Robot Velocista de Competición basado en CORBA

Luis Jiménez Gañán

marzo de 2006

Agradecimientos

Agradezco en primer lugar el apoyo de mi familia que siempre ha estado ahí. En especial a mis padres José Luis y Julia, que han seguido la evolución del proyecto desde Almazán, a Isabel que me ha echado una mano aunque no llegara a entenderlo y a Miguel, con sus comentarios siempre acertados.

A Ricardo, por confiar en mí, por esas clases magistrales tomando un café o en la sobremesa y, sobre todo, por enseñarme siempre ese otro enfoque que tienen todas las cosas.

A Carlos, porque sin su ayuda no habría sido capaz de resolver muchos de los problemas más complicados, porque siempre ha estado dispuesto a echarme una mano o a explicarme lo que hiciera falta.

A José Emilio, por infinidad cosas, por echarme una mano con las pruebas, por esas fotos, por tantas veces que me ha prestado herramientas y por enviarme archivos que me dejaba en el departamento.

A Raquel, por estar siempre pendiente y dispuesta a echarme una mano, por esos consejos, esas correcciones y esos ánimos que tanto hacen falta a última hora. Mil gracias.

A Álvaro, por escucharme y apoyarme en los días malos en que parece que el proyecto no se va a terminar nunca. A Paco, que es el culpable de varias de las mejores ideas del robot. A David, que siempre ha aportado la visión más "química" de cualquier cosa. A Quique, por echarme una mano con el robot y con esa puesta a punto. A José Luis, que ha compartido conmigo esos nervios de última hora y esas prisas por imprimir. A Elena, por ese apoyo y por distraerme de las preocupaciones del proyecto. A Darío, que no ha dudado en echarme una mano cuando me hacía falta. A todos los compañeros del DISAM, por esos ratos de desestres despues de comer. A Adolfo a Ignacio por mostrar interés. A Jorge y Nera, por esos pequeños consejos tan acertados.

A todos los compañeros de clase y amigos, a María, Guillermo, Alberto,

Juan Pablo, Marci, Dani, Pepe y a todos los demás. Porque gracias a vosotros se han pasado volando los últimos años de la carrera. A Jose, María, Chema, Fer, Manu, por esos partidos de fútbol, esas cañas y esos ratos tan buenos.

A todos mis amigos en general, a Ricardo, Julio y a todos los demás, por los buenos y malos momentos. Por insistir en que vaya más fines de semana a Almazán, por los ratos en la peña ...

A los profesores que me he encontrado por la carrera, a los buenos y los malos, que han conseguido que aprenda algo, tanto si estaba relacionado con sus asignaturas como si no. Y a todos los alumnos de esta escuela con los que alguna vez he coincidido y que alguna vez me han aportado algo diferente a lo convencional.

A quienes me he olvidado en la lista de agradecimientos y, en general, a todos los que han contribuido para que haya podido terminar este proyecto. A todos, ¡GRACIAS!

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo se describen las razones por las que se ha realizado este proyecto y se definen sus objetivos. Además se inica brevemente como se ha realizado el proyecto y como está organizado este documento.

1.1. Motivación

Resulta obvio que en la construcción de sistemas complejos de control intervienen muchos factores que determinan la necesidad de emplear arquitecturas software adecuadas. Esta adecuación se mide por el grado en que dichas arquitecturas permitan obtener tanto *características funcionales* determinadas (como son la satisfaccion de requisitos estrictamente necesarios en el ámbito del control) como *características no funcionales* (que son críticas a la hora de contruir un sistema software complejo).

La estructuración jerárquica de los sistemas de control tiene una gran importancia cuando de esta estructura se deriva la facilidad de construccion y descpliegue o incluso el*correcta funcionamiento* en condiciones nominales o incluso de fallo parcial.

De especial importancia es este aspecto cuando los niveles de inteligencia están aumentando de forma ubiqua en todos los elementos de un sistema de control que llega a consistir en una colectividad de agentes distribuidos que llegan hasta el máximo nivel de empotramiento.

Este proyecto fin de carrera se sitúa en la confluencia de ambas necesidades —empotramiento y arquitectura software— tratando de ofrecer una solución general al problema concreto del control escalable de microrobots

de competición.

Obviamente en este proyecto no se trata de ofrecer una solución definitiva a todos los problemas de arquitecuras complejas empotradas —tarea posiblemente demasiado ambiciosa para un proyecto fin de carrera— sino de proponer una arquitectura de control concreta que se ofrece como una solución viable a uno de los problemas planteados por la necesidad de control sofisticado que los microrobots de competición plantean.

1.2. Contexto

Este trabajo se enmarca dentro del proyecto de investigación a largo plazo CS^2 — Complex Software-intensive Control Systems. Este es un proyecto de investigación en tecnologías software para sistemas complejos de control realizado por el Autonomous Systems Laboratory (ASLab) de la UPM, dentro del Departamento de Automática de la ETS de Ingenieros Industriales de la Universidad Politécnica de Madrid.

El objetivo de este proyecto investigador es la definición de arquitecturas de control integrado para la construcción de sistemas complejos de control y el desarrollo de tecnologías software para su construcción. Las líneas maestras son simples y guían todo el desarrollo del proyecto:

- Modularidad
- Seguimiento de estándares
- Reutilizabilidad
- Diseño basado en patrones
- Independencia del dominio

En este contexto, se ha definido una plataforma software genérica de base denominada **Integrated Control Architecture** (**ICa**) [18] que proporciona los criterios de diseño software centrales. La arquitectura **ICa** es una meta-arquitectura software que ofrece una serie importante de ventajas frente a otros enfoques :

Coherente y unificada: La arquitectura **ICa** proporciona integración, vertical y horizontal, total y uniforme; por ejemplo permite emplear un

única tecnología en todos los niveles en la verticalidad del sistema de control (desde la decisión estratégica hasta los dispositivos empotrados de campo) así como la integración horizontal de unidades de negocio o empresas extendidas.

Clara: El modelo empleado en ella es un modelo preciso: el modelo de objetos ditribuidos de tiempo real que constituye la base de las plataformas estado del arte en este campo.

Flexible y extensible: Permite su adaptación a distintos contextos de ejecucion y distintos dominios al realizar un mínimo de compromisos de diseño; lo que permite la reutilización modular de componentes y la incorporación de nuevos componentes en dominios concretos.

Abierta: Permite la interoperabilidad con sistemas ajenos (tanto heredados como futuros).

Portable: Gracias a basarse en estándares internacionales.

Esta arquitectura —o metaarquitectura como debe ser considerada **ICa** en realidad [14]— se ha venido desarrollando en nuestro departamento durante los últimos años y, gracias a diferentes proyectos de I+D, se ha aplicado con éxito en multiples ámbitos de control:

- Control estratégico de procesos de fabricacion de cemento [16].
- Gestión de emergencias en plantas químicas [17].
- Sistemas de monitorización distribuida de tiempo real de produccion y distribución de energía eléctrica [6].
- Robots móviles cooperantes [15].
- Bucles de control en red [13].
- Protección de subestaciones eléctricas [12].
- *etc.*

La característica primaria de **ICa** es su enfoque modular. Los sistemas se contruyen por medio de módulos reutilizables sobre *frameworks* de sistemas distribuidos de objetos de tiempo real. La organización concreta de

los módulos viene dada por su arquitectura de aplicación, que se deriva de la metaarquitectura por medio del uso de patrones de diseño [19]. Su despliegue se hace según las necesidades y restricciones de la aplicacion, explotando la reubicabilidad de los componentes (ver Figura 1.1).

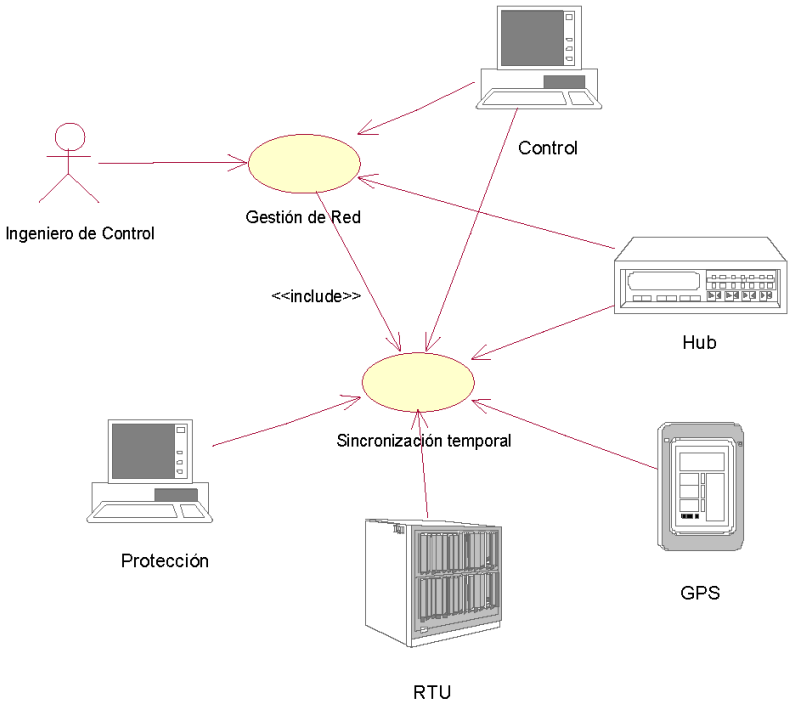

Figura 1.1: Diagrama de caso de uso para un despliegue de una aplicación de control y protección de redes.

1.3. Objetivos del proyecto final de carrera

El objetivo de este Proyecto Final de Carrera es dotar a un robot móvil, de los que se usan en competiciones de microrrobots, de lo necesario para poder realizar sobre el mismo un control de alto nivel.

Para conseguir éste objetivo se realizará un robot de competición con tecnología software convencional, concretamente un robot para participar en la categoría de velocistas que hay, o había, en las competiciones de robots como Hispabot, Robolid o Ciutat de Mataró.

Con la intención de poder controlar el robot de manera remota se le dotará de algún tipo de comunicación inalámbrica, de una plataforma sobre la que se puedan ejecutar aplicaciones basadas en CORBA y se realizará una aplicación que permita realizar la monitorización del estado del robot y el control del mismo desde un ordenador remoto.

Se empleará tecnología CORBA en vez de otras arquitecturas de desarrollo de software distribuido, como DCOM de Microsoft o Java RMI, por la independencia que tiene CORBA de la plataforma y del lenguaje de programación empleado, pero además y muy importante, porque CORBA es la única de estas arquitecturas sobre la que se pueden construir aplicaciones de tiempo real, las cuales, aunque no van a ser usadas en este proyecto, pueden ser requeridas en una aplicación o extensión de este Proyecto Final de Carrera.

1.4. Metodología

Para el desarrollo del proyecto se ha seguido una metodología evolutiva, que encaja con la naturaleza escalable del proyecto final. Se pueden distinguir tres grandes bloques.

- Un primer bloque en el que se construye un robot de competición mediante tecnología software convencional. Esta parte es muy amplia porque incluye la construcción mecánica del robot, la sensorización del mismo y su control mediante un microcontrolador. El resultado de esto es un robot apto para participar en competiciones, de hecho este es el estado en el que ha participado en algunas de ellas.

- El siguiente paso supone añadirle un nivel superior de inteligencia. Como el objetivo es construir un robot basado en CORBA he escogido la plataforma *Gumstix*[1] sobre la que corre un sistema operativo linux. Se trata de una placa de pequeño tamaño y bajo consumo que se puede conectar a una red LAN mediante Ethernet y WiFi y que podrá ejecutar una aplicación CORBA.

- La última fase consiste en proporcionarle una tercera aplicación que se comunicará con la placa *gumstix* mediante CORBA (la discusión sobre cual de las dos aplicaciones hace de servidor o cliente, o si ambos

[1]Información disponible en `www.gumstix.com` y `www.gumstix.org`

desempeñan los dos roles se decidirá más adelante en función de la aplicación) que se encargará de un control de alto nivel y que tendrá una interfaz gráfica para interaccionar con el robot y para monitorizar su estado.

1.5. Estructura del documento

Esta memoria de Proyecto Fin de Carrera se encuentra organizada en 6 capítulos y 2 apéndices, cuyo contenido se resume a continuación:

Capítulo 1: Introducción En este capítulo se presenta el proyecto y se da una idea global de su entorno, alcance y de las ideas básicas.

Capítulo 2: Robots de competición Se muestran los distintos tipos de robots de características similares al del objeto de este Proyecto Fin de Carrera, así como los tipos de pruebas en los que éstos participan y las innovaciones más destacables que emplean.

Capítulo 3: Hyperion, un velocista muy particular Ésta es la parte que describe el robot en sí, es decir, cómo es físicamente, qué ha de ser capaz de hacer y qué hardware lleva incorporado para ello.

Capítulo 4: Plataformas software Aquí se describen las herramientas software más importantes empleadas en el desarrollo del proyecto así como su uso.

Capítulo 5: Implementación software En este apartado se detalla el software que se ha implementado para el Proyecto Fin de Carrera, desde la distribución de tareas entre los diferentes microprocesadores hasta las aplicaciones definitivas en cada uno de ellos.

Capítulo 6: Conclusiones y líneas futuras En este capítulo se resumen las principales conclusiones que se han obtenido con el desarrollo del proyecto. Además se exponen algunas de las líneas de investigación en las que se puede seguir trabajando para ampliar lo que se ha conseguido con el mismo.

Apéndice A: Programación del proyecto En este apéndice se incluye un diagra de Gantt con la programación del proyecto.

Apéndice B: Otras herramientas Aquí se describen brevemente otras aplicaciones empleadas durante el desarrollo del proyecto.

Capítulo 2

Robots de competición

En este capítulo se van a describir los tipos de competiciones en las que participan distintos tipos de robot. También se describen los robot y la tecnología que emplean.

2.1. Presentación

Tanto a nivel nacional como a nivel internacional existe una gran cantidad de competiciones en las que el objetivo principal es que uno o varios robots autónomos[1] realicen una determinada tarea tratando de conseguir un mejor resultado en términos de tiempo u objetivos alcanzados que el resto de participantes, o enfrentándose literalmente a ellos.

La mayoría de estas competiciones están destinadas a estudiantes (principalmente de ingeniería, pero cada vez más también a alumnos más jóvenes en institutos) y tienen asociada una finalidad educativa importante ya que implican una aplicación práctica de los conocimientos adquiridos en materias como electrónica, robótica, regulación automática y programación. Además permiten desarrollar capacidades de trabajo en equipo y plantean un reto importante al enfrentar a los participantes a problemas reales que han de resolver de manera ingenieril.

En la Escuela Técnica Superior de Ingenieros Industriales de la Univer-

[1]El nivel de autonomía depende de cada competición, en la mayoría de ellas se exige un robot totalmente autónomo, pero hay categorías en las que es necesaria la intervención de un operador humano, y cada vez hay más competiciones en las que varios robots tienen que colaborar para conseguir un objetivo común

sidad Politécnica de Madrid se celebra desde el año 2000 se organiza una de estas competiciones, **Cybertech**. Se trata de una competición en la participan alumnos de la escuela desde los primeros hasta los últimos cursos, y en ella compiten en pruebas de rastreadores, velocistas y en una prueba única en el mundo como es *Robotaurus*, en la que los robots de los participantes intentan torear a un toro que también es un robot.

Asociada a la competición de Cybertech existe también en la escuela una asignatura de libre elección que ayuda a los alumnos a construirse dichos robots.

La experiencia adquirida desde la participación en esta competición hasta la organización de la misma y la colaboración con la asignatura han permitido al autor de este proyecto tener una cierta práctica en la construcción de robots y en el funcionamiento de dichas competiciones.

2.2. Competiciones más importantes

Dentro de la muchas competiciones tanto nacionales como internacionales voy a destacar algunas de ellas que destacan por una amplia participación, por el elevado nivel tecnológico que en ellas se exhibe o por el empeño que tienen en el aprendizaje de los participantes en cuestión de robótica.

Eurobot. Se trata de una competición que se desarrolla cada año en una ciudad europea, aunque también participan equipos de otros continentes. La única prueba en la que se compite en este concurso cambia cada año manteniendo siempre una serie de características como el desarrollo de las eliminatorias, que siempre enfrenta a dos equipos en un partido de minuto y medio, y unas dimensiones de los robots de 20x30 cm. Para conocer las normas de cada año se puede acudir a `www.eurobot.org`.

FIRST *For Inspiration and Recognition of Science and Technology.* Es la competición más importante de robots en Estados Unidos y Canada para estudiantes desde colegios a universidades y para ingenieros. Hay pruebas nuevas cada año y los participantes tienen 6 intensas semanas para diseñar, construir y probar sus prototipos. Toda la información disponible en su página web `www.usfirst.org`.

RoboCup es un proyecto internacional con la intención de promocionar la inteligencia artificial y la robótica. Tienen un objetivo concreto a largo

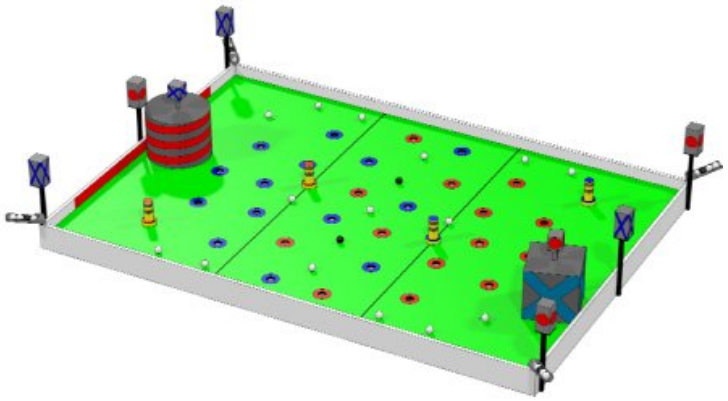

Figura 2.1: Funny Golf: prueba de Eurobot 2006

plazo, ser capaces de crear un equipo de robots humanoides autónomos que en el año 2050 sea capaz de ganar al mejor equipo humano de fútbol de mundo. Cada año organizan una competición en una ciudad diferente a lo largo de todo el planeta en las que hay diferentes pruebas relacionadas con el fútbol. Más información en `www.robocup.org`.

All-Japan Robot-Sumo Tournament es una competición de robots que se realiza en Japón y cuyos momentos más esperados son las competiciones de sumo en las categorías de 3 y 5 kg. Presume de ser la competición de robots más grande del mundo con 3000 robots participantes

Grand Challenge es una competición que se ha realizado en dos ocasiones por la Agencia de Investigación Avanzada del Departemento de Defensa de Estados Unidos. El objetivo es construir un vehículo autónomo capaz de navegar a lo largo de un desierto en una prueba de más de 200 km. Grandes empresas de todo el mundo como Intel o Volkswagen emplean en esta competición su más avanzada tecnología. Todos los datos de la competición del año 2005 se pueden ver en `www.grandchallenge.org`.

Hispabot es una de las competiciones más importantes de microrrobots dentro del panorama nacional. Con una gran tradición se celebra en la localidad de Alcalá de Henares esta competición en la que, además de pruebas estándar como las de sumo o de laberinto, se realiza la selección de los 3 equipos españoles que participan ese año en Eurobot. Más información disponible en la página web del Departamento de Electrónica de la Universidad de Alcalá `www.depeca.uah.es`.

2.3. Tipos de pruebas

Aunque existen infinidad de pruebas en las diferentes competiciones de microrrobots, hay una serie de ellas que, ya sea por su simplicidad o por el interés especial que despiertan, se pueden ver en muchas de estas competiciones. Si bien es cierto que la organización de cada competición tiene reglamentos propios y tratan muchas veces de añadir algo que diferencie sus pruebas de otras, muchas de las características son comunes, y estas son las que se exponen a continuación.

2.3.1. Rastreadores

La prueba de rastreadores es sin duda la más extendida y practicada de todas las pruebas disponibles. Las bases de la prueba son muy sencillas, se trata de que el robot siga un circuito marcado por una línea negra sobre un fondo blanco (también se realiza al revés, usando líneas blancas sobre un fondo negro). El robot ha de ser totalmente autónomo, como en todas las pruebas "estándar", y en general tendrá unas dimensiones limitadas. Lo más habitual son unas dimensiones máximas de aproximadamente 20x30 cm, la altura y el peso no suelen sufrir restricciones.

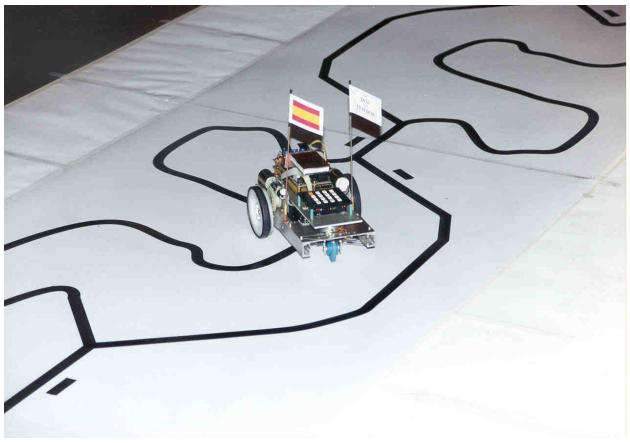

Figura 2.2: Un rastreador en funcionamiento

Pero esas solo son las características básicas de la prueba, porque siempre hay complicaciones adicionales:

- La trayectoria a seguir acostumbra a ser muy sinuosa y en muchos casos

las curvas pasan a ser ángulos, incluso más cerrados que un ángulo recto.

- Es muy habitual encontrarse con bifurcaciones, en algún punto de la trayectoria ésta se divide en dos previa indicación de cuál es el camino correcto mediante una marca en paralelo a la línea a seguir antes de llegar a la citada bifurcación. Que un robot se vaya por el camino equivocado puede suponer una pérdida de tiempo, de puntos y llevar a una descalificación.
- Existen otras variaciones como la existencia de tramos discontinuos en la línea que marca la trayectoria en la trayectoria o caminos que terminan en una calle cortada, pero estas dificultades no se ven en muchas competiciones.

2.3.2. Velocistas

La competición de velocistas es otra de las pruebas más habituales. Guarda bastante relación con la de rastreadores puesto que también se trata de seguir un circuito marcado con líneas negras sobre un fondo blanco, pero mientras a los robots rastreadores se les exige que sean capaces de seguir un camino complicado, lleno de curvas y bifurcaciones, lo que se espera de los robots velocistas es que, como su propio nombre indica, sean capaces de desarrollar altas velocidades sobre el circuito. Siguiendo esta filosofía la forma de los circuitos es mucho más sencilla, reduciéndose en algunos casos a un simple óvalo.

Otra diferencia significativa es que los robots no están obligados a mantenerse sobre una línea, de hecho los circuitos de velocistas están compuestos por dos líneas. Los robots pueden elegir por cual de éstas lineas ir, o incluso pueden ir entre ambas. El robot quedará descalificado si sale completamente del espacio delimitado por dichas líneas, aunque en algunas versiones de la prueba existen otras dos líneas (se pueden ver en la figura 2.3), generalmente de otro color, alrededor de las del circuito, de manera que si un robot pisa alguna de esas líneas quedará descalificado.

