Desarrollo e implantación de plataforma robótica móvil en entorno distribuido

Franciso J. Arjonilla García

18 de octubre de 2011

A mi abuelo
que en paz descanse

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivación

El desarrollo de nuevos algoritmos en inteligencia artificial y ciencia cognitiva necesitan ser probados para que resulten útiles en la mejora de la calidad de vida de las personas. Actualmente se está realizando un gran esfuerzo para el desarrollo de nuevas tecnologías en estos campos, sin embargo, su uso se limita a entornos simulados y la obtención de información. El objetivo de este proyecto es proporcionar una plataforma robótica para probar en entornos reales los sistemas de control que se encuentran en fase de investigación.

La ingeniería de un sistema de estas características no es trivial, y en general se recurre a la adquisición de robots preparados para investigación. No obstante la combinación de actuadores y sensores deseados debe hacerse a medida y este caso no es una excepción. Uno de los elementos diferenciadores que existen es que los componentes que forman el *cuerpo* del robot deben ser independientes a los que forman la *mente*, además la capacidad de cómputo de la *mente* no está definido y en principio no se impondrán límites a su escalabilidad. Por ello se ha optado por el uso de la tecnología CORBA que capacita al robot a funcionar en entorno distribuido, es decir, con el procesamiento de la información realizándose en sistemas externos al robot propiamente dicho.

Este proyecto continúa la labor de otros proyectos fin de carrera completándolos donde sea necesario, desarrollando las capacidades ausentes e integrando todo ello para que el Laboratorio de Sistemas Autónomos de la Universidad Politécnica de Madrid disponga de una plataforma de pruebas robusta y funcional.

1.2. Estructura del documento

El capítulo 2 presenta una visión general sobre los objetivos del proyecto. El capítulo 3 ofrece una visión detallada de los requisitos que debe tener el robot y las especificaciones de los requisitos. El proyecto parte desde unos componentes ya seleccionados y adquiridos, una estructura básica y algunos módulos ya realizados; este sistema de partida es descrito en el capítulo 4. Los capítulos 5 a 7 describen el trabajo de diseño y desarrollo realizado, separados en la parte hardware, las herramientas creadas que no participan directamente en el funcionamiento y la programación software, respectivamente. El capítulo 8 presenta las pruebas realizadas para cumplir con los requisitos. En el siguiente capítulo se analiza la organización de los recursos antes y durante el proyecto para terminar en el capítulo 10 con las

conclusiones, haciendo énfasis en la trascendencia de las tareas realizadas y posibles mejoras a realizar en el futuro. Finalmente en los anexos aparecen los trabajos más detallados que por su amplitud resulta conveniente incluirlos separados de la redacción del proyecto.

1.3. Estado del arte

En esta sección se hará un recorrido por el estado de la técnica en lo que se refiere a robots para investigación. Se realizará una clasificación de los tipos de robots existentes en cuanto al campo de utilidad, el entorno de trabajo y la autonomía, para inferir las características de robots móviles para investigación, que es el caso que se aborda en este proyecto, y se darán algunos ejemplos.

1.3.1. Robots autónomos

Un robot autónomo es aquel capaz de realizar tareas relativamente complejas en entornos no estructurados y desconocidos sin la orientación continua de un operador humano. Un grado de autonomía alto ya es una realidad en campos como la limpieza de domicilios y robots corta césped, y es necesario en campos más críticos como la exploración del espacio y el control en industrias con productos peligrosos.

La mayor parte de las fábricas más avanzadas incorporan robots autónomos dentro de su entorno directo. La localización y orientación tanto del producto a transformar como de las herramientas a usar está cada vez menos fijado por el proceso de producción y es el robot el que debe decidir ante una situación no contemplada a priori.

Otro área de investigación en robótica es dotar al robot de las capacidades suficientes para hacerle capaz de desenvolverse en su entorno, ya sea en tierra, interiores, sumergido, aéreo o espacial.

Un robot completamente autónomo debe tener como mínimo estas capacidades:

- Obtener datos del estado de su entorno.
- Operar durante tiempos prolongados sin intervención humana.
- Manipular parte del entorno y desplazarse para satisfacer sus objetivos.
- Evitar las situaciones potencialmente peligrosas para las personas, bienes y para sí mismo, a no ser que sea parte de su diseño.

A medida que un robot tiene mayores grados de autonomía también puede adquirir, mediante el aprendizaje, nuevas habilidades como la adaptación a entornos cambiantes y el reajuste de los métodos usados para realizar la tarea que tiene encomendada.

La robótica comercial autónoma ha logrado introducir algunos de los progresos realizados en los laboratorios.

Auto mantenimiento Para la autonomía completa es necesario que un robot sea capaz de cuidar de sí mismo. Los juguetes robotizados y las aspiradoras domésticas son capaces de localizar y acoplarse a estaciones de carga cuando su nivel de baterías se reduce.

El mantenimiento se basa en parte en la capacidad del robot para conocer el estado de sí mismo. El conocimiento de este estado se puede aproximar con modelos internos. En general,

estos modelos no son suficientemente exactos y la lectura con sensores propioceptivos aportan información más fidedigna con menos esfuerzo de desarrollo.

