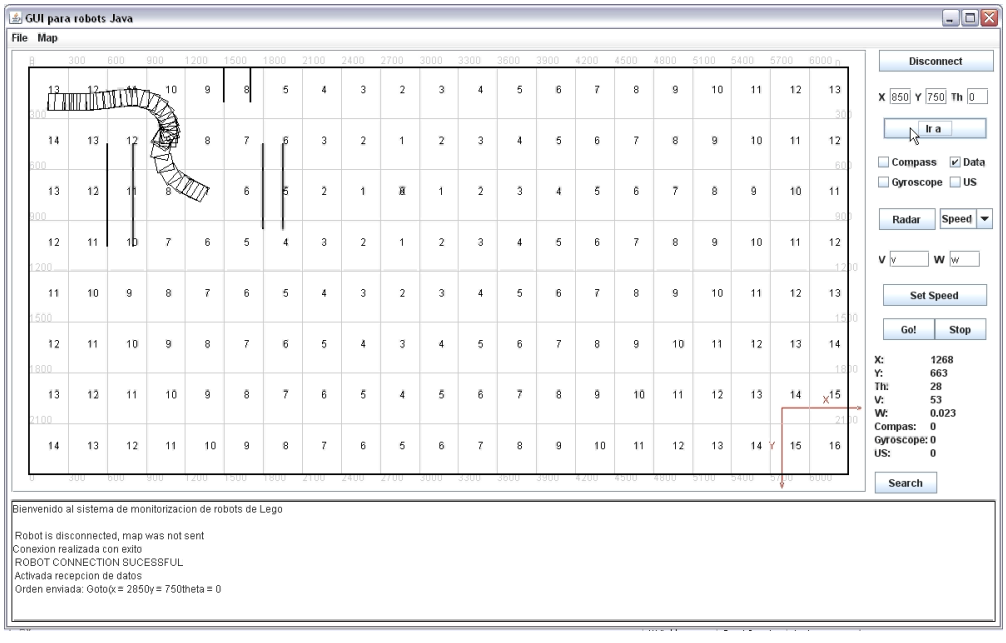

Fig. 3.11: Captura de la aplicación de monitorización en funcionamiento

La aplicación de control permite monitorizar todos los sensores de los que disponga el robot, así como enviar comandos de movimiento al robot, estableciendo nuevos puntos objetivo. También permite dar una velocidad lineal y angular para que se mueva libremente sin tener ningún objetivo, a modo de control remoto.

También dispone de un interfaz para cargar un archivo de mapa al robot. Este archivo contiene el tamaño de celda, el tamaño del rectángulo del mapa, y las paredes de dentro del rectángulo que pueda tener. Para representar obstáculos prefijados deben indicarse paredes en el archivo, y cuando el robot detecta uno por si mismo, lo representa dibujando cuatro paredes en los lados de la celda. La estructura de este archivo, así como un ejemplo utilizado, puede verse en el anexo D.

En la parte más grande de la pantalla, se ve el mapa que se está usando, así como un rectángulo que representa al robot a la escala adecuada. Tiene regla en la parte superior, inferior y lateral para poder ver en que punto del espacio se encuentra. Cuando el robot detecta un obstáculo, lo comunica a la aplicación de control, que tiene constantemente un hilo escuchando en el puerto COM del bluetooth [11].

El manual completo de la aplicación se encuentra en el anexo B.

3.4.3. Seguimiento visual de objetos

En el robot diferencial además se incluyó la funcionalidad de *tracking* o seguimiento de objetos, utilizando la cámara. Esta cámara permite detectar objetos de colores predeterminados y localizarlos respecto al robot y estimar su tamaño. La tarea de ejemplo implementada en el robot, consiste en buscar una pelota de color rojo y cogerla con las pinzas.

El objetivo del algoritmo iterativo de seguimiento, es conseguir que la imagen actual capturada por la cámara se parezca al objetivo explicado en la figura 3.12, conseguir centrar el punto *c* (centroide del blob) respecto al centro de la imagen C y conseguir que el area del blob, a, sea de un tamaño determinado (A), que nos indicaria que estamos a una distancia adecuada para atrapar el objeto.[15].

La cámara utilizada es la NXTCam 2.4.1, debe calibrarse previamente con el programa CamView [12]. Este programa permite configurar los colores que queremos que detecte la camara, dando los rangos de RGB [13] a tener en cuenta. Este proceso es delicado, ya que si damos demasiada gama de colores puede acabar mezclando zonas del suelo u otro objeto con el objeto del color deseado, y si se da un rango demasiado pequeño puede que nunca llegue a ver el objetivo con el suficiente tamaño como para cogerlo.