Generalmente las eliminatorias de robots velocistas se corren uno contra uno. Lo habitual es que primero se realice una clasificatoria por tiempos para, a continuación, ir eliminando participantes por eliminatorias en las que un robot se medirá a otro. Estas carreras se realizan haciendo salir a cada robot desde puntos opuestos del circuito, terminando la misma cuando uno de ellos

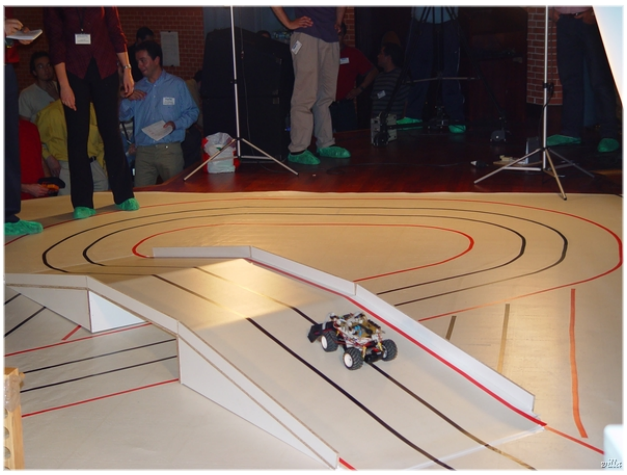

Figura 2.3: Robot velocista en un circuito con paso elevado

alcance al otro o tras un número determinado de vueltas, ganando el robot que haya sido capaz de acercarse más a su contrincante.

Como ya se ha comentado, los circuitos de velocistas son bastante sencillos, pero tratando de que tengan curvas en ambas direcciones se pueden encontrar circuitos con forma de "ocho" en los que el cruce se realiza mediante un paso elevado. Esto introduce nuevas dificultades como son las diferentes condiciones de iluminación debajo del puente y el puente en sí, ya que los desniveles y cambios de rasante pueden obligar a replantear el sistema motriz y la situación de los sensores.

2.3.3. Sumo

Una de las pruebas que más participantes y espectadores congrega es la competición de sumo. Es una prueba muy espectacular en la que dos robots, al igual que en las pruebas de sumo real, se enfrentan sobre un ring circular tratando de conseguir expulsar a su oponente del mismo.

Para la prueba de sumo es muy importante la estrategia, pero también la fuerza bruta a la hora de empujar al contrario, por eso tanto las dimensiones como el peso están siempre limitados. Generalmente no pueden pesar más de 3 kg, y el tamaño máximo suele ser de 20x20 cm con todos los mecanismos plegados, aunque durante la prueba tiene la posibilidad de desplegar partes móviles siempre que estas no se desprendan del robot.

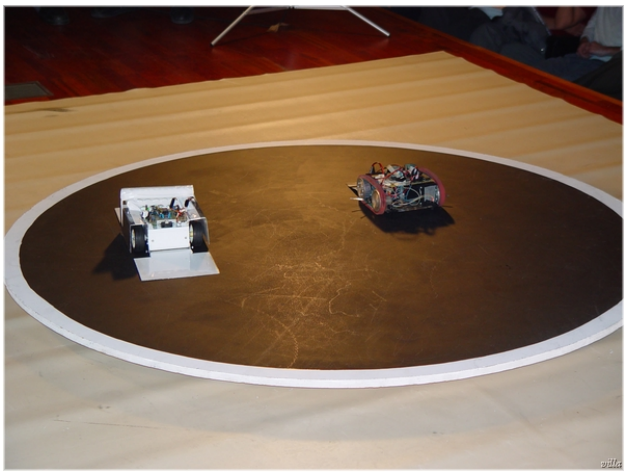

Figura 2.4: Combate de sumo

Es una prueba que varía poco en las diferentes competiciones, pero sí que existe una pequeña modificación de la misma que es esencialmente la misma pero con robots más pequeños. Se conoce con el nombre de "minisumo" y lo único que cambian son las dimensiones del robot, que se reducen hasta los 10x10 cm y el peso que se queda en tan solo 500 g. También existe la categoría de "micro-sumo" en la que los robots son todavía más pequeños.

2.3.4. Laberinto

Bajo este nombre se engloban una serie de pruebas en las que el objetivo final es que el robot sea capaz de salir de un laberinto. El robot siempre está limitado en tamaño, tanto las dimensiones en planta como en altura. Generalmente el laberinto está formado por paredes ortogonales que serán más altas que la altura máxima del robot.

El circuito puede ser conocido o desconocido. Cuando el circuito es desconocido es posible que este tenga una entrada y una salida, o que solo tenga una salida que ha de encontrar partiendo de un punto del interior. Si el circuito es conocido lo más habitual es que no esté determinado el punto de origen del robot, con lo que el robot primero tiene que identificar la posición en la que se encuentra para despues llegar hasta la salida lo más rápido posible. En algunos casos puede haber paredes móviles que cambien la disposición del circuito.

Figura 2.5: Robot en una prueba de Laberinto

Además la prueba se puede complicar más añadiendo una tarea a realizar en algún punto del laberinto como encestar una pelota, apagar una vela o pinchar globos. La realización de estas tareas supone una bonificación.

2.3.5. Otras pruebas

Se pueden encontrar todo tipo de pruebas a las que se tienen que enfrentar pequeños robots, pero es imposible describirlas todas y eso no es el objetivo de este proyecto. Pero es interesante destacar que existen pruebas muy curiosas como las de robots limpiadores, que tratan de limpiar un determinado recinto; los robots bomberos que van apagando fuegos (velas en realidad) en lo que representa la planta de una casa; robots que se dedican a la caza del zorro, concretamente tienen que atrapar a otro robot antes que sus contrincantes; incluso existen robots toreros en la prueba conocida como Robotaurus en Cybertech, en la que un robot trata de torear un toro que también es un robot.

2.4. Tecnología empleada

Las posibilidades son infinitas cuando se trata de diseñar un robot, pero cuando se habla de microrrobots de competición hay una serie de elementos que son muy usados en ellos. Al fin y al cabo todos tienen que realizar una

serie de tareas similares.

También hay que tener en cuenta que uno de los factores importantes en la realización de estos robots es el precio. Por tanto los elementos más habituales suelen ser componentes de bajo coste.

2.4.1. Microcontrolador

En un robot tendrá que haber un elemento que se encargue de interpretar las señales de los sensores, determinar qué tiene que hacer el robot y mandar las órdenes a los actuadores.

Este elemento va a ser el microcontrolador. Un microcontrolador es un dispositivo electrónico capaz de llevar a cabo procesos lógicos. Estos procesos son programados por el usuario en lenguaje ensamblador, o en C si se dispone de un compilador de C para ese microcontrolador, y grabados en la memoria de éste.

A diferencia de un microprocesador, el microcontrolador incluye dentro del circuito integrado los elementos necesarios para hacer funcionar un sistema. Es decir, no solo incluye una CPU *(Central Processing Unit o Unidad Central de Proceso)* sino también memoria, que a grandes rasgos será memoria no volátil para almacenar el programa y memoria volátil para los datos, puertos de entrada/salida y algunos periféricos.

Arquitectura

La mayoría de los computadores se construyen siguiendo dos arquitecturas diferentes, la arquitectura Von Neumann y la arquitectura Harvard.

La arquitectura **Von Neumann** se caracteriza por tener una memoria única donde se encuentran tanto los datos como la memoria de programa. El tamaño de la unidad de memoria queda fijado por el ancho del bus, así un microcontrolador de 8 bits manejará unidades, tanto de datos como de programa, de 8 bits. Si tiene que acceder a un dato o instrucción de más de un byte tendrá que realizar varios accesos a memoria.

De entre los microcontroladores de 8 bits que se usan habitualmente en robots hay que destacar el 68HC11 de Motorola, cuyo uso era masivo hasta hace algunos años y que usa una arquitectura Von Neumann.

Aunque inicialmente todos los microcontroladores adoptaron la arquitectura clásica de Von Neumann, en el momento presente se impone la arqui-

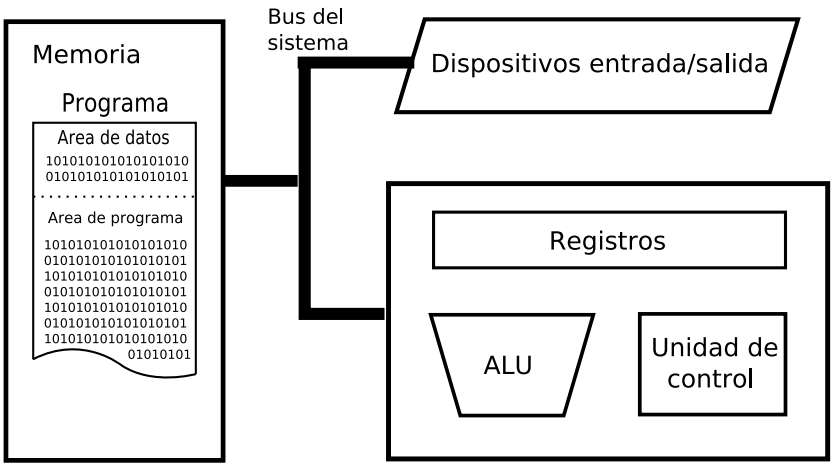

Figura 2.6: Esquema de la arquitectura Von Neumann

tectura **Harvard**. La arquitectura Harvard dispone de dos memorias independientes, una que contiene sólo instrucciones y otra, sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en las dos memorias.

Como ya se ha comentado, actualmente la mayoría de microcontroladores emplean arquitectura Harvard. Muchos fabricantes ponen a la venta microcontroladores de 8 bits, pero entre los aficionados a los microrrobots son muy conocidos los microcontroladores PIC de Microchip y los AVR de Atmel. Su éxito se basa en su bajo coste, la existencia de entornos de desarrollo y herramientas gratuitas y la disponibilidad de muchos de éstos modelos en encapsulados DIP *(Dual In-line Package)* muy sencillos de utilizar en prototipos y diseños "caseros".

Ordenadores empotrados

Si se requiere una capacidad de procesamiento más elevada, o si es imprescindible el uso de un sistema operativo, se puede recurrir a usar en el robot un ordenador empotrado. En la industria se usan habitualmente computadores de este estilo, pero su uso en microrrobots es bastante reducido porque su consumo energético es algo elevado y su precio también.

Dentro de esta categoría hay que destacar el PC/104. PC/104 o PC104 es un estándar que define el formato de la placa *(form factor)* y el bus del sistema. Se pueden encontrar en el mercado infinidad de sistemas que cumplen

dicho estándar o alguno parecido (como EBS o EPIC).

2.4.2. Sensores

El tipo de sensores necesarios en un microrrobot depende totalmente de la prueba para la que está pensado, aunque hay algunos que es imprescindible conocer si se piensa construir un robot de los que estamos tratando.

CNY70 El sensor CNY70 es un sensor de infrarrojos de corta distancia que funciona por reflexión. Está compuesto por un LED y un fototransistor. El LED emite luz infrarroja, ésta es reflejada contra un objeto, y por el fototransistor circulará una intensidad proporcional a la cantidad de luz reflejada.

Se usa principalmente para detectar si el suelo debajo del sensor es blanco o negro. De esta manera se localiza una línea negra sobre un fondo blanco, se localizan los bordes de los recintos o se detecta el paso delante del sensor de una franja blanca o negra para usarlo como *encoder*.

Sensores de contacto Estos sensores son simples interruptores que se usan para detectar una colisión física. La salida tiene una resistencia de *pull-up* o *pull-down*, el interruptor cierra un circuito eléctrico y cambia la tensión de la salida. Se suelen usar añadiéndoles un circuito antirrebotes. Pueden tener forma de bigote que activarán una señal cuando éste toque con algo.

Sensores de ultrasonidos Son sensores que permiten medir la distancia a los objetos que tiene delante. El funcionamiento consiste en medir el tiempo que tarda en volver un ultrasonido emitido por el sensor y rebotado en el objeto.

Pueden medir distancias a objetos entre unos centímetros y varios metros. El ángulo en el que detecta objetos es bastante amplio aunque son muy dependientes del objeto a detectar y es muy importante el material del objeto y sobre todo la orientación. Hay que tener en cuenta que puede haber varios ecos si hay paredes en varios lados.

Para realizar una medida es necesario indicarle al sensor que comience a medir mediante un pulso en una patilla, a lo que el sensor responderá con otro pulso cuya duración es proporcional a la distancia medida. Ésto suele llevar bastante tiempo. También hay algunos modelos

que se pueden controlar mediante un bus I2C, que miden la distancia continuamente y ésta se lee a través del bus.

GP2Dxx Los sensores de la familia GP2Dxx de Sharp son sensores medidores de distancia, pero a diferencia de los de ultrasonidos funcionan por infrarrojos. Este sensor mide constantemente la distancia al objeto que tenga delante y su salida es una señal analógica cuya tensión es función de la distancia según una curva conocida.

Son más sencillos de manejar ya que se leen mediante el convertidor analógico/digital que muchos microcontroladores llevan integrado, y son algo más rápidos que los de ultrasonidos, pero su rango es menor y su exactitud disminuye mucho con la distancia.

Los más comunes de la familia son los sensores GP2D12 y GP2D120, que miden distancias en los rangos de 10 a 80 cm y de 4 a 30 cm respectivamente. Su salida es analógica como se ha descrito anteriormente, aunque hay algunos miembros de la familia cuya salida es digital.

Brújula electrónica Es un sensor de campos magnéticos suficientemente sensible para captar el campo magnético de la tierra. Es importante incluir una brújula electrónica en cualquier robot que precise de un sistema de navegación, es habitual en los robots de laberintos y en cualquier prueba en la que sea importante conocer la orientación.

Pueden encontrarse brújulas con interfaz a través del bus I2C y también brújulas en las que la salida es un pulso cuya duración indica la posición de la misma.

Encoders Un *encoder* es un dispositivo que sirve para medir el giro de un eje. Si este eje está acoplado al sistema motriz del robot sirve para medir el avance del mismo y también su velocidad.

Existen encoders absolutos y relativos. Los encoders absolutos indican la posición exacta del eje, mientras que los incrementales solo indican el avance de una posición a otra, con lo que hay que llevar la cuenta de cuantos pasos han transcurrido desde una posición de referencia.

Los encoders comerciales son muy caros, así que se suelen construir artesanalmente usando sensores de infrarrojos y discos perforados o con secciones de colores acoplados a algún eje de la transmisión. Son muy útiles tanto para implementar un control de velocidad, como simplemente para medir la distancia recorrida.

Cámaras, el uso de cámaras de vídeo está algo limitado en los microrrobots por el precio y por la capacidad de procesamiento que requieren. Sin embargo, hay cámaras que ya incorporan una placa con cierto nivel de procesamiento que permiten que su uso sea algo menos complicado. Aún así no resulta sencillo utilizarlas sin un ordenador con cierta capacidad de procesamiento.

2.4.3. Actuadores

Salvo en pruebas muy específicas, los actuadores que llevan este tipo de robots se reducen a dos: motores y servos.

Servos

Un servo es un motor con una reducción bastante grande y control de posición, es decir, con un servo podemos hacer que un eje se coloque en una posición exacta dentro de su rango de acción (que es, típicamente, ligeramente superior a 180 °) ejerciendo un alto par.

Además es un elemento muy barato que se controla de manera muy sencilla, por que lo que es la opción más usada siempre que se quiere posicionar una parte móvil de robot en una posición concreta. Ejemplos de su uso pueden ser para colocar la orientación de las ruedas directrices o para mover un brazo que tenga una función determinada.

Cabe la posibilidad de eliminarles el tope mecánico y la realimentación del sistema de control para conseguir que su movimiento sea continuo. De esta manera se pueden usar como si de un motor con una reductora incorporada se tratara.

Motores

Se utilizan para el desplazamiento de los robots. Se emplean dos tipos de motores, motores de corriente continua o motores paso a paso.

Los motores de corriente continua son los más utilizados por ser muy baratos, sencillos de usar y por poderse encontrar en una amplia gama de tensiones de entrada y potencia. La velocidad de giro se controla con una señal PWM *(ulse-width modulation o modulación por anchura de pulso)* introducida en una etapa de potencia que puede reducirse a un puente en H. Es muy habitual el uso de optoacopladores para aislar eléctricamente el circuito de control del de potencia y reducir posibles interferencias.

Los motores paso a paso también se usan en microrrobots. Su manejo es mucho más complicado y se necesitará cierta electrónica dedicada exclusivamente a ello, pero se puede especificar con mucha exactitud como va a ser su movimiento e incluso bloquearlos en ciertas posiciones.

Capítulo 3

Hyperion, un velocista muy particular

En este apartado se describe como es el robot físicamente, es decir, cómo está construido, qué es capaz de hacer y todo lo que lleva incorporado para eso.

3.1. Introducción

En este proyecto se describe el proceso de creación de un robot velocista. Primero se va a realizar el mismo empleando tecnología software convencional y más tarde se le añadirá todo lo necesario para poder realizar sobre él un control de alto nivel.

El nombre del robot es Hyperion, cuyo nombre proviene del de uno de los 12 titanes de la mitología griega.

Como ya se ha expresado en el capítulo 2 la prueba de robots velocistas es una prueba en la que los robots tienen que recorrer lo más rápido posible un circuito marcado por una línea negra sobre un fondo blanco. Dicho circuito no tiene curvas muy cerradas ni bifurcaciones de ningún tipo.

Lo que sí se ha de tener en cuenta es la posibilidad de que haya un paso elevado en el circuito, y como consecuencia también hay que tener en cuenta la diferente iluminación debajo de dicho paso.

Además de la parte mecánica hay que tener en cuenta el objetivo del proyecto, se ha de dotar al microrrobot de lo necesario para controlar el robot

en bajo nivel y para poder realizar sobre el mismo un control de alto nivel mediante software distribuido basado en CORBA. Esto incluye desde lo necesario para controlar el hardware hasta la arquitectura sobre la que se puedan ejecutar aplicaciones CORBA sin olvidarnos de una conexión inalámbrica.

3.2. Estructura física

La principal característica del robot ha de ser su elevada velocidad. Para este objetivo la estructura física es lo más importante.

Por otro lado este es el aspecto en el que hay que mirar las restricciones de tamaño que ha de cumplir el robot. Éstos límites, en varias de las competiciones en las que existe esta prueba en España, son 30 cm de largo por 20 cm de ancho, mientras que en alguna de éstas competiciones existe también una limitación de la altura del robot de 15 cm. Para que el robot sea más genérico vamos a cumplir todas estas restricciones.

3.2.1. Chasis

A la hora de decidir cómo va a ser el chasis se plantean dos elecciones importantes. Una es la distribución mecánica en sí y otra es escoger los componentes o un chasis ya hecho.

Una distribución mecánica con una rueda que sea al mismo tiempo motriz y directriz es una distribución muy útil para robots rastreadores, porque permite giros muy elevados y es sencilla de controlar, pero queda desechada porque es complicado conseguir que un robot con esta estructura sea rápido. Además no necesitamos tanta capacidad de giro.

Las dos mejores distribuciones para el robot velocista son:

- Dos ruedas motrices independientes con una rueda loca. Con esta distribución el giro se produce mediante la diferencia de velocidad entre una rueda y la otra. Es posible mediante esta distribución conseguir un robot rápido, y el mecanismo de giro es sencillo en teoría, aunque habría que comprobar si es posible realizar pequeños giros a elevada velocidad.

  Si se elige esta disposición no es fácil usar un chasis ya fabricado, aunque resultaría bastante sencillo construir uno con dos motores y dos

reductoras idénticas que desarrollaran la velocidad que se fije como objetivo para el robot.

- Dos ruedas directrices. Es la disposición que llevan los coches de calle. La tracción puede estar en las ruedas del eje delantero o en las del trasero indistintamente, aunque es importante que haya un diferencial en el eje de tracción para que no se obligue a patinar a ninguna rueda.

  La dirección es muy sencilla de controlar porque se puede realizar mediante un servo que mueve una cremallera que a su vez hace girar las ruedas. Además no hay restricciones particulares para el motor salvo la colocación de la reducción mecánica y el diferencial.

Finalmente se decide optar por la última disposición ya que en ella se pueden realizar sendos controles de velocidad y de posición de manera independiente, mientras en la primera disposición ésto no es posible.

Con la intención de hacer un velocista rápido se decide comprar un chasis de un vehículo de radio control. Ya que incorporan tanto la transmisión con el diferencial, como un sistema de suspensión que puede ayudar mucho en las curvas. Tratando de acercarnos a las dimensiones máximas buscamos un coche de radio control de escala 1:10 que cumpla con los límites de 20x30 cm.

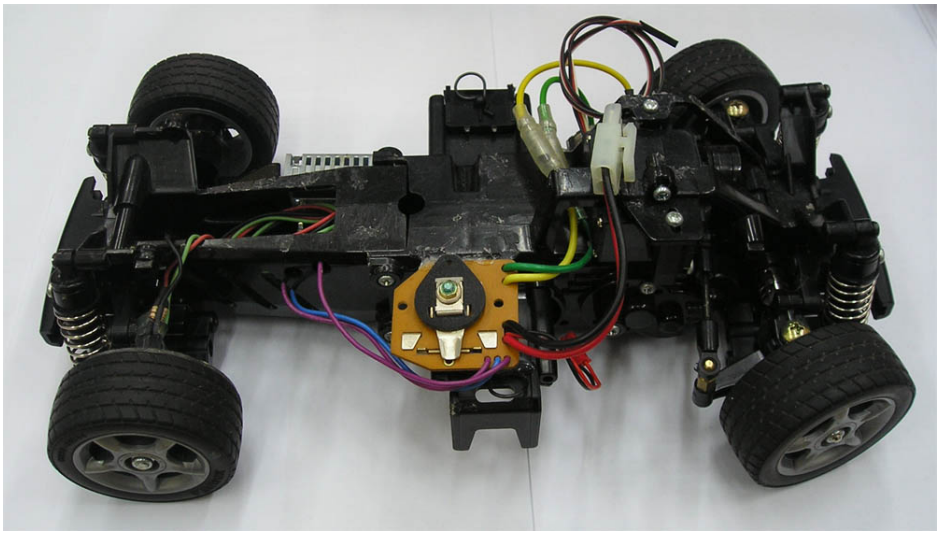

Figura 3.1: Chasis empleado para el robot

Finalmente se compra un coche de radio control que cumple esos requisitos. Se descubre que su radio mínimo de giro es algo superior que el que se

puede encontrar en una prueba de velocistas, pero éste se reduce modificando ligeramente la estructura de la dirección.

El chasis incluye toda la transmisión, incluido el motor. La velocidad que puede desarrollar es muy superior a la necesaria para la prueba. Esto implica que para ir una velocidad apta para la competición, la tensión a aplicar al motor ha de ser muy pequeña, lo que supone un par muy pequeño y hace que tarde mucho en alcanzar la velocidad de referencia. Pero se puede solucionar añadiendo un control de velocidad.

El motor que integra el chasis es un motor de competición, puede desarrollar una gran potencia pero también tiene un consumo bastante elevado, lo que tiene dos consecuencias:

- La alimentación del motor ha de ser independiente de la alimentación de la electrónica, para aumentar la duración de las baterías y para independizar los dos circuitos y evitar interferencias en la alimentación de la electrónica.

- Hay que usar una etapa de potencia que pueda proporcionar la intensidad que consume el motor. No se puede usar un integrado convencional como es el L298 que incluye dos puentes en "H", ya que incluso usando ambos canales en paralelo no podríamos alimentar el motor a más de 3 Amperios. Se decide usar una batería de 7,2 v para el motor que irá alojada en la parte baja del chasis, en un alojamiento específico para ella y otra batería para la electrónica cuya tensión y capacidad dependerá de la misma.

  La siguiente posibilidad que se plantea es montar un puente en "H" con componentes discretos usando transistores MOSFET de baja resistencia en conducción. Se monta una etapa de potencia usando 2 transitores MOSFET de canal n y 2 de canal p cuyas resistencias en conducción son de 0,018 y 0,06 Ω respectivamente y que pueden suministrar una intensidad de 31 A y se prueba en un uso intensivo del motor comprobando que el calentamiento de los transistores es muy bajo.