Los sistemas propioceptivos más importantes incluyen:

- Encoders
- Sensores térmicos
- Sensores de efecto Hall
- Sensores de inclinación
- Sensores químicos

Percepción del entorno La exterocepción se define como la capacidad para percibir el estado del entorno. Para realizar sus tareas los robots autónomos necesitan sensores que actualicen el estado interno del entorno con los datos proporcionados por sensores que midan las perturbaciones en la interacción del robot con el entorno, como por ejemplo la cantidad de suciedad a limpiar en una determinada zona. Sensores exteroceptivos son:

- Sensores de sonido
- Sensores táctiles
- Sensores de contacto
- Sensores ópticos
- Sensores de proximidad

En general los sensores se clasifican como propioceptivos o exteroceptivos por las variables que miden, si son propias del robot independientemente del entorno o son del entorno.

Ejecución de tareas Tras la percepción del estado interno y externo el siguiente paso consiste en realizar una tarea física que modifique este estado. Esta tarea puede ser parte de una secuencia de subtareas o ser una tarea condicional. Como ejemplo, un robot patrulla puede tener una secuencia de movimientos preconfigurada que se ve alterada cuando detecta un intruso.

1.3.2. Robótica móvil

Locomoción

Existe una gran variedad de modos de desplazarse sobre superficies. Los más empleados en robótica son ruedas, cadenas y patas. Los robots de ruedas son los más empleados con diferencia. Son más sencillos, más económicos y la carga que pueden transportar es relativamente mayor. En general, Para una misma carga útil, tanto los robots con patas como los que llevan cadenas son más complicados y pesados. Además, la disponibilidad de robots de ruedas es mucho mayor, pues es posible transformar coches de radio control.

No obstante los robots de ruedas pueden tener dificultades para moverse en terrenos irregulares o para superar obstáculos mayores que aproximadamente 0,4 veces el radio de sus

Figura 1.1: El próximo robot de exploración marciana, Curiosity.

ruedas. Para estos terrenos las cadenas son la opción más conveniente. Las cadenas son más resistentes y pueden superar obstáculos mayores e incluso zanjas. Sin embargo el giro de estas ruedas provoca el deslizamiento de las cadenas que reducen la eficacia del desplazamiento. En odometría las cadenas son poco prácticas pues el error acumulativo es muy grande y depende mucho del terreno.

Los robots con patas son los más aptos para desplazarse por terrenos muy accidentados. Su mayor complejidad, proveniente del número de grados de libertad, hace que el control de estos robots suponga un verdadero reto. Cada pata necesita varios motores cada uno con su sistema de control, más luego el control de la coordinación de todos los motores. Actualmente es un área de investigación muy activo que está siendo liderado por el robot BigDog, de la Agencia de Proyectos Avanzados de Investigación de la Defensa de Estados Unidos. Cuenta con actuadores hidráulicos y la suavidad de sus movimientos se asemeja al de un mamífero pequeño.

Localización y navegación en interiores

Moverse por el interior de un edificio puede ser complejo cuando no se conoce el mapa, más aún si únicamente se percibe una franja limitada del entorno en forma de distancias a objetos detectados. Un robot debe saber en qué lugar se encuentra y ser capaz de moverse de un punto a otro. Los robots comerciales actuales como *Roomba* pueden desplazarse de forma autónoma entre habitaciones detectando características del entorno, ya sea por contacto o por cualquier otro medio.

Los sistemas más avanzados usan información de varios sensores simultáneamente, realizando localización y navegación simultáneamente (SLAM). Pueden condicionar el uso de cada sensor al que proporcione los datos más fiables y reconstruir el mapa partiendo de esa nueva información.

En general, los robots de interiores limitan su movimiento al alcanzable por una silla de ruedas, controlando ascensores y puertas electrónicas en un mapa 2D. Los costes de investigación se abaratan, mientras que la autonomía necesaria para subir escaleras y abrir puertas

Figura 1.2: Uno de los robots iRobot usados en el reciente desastre nuclear de Fukushima.

son temas de investigación candentes.

Localización y navegación en exteriores

La autonomía en exteriores es alcanzable más fácilmente que en interiores. Los obstáculos raramente aparecen y son fácilmente sorteables. No cortan el paso y no existen riesgos de choque o necesidad de búsqueda de caminos, pues una desviación del rumbo no implica necesariamente que el robot impacte contra un obstáculo.

Algunos ejemplos de robots autónomos en exteriores son aeronaves no tripuladas, misiles de crucero y robots submarinos. Algunos de estos robots logran realizar misiones sin interacción humana alguna. Parte de esta tecnología también se ha introducido en aeronaves tripuladas donde el control lo realiza una máquina y el piloto guía al sistema de control sin tener acceso directo a los accionamientos.

Aún así, la autonomía de vehículos terrestres es más compleja que los no terrestres, debido a irregularidades en el terreno, diferencia de densidades de superficie, distintas condiciones climatológicas y de la superficie y posibles inestabilidades del entorno percibido.

Una de las aplicaciones más notables de la navegación en exteriores son los rovers de exploración marciana. Los robots Sojourner, Spirit, Opportunity y el más reciente Curiosity, programado para ser lanzado en diciembre de este año 2011, pueden encontrar la posición del sol y navegar por sus propias rutas a destinos sobre la marcha mediante:

- Creación de un mapa 3D con las características topográficas de la zona.
- Distinción de terrenos seguros e inseguros.