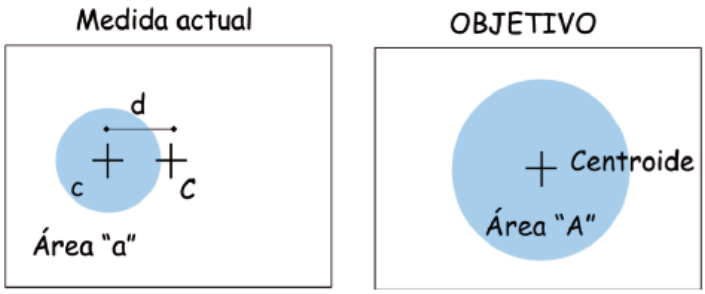

Fig. 3.12: Seguimiento de un objeto. Izquierda: procesamiento de una imagen capturada por la cámara. Derecha: imagen que queremos conseguir obtener, porque nos indicara que el seguimiento puede finalizar (el objeto estará centrado y cerca).

Estos son los pasos que se realizan en cada iteración del proceso de seguimiento:

- Recoge todos los blobs o rectángulos de colores identificables por la cámara que está viendo.
- Analízalos para coger el más grande de color rojo.
- Si es más grande que cierto umbral, gira hacia el lado por el que lo has visto (intentar centrar el punto c, o si está de frente avanza recto.
- Cuando estés lo suficientemente cerca, el tamaño del blob será mayor que cierto umbral, en ese caso abre la pinza, y sigue avanzando.
- Si es más grande que otro tamaño dado cierra la pinza y para los motores.
- Si al cabo de un segundo, sigo viendo un blob de tamaño grande justo enfrente, es que tengo la pelota cogida por las pinzas (cabe la posibilidad de que la hubiera empujado al intentar cogerla).

En la carpeta vídeos del cd, hay dos videos en los que se puede ver el robot haciendo el seguimiento (`tracking1.mp4`, `tracking2.mp4`).

3.4.4. Radar

Como complemento a la evitación de obstáculos, se montó sobre el robot de tipo coche una plataforma acoplada a un motor. Encima de esa plataforma está el sensor de ultrasonidos, con el objetivo de detectar obstáculos cuando se está navegando, y de utilizarse como radar para dibujar el entorno de vez en cuando, para mantener un mapa sencillo del entorno actual.

Cuando el usuario ordena desde la aplicación de control que se utilice el radar, indicando la velocidad, y por tanto la precisión que este dará, la plataforma empieza a girar. Mientras va girando, se van almacenando las lecturas del sensor de ultrasonidos, concretamente una por cada grado que se gira, obteniendo al final un total de 360 lecturas.

Esta información se transmite al ordenador de control, el cual teniendo en cuenta la posición y orientación del robot, traslada esos puntos a las coordenadas X e Y del mapa, plasmándolos y uniéndolos con lineas.

De esta manera se obtiene una visión del entorno que rodea al robot, con paredes y obstáculos.

3.4.5. Resultados y conclusiones

Seguimiento visual El *tracking* funciona muy bien, pero solo para el color rojo, ya que es el único que es el que más gama puede ver sin confundirlo con otro color. Por ejemplo con el azul o el verde, cuando el azul es muy oscuro, porque le pega la sombra o es la parte de abajo de la pelota que está menos iluminada, lo confunde con el negro. También existen problemas si la luz no es muy fuerte, ya que entonces ve todo de color anaranjado, siendo incapaz también de reconocer la pelota.

Se realizaron diversas pruebas en distintos escenarios, con más y menos luz, y los resultados fueron que sólo en ambientes muy iluminados era capaz de coger la bola. De cinco pruebas fue capaz de cogerla a la primera dos veces, en otras dos ocasiones no retuvo la pelota en las pinzas en el primer intento, pero si en el segundo, y tan solo en una ocasión no fue capaz de cogerla, ya que al acercarse la empujó demasiado lejos, no pudiendo rastrearla de nuevo.

Comunicaciones y monitorización La aplicación de monitorización se fue ampliando conforme se iban utilizando más sensores. Su funcionamiento se puede ver en todos los vídeos de "captura". Las comunicaciones no dieron ningún problema, ya que se trabajó en entornos cerrados y pequeños, donde el alcance del bluetooth no daba lugar a perdidas de tramas, ni a desconexiones del robot o del PC.

Fig. 3.13: Robot cogiendo la bola

Radar Los resultados de las capturas del radar no son muy precisos. En la figura 3.14 puede observarse que la representación del entorno es demasiado pobre, aunque si que es capaz de detectar las paredes, el pasillo y la caja con el ordenador encima, no es suficiente como para tener una representación real del espacio. Además solo es capaz de ver las paredes que se encuentren en posición completamente perpendicular al sensor, ese es el motivo de cubrir las esquinas del pasillo con cajas perpendiculares a la trayectoria en linea recta del sensor.