  Pero ésta etapa de potencia ocupa mucho espacio, se ha necesitado una placa de 66x121 mm para realizarla y como el robot no tiene la necesidad de invertir el sentido de la marcha se decide prescindir del puente en "H" y reducir la etapa de potencia a un único transistor SUP70N03-09 cuya resistencia en conducción es de 0,015 Ω que se dispara directamente desde el microcontrolador mediante un PWM y a

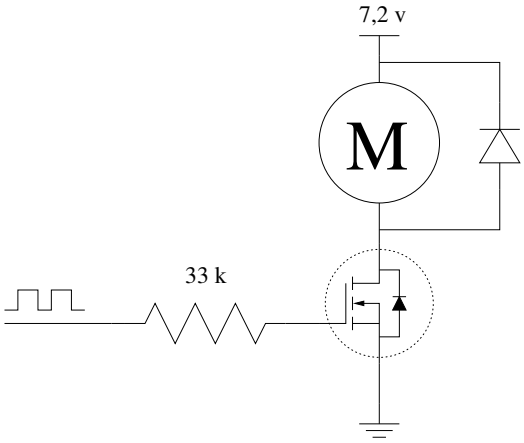

Figura 3.2: Esquema de la etapa de potencia

un diodo necesario para que siga conduciendo la intensidad residual cuando el transistor deja de conducir. De esta forma se reduce mucho su tamaño y se puede incluir en una placa que contenga otros circuitos del robot.

Dirección

El chasis está preparado para acoplarle un servo de dimensiones estándar que se encarga de controlar la dirección. Un servo se controla mediante pulsos de anchura variable en los que la anchura del pulso determina la posición a la que se mueve el servo. Será el microcontrolador el que se encargue directamente de generar esos pulsos con lo que, lo único que hay que añadir para que funcione es la alimentación del servo. El servo se alimenta a 6 v y, teniendo en cuenta que la batería para el motor es de 7,2 v nominales (que en la práctica puede llegar hasta 8 v) una manera muy sencilla de alimentar el servo es colocar un regulador lineal 7806 que se encarga de proporcionar exactamente 6 v siempre que la tensión a su entrada está por encima de unos 7 v.

El 7806 sólo necesita dos condensadores (uno entre la entrada y tierra y otro entre la salida y tierra) para funcionar, con lo que se colocará en la misma placa que la etapa de potencia del motor.

3.3. Sensorización

Los sensores van a ser el único medio a través del cual el robot reciba información de su entorno. Primero tenemos que determinar qué información necesita obtener el robot sobre su entorno o sobre sí mismo.

3.3.1. Sensores exteroceptivos

Los sensores exteroceptivos son los que proporcionan información sobre el entorno que rodea al robot. Lo único que necesita conocer el robot sobre su entorno es la situación de la línea negra que tiene que seguir. Para esto, la primera solución que se plantea es emplear sensores de infrarrojos CNY70. Se pueden leer tanto de manera digital (usando un Schmitt Trigger) como de manera analógica a través de un convertidor analógico a digital. Ambas opciones son muy rápidas, permitiendo la última, además, calibrar el sensor mediante software en la misma pista de competición.

Cuando se usan sensores de infrarrojos como los CNY70, es decir, que pueden leer el nivel de oscuridad del suelo en puntos concretos, es importante el número y la distribución de éstos.

Éstos fueron los sensores empleados en una primera versión del robot velocista. Tenía seis sensores CNY70 colocados delante del robot formando una línea recta perpendicular al avance del robot. Dos de ellos muy juntos tratando de permanecer sobre la línea cuando el robot se encuentre centrado y dos más a cada lado más alejados para situarse cuando se ha separado más de la referencia.

Dicha distribución de sensores tenía un comportamiento aceptable pero el objetivo de hacer un robot muy rápido exigía conocer la situación de la línea con más antelación. Con esta mentalidad se plantea la posibilidad de usar una cámara, pero esta posibilidad se desecha por lo complicado de su uso con un microcontrolador para realizar una tarea relativamente sencilla como es localizar una línea negra.

La siguiente opción resulta más atractiva, se trata de usar un sensor formado por una matriz lineal de sensores *(o Linear Sensor Array)*. Los principales fabricantes de optoelectrónica cuentan en sus catálogos con productos como estos. Básicamente se trata de juntar, en un único integrado, un cierto número de píxeles fotosensibles que integran la luz que incide sobre ellos en un determinado tiempo para generar una salida digital o analógica que

represente la exposición de cada píxel. Constituye una cámara lineal que es suficientemente simple como para ser procesada por un microcontrolador.

Un sensor de éstos, enfocado hacia la línea a cierta dista distancia delante del robot nos puede permitir conocer el nivel de oscuridad de muchos puntos en dicha línea con suficiente antelación, de manera que nos podría permitir más velocidad.

Este tipo de sensores se pueden encontrar con diferente número de píxeles y tamaño de los mismos. En cuanto al número de píxeles hay sensores de 64, 128, ... y hasta más de 1500, pero para localizar la línea será suficiente uno con 64 puntos. El fabricante de referencia en este tipo de sensores es TAOS Inc así que la primera opción era comprar su sensor TSL201-R, pero no se encuentra en las tiendas y a través de su distribuidor en España no se pueden conseguir unidades sueltas.

iC-LA

Tras buscar otros modelos de diferentes fabricantes, el sensor comprado finalmente es iC-LA de iC-Haus[1]. Se trata de 64 fotodiodos colocados en línea cada uno con una superficie sensible de 183 x 200 µm y una distancia de uno a otro de 200 µm.

Su manejo tiene cierto nivel de complejidad. Para que funcione, además de alimentarlo hay que pasarle una señal de reloj que marcará su funcionamiento. La lectura comienza mediante un pulso en una de las patillas del sensor, a partir de ahí éste integrará la luz que reciba, y transcurrido un determinado número de ciclos de reloj el sensor pone en la salida secuencialmente una tensión proporcional a la luz que ha incidido en cada píxel. Lo más complicado es sincronizar el convertidor analógico digital para que realice las 64 conversiones a la misma velocidad que las envía el sensor.

Otro problema que plantea dicho sensor es su encapsulado. El sensor solo está disponible en encapsulado BGA *(Ball Grid Array)*. Es un encapsulado pensado para una alta densidad de pines y soldado en horno en el que los pines son pequeñas semiesferas de estaño colocadas debajo del chip, a las que no se puede acceder mediante un soldador.

Afortunadamente el encapsulado que usa el sensor solo tiene 10 pines distribuidos en dos filas cerca de los bordes del chip, lo que permite soldarlo a una placa de circuito impreso, diseñada específicamente para el sensor, en

[1]Más información en `www.ichaus.com`

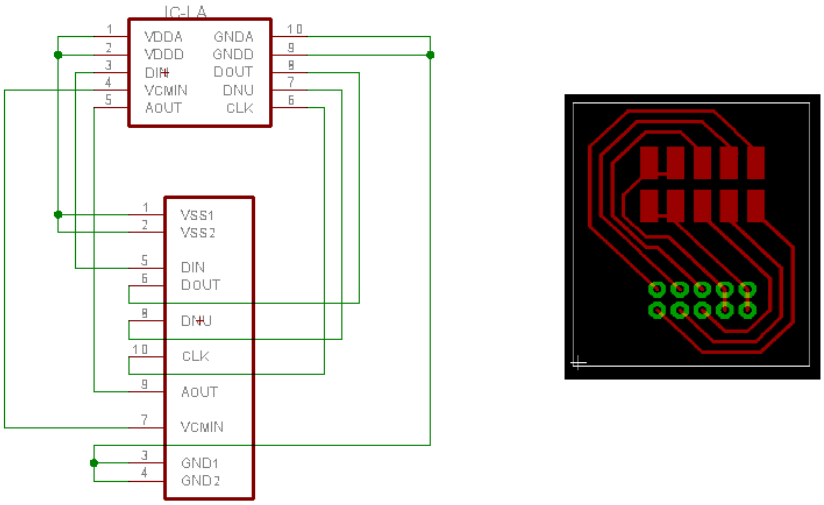

Figura 3.3: Placa para el conexionado del sensor lineal

la que hay 10 pads (o zonas de soldadura) de montaje superficial suficientemente grandes para llegar a la semiesfera de cada conector debajo del chip y sobresalir del mismo para poderlo calentar con el soldador. En la misma placa colocamos también un conector de cable plano de 10 hilos estándar que se usa para conectar el sensor.

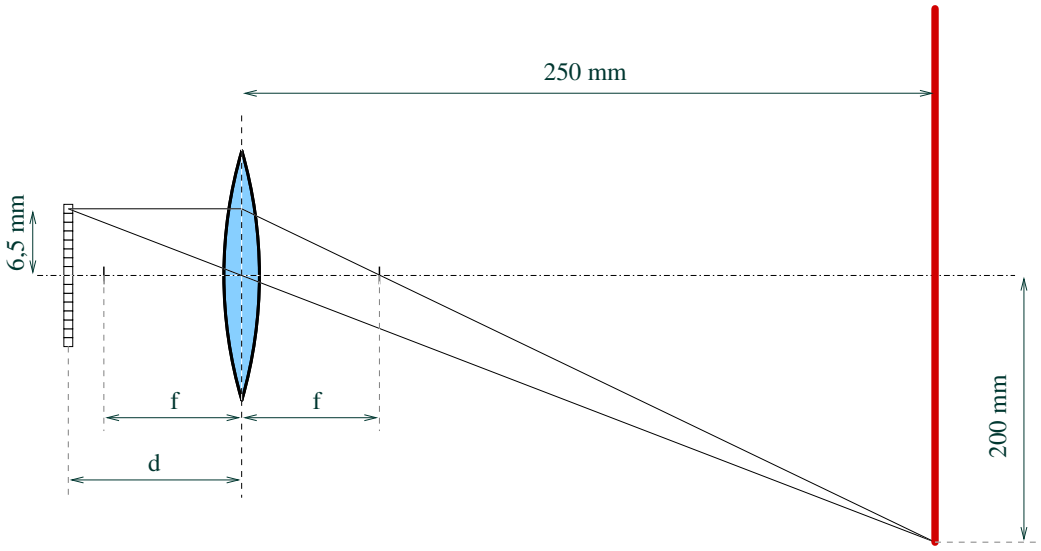

Figura 3.4: Diagrama para el cálculo de la lente del sensor

Este sensor irá situado lo más alto posible, es decir, a casi 15 cm de altura que es la altura máxima del coche. Su misión es conocer la posición de la línea con cierta antelación aunque dicha antelación no podrá ser demasiado grande porque el sensor no está colocado muy arriba y la superficie se vería muy deformada. Tratamos de que el sensor esté enfocado a una distancia de 20 cm delante del robot (calculando la longitud de la hipotenusa el sensor quedará a 25 cm de la línea que estará enfocada), y de que abarque una línea de unos 40 cm.

Observando la figura 3.4 es fácil deducir:

$$\frac{6{,}5}{d} = \frac{200}{250}$$

$$d = \frac{32{,}5}{4} = 8{,}125\ mm$$

y también:

$$\frac{6{,}5}{200} = \frac{d - f}{f}$$

$$6{,}5 * f = 1625 - 200 * f$$

$$f = \frac{1625}{206{,}5} = 7{,}87\ mm$$

Como no es sencillo encontrar una lente de una distancia focal tan pequeña se hicieron varias pruebas superponiendo dos lentes, pero ésto suponía varios problemas constructivos al ser muy difícil colocarlas exactamente a la distancia requerida.

Ante estos problemas se prueba una solución más sencilla que es sustituir la lente por un *pinhole*, un pequeño orificio en una cartulina colocada a la misma distancia que la lente. Esto implica que llega una cantidad de luz más pequeña al sensor y una imagen algo desenfocada, pero eso no es demasiado. Tras hacer varias pruebas se concluye que la imagen que se obtiene de la línea tiene suficiente luz y está suficientemente enfocada para localizar la línea con exactitud, por tanto se sustituye finalmente la lente por una carcasa alrededor del sensor con dicha cartulina.

Como este sensor mide la luz visible que recibe de la superficie a la que está enfocado, es muy sensible a las condiciones de iluminación. Para reducir esta dependencia colocamos junto al sensor una fila de LEDs rojos de alta luminosidad iluminando la zona de la que lee el sensor. La razón de que sean LEDs rojos es porque su eficiencia es muy alta y porque, atendiendo a las especificaciones del sensor iC-LA, la máxima sensibilidad se encuentra en la luz roja.

3.3.2. Sensores propioceptivos

Los sensores propioceptivos son aquellos que se usan para captar información del estado del propio robot. En el robot Hyperion las variables internas más importantes son dos: la posición de la dirección y la velocidad real.

La orientación de las ruedas directrices se establece en el robot mediante un servo y, ya que este movimiento no tiene nada que se le pueda oponer y que el servo tiene su propio control de posición, podemos tomar la referencia que le damos al servo como medida exacta de la posición de éste.

Con la velocidad no ocurre lo mismo. La velocidad del motor se controla mediante un tren de pulsos, en teoría y sin tener en cuenta rozamientos y demás perturbaciones la velocidad del robot debería ser proporcional al valor efectivo de dicha señal, pero esto no es así, ya que el robot tiene rozamientos, una inercia muy grande que hace que el sistema sea muy lento y podría haber perturbaciones como las cuestas que hay en algunos circuitos.

Encoder

Necesitamos medir el avance real del motor, es decir, necesitamos un encoder. Los encoders comerciales tienen un precio demasiado elevado y sería bastante complicado acoplarle uno al chasis así que optamos por construir uno artesanal.

Al encontrarse la reducción mecánica escondida debajo de una carcasa el único punto de la transmisión en el que es sencillo medir el avance es en las propias ruedas. Usamos una de las ruedas traseras que, al no ser directrices, están más libres y hay espacio para colocar el sensor.

Lo más sencillo de acoplar es un disco con franjas blancas y negras que serán detectadas por un sensor de infrarrojos de reflexión. El número de franjas determinará la resolución del encoder, por lo que interesa un número elevado de ellas, pero cuanto mayor sea el número de ellas más difícil resulta

detectar el cambio de color.

Por su bajo coste y lo probado que está, se va a usar un sensor de infrarrojos CNY70. Cómo mejor funciona este sensor es muy cerca de la superficie de reflexión así que lo colocamos a apenas 1 mm del disco con las franjas.

Tras realizar varios cálculos según las especificaciones del sensor se obtiene que, si las franjas blancas y negras que ve el sensor lineal son de 2,5 mm la reflexión en el centro de la franja blanca será de aproximadamente un 65 % de la reflexión máxima, mientras que en el centro de la franja negrá será de apoximadamente un 35 %. Si se monta un circuito típico para el CNY70 en el que sobre un blanco absoluto el sensor devuelva 0 v y 5 v sobre un negro absoluto, con el disco de franjas descrito, a la salida del sensor se obtendría una senoide cuyos picos estarían en 1,75 v y 3,25 v.

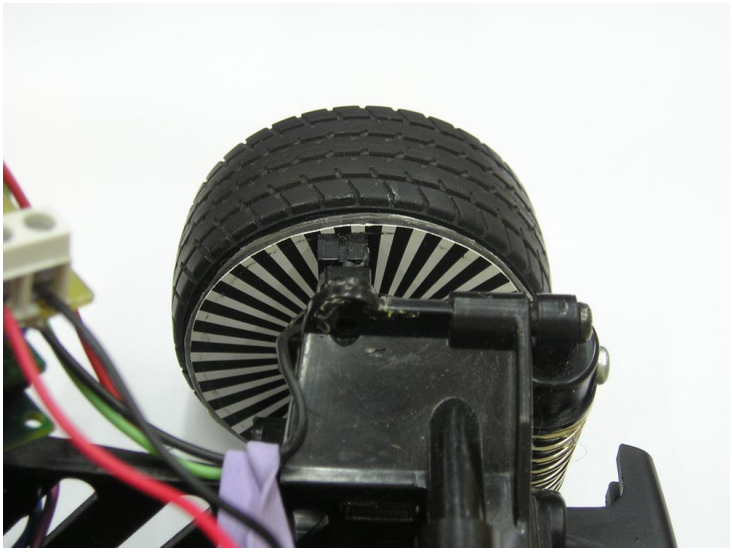

Figura 3.5: Encoder artesanal montado en una de las ruedas

Siguiendo estos cálculos construimos un disco con 36 franjas blancas y 36 franjas negras que permitirán detectar un cambio de nivel por cada 2,5 mm de avance del robot.

Para que el micocontrolador pueda contar cada uno de estos pasos por las franjas blancas y negras hay que convertir esta señal en un tren de pulsos, haciendo que por debajo de cierto valor de la senoide la señal que le llega al micro sea un cero lógico y por encima del mismo sea un uno. Ésto se consigue mediante un comparador y se puede realizar con un amplificador operacional

sin realimentación, pero un montaje así de sencillo podría provocar rebotes en el paso por la tensión umbral, lo que implica que hay que montar un comparador con histéresis. De esta forma el umbral con el que pasa de 0 a 1 es algo superior que el que umbral con el que cambia de 1 a 0 y no se producen rebotes en la señal.

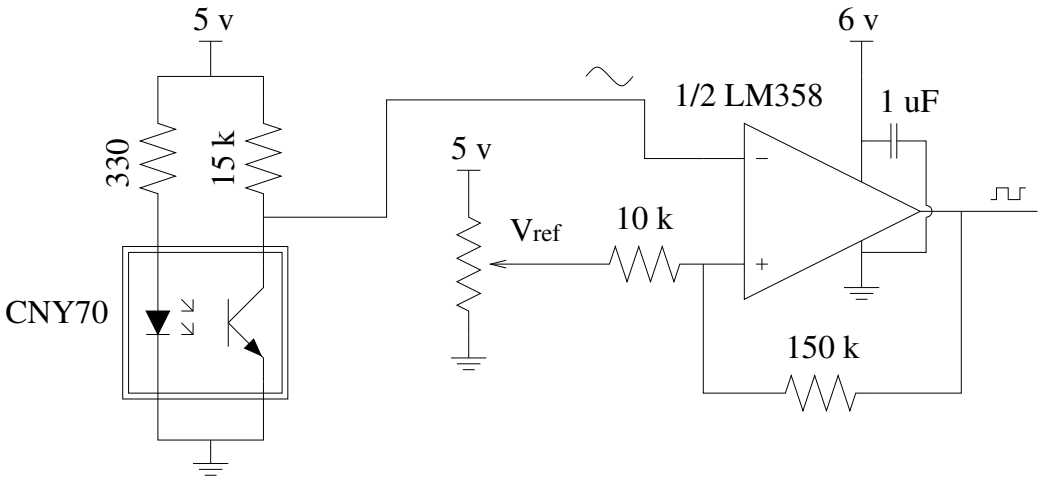

Figura 3.6: Esquema eléctrico del sensor CNY70 y el comparador del encoder

Se ha usado el integrado LM358 para realizar el comparador. El LM358 incluye 2 amplificadores operacionales de polarización simple, es decir, que no necesitan alimentación simétrica $\pm V_{cc}$ sino que se alimentan entre una cierta V_{cc} y tierra. Los operacionales de este integrado no son *rail-to-rail (o carril a carril)*, por tanto en saturación la salida no llega a los niveles de la entrada. En concreto la salida en estos operacionales puede bajar hasta 0 v y subir hasta 1,5 v por debajo de la tensión de alimentación. Si ésta es de 5 v la salida del comparador será un tren de pulsos entre 0 y 3,5 v que se puede leer de manera digital en cualquier entrada digital del microcontrolador

Para el cálculo de las resistencias basta con calcular las ecuaciones que determinan los dos umbrales del comparador y tener en cuenta la histéresis que tenemos. En este caso se ha calculado para que la franja de histéresis sea de 0,2 v. En cuanto a la tensión de referencia, ésta tendrá que estar centrada entre los valores extremos de la senoidal, y para que se pueda centrar exactamente con la señal real lo que hacemos es fijarla mediante un potenciómetro de 4,7 kΩ (suficientemente pequeño para que la tensión varíe poco con la intensidad que circula por la resistencia de la realimentación).

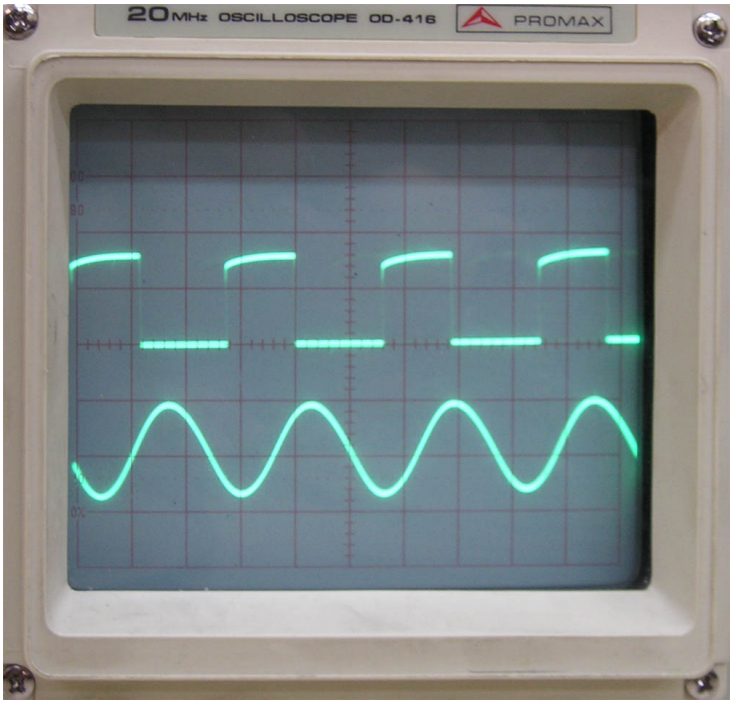

Figura 3.7: Salida del encoder y ésta tras pasarla por el comparador

3.4. Inteligencia

Como ya se ha comentado anteriormente no solo es necesario incorporar al robot de un procesador capaz de decidir las acciones a tomar para realizar la prueba de velocista, sino que también ha de ser posible realizar un control de alto nivel sobre el mismo mediante software distribuido basado en CORBA.

La plataforma que ha de gobernar el robot tiene 3 exigencias principales. Ha de ser capaz de ejecutar un ORB así como las aplicaciones específicas de control del robot, ha de ser capaz de interactuar con el hardware del robot de una manera sencilla y necesita poder comunicarse de manera inalámbrica con otro ordenador.

Se plantea la posibilidad de usar un microcontrolador, porque no solo incluye una CPU *(Central Processing Unit o Unidad Central de Proceso)* que ejecutará las instrucciones que controlen el comportamiento del robot, sino que además lleva integrados varios puertos de entrada/salida y ciertos dispositivos muy útiles para para la interacción con el hardware del robot como contadores, temporizadores, convertidor analógico a digital, etc.

Pero esta elección tiene un grave inconveniente que es la dificultad de instalar en él un sistema operativo sobre el que poder instalar el ORB. Otro problema sería conseguir una comunicación inalámbrica con otro ordenador. Se podría hacer un puente inalámbrico mediante Bluetooth en una conexión serie, pero no es una comunicación muy apropiada para CORBA.

Otra posibilidad es emplear algo más compatible con un PC, como pueda ser una placa miniITX o un PC104 en la que poder instalar un sistema operativo como Linux o VxWorks y sobre la que usar el ORB de manera similar a como se usa en un PC. De esta manera parece solucionada la parte más relacionada con CORBA pero sigue pendiente la interacción con el hardware. Para esta tarea se puede pensar en añadir un microcontrolador que, comunicado de alguna manera con la otra placa (UART, I2C, SPI...) desempeñe el papel de puente entre ésta y el hardware.

3.4.1. Gumstix

En la línea de la última posibilidad propuesta existe en el mercado una plataforma que ha sido denominada como el ordenador en miniatura completamente funcional (*FFMC: Full Function Miniature Computer*) más pequeño del mundo. Se conoce como Gumstix[2] y físicamente es una placa de 8x2 cm que aloja un microprocesador Intel XScale PXA255 a 200 ó 400 MHz, 64 MB de memoria RAM, 4 ó 16 MB de memoria Flash y la posibilidad de llevar incorporada conexión Bluetooth™.