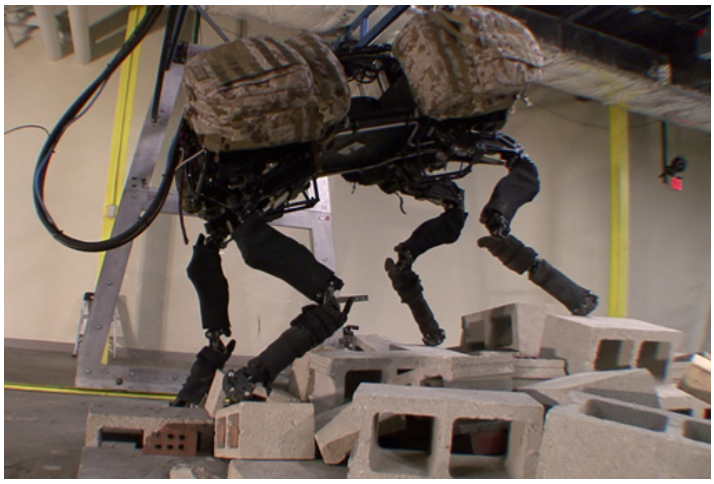


Figura 1.3: Robot de patas BigDog.

- Cálculo de rutas posibles hacia el destino prefijado.
- Navegación a lo largo de la ruta calculada.

Dos proyectos militares estadounidenses, el DARPA Grand Challenge y el DARPA Urban Challenge, promueven la investigación en navegación en exteriores, el primero en zonas rurales y el segundo en urbanas, habiendo logrado a fecha de hoy gran autonomía en los vehículos desarrollados.

En [13] se ha llegado a una solución bastante buena para el problema de control de robots móviles.

Diseño de ruedas

Se puede clasificar los robots móviles terrestres con ruedas según su estructura:

Uniciclo Es uno de los diseños más sencillos tanto mecánicamente como desde el punto de vista del control. El robot puede trazar curvas, girar sobre sí mismo e ir recto. El centro de giro se puede situar en cualquier punto del eje perpendicular a la dirección de avance de la rueda. Dada la inestabilidad de un robot de una rueda, se suele implementar con dos o más ruedas en cada lateral del robot que deslizan en los giros.

Ackerman Este tipo de diseño es el usado por los vehículos de carretera. La característica principal es que todas las ruedas se desplazan perpendicularmente al centro de giro del vehículo, siendo la configuración más típica dos ruedas motrices traseras y dos ruedas directrices delanteras. Es más eficiente que el uniciclo porque las ruedas no sufren deslizamiento, sin embargo, no permite curvas con radio de giro pequeño. Tiene buena estabilidad a altas velocidades.

Diseño sincronizado En este diseño todas las ruedas son tanto directrices como motrices. Permite un rango de movimientos omnidireccional. Cada rueda puede girarse sobre sí misma para ajustarse al movimiento requerido sin sufrir deslizamiento. Se está investigando en este tipo de robots para la próxima generación de vehículos lunares (Figura 1.4).

Figura 1.4: Vehículo lunar de la NASA en investigación.

1.3.3. Ejemplos de robots para investigación

Urbano

Urbano es un robot fruto de la investigación en Control Inteligente y Tecnología del Habla de la Universidad Politécnica de Madrid. Fue diseñado para realizar la función de guía en ferias y museos, aunque ha servido para realizar múltiples experimentos en robótica móvil.

Urbano ha sido habilitado para ser operado mediante lenguaje hablado y se puede interaccionar con él presencialmente o a distancia a través de Internet.

Es capaz de realizar SLAM (Simultaneous Localization And Mapping) en grandes edificios utilizando sensores láser para el posicionamiento y una red de sensores infrarrojos y ultrasónicos para detectar obstáculos fijos y móviles y poder así evitarlos para llegar al destino previamente programado.

Ha sido provisto de un rostro artificial y un brazo articulado a semejanza de un brazo humano con el fin de empatizar con los visitantes de los museos y a los que puede mostrar su estado interno en cuanto a baterías y otras variables, mostrando distintas emociones.

Las últimas investigaciones realizadas con este robot han consistido en el desarrollo de un generador de presentaciones por el cual el robot, partiendo de una base de datos de presentaciones, selecciona los mejores párrafos basándose en el entendimiento semántico de las características de los párrafos y de los criterios de calidad apropiados para una presentación pública. Es capaz de aprender para optimizar la calidad de las presentaciones usando lógica difusa para la toma de decisiones y representa las creencias del robot en cuanto a lo que es bueno, malo o indiferente para una presentación. Las creencias del robot continúan evolucionando para coincidir con las opiniones del público usando algoritmos genéticos para la adaptación de las reglas que guían esta evolución [2].

Khepera

Los robots Khepera son una serie de pequeños robots con múltiples funcionalidades y posibilidad de expandir estas funciones con tarjetas de extensión, como comunicación inalámbrica

Figura 1.5: El robot Urbano

y otros sensores. Han sido desarrollados por la empresa K-Team Corporation. Es de especial utilidad en la investigación de las capacidades de enjambres de robots trabajando para un objetivo común. En [3] hay más información sobre un proyecto de investigación que ha usado estos robots.