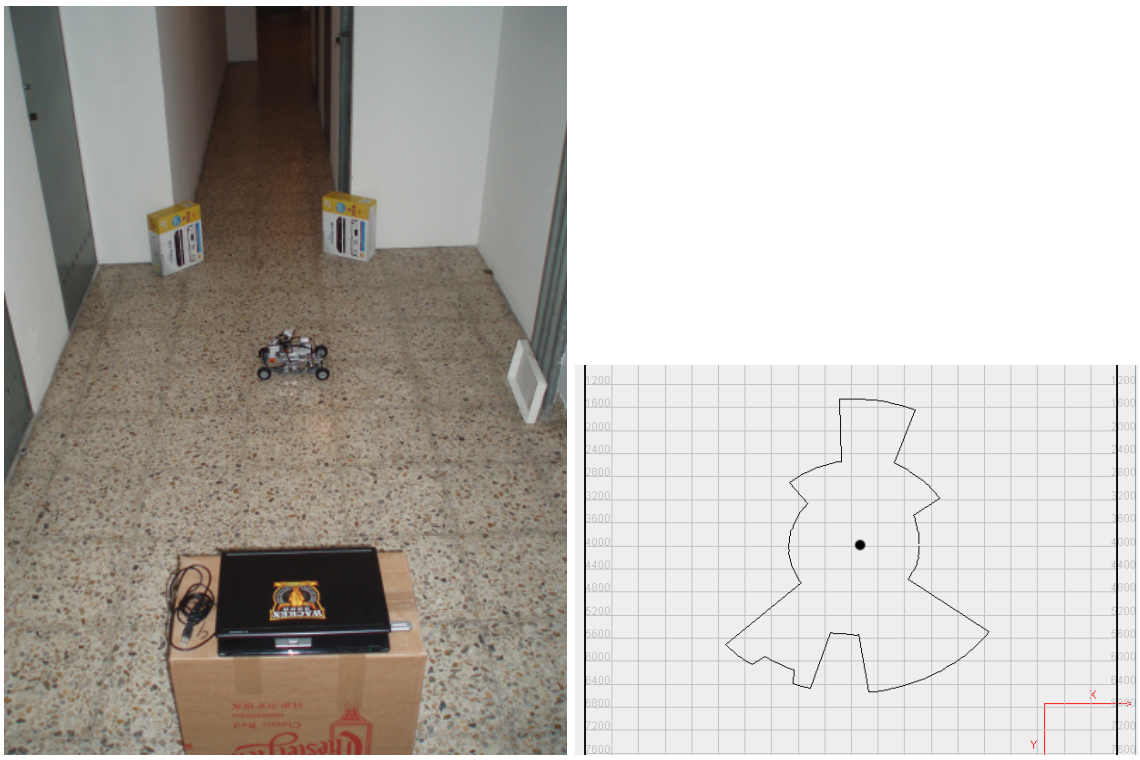

Fig. 3.14: Imagen real y captura del radar. La posición del robot se representa con el punto central negro, y las lineas representan a la distancia que el radar ha detectado obstáculos en todas las direcciones.

4
Conclusiones

4.1. Conclusiones sobre los resultados obtenidos

Esta sección resume las conclusiones obtenidas a partir de los distintos módulos desarrollados.

En cuanto a temas relacionados con la cinemática, se puede concluir que los sensores y odometría de estos robots no dan una precisión alta, y por tanto no pueden ser completamente autónomos. La precisión de los sensores y los componentes mecánicos, como era de esperar, no dan robustez y precisión para tareas robóticas de larga duración. Pero si que puede ser de utilidad en asignaturas de mecánica o robótica ya que la precisión si es suficiente como para aprender a trabajar con robots, y que estos conocimientos sean aplicados más adelante en robots de más tamaño y más posibilidades.

Acerca de la navegación, los resultados son buenos salvo la perdida de precisión en la estimación de la posición con el tiempo. Si que se ve que el sistema responde como debería, llevando trayectorias óptimas y evitando obstáculos. La poca fiabilidad de la estimación de la posición a largo plazo hace que la navegación no pueda utilizarse en grandes distancias, pero en un entorno más o menos pequeño de unos 5x5 metros los resultados son muy positivos. La utilización de técnicas estudiadas en asignaturas como inteligencia artificial o robótica pueden mejorar los resultados obtenidos añadiendo nuevos tipos de mapas, o posicionamiento mediante balizas, o mediante nuevos sensores como un receptor GPS.