Pero el tamaño no es la única ventaja, sino que esta plataforma está preparada para ser acompañada de diferentes unidades de expansión, mediante las cuales se puede tener conexión Ethernet, más memoria de almacenamiento, WiFi, una tarjeta de sonido e incluso un microcontrolador de propósito general muy útil para interactuar con el hardware.

Las unidades de expansión son imprescindibles para alimentar la placa principal o para tener una conexión serie en la que por defecto hay una consola activa (es la manera más sencilla de comenzar a manejar el Gumstix). La variante *connex* de Gumstix tiene 2 conectores en los que conectar las unidades de expansión, se encuentran uno a cada lado y son conectores específicos de 60 y de 92 hilos. De este modo se pueden conectar solo 2 unidades

[2]Todas las características de estas placas se pueden observar en la página del fabricante `www.gumstix.com`. Toda la documentación necesaria sobre las placas, su utilización y sobre el entorno de desarrollo se puede encontrar en `www.gumstix.org`.

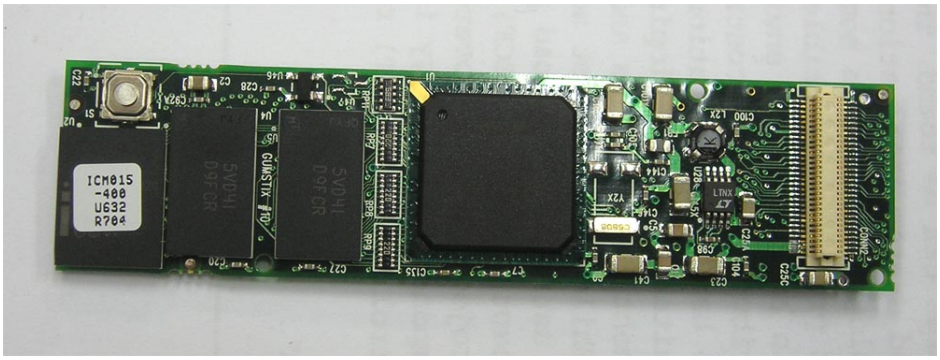

Figura 3.8: Gumstix

de expansión al mismo tiempo (excepto la unidad *tweener* que tiene tanto el conector macho como el hembra de 60 hilos y permite colocarlo entre el Gumstix y otra unidad que se acople a dicho conector).

Otra característica a tener en cuenta es que Gumstix viene con Linux instalado (concretamente con un kernel 2.6.11 y varias aplicaciones como un servidor web, un navegador web, un servidor ssh) y además todas las herramientas de desarrollo son de código abierto.

Para este proyecto se ha decidido usar la plataforma *Gumstix connex 400xm* que tiene memoria extendida, es decir, 16 MB de memoria Flash y funciona a 400 Mhz. Existe una versión de esta placa con conexión Bluetooh integrada pero no se compra porque se tiene intención de realizar la conexión inalámbrica mediante WiFi a través de una placa de expansión.

Además se van a usar 2 placas como accesorio: la primera es *netCF*, una placa que incluye un puerto Ethernet y un zócalo Compact Flash en el que se planea conectar una tarjeta inalámbrica WiFi CF. Con esta placa dotamos al Gumstix de dos interfaces de red, una red cableada y una inalámbrica con las que podremos conectarlo a la red en la que habrá al menos un ordenador con el que pueda interactuar a través de CORBA.

La segunda es *Robostix*, una placa que incluye un microcontrolador de 8 bits Atmel ATmega128 junto con todo lo necesario para su funcionamiento y conectores para tener acceso a prácticamente todos los pines de entrada/salida del mismo. Esta placa se compra sin los conectores para poder colocarlos por el lado contrario al que vienen ya que al montar las 3 placas juntas tocarían con la placa netCF.

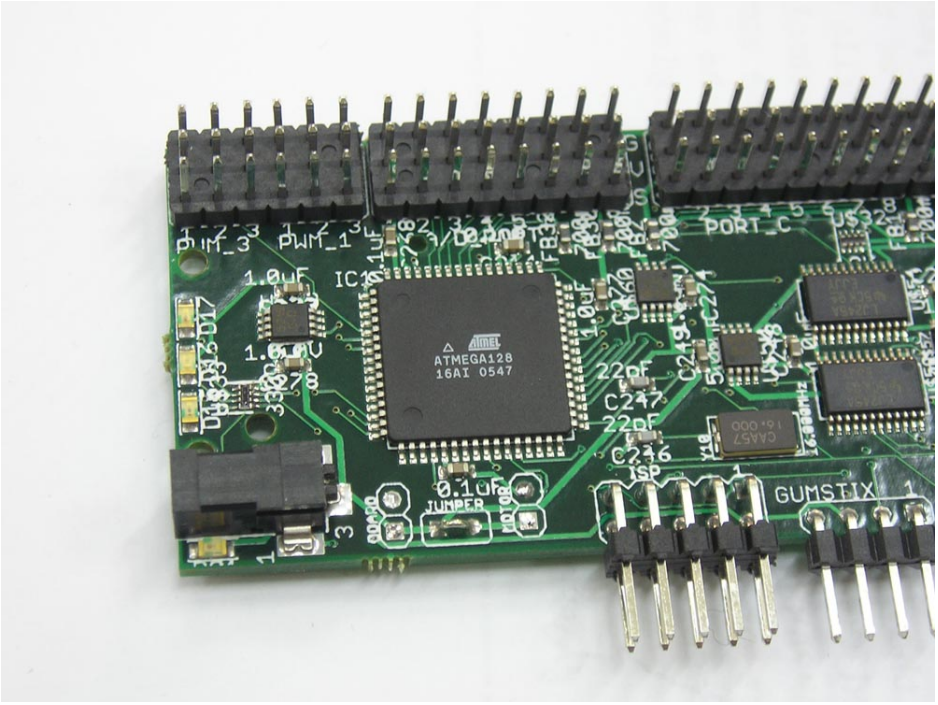

Figura 3.9: Placa de expansión Robostix

La conexión inalámbrica

Se realizará usando la tarjeta DCF-660W de D-Link que es una tarjeta Compact Flash compatible con los slots tipo I y II, que cumple con el estándar IEEE 802.11b y que permitirá conectar el Gumstix a una red inalámbrica. Se ha escogido esta tarjeta, cuya velocidad máxima de transferencia es de 11 Mbps, frente a otras que alcanzan los 54 Mbps porque el chip que incorpora (prism 2.5) está soportado por el driver de red inalámbrica que incorpora el entorno de desarrollo del Gumstix.

En la figura 3.10 se puede de la tarjeta DCF-660W insertada en el zócalo Compact Flash de la placa de expansión netCF.

3.4.2. Microcontrolador ATmega128

El elemento fundamental de la placa Robostix es el microcontrolador ATmega128 de la familia AVR de Atmel. Se trata de una familia de microcontroladores de 8 bits con un juego de instrucciones reducido *(RISC)*,

Figura 3.10: Placa de expansión netCF y tarjeta WiFi

arquitectura Harvard y que integran una gran variedad de periféricos.

El ATmega128 incluye 128 Kbytes de memoria Flash para programas junto con 4 Kbytes de memoria SRAM y 4 Kbytes de memoria EEPROM. Tiene un juego de 133 instrucciones, la mayoría de las cuales son de un ciclo de ejecución y puede ejecutar 16 millones de instrucciones por segundo. También incorpora dos temporizadores de 8 bits y dos de 16 bits, dos puertos de serie, convertidor analógico a digital de 10 bits de resolución y comparador analógico.

Atmel distribuye de manera gratuita un Entorno de Desarrollo Integrado *AVR Studio 4*. Es una herramienta muy completa que incluye ensamblador, compilador de C, simulador y depurador, aunque solo está disponible para Windows y el desarrollo del proyecto se hará mayoritariamente en un sistema Linux con herramientas de código abierto que también se encuentran disponibles.

La grabación de los programas en la memoria Flash se realiza mediante el protocolo ISP *(In-System Programmer)*. En la placa ya se encuentra la parte hardware del programador y en el entorno de desarrollo del Gumstix se ha incluido la aplicación *uisp* para grabar los programas.

En la placa Robostix también hay tres LEDs que resultan de gran ayu-

da a la hora de hacer programas de prueba o depurar los programas del microcontrolador.

3.4.3. PC

El objetivo del proyecto es poder realizar un control mediante software distribuido en el robot, por tanto en el proyecto se realizará una aplicación que sirva de ejemplo de dicho control y que se ejecute en un ordenador personal. Éste será también el ordenador que se emplee para manejar todas las herramientas de desarrollo empleadas en el proyecto. En concreto se trata de un ordenador con un microprocesador Pentium 4 a 2,26 GHz con un sistema operativo Linux Mandrake 10.0.

Este ordenador se podrá conectar de manera inalámbrica al Gumstix, bien a través de una red Ethernet y un punto de acceso inalámbrico o bien con un dispositivo inalámbrico.

3.5. Pruebas sobre el microrrobot

Con el velocista ya construido se han realizado ciertas pruebas que permitan comprobar el funcionamiento del mismo.

Capacidad de giro

Una de las funciones de la interfaz gráfica será tratar de representar la trayectoria recorrida por el robot. Pero para eso es necesario conocer el giro real del robot en función de la variable sobre la que se actúa para controlar la dirección, que es el ancho del pulso de la señal del servo.

Lo primero que ha realizado es, con el robot parado, buscar los límites del servo en los que la dirección choca contra su tope mecánico. Para esta función se implementa un programa que permite controlar directamente desde el teclado del ordenador la señal del servo.

A partir de aquí se hace una prueba “en pista” en la que se mide el radio de giro del velocista para una serie de posiciones del servo. Sobre estos datos se realiza un regresión que permite calcular la función que relaciona el radio de giro con la señal del servo.

De esos datos podemos concluir que el radio de giro mínimo es de 45 cm

girando hacia la derecha y de 40 cm girando hacia la izquierda. El peor radio de giro hacia la derecha es consecuencia de encontrarse en el lado derecho delantero del robot la reducción mecánica que le impide girar tanto como en el otro sentido.

Consumo de la electrónica

A la hora de incorporar al robot el ordenador empotrado uno de los parámetros a tener en cuenta es su consumo, ya que un ordenador empotrado puede consumir mucho, y los datos de consumos que da el fabricante no son suficientemente completos. Lo mismo ocurre con la conexión inalámbrica, así que lo que hacemos es medir los consumos en diferentes condiciones:

Consumos	***Con tarjeta WiFi***	***Sin tarjeta WiFi***
Procesador en reposo	420 mA	540 mA
Procesador trabajando	570 mA	690 mA
En reposo transmitiendo	430 mA	580 mA
Procesando y transmitiendo	580 mA	730 mA

Los datos de consumo en los que no está la tarjeta WiFi se han medido transmitiendo datos a través de la conexión de red Ethernet.

El consumo corresponde al Gumstix y las placas accesorias, es decir, incluye el la placa del microcontrolador y la que tiene el conector Ethernet y la ranura Compact Flash para la WiFi. Pero no se ha tenido en cuenta el consumo ni del motor ni del servo de la dirección cuya alimentación es independiente.

Funcionamiento como velocista

Independientemente del control que se realice sobre el robot desde un ordenador remoto, sólo con el microcontrolador éste ha de funcionar como un velocista, por tanto hay que comprobar que realmente es capaz de seguir una línea negra sobre un fondo blanco.

Tras varias pruebas en un pequeño circuito construido al efecto podemos determinar que el robot puede seguir dicha línea hasta una velocidad de 1,2 m/s.

La prueba más importante en este aspecto es la participación en una competición real. Así que se decidió participar con el robot en una competición

a nivel nacional como es Robolid en su edición del 2006.

La competición se realizó en Valladolid el día 16 de marzo de 2006 y el robot Hyperion partició, como no podía se de otra manera, en la prueba de velocistas. El comportamiento del robot fué satisfactorio y estuvo a la altura de las circunstancias clasificándose 8^{o} por tiempos para las rondas eliminatorias en las que cayó eliminado al enfrentarse al que finalmente sería el vencedor.

Capítulo 4

Plataformas software

En este apartado se describen las principales herramientas y tecnología software en las que se apoya el proyecto.

Primero se muestra el entorno de desarrollo que se usa en el desarrollo de aplicaciones para el *Gumstix*. A continuación, en el apartado 4.2 se explica qué es CORBA, qué se puede hacer con CORBA y cómo se ha utilizado. En la sección 4.3 se describen las herramientas disponibles, y las que se han utilizado para implementar aplicaciones para el microcontrolador *ATmega128* que controla el robot. Y por último, en el apartado 4.4 se presenta la biblioteca Qt con la que se ha realizado la interfaz gráfica.

4.1. Gumstix buildroot

Gumstix *buildroot*[1] es el entorno de desarrollo disponible para trabajar con el Gumstix. Se trata de una serie de scripts que se encargan de descargar y compilar todo lo necesario para reconstruir el sistema de lleva integrado el Gumstix, además de las herramientas para desarrollar aplicaciones para el mismo.

Para comenzar es necesario un cliente de SubVersion[2] con el que nos descargaremos la versión actual del entorno de desarrollo. Lo que tenemos que hacer es escribir en un terminal:

```
svn co http://svn.gumstix.com/gumstix-buildroot/trunk
```

[1] Toda la información se encuentra disponible en la página `www.gumstix.org`

[2] Subversion es un software de control de versiones.

```
                                                    gumstix-buildroot
cd gumstix-buildroot
```

En la carpeta principal se encuentra un *Makefile* en el que podremos seleccionar la versión del compilador *gcc* que queremos, la versión de las *binutils*[3], si queremos o no soporte para C++ y también podemos seleccionar, de entre una lista de aplicaciones, qué software vamos a querer en el Gumstix. En nuestro caso es importante dar el valor *true* a la variable *INSTALL_LIBSTDCPP* para tener soporte de C++, ya que el desarrollo de las aplicaciones basadas en CORBA lo realizaremos en C++.

Después solo es necesario indicarle que comience a compilar:

```
make defconfig
make
```

Lo primero que hará será descargarse desde internet el código fuente de todo lo que tiene que compilar. A continuación compila el compilador *gcc* para obtener un compilador cruzado que permitirá compilar todas las aplicaciones del gumstix. El último paso es compilar de manera cruzada tanto el kernel como el resto de aplicaciones que hemos seleccionado antes y que quedarán instaladas en el Gumstix.

El compilador cruzado es necesario porque no vamos a compilar las aplicaciones en la misma máquina en la que se van a ejecutar, las compilaremos sobre un PC con arquitectura i386 y se ejecutarán sobre el Gumstix que tiene arquitectura ARM. Para compilar desde una máquina con una arquitectura diferente se usa un compilador cruzado que generará archivos ejecutables que solo se podrán ejecutar desde una máquina cuya arquitectura sea para la que se ha compilado.

El resultado de esto es, por una parte, la estructura de directorios con todos los archivos que quedarán en el Gumstix. Todo esto queda en la carpeta `build_arm_nofpu/root`.

Por otra parte crea el archivo `root_fs_arm_nofpu` que contiene lo mismo en un solo archivo, es decir, es una imagen de lo que tendrá el Gumstix. Esta imagen se podrá grabar en la memoria Flash del Gumstix. Para esto tenemos que conectarnos a uno de los puertos de serie donde el Gumstix tiene cofigurada una consola e iniciar U-Boot (es un cargador de arranque o bootloader)

[3]Las binutils son un conjunto de herramientas para el desarrollo de software con las que se realizan diversas acciones relacionadas con los ejecutables.

que está instalado en el Gumstix y al que se puede acceder pulsando una tecla durante el arranque. El bootloader puede recibir esa imagen que se la enviaremos mediante el protocolo kermit[4] y grabarla en la memoria Flash[5].

De entre el resto de cosas que crea es importante destacar la carpeta `build_arm_nofpu/staging_dir` en la que se encontrarán todos los ejecutables del compilador cruzado y de las *binutils* (todos comienzan con el prefijo `arm-linux-`), las librerías y cualquier cosa que, generada por un proyecto, pueda ser usado por otro durante su compilación.

4.1.1. Ejemplo de programación

Para poder ver por primera vez al Gumstix ejecutar un programa propio podemos compilar el típico *hello world*. Por ejemplo, en C++ podemos crear `hello.cpp`:

```
#include <iostream>
using namespace std;

int main()
{
        cout << "Hello world from Gumstix" << endl;
        return 0;
}
```

Suponiendo que la carpeta build_arm_nofpu/staging_dir/bin se encuentra en el PATH podemos compilarlo y enviarlo al Gumstix mediante:

```
arm-linux-g++ hello.cpp -o hello
scp hello root@<ip_del_gumstix>:\root
```

Si ahora nos conectamos al Gumstix mediante ssh podemos ejecutarlo *(./hello)* y ver el mensaje en la consola. Todo esto supone que ya hemos configurado el interfaz de red ethernet o a través de la red inalámbrica, aunque también se puede enviar el archivo y conectarse a través del puerto de serie para ejecutarlo.

[4]Kermit es un protocolo de transferencia y gestión de ficheros y un juego de herramientas que permiten tranferencia de ficheros, emulación de terminales y conversión de juegos de caracteres entre máquinas que pueden tener distinto hardware o sistema operativo

[5]Todo este proceso se encuntra detallado en la sección *Tutorial* de la documentación del Gumstix

4.2. CORBA

La tecnología CORBA está orientada a los sistemas distribuidos heterogéneos. Un sistema distribuido heterogéneo es aquel compuesto por diferentes módulos software que interactúan entre sí sobre distintas plataformas hardware/software[6] unidas por una red de área local.

La *suite* de especificaciones CORBA *(Common Object Request Broker Architecture)* proporciona un conjunto de abstracciones flexibles y servicios concretos necesarios para posibilitar soluciones prácticas a los problemas que surgen en entornos distribuidos heterogéneos.

CORBA no es más que una especificación normativa para la tecnología de la gestión de objetos distribuidos (DOM). Esta tecnología DOM proporciona una interfaz de alto nivel situada en la cima de los servicios básicos de la programación distribuida. En los sistemas distribuidos la definición de la interfaz y ciertos servicios (como la búsqueda de módulos) son muy importantes. CORBA Proporciona un estándar para poder definir estas interfaces entre módulos, así como algunas herramientas para facilitar la implementación de dichas interfaces en el lenguaje de programación escogido.

CORBA es independiente tanto de la plataforma como del lenguaje de la aplicación. Esto significa que los objetos se pueden utilizar en cualquier plataforma que tenga una implementación del CORBA ORB (*Object Request Broker*) y que los objetos y los clientes se pueden implementar en cualquier lenguaje de programación por lo que al programador no le hará falta saber el lenguaje en que ha sido escrito otro objeto con el que se esté comunicando.

4.2.1. El grupo OMG

El OMG *(Object Management Group)* es una organización sin ánimo de lucro creada en 1989 con la misión de crear un mercado de programación basada en componentes, impulsando la introducción de objetos de programación estandarizada.

Su propósito principal es desarrollar una arquitectura única utilizando la tecnología de objetos para la integración de aplicaciones distribuidas garantizando la reusabilidad de componentes, la interoperabilidad y la portabilidad, basada en componentes de programación disponibles comercialmente.

[6]Pueden tener arquitecturas diversas y soportar diferentes sistemas operativos

El OMG es una organización de estandarización de carácter neutral e internacional, ampliamente reconocida. Hoy en día son miembros del OMG alrededor de mil distribuidores de software, desarrolladores y usuarios que trabajan en diferentes campos, incluyendo universidades e instituciones gubernamentales. Además, mantiene estrechas relaciones con otras organizaciones como ISO, W3C, etc. Sus estándares facilitan la interoperabilidad y portabilidad de aplicaciones construidas mediante objetos distribuidos. El OMG no produce implementaciones, sólo especificaciones de software que son fruto de la recopilación y elaboración de las ideas propuestas por los miembros del grupo OMG.

4.2.2. La norma CORBA

CORBA es una especificación normativa que resulta de un consenso entre los miembros del OMG. Esta norma cubre cinco grandes ámbitos que constituyen los sistemas de objetos distribuidos:

- Un lenguaje de descripción de interfaces, llamado **IDL** *(Interface Definition Language)*, traducciones de este lenguaje de especificación IDL a lenguajes de implementación (como pueden ser C++, Java, ADA, Python, etc.) y una infraestructura de distribución de objetos llamada **ORB** *(Object Request Broker)*.
- Una descripción de servicios, conocidos con el nombre de **CorbaServices**, que complementan el funcionamiento básico de los objetos que dan lugar a una aplicación. Cubren los servicios de nombres, de persistencia, de eventos, de transacciones, etc. El número de servicios se amplía continuamente para añadir nuevas capacidades a los sistemas desarrollados con CORBA.
- Una descripción de servicios orientados al desarrollo de aplicaciones finales, estructurados sobre los objetos y servicios CORBA. Con el nombre de **CorbaFacilities**, estas especificaciones cubren servicios de alto nivel (como los interfaces de usuario, los documentos compuestos, la administración de sistemas y redes, etc.) Pretende definir colecciones de objetos prefabricados para aplicaciones habituales.
- Una descripción de servicios verticales denominados **CorbaDomains**, que proveen funcionalidad de interés para usuarios finales en campos de aplicación particulares. Por ejemplo, existen proyectos en curso en sectores como: telecomunicaciones, finanzas, medicina, etc.

- Un protocolo genérico de intercomunicación, llamado **GIOP** *(General Inter-ORB Protocol)*, que define los mensajes y el empaquetado de los datos que se transmiten entre los objetos. Además define implementaciones de ese protocolo genérico, sobre diferentes protocolos de transporte, lo que permite la comunicación entre los diferentes ORBs consiguiendo la interoperabilidad entre elementos de diferentes vendedores. Como por ejemplo el IIOP para redes con la capa de transporte TCP.

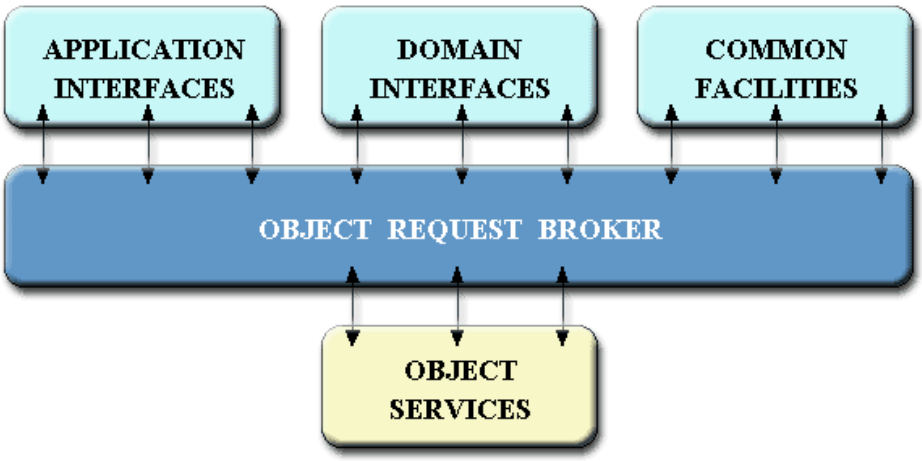

Figura 4.1: Servicio de objetos CORBA

4.2.3. La Tecnología CORBA

En esta sección se hace una descripción general de la arquitectura CORBA y su funcionamiento. Para un conocimiento más detallado es necesario recurrir a los documentos de especificación del OMG[7].