En su versión más básica, el robot Khepera-III dispone de sensores de ultrasonidos en todo su perímetro, sensores de proximidad basados en infrarrojos para rangos de distancias pequeñas y una configuración de ruedas con los que desplazarse como un móvil tipo uniciclo.

Tal como se expone en su página web, la característica más importante es su modularidad gracias al bus de extensiones incorporado para prácticamente cualquier configuración de sensores y actuadores. Incorpora dos motores DC de alta eficiencia y precisión.

Hace uso del sistema operativo GNU/Linux y de todas las herramientas de desarrollo típicamente usadas en este sistema operativo, como el compilador cruzado de C/C++. Se incluye con varios programas y utilidades que facilitan el desarrollo de aplicaciones de control.

Qbo

Qbo es un robot actualmente en desarrollo por la empresa thecorpora y su misión primaria es ejercer de plataforma robótica para investigación en interacciones sociales humano-robot. Para ello cuenta con una multitud de sensores específicos de esta tarea:

- Tres micrófonos, uno de ellos unidireccional.
- Dos altavoces de alta calidad.
- Dos cámaras de alta definición con cierres deslizantes a modo de párpados.
- LEDs en boca y nariz.

Figura 1.6: El robot Khepera con vista del interior.

- Sensores de proximidad.
- Una pantalla LCD de 20x4 caracteres.
- Otros sensores internos.

Su sistema de control está formado por varios procesadores en una disposición muy similar a la que se ha propuesto en este proyecto:

- Tarjeta Arduino para controlar los sensores y actuadores situados en el cuerpo del robot.
- Una tarjeta basada en un microcontrolador ARM para controlar los dispositivos situados en la cabeza del robot.
- Una placa Mini-ITX con procesador ATOM y tarjeta gráfica Nvidia ION donde realizar los cálculos pesados para reconocimiento de habla y visión artificial.

Todas estas características dotan al robot Qbo de múltiples capacidades entre las que se incluyen:

- Visión Estereoscópica.
- Sistema de Reconocimiento de Voz.
- Sistema para la Síntesis de la Voz.
- API y Panel de Control.
- Wifi & Bluetooth conexiones.
- Qbo puede sortear esquinas y no caerse gracias a los sensores.

Para potenciar la misión investigadora del robot todo su código ha sido liberado con licencias abiertas y está en desarrollo un portal web en el que los usuarios puedan publicar y divulgar sus algoritmos de control. Qbo está llamado a ser la referencia en cuanto a investigación en robots sociales.

Figura 1.7: El robot Qbo en desarrollo por la empresa thecorpora.

1.4. Marco del proyecto

El proyecto se ha desarrollado en el Laboratorio de Sistemas Autónomos[1] de la Universidad Politécnica de Madrid, donde se desarrollan proyectos encaminados a mejorar los sistemas de control industriales.

Figura 1.8: Logo de la UPM y ASLab.

El laboratorio ASLab tiene varias líneas de investigación abiertas, todas encaminadas hacia el cumplimiento de los objetivos a medio y largo plazo del grupo de investigación, que están recogidos en el proyecto ASys.

Las investigaciones realizadas producen sistemas que han de ser validadas mediante pruebas reales.

1.4.1. Proyecto ASys

ASys es el proyecto principal del grupo ASLab. Se trata de una línea de investigación a largo plazo en la que se enmarcan los demás proyectos. Uno de los objetivos principales es el desarrollo de tecnologías universales para la construcción de sistemas de gran autonomía.

Figura 1.9: Logo de ASys.

ASLab, aún enfocándose en el objetivo a largo plazo de ciencia y tecnología para autonomía, tiene algunas líneas de investigación más específicas alrededor de temas concretos:

- Arquitecturas de control integradas.
- Sistemas de control basados en modelos.
- Ontologías para sistemas autónomos.
- Procesos de desarrollo para controladores complejos.
- Componentes de control reutilizables.
- *Middleware* de tiempo real y plataformas para control distribuido.

[1] La página web de ASLab es `http://www.aslab.org`.

- Reubicación de componentes de control embebidos.
- Tecnología de sistemas conscientes.
- Implicaciones filosóficas de la tecnología de máquinas conscientes.

1.4.2. Otros proyectos de investigación

Algunos de los proyectos en los que ha trabajado o está trabajando el grupo ASLab son:

HUMANOBS Humanoides que aprenden habilidades socio-comunicativas por imitación. Financiado por el programa IST de la Comisión Europea.

COMPARE Una aproximación por componentes a sistemas embebidos y en tiempo real, financiado por el programa IST de la Comisión Europea.

ICEA Integrando Cognición, Emoción y Autonomía, financiado por el programa IST de la Comisión Europea.

C3 Control Cognitivo Consciente, financiado por el Ministerio de Educación y Ciencia.

1.5. Alcance del proyecto

El alcance del proyecto es el desarrollo y la implantación de una plataforma robótica móvil para navegación en interiores y exteriores en entorno distribuido con el middleware CORBA, usando en lo posible componentes hardware ya adquiridos, y la redacción de documentación en forma de manuales de cuanto sea necesario para operar el robot, con el objeto de permitir pruebas de sistemas de control en investigación de sistemas conscientes y autorreparables.