En los temas de monitorización, la aplicación funciona muy bien, dando lecturas en tiempo real de lo que está haciendo el robot, sin notar apenas retardo por el uso de bluetooth. Como el robot y estos experimentos se hicieron en distancias cortas, el canal bluetooth es suficiente para tener unas buenas comunicaciones, en caso de que las distancias entre el ordenador de control y el robot fueran más grandes quizá si que podrían detectarse perdidas de paquetes o mala recepción, pero no es el caso.

Como conclusión final se puede decir que los robots Lego Mindstorms son una buena base para aprender robótica, inteligencia artificial y mecánica, y que la implementación de algoritmos estándard utilizados en robótica son perfectamente aplicables. No proporcionan la precisión que dan sus hermanos mayores, pero si unos buenos resultados en nivel general. Además la alta flexibilidad a la hora de construir y programar hace que las posibilidades de investigación, ya sea como pruebas de concepto, o como recursos didácticos sean enormes.

4.2. Trabajo futuro

El bloque NXT ha evolucionado durante la realización de este proyecto al nuevo bloque NXT 2.0, asimismo la API de leJOS también ha sido ampliada y renovada en este periodo, por tanto el trabajo futuro pasaría por adaptar todos los módulos a los nuevos robots.

Además, el experimentar con nuevos sensores, nuevas formas de navegación, o distintos modelos de cinemáticas es otro camino por el que ampliar la investigación en estos robots. Por ejemplo se podrían construir robots humanoides, o *Segways*, y añadir funcionalidades para que fueran capaz de comunicarse entre ellos, ya sea directamente o a través del ordenador de control. Podrían construirse robots con más de un bloque, permitiendo así colocar más motores y más sensores, que coordinándose entre si de manera eficiente dieran resultados mejores.

En general, la gran flexibilidad en la construcción y programación hace que las posibilidades en un futuro de proyectos con Lego Mindstorms sean muy grandes.

4.3. Conclusiones personales

En el terreno personal, he aprendido y disfrutado muchísimo con este proyecto, ya que desde pequeño siempre me ha apasionado la robótica y las construcciones de Lego, y el poder realizar mi

proyecto fin de carrera uniéndolos ha sido una experiencia que poca gente puede vivir.

He podido desarrollar mis capacidades de programación, así como aprender mecánica, trigonometría y física. Cuando realizas proyectos de este tipo es cuando te das cuenta de que todas matemáticas y física que te enseñan en los primeros años de ingeniería, y que en su momento piensas que no vas a utilizar jamás, al final resultan ser importantísimos.

Me gustaría agradecer a los miembros del departamento la ayuda prestada, en forma de tutorías, material bibliográfico. Por la flexibilidad que me dieron para poder disponer de los robots de Lego en cualquier momento, y por la ayuda inmediata que recibí cuando más atascado estuvo el proyecto.

Bibliografía

[1] Jonathan B. Knudsen. The Unofficial Guide to LEGO MINDSTORMS Robots. O'Reilly 1999.

[2] Paul Wallich. Mindstorms Not just a kid's toys. Artículo en IEEE Spectrum. Noviembre 2001.

[3] D. Benedettelli, M. Casini, A. Garulli, A. Giannitrapani, A. Vicino. A LEGO Mindstorms experimental setup for multi-agent systems. In Proceedings of the 3rd IEEE Multi-Conference on Systems and Control, pp. 1230-1235, St. Petersburg (Russia), July 8-10, 2009.

[4] "Programmable Bricks". Projects. MIT Media Lab. http://llk.media.mit.edu/projects.php?id=135.

[5] All the tools to take your LEGO MINDSTORMS NXT to the Extreme. http://mindstorms.lego.com/ Overview/ NXTreme.aspx

[6] Software Bricx Command Center 3.3 [En linea] disponible en: http://bricxcc.sourceforge.net/

[7] Software Eclipse. http://www.eclipse.org/

[8] Lenguajes de programación Next Byte Codes & Not eXactly C. http://bricxcc.sourceforge.net/nbc/

[9] LEJOS: Java por Lego Mindstorms. http://lejos.sourceforge.net

[10] J.-C. Latombe. Robot motion planning. Kluwer Academic Publishers, Dordrecht, Netherlands, 1991.

[11] Palo Wireless . RFCOMM Protocol. http://www.palowireless.com/infotooth/tutorial/rfcomm.asp

[12] Software para calibración de cámara. NXT CamView. http://nxtcamview.sourceforge.net/ &PAGE_user_op=view_page&PAGE_id=78

[13] Charles A. Poynton Digital Video and HDTV: Algorithms and Interfaces. Morgan Kaufmann. 2003

[14] R. Siegwart, I.R. Nourbakhsh. Introduction to Autonomous Mobile Robots. MIT, 2004

[15] L. Montano, A.C. Murillo. Documentación de la asignatura Robótica de Servicio. CPS de Ingenieros. Universidad de Zaragoza. 2009-10