Common Object Request Broker Architecture (CORBA)

CORBA especifica un sistema que proporciona interoperabilidad entre sistemas que funcionan en entornos heterogéneos y distribuidos de forma transparente para el programador. Su diseño se basa en el modelo de objetos

[7]Más información en `www.omg.org`

del OMG, donde se definen las características externas que deben poseer los objetos para que puedan operar de forma independiente de la implementación.

Las características más destacables de CORBA son las siguientes:

- es una tecnología no propietaria
- una base fundamental de la especificación son los requisitos reales de la industria
- es una arquitectura extensible
- es independiente de la plataforma
- es independiente del lenguaje de programación
- proporciona múltiples servicios de aplicación
- servidores y clientes (proveedores y usuarios de servicios) pueden desarrollarse de manera totalmente independiente

En una aplicación desarrollada en CORBA los objetos distribuidos se utilizan de la misma forma en que serían utilizados si fueran objetos locales, esto es, la distribución y heterogeneidad del sistema queda oculta y el proceso de comunicación entre objetos es totalmente transparente para el programador.

Arquitectura general

Un objeto CORBA es una entidad que proporciona uno o más servicios a través de una interfaz conocida por los clientes que requieren dichos servicios. Un objeto CORBA se define como aquel que proporciona algún tipo de servicio, de modo que las entidades que sólo los utilizan no se consideran como tales.

El componente central de CORBA es el ORB *(Object Request Broker)*, el cual proporciona la infraestructura necesaria para identificar y localizar objetos, gestionar las conexiones y transportar los datos a través de la red de comunicación. En general el ORB está formado por la unión de varios componentes, aunque es posible interaccionar con él como si fuera una única entidad gracias a sus interfaces. El núcleo del ORB *(ORB Core)* es la parte fundamental de este componente, siendo el responsable de la gestión de las peticiones de servicio. La funcionalidad básica del ORB consiste en transmitir las peticiones desde los clientes hasta las implementaciones de los objetos servidores.

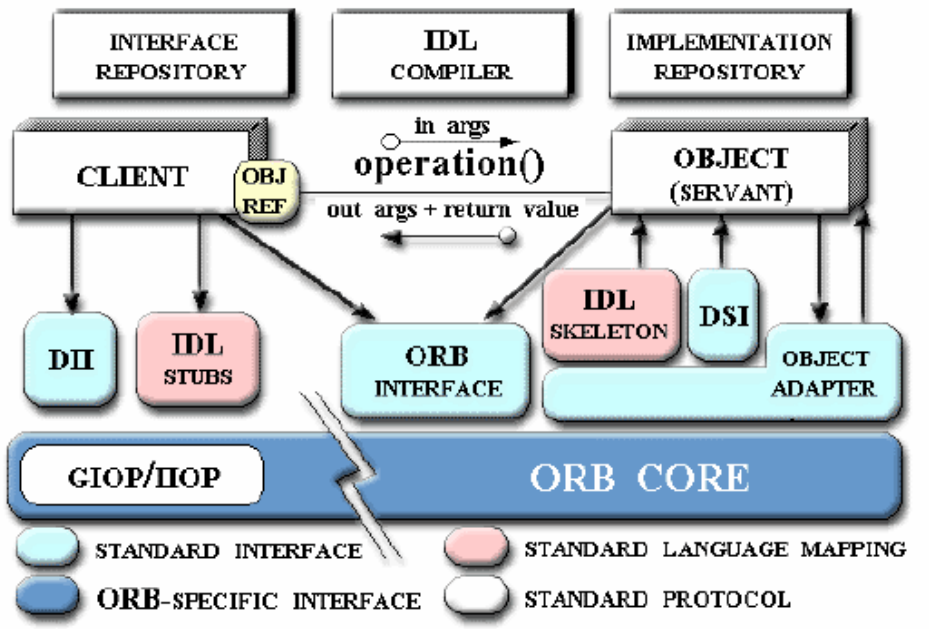

Figura 4.2: Componentes de la arquitectura CORBA

Los clientes realizan peticiones de servicio a los objetos, también llamados servidores, a través de interfaces de comunicación bien definidas. Las peticiones de servicio son eventos que transportan información relativa a los objetos o entidades implicadas en dicho servicio, información sobre la operación a realizar, parámetros, etc. En la información enviada se incluye una referencia de objeto del objeto proveedor del servicio; esta referencia es un nombre complejo que identifica de forma unívoca al objeto en cuestión dentro del sistema.

Para realizar una petición un cliente se comunica primero con el ORB a través de uno de los dos mecanismos existentes a su disposición: para el caso de invocación estática el *stub* o para el caso de la invocación dinámica el DII, *Dynamic Invocation Interface*.

- Invocación estática: el *stub* es un fragmento de código encargado de mapear o traducir las peticiones del cliente, que está implementado en un lenguaje determinado, y transmitírselas al ORB. Es gracias a los *stubs* que los clientes pueden ser programados en diferentes lenguajes de programación.

- Invocación dinámica: se basa en el uso de la DII. Permite realizar invocaciones sin necesidad de que el cliente sepa cierta información acerca del servidor, que sería necesaria en caso de utilizar el *stub*.

Una vez la petición llega al núcleo del ORB es transmitida hasta el lado del servidor, donde se sigue un proceso inverso, de nuevo a través de dos mecanismos alternativos: el *skeleton* del servidor, análogo al *stub* del cliente, o la *DSI (Dynamic Skeleton Interface)*, contrapartida de la DII. Los servicios que proporciona un servidor CORBA residen en la implementación del objeto, escrita en un lenguaje de programación determinado. La comunicación entre el ORB y dicha implementación la realiza el adaptador de objetos (*Object Adapter*, OA), el cual proporciona servicios como:

- generación e interpretación de referencias a objetos
- invocación de métodos de las implementaciones
- mantenimiento de la seguridad en las interacciones
- activación o desactivación de objetos
- mapeo de referencias a objetos
- registro de implementaciones

Existen adaptadores de objetos más complejos, que proporcionan servicios adicionales, y que son utilizados para aplicaciones específicas (por ejemplo, bases de datos). Dos adaptadores de objetos básicos son el BOA (*Basic Object Adapter*) y el POA (*Portable Object Adapter*). El primero ha quedado ya obsoleto, y suele utilizarse el segundo adaptador. El ORB proporciona además un POA preconfigurado que puede usarse si no se desea crear un POA específico para una aplicación, llamado *RootPOA*.

El adaptador de objetos necesita conocer cierta información acerca de la implementación del objeto y del sistema operativo en el que se ejecuta. Existe una base de datos que almacena esta información, llamada *Interface Repository* (IR), que es otro componente estándar de la arquitectura CORBA. El IR puede contener otra información relativa a la implementación como por ejemplo datos de depurado, versiones, información administrativa, etc.

Las interfaces de los objetos servidores pueden especificarse de dos maneras: o bien utilizando el lenguaje IDL, o bien almacenando información necesaria en el IR. Para el caso de la invocación dinámica la interfaz DII accede al IR en busca de ésta información en concreto cuando un cliente la

utiliza para realizar una petición. De este modo se hace posible que un cliente pueda invocar un servicio sin necesidad de conocer la descripción IDL de la interfaz del objeto servidor.

Por último, en el lado del servidor CORBA, los objetos que se encargan en última instancia de atender a las peticiones de servicio son los *servants*. Estos objetos contienen la implementación de las operaciones asociadas a cada servicio o método de la interfaz del objeto. No son visibles desde el lado del cliente; este sólo ve una entidad única a la que conoce como servidor. Dentro del servidor se crean y gestionan *servants* que atienden las peticiones que llegan al mismo. El adaptador de objetos es el encargado de hacer llegar dichas peticiones a los *servants*, crearlos o destruirlos, etc. Cada *servant* lleva asociado un identificador llamado *object ID*, valor que es utilizado por el adaptador de objetos para la gestión de los mismos.

Interoperabilidad entre ORBs

En la actualidad existen muchas implementaciones diferentes de ORBs, lo cual es una ventaja ya que cada desarrollador puede emplear aquella que mejor satisface sus necesidades. Pero esto crea también la necesidad de un mecanismo que permita a dos implementaciones diferentes comunicarse entre sí. También es necesario en determinadas situaciones proporcionar una infraestructura que permita a aplicaciones no basadas en CORBA comunicarse con aplicaciones basadas en esta arquitectura. Para satisfacer todos estos requisitos existe una especificación para lograr la interoperabilidad entre ORBs.

Las diferencias en la implementación del *broker* no son la única barrera que separa a objetos que funcionan en distintos ORBs, también existen otras dificultades como infraestructuras de seguridad o requisitos específicos de sistemas en vías de desarrollo. Debido a esto, los objetos que funcionan en diferentes dominios (ORBs y entornos de los mismos) necesitan un mecanismo que haga de puente entre ellos. Este mecanismo debe ser suficientemente flexible para que no sea necesaria una cantidad de operaciones de "traducción" inmanejable, ya que la eficiencia es uno de los objetivos de la especificación de CORBA. Esta característica es crítica en sistemas de control, donde suele ser necesario alcanzar unos niveles determinados de seguridad y predecibilidad.

La interoperabilidad puede alcanzarse a través muchos procedimientos, los cuales pueden clasificarse en dos tipos principales: inmediatos e interme-

diados (*immediate/mediated bridging*). En los procedimientos intermediados los elementos de un dominio que interaccionan con los de otro dominio distinto, son transformados a un formato interno acordado por ambos dominios de antemano, y bajo este formato la información pasa de un dominio al otro. Este formato interno de la información puede basarse en una especificación estándar, como en el caso del IIOP del OMG, o bien puede ser un formato acordado de forma privada. Los procedimientos inmediatos se basan en traducir directamente la información desde el formato utilizado por un dominio, al formato utilizado por el otro, sin pasar por estructuras intermedias de datos. Esta solución es mucho más rápida, pero es menos general.

CORBA IDL

Como ya hemos mencionado, el lenguaje IDL (*Interface Definition Language*) es un lenguaje utilizado para especificar interfaces CORBA. A través de un compilador específico que procesa las definiciones escritas en IDL, se genera código fuente en el lenguaje de programación deseado en cada caso, código que es utilizado en las aplicaciones para crear servidores CORBA ya que proporciona la infraestructura de *skeletons* y *stubs* necesaria para que estos objetos puedan comunicarse con el ORB. IDL es un estándar ISO, y tiene las siguientes características principales:

- su sintaxis es muy similar a la de C++
- soporta herencia múltiple de interfaces
- independiente de cualquier lenguaje de programación y/o compilador, puede mapearse a muchos lenguajes de programación.
- permite independizar el diseño de la interfaz de la implementación del objeto CORBA en cuestión. La implementación puede cambiarse por otra distinta, manteniendo la misma interfaz, de forma que desde el "exterior" el objeto sigue siendo el mismo y continúa ofreciendo idénticos servicios.

IDL no es un lenguaje de programación propiamente dicho, es únicamente un lenguaje declarativo. Tiene tres elementos principales: operaciones (métodos), interfaces(conjuntos de operaciones) y módulos (conjuntos de interfaces).

El IDL se mapea ("traduce") al compilar al lenguaje de programación deseado. Para el lenguaje C++, IDL sigue la siguiente tabla de conversión:

Tipo de datos IDL	*Tipo de datos C++*	*typedef*
short	CORBA::Short	short int
long	CORBA::Long	long int
unsigned short	CORBA::UShort	unsigned short
unsigned long	CORBA::ULong	unsigned long
float	CORBA::Float	float
double	CORBA::Double	double
char	CORBA::Char	char
boolean	CORBA::Boolean	unsigned char
octet	CORBA::Octet	unsigned char
enum	enum	enum

4.2.4. Bases de la construcción de aplicaciones CORBA

Para desarrollar un objeto CORBA se siguen, en general, los siguientes pasos:

1. **Diseño**: determinación de los servicios que debe proporcionar el objeto, e implementación de la/las interfaces IDL correspondientes.

2. **Compilación de IDL**: mediante el compilador de IDL se procesan las definiciones de interfaces y se generan los *skeletons* y *stubs* correspondientes, en un lenguaje de programación determinado.

3. **Implementación de servants**: utilizando las clases generadas por el compilador de IDL se crean los *servants* de la aplicación, que contienen la implementación de las operaciones del objeto CORBA.

4. **Implementación del servidor**: el objeto CORBA debe proporcionar la infraestructura base para que la comunicación con el ORB sea posible; debe crear un adaptador de objetos para sus *servants*, etc.

5. **Compilación**: se procede a la compilación de los archivos de código fuente. Debe incluirse el código generado automáticamente por el compilador de IDL (la parte correspondiente a los *skeletons*).

El proceso se reduce básicamente a tres fases: generación e integración de código correspondiente a las interfaces, implementación de la funcionalidad del objeto CORBA y por último, compilación. Estos pasos son los correspondientes para el desarrollo de un objeto CORBA, es decir, un servidor. Para

el desarrollo de un cliente, el proceso se reduce a la integración del código fuente generado correspondiente a los *stubs*.

4.2.5. ORB

Existe una gran cantidad de ORB's que se pueden emplear, los hay de pago y gratuitos, incluso algunos con soporte para tiempo real.

Para éste proyecto no es necesario soporte para tiempo real, ya que no se van a implementar aplicaciones que lo requieran, pero sí que se busca un ORB de código abierto.

El broker utilizado para este proyecto ha sido MICO (acrónimo de MICO Is CORBA). MICO es un broker de código abierto que implementa el estándar CORBA 2.3 y que ha sido elegido por varias razones:

- Es de código abierto.
- Está pensado para implementar el código en C++.
- Es un broker "sencillo", es decir, solo implementa lo que especifica el estándar, lo que lo hace apto para usarlo en arquitecturas empotradas.
- No requiere el uso de software propietario o de librerías especializadas.

Instalación de MICO

Como se van a comunicar dos arquitecturas diferentes usando CORBA, hay que compilar el broker para ambas plataformas Primero se instala la versión para el PC, es una instalación estándar y basta con seguir las instrucciones que vienen con el codigo fuente:

```
tar -zxvf mico-2.3.12RC2.tar.gz
cd mico
./configure --prefix=/opt2/mico --enable-final
make
make install
```

Lo siguiente es compilar la versión para la arquitectura ARM. Disponemos de un compilador cruzado resultado de la instalación del *gumstix buildroot* en el capítulo 4.1. El problema es que la compilación cruzada crea todos los ejecutables para la arquitectura ARM, sin darse cuenta de que algunos de los

ejecutables son para el sistema anfitrión (o *host*). La solución más sencilla, aunque no resulte muy elegante, es compilar otra versión para el sistema *host* lo más parecida a la versión para el ARM y con librerías estáticas, de manera que los ejecutables puedan ser sustituidos por los que crea el compilador cruzado cuando dé un error en la compilación intentando ejecutar uno de ellos.

```
tar -zxvf mico-2.3.12RC2.tar.gz
cd mico
./configure --prefix=/opt2/mico_static_minimum
         --enable-minimum-corba --enable-final --disable-shared
make
make install
```

Para terminar se compila la versión cruzada, para eso se definen las variables de entorno necesarias para localizar el compilador cruzado.

```
export CC=arm-linux-gcc
export CXX=arm-linux-g++
export AR=arm-linux-arm
export LD=arm-linux-ld
export CXXFLAGS="-ffunction-sections -fdata-sections"
export LDFLAGS="-Wl,--gc-sections"

tar -zxvf mico-2.3.12RC2.tar.gz
cd mico
./configure --prefix=/opt2/mico_arm --build=i386 --host=i386
       --target=arm-linux --enable-final --enable-minimum-corba
make
make install
```

La compilación se detendrá varias veces porque intentará ejecutar varios archivos que acaba de compilar para la plataforma ARM. Cuando uno da error, sustituirlo por el ejecutable de la versión anterior (hay que tener en cuenta una cosa, el primero que falla es mkdepend, que lo primero que intenta crear es un archivo `.depend` en la carpeta `orb`, este archvio tendrá que ser eliminado también al cambiar el ejecutable, porque este archivo ya ha sido creado pero le faltan líneas y si no se elminina no será completado).

Como la intención es crear los ejecutables para la plataforma ARM con librerías estáticas, hay que tener en cuenta que las librerías se añaden completas a los ejecutables y eso supone mucho código que no se ejecuta. La solución es compilar con los parámetros `-ffunction-sections -fdata-sections` y linkar con `-Wl,--gc-sections`, de esta forma hacemos que la unidad de división de las librerías sea cada función y sólo se incluyan las que se van a ejecutar.

Para terminar, se compila uno de los ejemplos que vienen con MICO, concretamente el que se encuentra en los ejemplos en `poa/account-1`.

Para compilarlo quitando las funciones de las librerías que no se usan se ha usado el siguiente Makefile:

```
MICO_ARM=/opt2/mico_arm
IDL=idl
CXX=arm-linux-g++
CXXFLAGS=-Os -I. -I$(MICO_ARM)/include -ffunction-sections
                                                -fdata-sections
LD=arm-linux-g++
LDFLAGS=--static -L$(MICO_ARM)/lib -Wl,--gc-sections
LDLIBS=-lm -lmico2.3.12RC2 -ldl -lpthread
STRIP=arm-linux-strip

all: client server

server: account.h account.o server.o
     $(LD) $(LDFLAGS) account.o server.o $(LDLIBS) -o $@;
                                                  $(STRIP) $@

client: account.h account.o client.o
     $(LD) $(LDFLAGS) account.o client.o $(LDLIBS) -o $@;
                                                  $(STRIP) $@

client.o: account.h client.cc
     $(CXX) $(CXXFLAGS) -c client.cc -o $@

server.o: account.h server.cc
     $(CXX) $(CXXFLAGS) -c server.cc -o $@
```

```
account.o: account.h account.cc
     $(CXX) $(CXXFLAGS) -c account.cc -o $@

account.h account.cc : account.idl
     $(IDL) account.idl
```

Para compilarlo es necesario que la carpeta `bin` de mico y la carpeta en la que se encuentra el compilador cruzado estén en el PATH.

Se puede observar que, después de enlazar los ficheros objeto y crear los ejecutables, éstos son pasados por *strip*, para que les quite todos los símbolos innecesarios y así reducir más su tamaño.

4.3. Herramientas de desarrollo para el microcontrolador

ATMEL, el fabricante de la familia de microcontroladores AVR a la que pertenece el microcontrolador ATmega128 que vamos a usar, pone a disposición de los programadores un Entorno de Desarrollo Integrado conocido como *AVR Studio 4*. Se trata de una herramienta gratuita que incluye ensamblador, compilador de C, simulador y depurador.

Sin embargo el proyecto se va a desarrollar en una máquina que tiene una distribución de GNU/Linux como sistema operativo, para el que también existen herramientas de desarrollo gratuitas.

Para desarrollar aplicaciones en lenguaje ensamblador existe un ensamblador llamado `avra` cuyo código fuente se puede encontrar en internet.

Aprovechando que la arquitectura del mismo está pensada para ser programado en C y que existe un muy buen compilador de C para esta familia de microcontroladores, vamos a emplear este lenguaje para programarlo. De este modo podremos realizar programas de cierta complejidad de manera más rápida que en lenguaje ensamblador.

Las herramientas imprescindibles para esto son tres:

avr-gcc Es un compilador de C y ensamblador. Se trata de la versión del compilador *gcc* de GNU para los microcontoladores AVR.

avr-binutils Las GNU Binutils son un grupo de herramientas de programación para la manipulación de código objeto. *avr-binutils* son las he-

rramientas para manejar los archivos que se obtienen al compilar con *avr-gcc*. Éstas tendrán que ser instaladas antes que avr-gcc.

avr-libc Es una librería de C para usar el compilador *gcc* con la familia de microcontroladores AVR.

Con estas tres herramientas se dispone de todo lo necesario para programar en C para los microcontroladores AVR. En la página web de la *avr-libc*, `www.nongnu.org/avr-libc`, se puede encontrar una amplia documentación y ejemplos además de una completa guía para la instalación de las tres herramientas.

Hay otras herramientas útiles, aunque no imprescindibles, para desarrollar aplicaciones para estos microcontroladores.

uisp Se trata de la herramienta que hay que usar para grabar los programas en el microcontrolador. En nuestro caso esta herramienta está ya instalada en el Gumstix, que es desde donde lo programaremos.

avrdude Es una herramienta similar a la anterior, se puede usar también para programar el microcontrolador.

avr-gdb Esta aplicación es la versión para los microcontroladores AVR del depurador *gdb* de GNU. Sirve para detectar errores en los programas.

SimulAVR Se trata de un simulador. Esta herramienta se usa conjuntamente con *avr-gdb* para depurar el código, detectar fallos y comportamiento erróneo para poder corregirlo.

AVaRice Permite usar el depurador *avr-gdb* a través de un interfaz JTAG, de manera que se pueda realizar una depuración en el mismo circuito de la aplicación final mientras se ejecuta el programa.

4.4. Qt

La biblioteca Qt (o simplemente Qt) es una herramienta de programación para desarrollar interfaces gráficas de usuario. Una interfaz gráfica de usuario (GUI) es un método para facilitar la interacción del usuario con el ordenador a través de la utilización de un conjunto de imágenes y objetos (como iconos o ventanas) además de texto. Surge como evolución de la línea de comandos de los primeros sistemas operativos y es pieza fundamental en un entorno gráfico.

Un poco de historia.

Inicialmente Qt apareció como biblioteca desarrollada por Trolltech[8] en 1992 siguiendo un desarrollo basado en el código abierto, pero no libre. Se usó activamente en el desarrollo del escritorio KDE (entre 1996 y 1998), con un notable éxito y rápida expansión. Esto fomentó el uso de Qt en programas comerciales para el escritorio, situación vista por el proyecto GNU como amenaza para el software libre.

Para contrarrestarla se plantearon dos ambiciosas iniciativas: por un lado el equipo de GNU en 1997 inició el desarrollo del entorno de escritorio GNOME con [GTK+] para GNU/Linux. Por otro lado se intenta construir una biblioteca compatible con Qt pero totalmente libre, llamada Harmony.

Qt cuenta actualmente con un sistema de doble licencia: por un lado dispone de una licencia GPL para el desarrollo de software de código abierto (open source) y software libre gratuito; y por otro una licencia de pago para el desarrollo de aplicaciones comerciales.

4.4.1. Designer

Como ayuda para el diseño de la interfaz gráfica con Qt se utiliza la herramienta *Designer*[9] (Figura 4.3).

Designer nos permite la creación de diálogos interactivos con distintos objetos (barras, displays ...). Para estos elementos se definen sus propiedades tales como el nombre, tamaño, tipo de letra, etc. Estos son clases dentro de la aplicación que estamos diseñando. Posteriormente será necesario implementar la funcionalidad mediante el código que se debe ejecutar cuando se recibe el evento correspondiente.

Resultado del trabajo con *Designer* se obtiene un archivo de extensión ".ui". Utilizando el compilador de ui (ui compiler) para generar la descripción de la clase ".h" y su implementación ".cpp" (implementación de la clase). Como la parte de Qt emplea las palabras clave "SLOT" y "SIGNAL" que no significan nada para C++, es necesario pasar el ".h" por el compilador de objetos de Qt MOC (Meta Object Compiler) que se encarga de crear "glue code" de modo que se genera código C++ correspondiente a los mecanismos de comunicación "SIGNAL-SLOT" (una especie de mapeo o traducción a

[8] www.trolltech.com

[9] www.trolltech.com/products/qt/designer.html

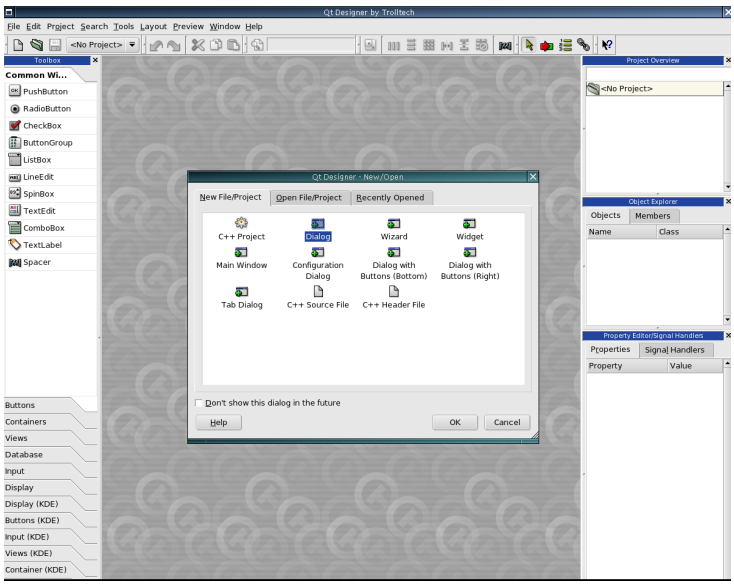

Figura 4.3: Herramienta para Qt: Designer

C++).