1.6. Objetivo del proyecto

El objetivo principal del proyecto es la ejecución de las tareas propuestas en el alcance del proyecto. Para ello es necesario analizar el sistema existente y el sistema deseado con detalle suficiente para poder planificar el tiempo de desarrollo. A partir de éste análisis se establecerán los requisitos que ha de cumplir el robot a la finalización del proyecto.

El proyecto tiene sentido siempre y cuando el robot permita realizar investigaciones en sistemas distribuidos, sistemas de tiempo real, algoritmos de navegación en interior y exterior, sistemas cognitivos de control y sistemas autorreparables. A pesar de ser un proyecto de desarrollo, establecerá la base necesaria para la investigación en dichos campos.

El desarrollo se dividirá en dos bloques:

Plataforma física:

- Se ha de diseñar la plataforma con objeto de integrar físicamente todos los componentes hardware.
- Se seleccionarán los componentes y los sensores que aumenten las capacidades del robot.
- Se diseñará el control de alimentaciones y sensado de nivel de baterías.
- Se integrarán los canales de señales de los equipos.

Diseño del software de la plataforma:

- Se usará CORBA en el despliegue del entorno distribuido.
- Se integrará el control remoto de los equipos con el desarrollo de módulos CORBA.
- Se deberá gestionar los errores, la tolerancia a fallos y las desconexiones de los dispositivos.
- Se implementarán las herramientas necesarias que faciliten el desarrollo y despliegue, así como las librerías de explotación de la plataforma, junto con plantillas, documentación, etc.

Capítulo 2

Requisitos

En este capítulo se describe en detalle cada uno de los requisitos del proyecto. El cuadro 2.1 ofrece una visión general del árbol de requisitos.

Número	**Descripción**
1	Establecer base de desarrollo
1.1	Integración de un ordenador de a bordo
1.1.1	Sistema operativo en tiempo real
1.1.2	Elaboración de un sistema de configuración e inicialización de servants
1.2	Desarrollo de tarjeta de alimentaciones
1.2.1	Los dispositivos se controlarán individualmente
1.2.2	Apagado manual de dispositivos
1.2.3	Apagado remoto de dispositivos
2	Integración de sensores y actuadores
2.1	Uso de la tecnología CORBA
2.2	Los objetos CORBA estarán disponibles cuando el dispositivo lo esté.
2.3	Recuperar funcionalidades automáticamente.
2.4	Integración de sensores propioceptivos
2.4.1	Integración de una brújula electrónica
2.4.2	Integración de acelerómetros
2.4.3	Integración de sensores de tensión e intensidad
2.5	Integración de sensores exteroceptivos
2.5.1	Integración de una cámara direccionable
2.5.1.1	Integración de la cámara estereoscópica existente
2.5.1.2	Integración de la muñeca
2.5.2	Integración del dispositivo láser
2.5.3	Integración del receptor GPS
2.5.3.1	Integración del sistema de correción diferencial del GPS
3	Uso de estándares y optimización en el desarrollo

Cuadro 2.1: Árbol de requisitos resumido.

2.1. Definición de los requisitos

Esta sección contiene tablas para cada requisito ofreciendo descripciones más específicas, la importancia que tiene en el proyecto, el motivo por el cual es necesario y el método de validación que se deberá usar para comprobar el cumplimiento del requisito. La numeración empleada describe la jerarquía de requisitos de manera que un requisito derivado describe detalles del superior.

Número	1
Nombre	Establecer base de desarrollo
Importancia	alta
Descripción	Se establecerá una plataforma base sobre la que desarrollar e integrar el resto de componentes software y hardware.
Justificación	Los módulos a desarrollar necesitan una base sobre la que funcionar.
Verificación	Es posible desarrollar e integrar módulos en la plataforma robótica base.

Número	1.1
Nombre	Integración de un ordenador de a bordo
Importancia	alta
Descripción	Se instalará un ordenador a bordo que ejecutará los servicios que el robot ofrezca a clientes remotos.
Justificación	Es necesario un sistema de procesamiento de información que entienda los protocolos del entorno distribuido y adapte las señales de los dispositivos.
Verificación	Se iniciará una sesión en modo consola remotamente desde otro ordenador.

Número	1.1.1
Nombre	Sistema operativo en tiempo real
Importancia	media
Descripción	El sistema operativo del ordenador de a bordo deberá contar con capacidades de tiempo real
Justificación	En la ejecución de una tarea, las acciones que realiza un robot puede ser inadecuado por realizarse más tarde de lo debido. Los tiempos de ejecución de los procesos han de ser predecibles.
Verificación	Se realizarán pruebas específicas que confirmen el correcto funcionamiento de procesos en tiempo real usando las herramientas para tal fin del que disponen los núcleos en tiempo real.

Número	1.1.2
Nombre	Elaboración de un sistema de configuración e inicialización de servants
Importancia	alta
Descripción	Se crearán los scripts y los archivos de configuración necesarios para que todos los módulos hagan uso de los mismos parámetros y se facilite la configuración común. El sistema de inicialización de servants se realizará usando las utilidades de arranque del sistema operativo.
Justificación	Durante la experimentación con el robot se realizarán cambios en la configuración que no deberían suponer más esfuerzo que el cambio del parámetro en un solo punto del sistema.
Verificación	Existirá un árbol de directorios desde el que se controlarán los parámetros más importantes de configuración de los servants.