Es necesario implementar el comportamiento de los distintos elementos del diálogo, es decir, indicar qué tipo de acción realizarán cuando se produzca un evento. Los archivos ".cpp" descritos anteriormente son creados directamente a partir del ".ui" generado por el *Designer*, por lo que no conviene tocarlos ya que cualquier cambio que se haga en ellos se perderá al modificar la interfaz.

El proceso seguido para la implementación de la funcionalidad característica de cada elemento de la interfaz diseñada es crear una clase que herede de la clase generada en la implementación del ui, de modo que en esta nueva clase se definan los métodos virtuales de la clase base.

Una vez esté implementada la funcionalidad es necesario pasar el ".h" por el MOC porque aparecen de nuevo mecanismos de "SIGNAL-SLOT" y es necesario generar el "glue code" para que el compilador de C++ pueda entenderlo.

Para generar el binario es necesario escribir la aplicación principal, en la que se instancia la clase derivada en un entorno de aplicación de Qt.

Es momento de reunir los archivos, compilarlos para generar los ficheros de código objeto, enlazar estos con las librerias de Qt y obtener de este modo

el binario de la aplicación.

Capítulo 5

Implementación software

En este capítulo se describe el software que se ha desarrollado a lo largo del proyecto. Se empieza presentando los diferentes computadores que intervienen y se describen las tareas que realizará cada uno. Posteriormente se explican los diferentes métodos de comunicación que se han empleado para terminar explicando las aplicaciones de cada uno de los computadores.

5.1. Planteamiento

Para la realización de este Proyecto Final de Carrera se han implementado aplicaciones que se ejecutan sobre 3 procesadores diferentes. El primero es un microcontrolador de 8 bits de ATMEL, un ATmega128 que lo hemos programado en C usando el juego de herramientas gratuitas que existen disponibles para programarlos. El segundo es el *Gumstix*, se trata de un pequeño ordenador empotrado con arquitectura ARM y sobre el que corre un sistema operativo GNU/Linux. El último es un ordenador personal sobre el que también corre un sistema operativo GNU/Linux.

Dado que se ha de desarrollar software para diferentes plataformas es importante tener en cuenta desde qué máquina se va a realizar el desarrollo. Por una parte porque el sistema operativo de esa máquina determinará qué herramientas se pueden usar, y por otro porque ha de ser capaz de compilar de manera cruzada aplicaciones que se han de ejecutar en las otras arquitecturas.

Durante la realización de este proyecto se va a usar como ordenador de desarrollo de las aplicaciones un Dell Dimension 8200 con un microprocesador Intel Pentium 4 a 2,26 GHz sobre el que corre un sistema operativo Linux

Mandrake 10.1. Además éste será uno de los tres computadores que se usen en el proyecto.

En el transcurso del proyecto se han implementado una gran cantidad de aplicaciones para las tres plataformas. El desarrollo de estas aplicaciones tenía dos objetivos:

- Las primeras aplicaciones tenían únicamente como objetivo la familiarización con las herramientas de desarrollo y con el funcionamiento del hardware.
- A continuación se realizaron diversas aplicaciones ya orientadas hacia la aplicación final que implementan pequeñas partes del funcionamiento de la misma. Estas aplicaciones pueden ser probadas de manera independiente y quitar mucha complejidad de las aplicaciones finales para las que la tarea principal será la integración de los bloques parciales.

Otro de los aspectos a tener en cuenta en la realización del proyecto es el funcionamiento que queremos para el software que controla el robot. Típicamente, un robot de estas características realiza ciclos en los que examina todas las variables de entrada, decide lo que va a hacer y pone en marcha los actuadores. Vamos a intentar que el robot, y por tanto el programa que funciona en el microcontrolador que lo controla directamente, funcionen de esta manera. Pero trataremos de que ésto se realice siguiendo una frecuencia determinada, es decir, determinando un intervalo de tiempo en el que el robot pueda hacer todas sus tareas y que además permita una respuesta suficientemente rápida ante cambios en el entorno.

Esto implica que tendremos que determinar la duración de todas las tareas a realizar por el microcontrolador y que será éste el que marque el ritmo a las demás aplicaciones. Para que todo pueda funcionar así también tendremos que determinar si la duración de lo que tienen que hacer cada uno de los computadores es inferior a la duración del ciclo.

5.2. Distribución del software

Al disponer de tres computadores no solo hemos de determinar la funcionalidad global, sino también cual va a ser la funcionalidad que se implemente en cada uno de ellos.

La idea fundamental es que el sistema ha de tener una cierta modularidad, haciendo que el robot pueda funcionar de manera sencilla aunque no disponga del control de alto nivel. Además, el sistema ha de ser abierto, la aplicación que se ejecuta en cada una de las plataformas ha de poder ser cambiada sin que implique un cambio en las aplicaciones de las otras dos plataformas.

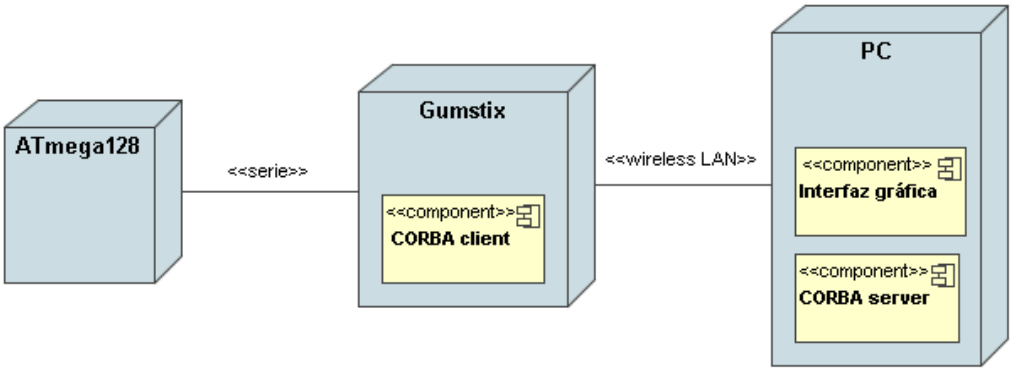

Figura 5.1: Diagrama de despliegue del software desarrollado

Microcontrolador

La aplicación del microcontrolador, al ser éste el que está en contacto directo con el hardware, será la que tenga que interpretar los datos de los sensores y generar las salidas que controlen los actuadores. También ha de ser capaz de hacer que el robot funcione como un robot velocista, es decir, ha de ser capaz de localizar la línea y seguirla.

Como ya se comentó en el capítulo 3 la relación de marchas del robot es muy larga, y esto hace que el sistema motriz sea muy lento, por lo que es imprescindible implementar un control de velocidad. Para que este control sea lo más rápido posible se ha de implementar sobre el microcontrolador y de este modo no se le añaden retardos adicionales debidos al tráfico de datos.

Por último, el robot ha de estar comunicado con el Gumstix, ha de ser capaz de transmitirle toda la información que maneja el microcontrolador y ha de poder recibir órdenes como la referencia de velocidad.

Gumstix

El Gumstix, a su vez, ha de servir de puente entre el microcontrolador y el ordenador que realice el control de alto nivel. Ha de estar comunicado con el microcontrolador para recibir todos los datos del robot y enviarle referencias; y tiene que ejecutar una aplicación basada en CORBA con la que se comunique con el ordenador remoto. La decisión de qué máquina hará de cliente o de servidor se decidirá más adelante.

También ha de tener la posibilidad de controlar el robot si no se puede comunicar la aplicación remota, o de realizar un registro de todo lo que le ocurre al microcontrolador que permita hacer un análisis a posteriori de lo que le ha ocurrido al robot.

PC remoto

La aplicación en el ordenador remoto será la que se encargue del control de alto nivel sobre el robot. El objetivo de este proyecto no es realizar este control sino hacer todo el sistema abierto que lo haga posible, por tanto la aplicación que se va a realizar para ejecutar en este ordenador será una aplicación que permita demostrar que se puede realizar ese control. Concretamente se va a implementar una monitorización gráfica de todo el estado del robot, de la trayectoria recorrida y un mando que permita establecer la velocidad de referencia del robot.

5.3. Estructura de comunicación

La comunicación entre los diferentes programas que integran el proyecto es una parte muy importante, así que lo primero va a ser determinar cómo va a ser esa comunicación.

5.3.1. Comunicación serie Gumstix-Robostix

Microcontrolador → Gumstix

Para poder tener control sobre el robot en un ordenador remoto es importante que se pueda disponer de todos los datos que maneja el microcontrolador. Por tanto, desde el microcontrolador enviaremos, en cada ciclo, todos los

datos de los sensores, las señales que se les pasan a los actuadores y también los valores significativos que se calculen en relación al estado del robot.

Para la comunicación entre el microcontrolador ATmega128 que hay en la placa Robostix y el Gumstix la posibilidad más sencilla es una comunicación por puerto de serie, ambos tienen dos puertos de serie en niveles de tensión TTL[1] en los que solo hay que comunicar la recepción (cable Rx) de uno de los puertos de uno con la transmisión (cable Tx) de uno de los puertos del otro y viceversa.

La velocidad de transmisión de datos ha de ser lo más rápida posible, así que probamos con las velocidades más rápidas que soportan el puerto de serie del Gumstix y del microcontrolador. Las velocidades más rápidas que soporta el puerto de serie del Gumstix son de 115200 y 230400 bps *(bits por segundo)* mientras que el microcontrolador soporta velocidades mayores. Aunque las velocidades que puede generar el generador de reloj para el puerto de serie del microcontrolador (cuando el cristal de cuarzo es de 16 MHZ como en nuestro caso) no son esas exactamente, las más parecidas son 117647 bps (2,1 % más que 115200) y 222222 bps (3,5 % menos que 230400). Tras varias pruebas con ambas velocidades se decide usar la de 115200 ya que con 230400 se produce algún error.

De todos los datos que hay que transmitir a través del puerto de serie, la lectura del sensor lineal es el más voluminoso. Se trata de la medida del la cantidad de luz en los 64 fotodiodos. En la conversión analógico a digital de esas medidas solo se van a considerar 8 bits ya que no es necesaria mucha resolución, ésto supone valores de entre 0 y 255. No podemos enviar directamente las medidas como un byte, porque muchas corresponderían a caracteres ASCII especiales que no son imprimibles y que no los vamos a reconocer.

Si enviamos cada uno de los dígitos de la medida (por ejemplo un '1', un '4' y un '3' para un 143) estaríamos enviando $64 * 3 = 192$ caracteres, si estamos usando el protocolo 8N1 (8 bits de datos y 1 de parada) esto duraría aproximadamente 14,7 ms que es mucho tiempo dentro de cada ciclo.

Podríamos enviarlo usando dos caracteres en hexadecimal pero seguiría siendo mucho tiempo. Como las medidas del sensor lineal están bastante lejos de los valores altos, decidimos enviar un único byte por cada medida usando

[1]Los niveles de tensión TTL son 5 v y 0 v, mientras un puerto de serie con interfaz RS-232 maneja tensiones de ±12 v

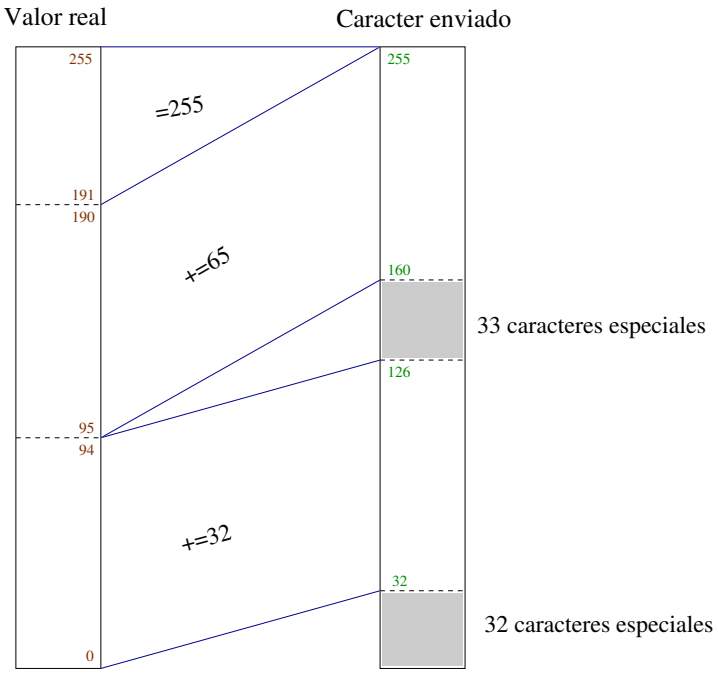

Figura 5.2: Transformación de las medidas para enviarlas

únicamente los 191 caracteres ASCII imprimibles[2]. Todas las medidas por encima del 190 las tratamos como si fueran un 190. En la figura 5.2 se puede ver gráficamente.

Esta conversión la realizamos en el microcontrolador antes de enviar los datos. Posteriormente la haremos a la inversa en el Gumstix para obtener las medidas originales.

Además de las medidas del sensor lineal los datos que se van a enviar desde el microcontrolador hasta el Gumstix son: un número de índice, posición del servo, velocidad medida en el encoder y velocidad de referencia. La trama es de la forma:

`n:<indice> l:<lineal> s:<servo> e:<encoder> r:<ref_vel>`

Donde:

<**indice**> Es el número de índice en 5 caracteres.

<**lineal**> Las 64 medidas del sensor lineal según se describe anteriormente.

[2]Los caracteres ASCII cuyos valores están entre 0 y 31 y entre 127 y 159 no son imprimibles

<**servo**> Es el valor del comparador que determina el ancho del pulso del servo en 5 caracteres (en unidades de 0,5 µs).

<**encoder**> Es el número de saltos que ha detectado el encoder en los últimos 5 ciclos expresado en 3 caracteres (resultarán dm/s).

<**ref_vel**> Es la referencia de velocidad, 2 caracteres.

Teniendo en cuenta el caracter de cambio de línea del final esto hace una trama de 94 caracteres que tardarán aproximadamente 7,2 ms en ser transmitidos.

Gumstix → Microcontrolador

Hacia el microcontrolador lo único que se transmiten son las referencias. Éstas son asíncronas y no tienen por qué transmitirse en todos los ciclos, de hecho solo se transmitirán cuando haya un cambio en las mismas.

Para la aplicación de demostración el único dato que transmitimos al robot es la referencia de velocidad. Este envío, al ser asíncrono y ser el microcontrolador el que recibe los datos, ha provocado más problemas y la necesidad de un código más robusto ante errores. Finalmente la trama que se usa es :

`r<ref_vel>f`

Siendo <ref_vel> la referencia de velocidad.

5.3.2. CORBA

La capacidad de proesamiento que se puede embarcar en un microrrobot es limitada, además no se puede tener una interacción a través de una aplicación gráfica. Esta es la razón por la que se decide que el control del robot se realice remotamente empleando software distribuido.

Con la intención de crear una plataforma abierta hemos usado CORBA como *middleware*[3], entre otras cosas porque CORBA es la única alternativa que permite realizar aplicaciones de tiempo real. De este modo, aunque en este proyecto no se van a implementar aplicaciones de tiempo real, se deja la puerta abierta para posibles aplicaciones futuras.

[3]Se conoce como middleware a los agentes software que hacen de intermediarios entre los diferentes componentes de una aplicación distribuida.

Para que sea un ordenador remoto el que realiza el control de alto nivel será la aplicación que hay en el ordenador del robot la que actúe como cliente. La aplicación sobre el PC remoto será la que haga de servidor; recibirá desde el robot las invocaciones que tendrán como parámetros de entrada el estado del robot y como parámetros de retorno las referencias que resultan del procesamiento de la información.

En la aplicación que se ha implementado existe un único método que implemente el servidor cuyo nombre es `enviarEstado` y que tiene como entrada el estado del robot y como retorno la velocidad de referencia.

Lo primero que hay que hacer para realizar una aplicación basada en CORBA es definir la interfaz mediante el lenguaje IDL[4]. En nuestro caso se ha creado el archivo `monitor.idl`:

```
struct estadoRobot {
        unsigned short id;
        short posicion;
        octet linea[64];
        unsigned short servo;
        unsigned short encoder;
};

interface Monitor {
        void enviarEstado(in estadoRobot estado, out unsigned
                                                    short ref_vel);
};
```

En él se puede ver la función `enviarEstado` y también la definición de un tipo *struct* `estadoRobot` en el que se incluye la información del estado del robot.

Este archivo se pasa por el compilador IDL y se obtienen los *client stubs* y los *server skeletons* que son las porciones de código que se incluirán en el cliente y en el servidor y que se encargan de la comunicación entre ellos.

[4]Más información sobre el lenguaje IDL en la sección 4.2.3

5.4. Microcontrolador

El microcontrolador se ha de encargar de controlar todo el hardware del robot. Aunque el microcontrolador que usamos, el ATmega128 de Atmel es un microcontrolador bastante potente, sus recursos son limitados y como muchas de las acciones a realizar se pueden hacer de varias maneras vamos a tratar de distribuir los recursos para que todo funcione en paralelo lo mejor posible.

Vamos a organizar primero todos los requisitos del hardware del robot agrupándolos por dispositivos:

Servo Para controlar el servo habrá que generar un tren de pulsos cuyo periodo total sea de entre 10 y 20 ms y cuya anchura de pulso esté entre 0,5 y 2,5 ms.

Motor El control del servo necesita únicamente un PWM *(Pulse-width modulation o modulación por ancho de pulso)*. Se trata de una señal cuya frecuencia no es determinante, su valor eficaz es el que determina la tensión media que se aplica al motor.

Encoder Para saber lo que avanza el robot se necesita una entrada en la que contar los pulsos que provienen del comparador con histéresis del encoder.

iC-LA La parte del robot más complicada de manejar es el sensor lineal. Resumiéndolo, el funcionamiento del sensor es como sigue. Al sensor hay que darle una señal de reloj, una onda cuadrada que determinará su velocidad de funcionamiento. Para que comience a funcionar hay que enviarle un pulso por otra patilla, a partir de entonces, y durante 66 ciclos, se integra la luz que recibe cada uno de los 64 fotodiodos cargándose un condensador. A partir del entonces el sensor coloca en otra patilla, y de manera secuencial, la tensión de cada uno de esos condensadores (en realidad son 66 medidas y no 64, ya que la primera y la última corresponden a la temperatura del sensor).

Esas tensiones las mediremos con el convertidor analógico digital. Lo más complicado es hacer que éste funcione de manera continua a la misma velocidad que el sensor envía las medidas. Para ello hay que determinar el tiempo que dura cada conversión y hacer que la señal de reloj del sensor tenga la misma frecuencia.

El convertidor analógico digital del microcontrolador tiene 10 bits de resolución, pero para detectar la línea negra no necesitamos tanta resolución y nos quedaremos con los 8 bits más significativos. De esta forma la conversión puede ser más rápida, de tan solo 13 µs por conversión. Esto fija la señal del reloj, que tendrá un periodo de 13 µs.

Un potenciómetro Cuando no se disponga de una velocidad de referencia proporcionada por el Gumstix o el PC remoto se usará un potenciómetro para indicar la velocidad de referencia del motor. Para poder medir la tensión en la patilla central del potenciómetro hay que conectarla a una entrada del convertidor analógico digital.

Para no usar más interrupciones de las imprescindibles vamos a intentar contar los pulsos del encoder y sacar todos los trenes de pulsos utilizando los contadores y temporizadores que tiene integrado el microcontrolador.

Tanto la señal del servo como la del motor pueden tener la misma frecuencia. Vamos a utilizar para ambos una señal de 10 ms de periodo. Generaremos ambas señales con el contador/temporizador 3, que es un contador de 16 bits, configurándolo con esos 10 ms de periodo. Para ello, teniendo en cuenta que el microcontrolador realiza 16 millones de ciclos por segundo, configuraremos dicho contador con un preescalado de 8 (el contador se incrementará cada 8 ciclos) y pondremos el valor de 20.000 como valor que haga que el contador vuelva a 0. Es decir, el contador contará constantemente de 0 a 20.000 tardando 10 ms en cada ciclo.

Hay tres comparadores asociados al temporizador 3, usaremos dos de ellos en el modo *inverted Fast PWM* en el que el pin de salida de cada uno de ellos se pone a '1' cuando se reinicia el contador y a '0' cuando el valor del contador y el valor de referencia del comparador coinciden. Es decir, dando al motor el valor 15.000 aplicaremos al motor el 75 % de la tensión de las baterías y dando al servo el valor de 2.600 se generarán pulsos de 1,3 ms.

La señal de reloj del sensor lineal la generaremos con el contador 2, que es de solo 8 bits. Lo usaremos también en modo comparador haciendo que la salida asociada a él cambie de estado cada 104 ciclos, es decir, cada 6,5 ms. De este modo creamos una señal cuadrada de 13 ms exactos de periodo.

Para el encoder usaremos el contador 1 (otro de 16 bits), pero en vez de configurarlo para que se incremente con los ciclos del micro, lo configuramos para que cuente los flancos de bajada de la señal que viene del comparador con histéresis. Recordando que el disco del encoder tiene 36 franjas blancas

y 36 negras y que la longitud de la rueda es de 18 cm, cada incremento del contador supondrá que el robot ha avanzado 5 mm.

Para las entradas analógicas simplemente conectamos la salida analógica del sensor y el cursor del potenciómetro de la referencia de velocidad a 2 de las 8 entradas que tiene el convertidor analógico digital. Para la referencia de velocidad leeremos la medida de manera "normal" mientras que para leer las tensiones de cada punto del sensor lineal usaremos el convertidor en modo continuo.

Por último, para la señal que indicará al sensor lineal que ha de arrancar usaremos una salida digital cualquiera.