Número	1.2
Nombre	Desarrollo de tarjeta de alimentaciones
Importancia	alta
Descripción	Se desarrollará una tarjeta que controle la alimentación de los distintos dispositivos que se instalen en el robot.
Justificación	El consumo en vehículos autónomos debe reducirse si se quiere aumentar la autonomía de las baterías. Se realizarán experimentos en los que los dispositivos se apagan espontáneamente.
verificación	Los dispositivos se encenderán al accionar un interruptor en la tarjeta de alimentaciones.

número	1.2.1
nombre	Los dispositivos se controlarán individualmente
importancia	media
descripción	La tarjeta se implementará con varios canales independientes que permitan el control individual de dispositivos a 12V y 24V.
justificación	Habrá momentos durante la experimentación con sistemas de control en los que interese mantener en funcionamiento unos dispositivos y apagar los demás.
verificación	Se podrá apagar cada dispositivo individualmente dejando los demás encendidos.

Número	1.2.2
Nombre	Apagado manual de dispositivos
Importancia	media
Descripción	Se podrá apagar manualmente cualquier dispositivo en cualquier momento mediante interruptor.
Justificación	Se realizarán experimentos en los cuales se simula el fallo arbitrario de dispositivos.
Verificación	Se podrá apagar cada dispositivo individualmente mediante interruptores fácilmente alcanzables.

Número	1.2.3
Nombre	Apagado remoto de dispositivos
Importancia	media
Descripción	Se podrá apagar remotamente cualquier dispositivo sin que sea necesario intervención humana alguna.
Justificación	En determinadas situaciones es deseable apagar dispositivos no funcionales o de baja importancia para reducir el consumo.
Verificación	Se cortará remotamente la alimentación de cada dispositivo conectado a la tarjeta de alimentaciones usando un programa que funcione autónomamente.

Número	2
Nombre	Integración de sensores y actuadores.
Importancia	alta
Descripción	Se integrarán los sensores y actuadores complementarios al robot base que permitan extender la funcionalidad del robot a navegación en interiores y exteriores.
Justificación	Todo sistema de control requiere de entradas y salidas para ser funcional.
Verificación	Se podrá controlar remotamente cada dispositivo.

Número	2.1
Nombre	Uso de la tecnología CORBA
Importancia	alta
Descripción	Se usará la tecnología CORBA en todos los servicios que se publiquen en el robot.
Justificación	El estándar CORBA es una tecnología probada para la creación de entornos distribuidos.
Verificación	Se podrán realizar todas las funciones del robot a través de interfaces CORBA ejecutadas remotamente.

Número	2.2
Nombre	Los objetos CORBA estarán disponibles cuando el dispositivo lo esté.
Importancia	alta
Descripción	Cuando un dispositivo falle, el objeto CORBA correspodiente dejará de estar disponible. Una vez vuelva a estar en funcionamiento el dispositivo, los objetos CORBA deberán ser capaces de reiniciarse por sí mismos.
Justificación	Para considerar que un robot es autónomo y tiene capacidades de autocuración no debe existir intervención humana en los procesos de recuperación de la funcionalidad.
Verificación	Para cada dispositivo, el objeto CORBA debe dar error al apagar el dispositivo y recuperar la funcionalidad original al encender de nuevo el dispositivo, sin que haya manipulación del ordenador de a bordo.

Número	2.3
Nombre	Recuperar funcionalidades automáticamente.
Importancia	alta
Descripción	El robot contará con capacidades para recuperar, sin intervención del operador, funcionalidades perdidas en los dispositivos.
Justificación	
Verificación	Los módulos CORBA en funcionamiento morirán al apagar el dispositivo correspondiente y se reiniciarán automáticamente cuando el dispositivo vuelva a estar disponible.

Número	2.4
Nombre	Integración de sensores propioceptivos
Importancia	media
Descripción	Se desarrollarán e integrarán sensores que permitan conocer el estado interno del robot.
Justificación	
Verificación	Se podrá comprobar remotamente la salida de los sensores propioceptivos.

Número	2.4.1
Nombre	Integración de una brújula electrónica
Importancia	media
Descripción	Se desarrollará e integrará una brújula electrónica.
Justificación	La brújula electrónica sirve como complemento de datos odométricos y del receptor GPS para conocer la orientación del robot.
Verificación	Se podrá comprobar remotamente la orientación del robot respecto del campo magnético terrestre.

Número	2.4.2
Nombre	Integración de acelerómetros
Importancia	baja
Descripción	Se desarrollará e integrará un sensor que mida aceleraciones
Justificación	Es posible que tanto en exteriores como en interiores el robot se desplace por terrenos accidentados o inclinados. Los sistemas de control deben estar al corriente de esta situación para adaptarse mejor al entorno.
Verificación	Se podrá comprobar remotamente la inclinación del robot en reposo respecto de la vertical.

Número	2.4.3
Nombre	Integración de sensores de tensión e intensidad
Importancia	media
Descripción	Se desarrollará e integrará un sensor que mida el consumo y el nivel de carga tanto de las baterías de la base robótica como de las baterías de dispositivos.
Justificación	Los sistemas de control deben estar al corriente de la salud de todos los componentes del robot para preveer fallos por falta de energía.
Verificación	Se podrá comprobar remotamente la tensión de las baterías y el consumo instantáneo.