5.4.1. Desarrollo del programa

El proceso que describe el programa, a grandes rasgos, es muy sencillo. Primero inicializa los periféricos y luego realiza indefinidamente un ciclo que incluye: lectura de todos los sensores; cálculo de la posición, velocidad y valores para el servo y el motor; envío de las señales a los actadores y envío de todos los datos por el puerto de serie. Este proceso está más detallado en el flujograma de la figura 5.3

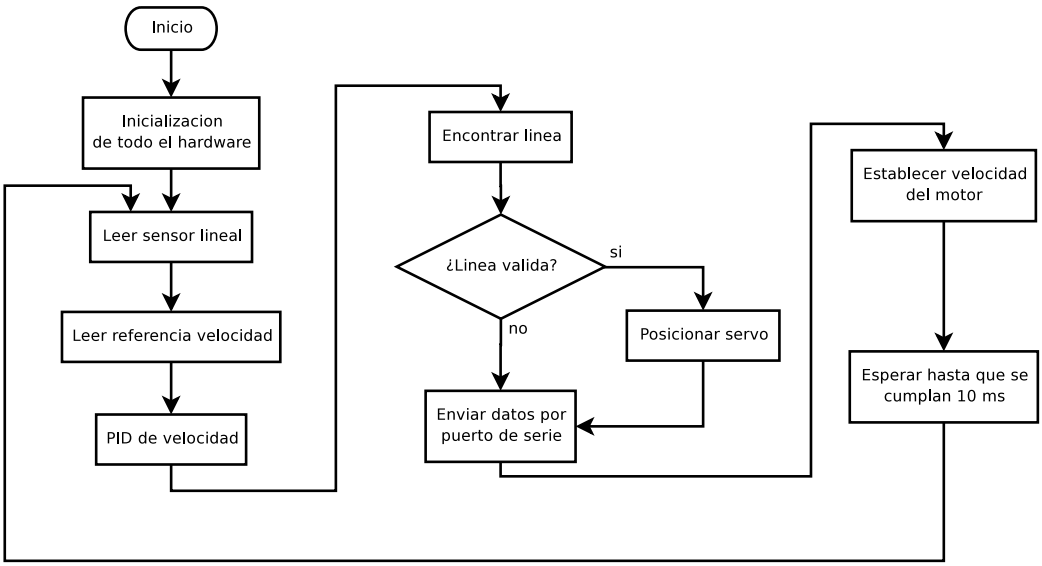

Figura 5.3: Flujograma del programa del microcontrolador

Se comprueba que la duración del ciclo que recorre indefinidamente el

programa es inferior a 10 ms, por tanto se decide usar el mismo contador de 10 ms que genera las señales del servo y del motor para hacer que el ciclo dure exactamente 10 ms. Como consecuencia de esto se tomarán muestras del estado del robot con un intervalo fijo 100 veces por segundo.

Además, de manera asíncrona, está preparado para recibir por el puerto de serie una nueva referencia de velocidad. Para esto se ha configurado la interrupción que salta cuando ser recibe un caracter por el puerto de serie, y se han habilitado las interrupciones de manera local al inicio del programa.

Las funciones que se realizan en la mayoría de los bloques del diagrama están ya descritas o no tienen dificultad, pero hay alguna de ellas que se va a explicar con mayor detalle.

5.4.2. Control de velocidad

Como ya se ha comentado anteriormente, la relación de marchas del robot es muy larga y, por consecuencia, la velocidad máxima también. Pero aunque en la prueba de velocistas se requiere mucha velocidad, la velocidad máxima del robot es muy superior a la que se necesita. Como la velocidad del motor en vacío es proporcional a la tensión de alimentación, si queremos una velocidad reducida, tenemos que aplicar una tensión muy baja, con lo que el par es muy reducido y el robot tarda mucho en alcanzar la velocidad de referencia. Además, de esta manera tendríamos muchos problemas con el paso elevado que hay en algunos circuitos de velocistas; no tendríamos par para subir la cuesta arriba y el coche se dispararía en la cuesta abajo.

Por todo esto se decide realizar un control de la velocidad del robot. Lo primero que hay que hacer es identificar el sistema, para lo cual la primera opción es calcularlo matemáticamente. Este intento no es fructífero, porque aunque es posible obtener un modelo del comportamiento del motor, es mucho más complicado modelar las resistencias y las inercias del robot.

Como tenemos acceso total al sistema, decidimos identificarlo de manera experimental. Para ello introducimos una secuencia de valores en la entrada del sistema (que es el ciclo de carga del tren de pulsos que va a la etapa de potencia) y registramos los valores de velocidad medidos con el encoder. Tomamos varias series de medidas como ésta.

En este punto hay que tener en cuenta que con el ciclo del programa del microcontrolador en 10 ms, en cada ciclo el encoder apenas genera 2 o 3 pulsos cuando la velocidad del robot está cerca de la que se quiere para el

robot, 1 a 1,5 m/s. Con el objetivo de suavizar un poco los datos de velocidad y de redondearlos un poco, decidimos sumar el avance del encoder en los 5 últimos ciclos. De este modo la medida de la velocidad se realiza en decímetros por segundo y la resolución que tenemos es mayor ya que trabajaremos con valores de entre 10 y 15.

Con los datos que hemos registrado de entrada y salida del sistema arrancamos la herramienta de identificación de sistemas de Matlab. Usando unos datos para identificar y otros para comprobar terminamos obteniendo un sistema de segundo orden cuyo comportamiento se parece razonablemente al del sistema real. Este modelo del sistema será el que usaremos para diseñar un regulador y simularlo.

Usando las tablas de Ziegler-Nichols[5] obtenemos los valores de las constantes de un regulador PID. Usando la herramienta de simulación de Matlab, simulink, y el modelo del sistema que hemos obtenido previamente, probamos el regulador y lo modificamos hasta conseguir un sistema realimentado suficientemente rápido y sin apenas sobreoscilación.

Para implementar el regulador lo hemos discretizado y hemos obtenido una ecuación en diferencias. Usando esta ecuación el microcontrolador calcula el nuevo valor que ha de enviar al motor.

En la gráfica 5.4 se pueden ver los datos de velocidad del robot frente a una secuencia de valores de la velocidad de referencia. Se puede ver que el sistema de velocidad es ahora mucho más rápido y que en régimen permanente el robot sigue muy de cerca la referencia. La velocidad está suavizada para que se aprecie mejor.

5.4.3. Detección de la línea

Las medidas de los 64 valores del sensor lineal no son demasiado buenas. Para empezar, al haber sustituido la óptica por un pequeño orificio, la línea se ve "borrosa", cada pixel está apuntando a una zona que se superpone con la zona del de al lado y esto provoca que los bordes de la línea no se vean nítidos.

Además, los píxeles de los extremos reciben bastante menos luz que los del centro, por tanto no podemos asignar un umbral para determinar qué píxeles están viendo una zona blanca y qué píxeles están viendo una zona negra.

[5]Las tablas de Ziegler-Nichos son unas tablas experimentales que sirven para dar valores a las constantes de un regulador P, PI o PID

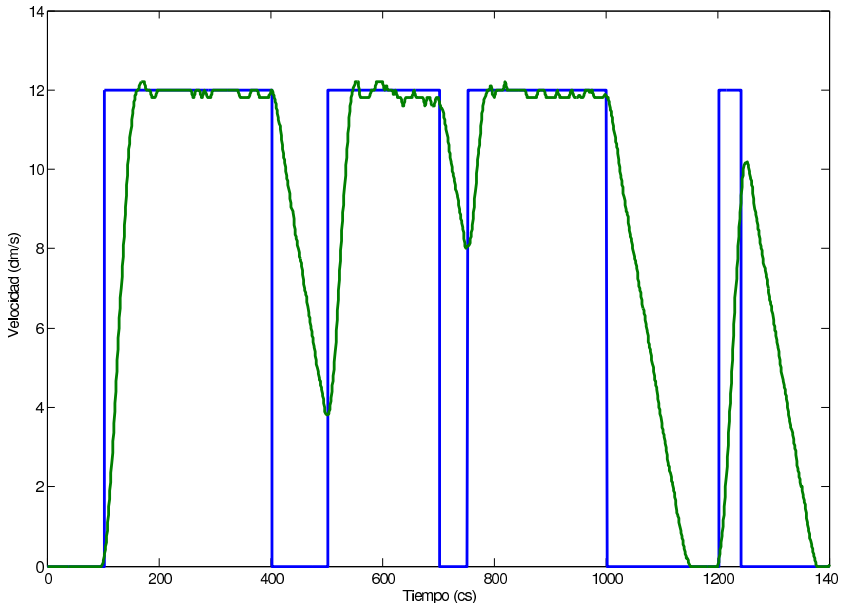

Figura 5.4: Datos de velocidad usando el regulador PID

Pero la línea se puede localizar buscando sus bordes. Hay que buscar un cambio de blanco a negro y uno de negro a blanco a una distancia determinada. El algoritmo utilizado para encontrarlos es muy sencillo, como la línea está algo difuminada los saltos entre cada pixel y su consecutivo no son tan grandes, pero si miramos los incrementos entre dos píxeles con otro entre medias, el salto es bastante grande en los bordes de la línea. Así que buscamos el máximo escalón de bajada y de subida entre píxeles dejando uno entre medias y luego comprobamos que el máximo escalón de subida esté a la derecha del de bajada y a una distancia equivalente a la anchura de la línea. Si se dan estas condiciones podemos garantizar que la línea está entre ambos escalones.

5.5. Gumstix: CORBA client

La aplicación que se ejecuta sobre la plataforma Gumstix es la más sencilla de las tres. Tiene únicamente dos tareas que realizar. Por un lado ha de leer del puerto de serie los datos del estado del robot enviados por el microcontrolador que se encuentra en la placa Robostix. Además tendrá que enviar al microcontrolador, también por el puerto de serie las nuevas referencias de

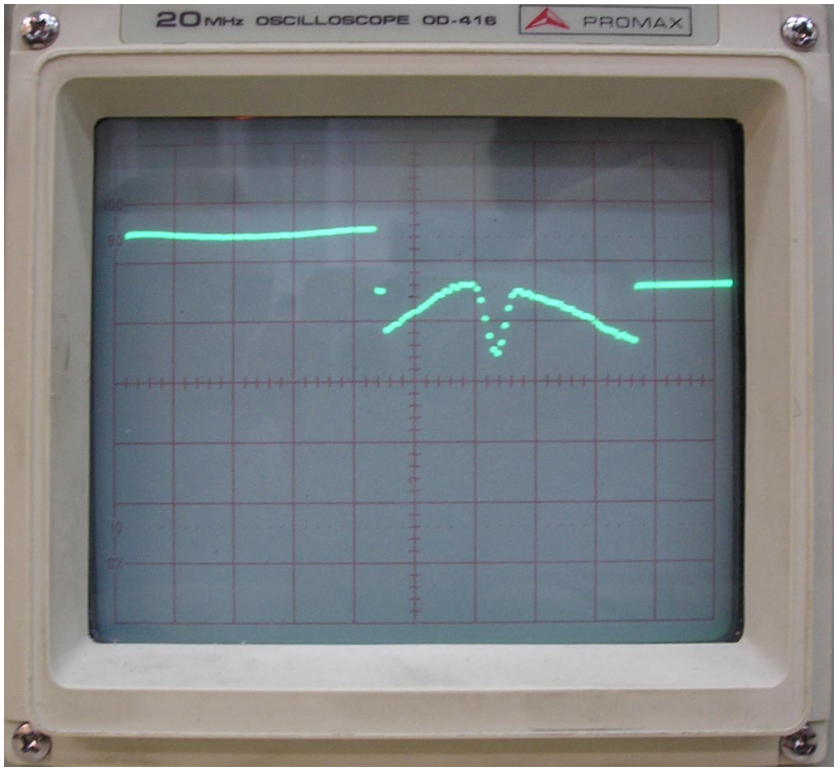

Figura 5.5: Aspecto de las 64 valores analógicos a la salida del sensor lineal

velocidad (en la aplicación que se ha implementado la referencia de velocidad es el único parámetro que se transmite, pero se puede hacer igual para otros valores). Por otro lado la aplicación del Gumstix ha de ser el cliente de CORBA que realizará con cada conjunto de datos las llamadas oportunas a las funciones implementadas en el servidor CORBA.

Adicionalmente dicha aplicación se puede encargar también de registrar todos los datos del estado del robot en un fichero. Ésto es lo que se ha hecho en varias aplicaciones previas al funcionamiento de la conexión inalámbrica para tomar valores del comportamiento del robot.

Todo el desarrollo de los programas para la plataforma Gumstix se ha realizado en lenguaje C++ ya que el broker CORBA utilizado, MICO, solo soporta este lenguaje.

Para la comunicación por el puerto serie se ha creado una clase llamada *puerto_serie* que, basándose en la librería *termios*, se encarga de toda la comunicación por el puerto serie haciendo que el programa principal pueda

escribir y leer de dicho puerto como si de un fichero cualquiera se tratara.

Para manejar el estado del robot se ha creado la clase *estado_robot* con todas las funciones necesarias para obtener los datos del estado del robot desde el puerto serie, analizarlos y prepararlos para realizar las llamadas al servidor CORBA con ellos.

La aplicación principal se encargará, no solo de instanciar y manejar los objetos de las clases arriba descritas, sino también de inicializar el ORB[6]. A continuación obtendrá la referencia al objeto CORBA que hace de servidor a partir del IOR[7], que se lo pasamos como argumento al ejecutar la aplicación.

A partir de este momento la aplicación se dedicará a leer constantemente el estado del robot desde el puerto serie, interpretar esos datos y realizar una llamada al servidor CORBA. En caso de que la referencia de velocidad que se obtiene del servidor sea diferente de la que tiene el robot, se la transmitirá vía serie.

Para que la aplicación sea lo más independiente posible, ésta se compila usando las librerías estáticas del ORB, para que la implementación del mismo quede dentro del ejecutable y no tenga que cargar librerías de manera dinámica. Para que no se incluyan en el ejecutable las librerías completas, sino solamente la parte que se usa, éstas habrán tenido que se compiladas (al igual que la aplicación final) usando las opciones del compilador g++ *-ffunction-sections -fdata-sections*. D este modo se consigue que la unidad en la que quedan estructuradas las librerías sea cada función, de manera que solo se incluyen las funciones que sí que se usan.

Como la memoria del Gumstix es de tan solo 16 Mb tras generar el ejecutable (de nombre *serie_rxtx*) pasamos éste por la aplicación *strip*. Strip es una *binutil* que se encarga de eliminar del ejecutable los símbolos, de manera que se reduce bastante el tamaño de los ficheros.

Makefile

Para realizar una compilación de todos los archivos que componen la aplicación (y todas las aplicaciones que se han desarrollado en el proyecto)

[6]El ORB, Object Request Broker, es el componente que se encargará de la función de *middleware* o intermediario en la comunicación entre el cliente y el servidor CORBA

[7]El IOR o *Interoperable Object Reference* es un identificador global único para un objeto CORBA, si se conoce el IOR de un objeto éste puede ser invocado sin tener que saber en qué máquina se encuentra o cómo funciona

se va a usar la herramienta *make*.

Make es una herramienta de generación de código, muy usada en los sistemas operativos tipo GNU/Linux. Cuando es ejecutado éste busca un archivo de nombre *Makefile* en el que se encuentra toda la información necesaria para compilar la aplicación. La parte más importante del documento es lo que se conoce como dependencias, en la que se describen las relaciones entre los ficheros de código de manera que cuando uno de ellos cambia, make sabrá exactamente cuales son los archivos que tiene que recompilar y cuales no es necesario.

Make también es muy usado para la limpieza de los archivos temporales en la compilación.

El fichero *Makefile* correspondiente a la aplicación del Gumstix es el siguiente:

```
TARGET=serie_rxtx

MICO_ARM=/opt2/mico_arm
CXX=arm-linux-g++
CXXFLAGS=-c -Os -I. -I$(MICO_ARM)/include -ffunction-sections
                                                    -fdata-sections
LD=arm-linux-g++
LDFLAGS=-Wl,--gc-sections
MICO_STATIC_LIB=$(MICO_ARM)/lib/libmico2.3.12RC2.a
LDLIBS=-lm -ldl -lpthread
STRIP=arm-linux-strip
IDL=idl

all: $(TARGET)

$(TARGET): serie.o estado_robot.o puerto_serie.o monitor.o
$(LD) $(LDFLAGS) serie.o estado_robot.o puerto_serie.o
      monitor.o $(MICO_STATIC_LIB) $(LDLIBS) -o $@; $(STRIP) $@

serie.o: serie.cpp estado_robot.h monitor.h
$(CXX) $(CXXFLAGS) serie.cpp

estado_robot.o: estado_robot.cpp estado_robot.h monitor.h
$(CXX) $(CXXFLAGS) estado_robot.cpp
```

```
puerto_serie.o: puerto_serie.cpp puerto_serie.h
$(CXX) $(CXXFLAGS) puerto_serie.cpp

monitor.h monitor.cc: monitor.idl
$(IDL) monitor.idl

monitor.o: monitor.cc monitor.h
$(CXX) $(CXXFLAGS) monitor.cc

clean:
rm *.o $(TARGET) monitor.h monitor.cc
```

5.6. PC: CORBA servant e interfaz gráfica

Será el PC remoto el que se encargue del control de alto nivel del robot. Para ello tendrá que implemetar un servidor CORBA y una aplicación gráfica que permita observar la evolución del estado.

5.6.1. Aplicación gráfica

Para la interfaz gráfica se ha optado por usar las librerías Qt, empleando la herramienta *Designer* para componer gráficamente dicha interfaz.

La interfaz se compone de múltiples *widgets* o pequeños elementos como cuadros de texto, barras de desplaziemto o indicadores. Hay multitud de *widgets* estandar, pero para nuestra aplicación se han definido dos nuevos. Uno se llama *LineaField* y será una imagen en la que la línea superior será siempre la última línea que ha captado el sensor lineal y el resto de la imagen se forma desplazando las líneas anteriores, con lo que siempre se ven las últimas 64 líneas recibidas.

El otro *widget* que se ha definido es *TrayectField*, se trata de otro widget gráfico en el que se va a representar la trayectoria que recorre el robot.

Con la herramienta *Designer* se ha creado la interfaz integrando los diferentes widgets. La conexión entre los mismos se basa en un mecanismo de *signals* y *slots*. Las *signals* son señales que se activan cuando ocurre algo en el widget como un cambio de un valor, o un click de ratón; mientras que los

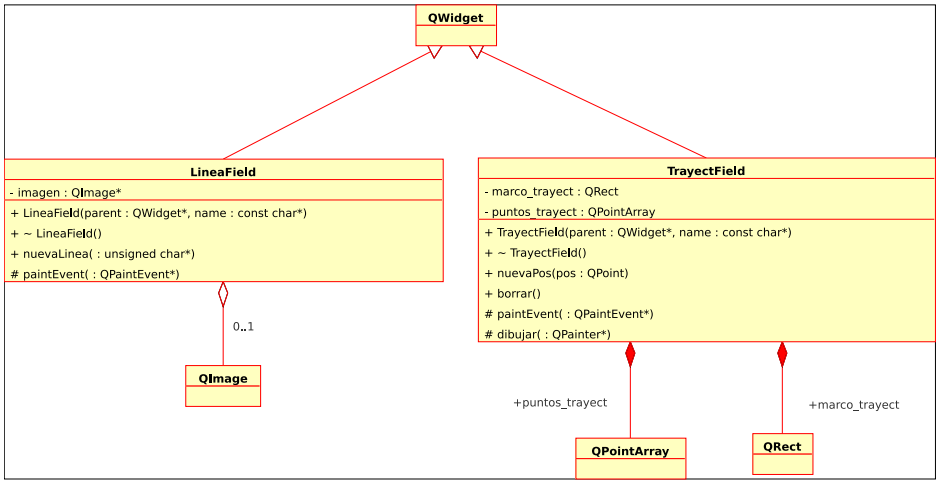

Figura 5.6: Diagrama de clases de los widgets implementados

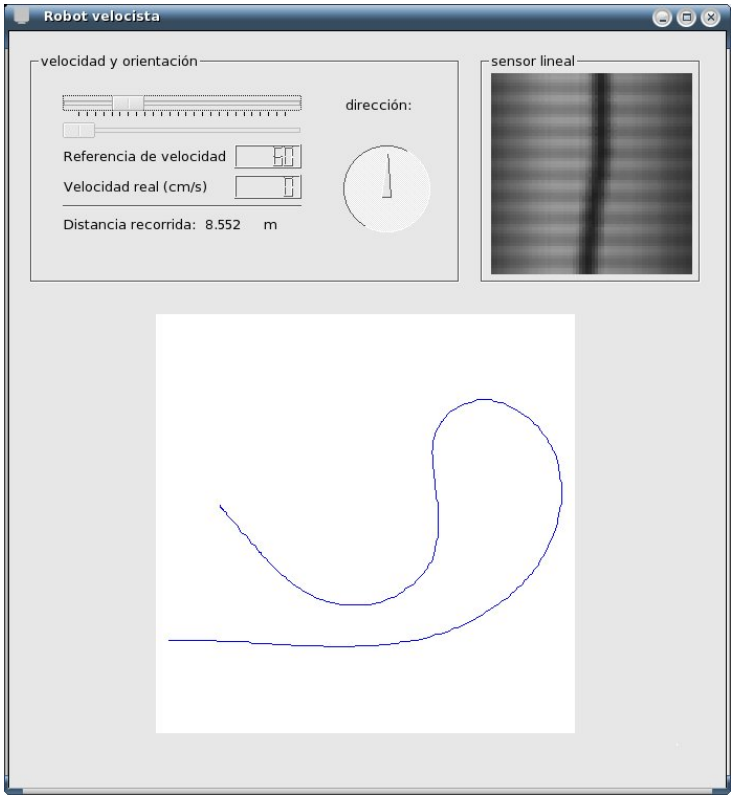

Figura 5.7: Aspecto de la interfaz gráfica

slots son las acciones predefinidas que tienen dichos *widgets* (como mostrar un cierto valor). Conectando una determinada *signal* de un *widget* con un *slot* se puede conseguir, por ejemplo, que al mover un deslizador se muestre en un *display* su valor. De esta manera, dentro de la propia herramienta resulta muy sencillo comunicar unos elementos de la interfaz con otros.

Pero conseguir que los sucesos en los *widgets* provoquen la ejecución de código externo a la aplicación gráfica es algo más complicado. Es necesario definir nuevos *slots* virtuales cuya funcionalidad se implementará en una clase derivada. El resultado del trabajo con *Designer* es un archivo de extensión .ui, `robot_formbase.ui`. Este archivo hay que pasarlo por el compilador UI que generará los archivos `robot_formbase.h` y `robot_formbase.cpp` con la definición de la clase *robot_formbase*. De esta última derivará la clase *robot_form* en la que se encuentran las funciones que transmitirán al servidor de CORBA lo que el usuario haya cambiado en la aplicación gráfica.

A continuación hay que pasar todos los archivos de extensión .h por el compilador MOC *(Meta Object Compiler)* que creará código para adaptar a C++ las *signals* y los *slots* que se han empleado. Por último habrá que compilar todos los ficheros de código (incluidos los generados por el compilador MOC) y enlazar todos los ficheros objeto.

5.6.2. CORBA servant

La otra parte de la aplicación que se ejecuta en el PC remoto es el servidor CORBA. Para ello se ha creado la clase *Monitor_impl* que hereda de *POA_Monitor*. La clase *POA_Monitor* forma parte de lo que se conoce como *server skeletons*, es decir, es código que crea el compilador IDL a partir de la definición de la interfaz.

La clase *Monitor_impl* es la que implementa la funcionalidad del servidor, por lo tanto en ella habrá que definir la función *enviarEstado* que está descrita en el archivo de interfaz `monitor.idl`. El encabezado queda:

```
void Monitor_impl::enviarEstado(const estadoRobot&, MICO_UShort&);
```

Es en esta función donde se calcula la distancia recorrida, la posición del robot y su trayectoria; además es en esta función donde, cada 5 veces que es llamada, se mandan las señales necesarias para que se actualicen los indicadores de la aplicación gráfica con los valores medios del estado del robot en esos 5 ciclos. No se realiza con todas las llamadas porque no es necesario actualizar tan rápidamente la visualización y supondría un cálculo

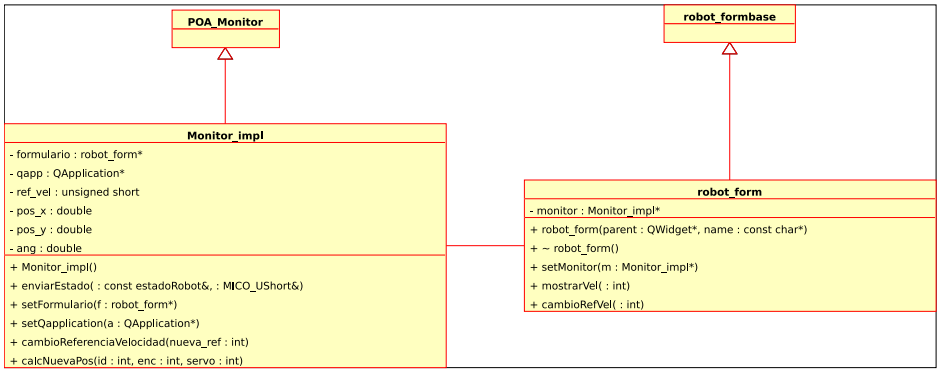

Figura 5.8: Diagrama de clases del servidor CORBA

innecesario.