Número	2.5
Nombre	Integración de sensores exteroceptivos.
Importancia	alta
Descripción	Se desarrollarán e integrarán sensores que permitan conocer el entorno del robot, tanto en exteriores como en interiores.
Justificación	Un robot autónomo necesita conocer el entorno que le rodea para poder manipularlo y desenvolverse en él.
Verificación	Se podrá comprobar remotamente variables asociadas al entorno del robot.

Número	2.5.1
Nombre	Integración de una cámara direccionable
Importancia	media
Descripción	Se desarrollará e integrará un sistema de visionado que permita tomar imágenes y tomar datos de profundidad en todos los ángulos.
Justificación	La visión es uno de los sentidos más complejos y a la vez más útiles para el control de robots en movimiento.
Verificación	Se podrá controlar remotamente la dirección en la que enfoca la cámara y recibir imágenes de ésta.

Número	2.5.1.1
Nombre	Integración de la cámara estereoscópica existente
Importancia	media
Descripción	Se integrará la cámara estereoscópica que dispone el laboratorio en el conjunto del robot.
Justificación	
Verificación	Se podrá capturar imágenes estereoscópicas remotamente.

Número	2.5.1.2
Nombre	Integración de la muñeca
Importancia	media
Descripción	Se desarrollará e integrará un sistema de movimiento de la cámara utilizando el dispositivo powercube de dos grados de libertad que posee el laboratorio.
Justificación	Cumplimiento del requisito 2.5.1 sobre direccionamiento de la cámara.
Verificación	Se podrá controlar remotamente la dirección a la que apunta la cámara.

Número	2.5.2
Nombre	Integración del dispositivo láser
Importancia	alta
Descripción	Se desarrollará e integrará un sistema de sensado de distancias en interiores a partir del dispositivo láser montado en el robot.
Justificación	Es necesario un sistema de detección de obstáculos para la navegación en interiores.
Verificación	Se podrá comprobar remotamente la lectura de las distancias a objetos próximos.

Número	2.5.3
Nombre	Integración del receptor GPS
Importancia	alta
Descripción	Se desarrollará e integrará un sistema de recepción de posición en exteriores basado en el receptor GPS existente.
Justificación	Es necesario un sistema de localización para navegación en exteriores.
Verificación	Se podrá comprobar remotamente la posición y velocidad del robot.

Número	2.5.3.1
Nombre	Integración del sistema de correción diferencial del GPS
Importancia	baja
Descripción	Se montará una estación base para la lectura y envío por radio de correcciones diferenciales basado en el receptor GPSd existente.
Justificación	Es preferible que el sistema de localización tenga una buena precisión en la posición.
Verificación	Se podrá comprobar remotamente la posición del robot en exteriores con un error menor a medio metro.

Número	3
Nombre	Uso de estándares y optimización en el desarrollo
Importancia	media
Descripción	Se hará uso de herramientas y normas estándar donde sea posible. Donde no existan normas estándar se unificará en librerías y scripts comunes a todos los módulos el código que sea posible compartir.
Justificación	El desarrollo se facilita con el uso de estándares y se asegura la continuidad a medio-largo plazo.
Verificación	El uso de software propietario se limita a lo estrictamente necesario.

2.2. Vista general del proyecto

En la figura 2.1 se da una visión general del proyecto. Se ha separado en dos bloques, la parte software y la parte hardware. La parte software está desarrollado con las utilidades, librerías y scripts que se crearán y compondrán el entorno de desarrollo, que no interviene durante la operación del robot. La parte software necesita un sistema operativo sobre el que ejecutar y poder comunicarse con el hardware; también se implantará como parte del proyecto.

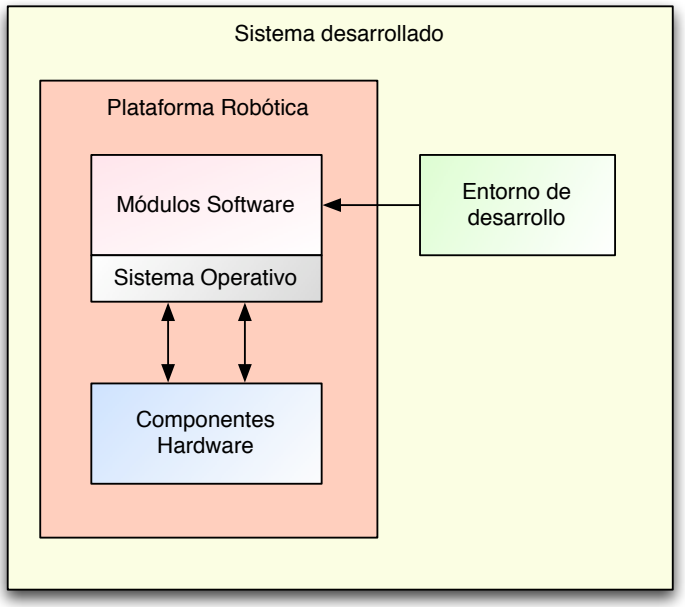


Figura 2.1: Vista general del proyecto.

Capítulo 3

Sistema de partida

El desarrollo del robot ha sido una constante evolución y lo seguirá siendo, incorporando nuevos dispositivos a medida que aparezcan y las necesidades así lo soliciten, y adaptando las tecnologías que surjan para un mejor aprovechamiento de las capacidades del robot. Por ello, este proyecto no parte desde cero, sino que existe una base desde la cual se han ido desarrollando otros proyectos y que han ido completando el robot.