5.6.3. Aplicación principal

La aplicación principal únicamente se ha de encargar de iniciar el servidor CORBA y la interfáz gráfica. Para eso primero inicializa el ORB y una variable QApplication que es la aplicación gráfica. A continuación instancia un objeto de la clase Monitor_impl y uno de la clase robot_form. Cada uno de estos objectos tiene un atributo que es un puntero al otro de manera que se pueden pasar los datos.

El mayor inconveniente que plantea la integración de ambas aplicaciones es que ambas requieren el control del hilo de ejecución para su funcionamiento, por eso es necesario crear un segundo hilo o *thread* de manera que cada uno de los hilos quede indefinidamente en la QApplication de Qt o en el CORBA servant.

Ejecución

El resultado de la compilación es un ejecutable de nombre *monitor*. Para lanzarlo solo habrá que ejecutar `./monitor` en una consola y se abrirá la ventana gráfica. Pero además, al ejecutarlo imprimirá en la consola el IOR (la referencia del CORBA servant) que tendrá un aspecto como:

```
IOR:010000001000000049444c3a4d6f6e69746f723a312e300002000000000
000002f000000010100000b000000706f686c2e61736c616200008382000013
```

```
0000002f363535372f313134333734303232372f5f3000010000002400000000
1000000010000000100000014000000010000000100010000000000090101000
00000000
```

Ahora solo hay que copiarlo y pegarlo como argumento al ejecutar la aplicación cliente que corre sobre el Gumstix. A partir de ese momento, todos los datos del estado del robot que esté proporcionando el microcontrolador se transmitirán al PC a través de la invocación remota de la aplicación *enviarEstado*. Y la llamada a dicha función también provocará que la referencia de velocidad seleccionada en la interfaz gráfica se envíe al microcontrolador.

5.7. Pruebas con CORBA

Con la aplicación ya completa, se prueba toda en conjunto, desde el microcontrolador hasta el PC remoto con el servidor CORBA. Como todas las partes se había probado ya de manera individual o unas con otras la aplicación completa funciona sin problemas.

Pero ahora, de cara a una posible ampliación de la aplicación, o pensando en emplear esta plataforma sobre otros robots se plantea la duda de cuántas peticiones al servidor CORBA se podrían realizar.

Para eso se decide realizar mediciones de tiempos de lo que dura la invocación de un método remoto, que incluirá tanto las conversiones de los datos, el envío de los mismos, ejecución del método remoto y vuelta de los datos. Para que estos datos sean más reales se han tomado en la aplicación real. Más concretamente se han implementado en el cliente CORBA que se ejecuta sobre el Gumstix midiendo el tiempo que tarda en ejecutarse el método remoto *enviarEstado*.

El aspecto de los datos obtenidos puede observarse en la figura 5.9. En la figura se puede observar como la mayoría de las llamadas es resuelta en menos de 8 ms pero que la duración de las mismas se alarga, en algunos casos hasta cerca de 40 ms.

Para ver más claramente la duración de las invocaciones se ha realizado también la figura 5.10 en la que se ve la probabilidad que tiene una llamada de tardar un cierto tiempo.

Concretamente la duración media de dichas llamadas es de 8,225 ms con una desviación típica de 3,25 ms. Además el 75 % de las llamadas ha durado menos de 9 ms.

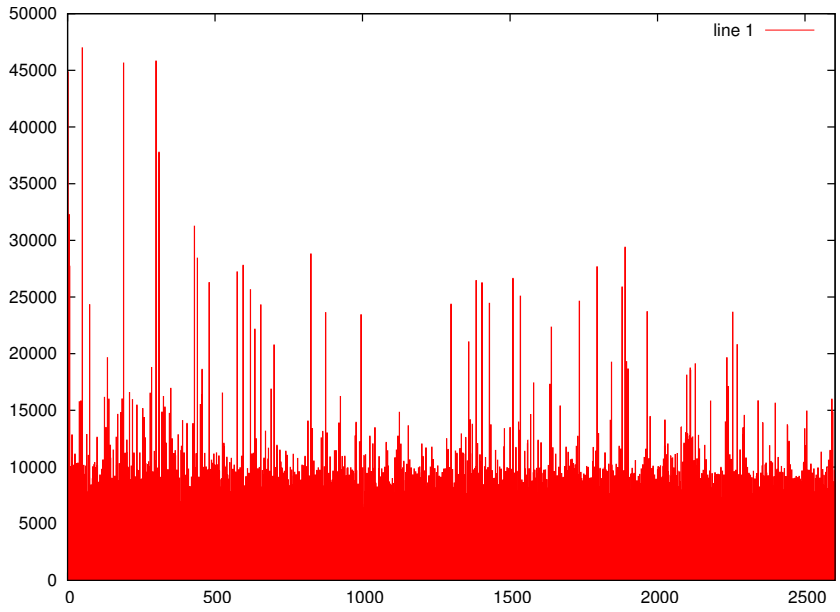

Figura 5.9: Tiempos de las llamadas a un método remoto.

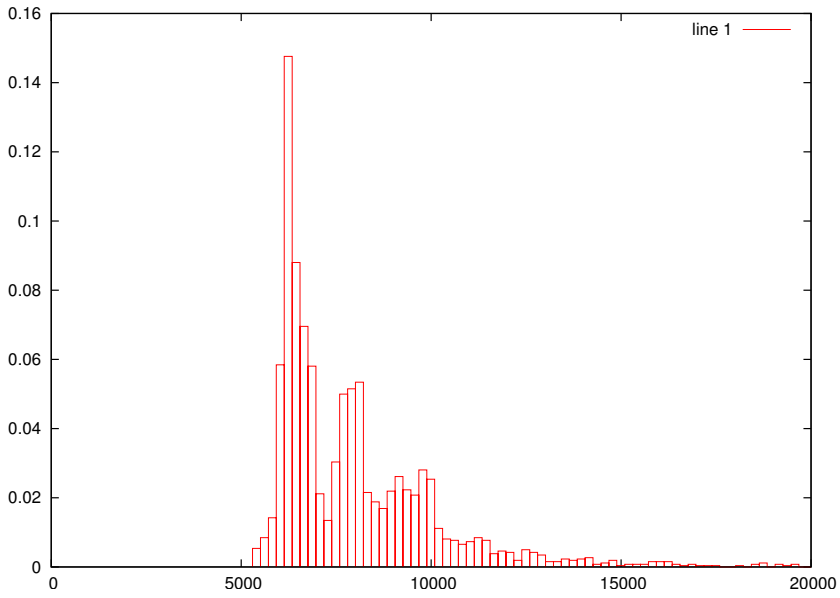

Figura 5.10: Histograma de duración de las llamadas a métodos remotos.

Capítulo 6

Conclusiones y líneas futuras

Para finalizar con la memoria de este proyecto fin de carrera titulado **"Robot Velocista de Competición basado en CORBA"** en este capítulo se describen las conclusiones que se sacan del trabajo realizado así como posibles líneas de investigación que podrían seguirse en el futuro y revertirían en mejoras para el mismo.

En la Sección 6.1 se recuerdan los objetivos, el trabajo realizado así como las conclusiones que se sacan del proyecto que se describe en esta memoria.

En la Sección 6.2 se describen las líneas de trabajo que quedan abiertas tras la conclusión de este proyecto que supondrían mejoras de la aplicación desarrollada.

6.1. Conclusiones

El objetivo de este proyecto es dotar a un robot movil, de los que se usan en competiciones de microrrobots, de lo necesario para poder realizar sobre el mismo un control de alto nivel de manera remota a través de CORBA. Para cumplirlo se han realizado una serie de tareas que se pueden resumir en estas:

- Se ha contruido un robot velocista controlado por un microcontrolador que es capáz de competir contra otros de similares características. Como innovación más importante en este aspecto se ha empleado un sensor lineal que resulta un gran avance respecto al uso habitual de sensores de infrarrojos CNY70.

- Ha sido posible la instalación en el microrrobot de un computador empotrado que se puede conectar a cualquier red inalámbrica IEEE 802.11b sin que esto suponga un problema físicamente ni en cuestión de consumo.
- Se ha compilado un ORB para el computador empotrado y se han implementado las aplicaciones que invocan, desde el robot, métodos remotos que se ejecutan en otro ordenador.
- Por último se ha creado una aplicación que se ejecuta en un PC convencional y que implementa dichos métodos remotos y una interfaz gráfica que permite la interacción del usuario con el robot.

El resultado concreto del proyecto es, por un lado un microrrobot velocista que se puede controlar de manera remota mediante una aplicación gráfica que también permite observar el estado del mismo; y por otro lado se tiene una plataforma que se puede incorporar a cualquier robot móvil y que permite una elevada capacidad de procesamiento y ejecución de aplicaciones remotas a través de CORBA.

6.2. Líneas futuras

Tras comprobar que el sistema utilizado en el proyecto es viable para controlar un velocista mediante CORBA se puede pensar en controlar cualquier robot móvil de la misma manera.

Directamente a partir de este punto se pueden abordar varios proyectos que se apoyen en lo que se ha conseguido con éste:

- La continuación natural del proyecto sería implementar un verdadero control de alto nivel que pueda dotar al velocista de una ventaja real respecto al estado actual. La aplicación remota que se ha implementado solo incluye una visualización del estado del robot y una intervención muy básica sobre el comportamiento del mismo.

  Pero el sistema está preparado para que remotamente se realize mucho más procesamiento. Por ejemplo es posible un aprendizaje del circuito que el robot está recorriendo para identificar tramos en los que se pueda desarrollar mayor velocidad, implementar rutinas más robustas que

permitan detectar condiciones de fallo de los sensores, patinaje del robot o errores en cualquier otro componente, realizar un ajuste en línea de los parámetros de los controladores ...

- El otro camino que se abre con este proyecto es el de apovechar la conclusión satisfactaria que se ha conseguido con el comutador integrado Gumstix y la conexión WiFi para aplicarlo a cualquier robot móvil. Por sus características es una plataforma apta para ser incorporada en varios robots y conseguir una colaboración entre ellos.

  El desarrollo de robots colaborativos es un tema en el que se está trabajando mucho actualmente para taras muy diversas y que va a tener un desarrollo muy importante durante los próximos años.

Apéndice A

Programación del proyecto

Al comienzo del proyecto se realizó una programación aproximada. Durante la realización del mismo no se ha llevado estrictamente esa programación. Pero ha sido muy útil de cara a redistribuciones del tiempo en revisiones intermedias del avance del proyecto.

Esta programación se ha plasmado en un diagrama de Gantt que se puede observar en la figura A.1. Las tareas concretas que aparecen en el diagrama han variado ligeramente de las que se propusieron en un principio, debido a las elecciones que se tomaron. Pero básicamente siguen el proceso que ya se ha comentado de desarrollo del proyecto.

En una primera fase se encuentran las actividades relacionadas con la construcción del robot velocista. Éstas comienzan con un periodo de formación y documentación, compras, desarrollo de aplicaciones, integración y pruebas.

En la siguiente fase incluye la selección y uso del computador empotrado que se ha usado en el proyecto, así como la formación en lo que a CORBA se refiere y la implementación de las aplicaciones cliente, servidor e interfaz gráfica.

Por último se realizan las pruebas de integración de todas las aplicaciones y se deja tiempo para redactar la memoria del documento.

Id	Nombre de tarea	Duración	Comienzo	Fin	Predeceso
1	Formación en microrrobots	2 mss	mar 01/02/05	lun 28/03/05	
2	Formación en microcontroladores	6 sem.	mar 29/03/05	lun 09/05/05	1
3	Selección de componentes	3 sem.	mar 10/05/05	lun 30/05/05	2
4	Compra de componentes	5 sem.	mar 31/05/05	lun 04/07/05	3
5	Pruebas con el microcontrolador	1 ms	mar 05/07/05	lun 01/08/05	4
6	Montaje del sensor lineal	2 sem.	mar 05/07/05	lun 18/07/05	4
7	Manejo sensor lineal	1 ms	mar 02/08/05	lun 29/08/05	5;6
8	Montaje servo	1 sem	mar 05/07/05	lun 11/07/05	4
9	Montaje encoder	2 sem.	mar 05/07/05	lun 18/07/05	4
10	Diseño placa integración componentes	2 sem.	mar 05/07/05	lun 18/07/05	4
11	Montaje placa integración componentes	1 sem	mar 30/08/05	lun 05/09/05	10;9;8;7
12	Pruebas robot inicial	3 sem.	mar 06/09/05	lun 26/09/05	11
13	Formación en CORBA	2 sem.	mar 27/09/05	lun 10/10/05	12
14	Pruebas con CORBA	3 sem.	mar 11/10/05	lun 31/10/05	13
15	Selección computador integrado	1 ms	mar 27/09/05	lun 24/10/05	12
16	Compra computador empotrado	2 sem.	mar 25/10/05	lun 07/11/05	15
17	Integración computador-microcontrolador	1 ms	mar 08/11/05	lun 05/12/05	16
18	Diseño apps cliente y servidor CORBA	1 sem	mar 01/11/05	lun 07/11/05	14
19	Implementación aplicaciones CORBA	2 sem.	mar 08/11/05	lun 21/11/05	18
20	Formación Qt	1 sem	mar 27/09/05	lun 03/10/05	12
21	Realización aplicación gráfica	2 sem.	mar 22/11/05	lun 05/12/05	20;19
22	Pruebas sistema completo	1 ms	mar 06/12/05	lun 02/01/06	17;19;21
23	Redacción memoria	1 ms	mar 03/01/06	lun 30/01/06	22

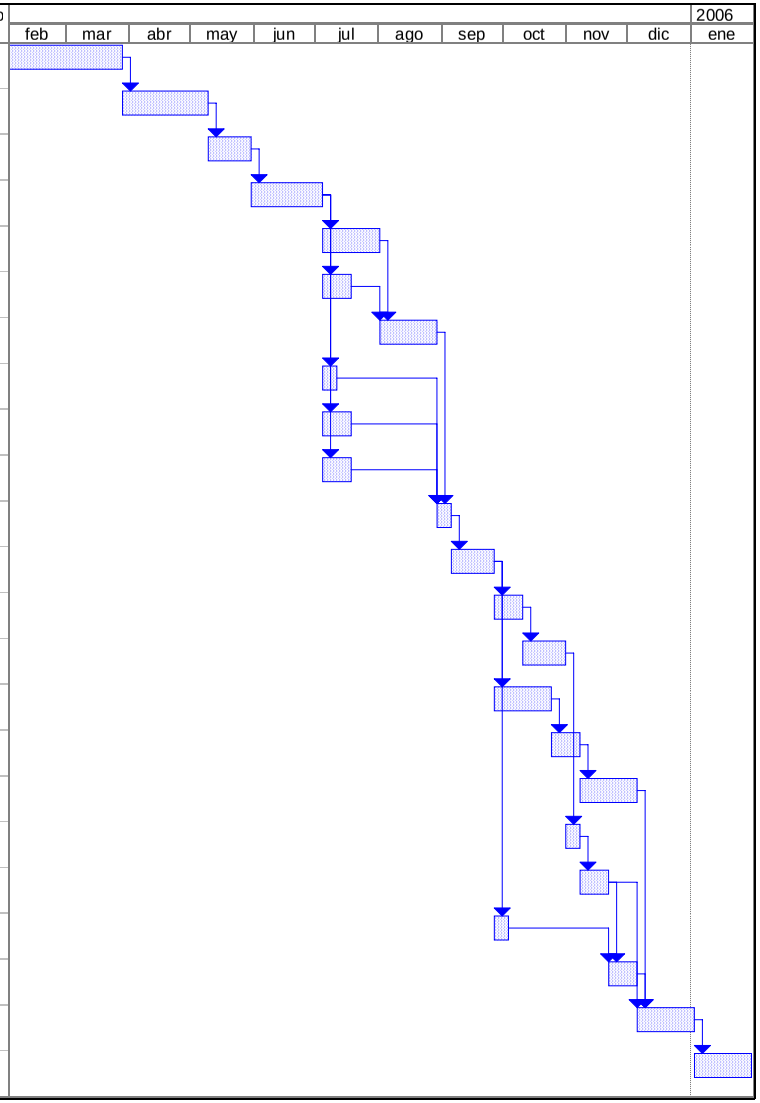

Figura A.1: Diagrama de Gantt del proyecto

Apéndice B

Otras herramientas

Además de las herramientas descritas en el Capítulo 4, en la realización de este proyecto y su correspondiente memoria se han utilizado otras. A continuación se muestra una lista de las mismas así como en qué se empleó cada una de ellas.

LaTeX: es un conjunto de macros de TeX. La idea principal de LaTeX es ayudar a quien escribe un documento a centrarse en el contenido más que en la forma. La calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea. LaTeX es Software Libre bajo licencia LPPL. Se ha utilizado para escribir esta memoria que recoge el trabajo realizado en el proyecto.

Kile: es un editor sencillo para TeX y LaTeX. Se ha utilizado para generar esta memoria. Se utilizan las funciones integradas de las que dispone, que hacen que la creación del documento sea más sencilla (autocompleta comandos TeX/ LaTeX posibilita generar y ver el documento con un solo click, posee un sistema de corrección ortográfica, tiene un asistente para introducir la bibliografía ...).

GIMP(GNU Image Manipulation Program): es un programa de manipulación de imágenes del proyecto GNU. Es la alternativa más firme del software libre al popular programa de retoque fotográfico Photoshop. En este proyecto se ha utilizado para hacer capturas de algunas de las aplicaciones empleadas así como para el retoque de otras de las imágenes que aparecen en este documento.

Open Office.org: es un proyecto basado en el código abierto para crear una

suite ofimática. Es bastante compatible con los formatos de fichero de Microsoft Office, ya que puede leer directamente los archivos creados con dicha *suite* ofimática, aunque tiene su propio formato de archivos basado en el estándar XML. Se ha utilizado el módulo "Calc"(hoja de cálculo) en la manipulación de datos provenientes del robot.

Matlab: es un programa interactivo para computación numérica y visualización de datos. Es ampliamente usado por Ingenieros de Control en el análisis y diseño, posee además una extraordinaria versatilidad y capacidad para resolver problemas en matemática aplicada, física, química, ingeniería, finanzas y muchas otras aplicaciones. Se ha empleado la herramienta (o toolbox) de identificación de sitemas para idenntificar el sistema de velocidad y la herramienta de simulación, Simulink, para simular el regulador antes de implementarlo.

Xfig: es una completa herramienta de dibujo, con ella podemos realizar cualquier tipo de dibujo vectorial en Linux: gráficos, planos, esquemas, bocetos, diagramas de flujo .... Se ha usado para realizar varios de los diagramas y dibujos vectoriales que hay en esta memoria.

Umbrello UML Modeller: es un programa que permite crear diagramas UML de software o de otros sistemas en un formato estándar. Permite la generación de código para multitud de lenguajes a partir de los diagramas y también el trazado de los mismos a partir del código. Varios de los diagramas UML realizados durante el proyecto se han realizado con Umbrello.

Subversion: es un software de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Es software libre y se lo conoce también como svn por ser ese el nombre de la herramienta de línea de mandatos. Se ha empleado para descargar la última versión del entorno de desarrollo del *Gumstix*.

Octave: Octave o GNU Octave es un programa software libre para realizar cálculos numéricos. Como indica su nombre es parte de proyecto GNU. Es considerado como el equivalente GNU a Matlab.

Bibliografía

[1] Avr libc user manual, 2005.

[2] Mico is corba, an open source corba 2.3 implementation, 2005.

[3] M.I. Alarcón, P. Rodríguez, L.B. Almeida, R. Sanz, L. Fontaine, P. Gómez, X. Alamán, P.Ñordin, H. Bejder, and E. de Pablo. Heterogeneous integration architecture for intelligent control. *Intelligent Systems Engineering*, 1994.

[4] Scott W. Ambler. *The Elements of UML 2.0 Style*. Cambridge University Press, 2005.

[5] Rafael Chinchilla. *Installing ICa*. Hard Real Time CORBA IST Project, 2003.

[6] José Antonio Clavijo, Miguel José Segarra, Ricardo Sanz, Agustín Jiménez, Carlos Baeza, Carlos Moreno, Ramón Vázquez, Francisco Javier Díaz, and Antonio Díez. Real-time video for distributed control systems. In *Proceedings of IFAC Workshop on Algorithms and Architectures for Real-time Control, AARTC'2000*, Palma de Mallorca, Spain, 2000.

[7] Matthias Kalle Dalheimer. *Programming with Qt*. O'Reilly, 2nd edition, 2002.

[8] Helmut Kopk and Patrick W. Daly. *Guide to LaTeX*. Addison Wesley, 4th edition, 2003.

[9] Steve Vinoski Michi Henning. *Advanced CORBA programming with C++*. Addison Westley, 1999.

[10] Frank Mittelbach and Michel Goossens. *The LaTeX Companion*. Addison Wesley, 2nd edition, 2004.

[11] Jeremy L. Rosenberger. *Teach Yourself Corba in 14 Days*. Sams, 1998.

[12] Ricardo Sanz. Embedding interoperable objects in automation systems. In *Proceedings of 28th IECON, Annual Conference of the IEEE Industrial Electronics Society*, pages 2261–2265, Sevilla, Spain, November 5-8 2002. IEEE Catalog number: 02CH37363.

[13] Ricardo Sanz. The IST HRTC project. In *OMG Real-Time and Embedded Distributed Object Computing Workshop*, Washington, USA, July 14-17 2003. OMG.

[14] Ricardo Sanz, Idoia Alarcón, Miguel J. Segarra, Angel de Antonio, and José A. Clavijo. Progressive domain focalization in intelligent control systems. *Control Engineering Practice*, 7(5):665–671, May 1999.

[15] Ricardo Sanz, Fernando Matía, and Eugenio A. Puente. The ICa approach to intelligent autonomous systems. In Spyros Tzafestas, editor, *Advances in Autonomous Intelligent Systems*, Microprocessor-Based and Intelligent Systems Engineering, chapter 4, pages 71–92. Kluwer Academic Publishers, Dordretch, NL, 1999.

[16] Ricardo Sanz, Miguel Segarra, Angel de Antonio, and Idoia Alarcón. A CORBA-based architecture for strategic process control. In *Proceedings of IFAC Conference on New Technologies for Computer Control*, Hong Kong, P.R. of China, 19-21 November 2001.

[17] Ricardo Sanz, Miguel Segarra, Angel de Antonio, Idoia Alarcón, Fernando Matía, and Agustín Jiménez. Plant-wide risk management using distributed objects. In *IFAC SAFEPROCESS'2000*, Budapest, Hungary, 2000.

[18] Ricardo Sanz, Miguel J. Segarra, Angel de Antonio, and José A. Clavijo. ICa: Middleware for intelligent process control. In *IEEE International Symposium on Intelligent Control, ISIC'1999*, Cambridge, USA, 1999.

[19] Ricardo Sanz and Janusz Zalewski. Pattern-based control systems engineering. *IEEE Control Systems Magazine*, 23(3):43–60, June 2003.

[20] Irene Hyna Tobias Oetiker, Hubert Partl and Elisabeth Schlegl. The not so short introduction to latex 2e, 2005. www.ctan.org.