Este capítulo describe los dispositivos que ya están adquiridos, qué se ha desarrollado con ellos antes del inicio de este proyecto y qué nivel de integración cuentan en el conjunto del robot.

3.1. Descripción del robot base

La plataforma robótica móvil que se ha tomado como base es un robot Pioneer 2-AT8 de la casa mobilerobots. Este robot ha sido ampliamente utilizado durante la última década por laboratorios de investigación en todo el mundo Sus características principales son:

- Cuerpo robusto en aluminio.
- Robot tipo (2,0)[1] con 4 ruedas controladas con regulador.
- Microcontrolador H8S integrado. Interfaz por puerto serie.
- Baterías de ácido-plomo integradas, accesibles por compuerta en parte trasera.
- 16 sensores de proximidad por ultrasonidos, distribuidos por el perímetro del robot.
- Dimensiones 500x490x240 mm
- Peso 15Kg

Además del robot base se ha incluido una estructura en su parte inferior con sensores de contacto que detienen el robot, como medida de seguridad ante posibles choques. Esta estructura sobresale por la parte anterior y posterior y también protege las ruedas.

Al comienzo de este proyecto la base robótica se encontraba plenamente operativa e integrada en el conjunto. El software para controlarlo estaba bien implementado y todas las

[1]Según la clasificación de robot propuesta en [4]

Figura 3.1: La plataforma robótica de investigación tomada como base.

funciones estaban disponibles a través de una interfaz CORBA[2] que mapeaba las funciones de la biblioteca ARIA destinadas a este robot, incluyendo el movimiento del robot, la lectura de los ultrasonidos y la lectura de los sensores de contacto. Este proyecto se desarrolló en [16]. Existe una rutina de comprobación en el servant CORBA por el cual hace que el robot emita sonidos característicos cuando la inicialización ha sido correcta y está preparado para recibir órdenes. La publicación en el servidor de nombres también estaba implementada.

3.2. Descripción de la muñeca

La muñeca consiste en un actuador robotizado de dos grados de libertad. El dispositivo que cumple esta función ya está disponible y montado sobre el robot, y se trata del modelo PW-070 del fabricante SCHUNK. El cableado estaba pendiente de integrar en el robot.

El software existente para controlar este dispositivo se encontraba parcialmente desarrollado. Era posible enviar órdenes de movimiento sencillas de manera poco estructurada, desde un ordenador conectado directamente por puerto serie. No existía servant CORBA ni interfaz definida.

El protocolo está descrito en el manual correspondiente que puede encontrarse en el subdirectorio `doc` dentro árbol de directorios del código fuente de la muñeca, en el repositorio SVN. Resumiendo, se trata de un protocolo cliente-servidor en el que la muñeca actúa como servidor, tanto para recibir los comandos de movimiento como para enviar datos relativos a su estado interno, como temperatura y consumo instantáneo.

3.3. Descripción de la cámara

La cámara estaba preparada y montada sobre la muñeca. A pesar de usar una interfaz FireWire, la alimentación de la cámara es independiente y el cableado estaba pendiente de integrar, es decir, que tan sólo estaba resuelto la conexión a ordenadores de sobremesa para trabajar con la cámara desmontada.

[2]Se da una descripción somera de esta tecnología en la sección E

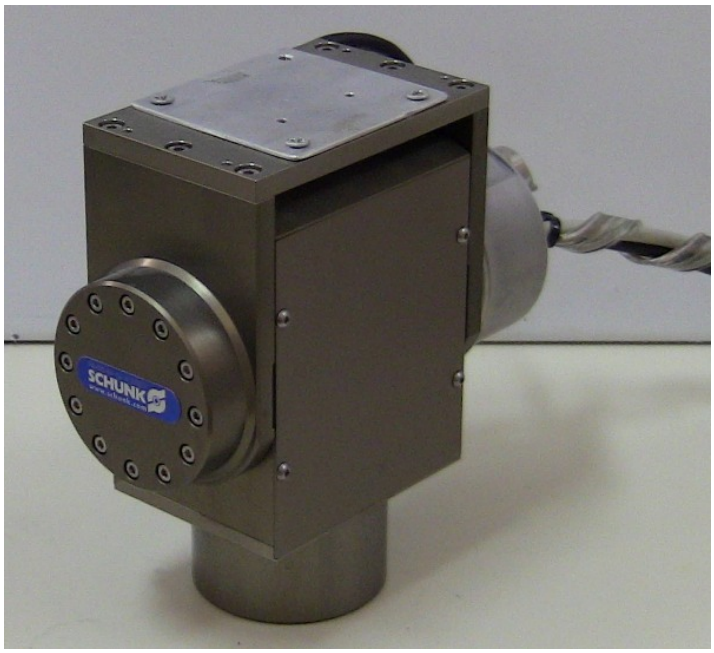


Figura 3.2: La muñeca usada en el movimiento de la cámara.

El servant CORBA estaba programado y funcionando. Está basado en un driver para linux realizado en el laboratorio para esta aplicación con soporte para la captura síncrona de ambos sensores y múltiples resoluciones. El envío de los datos de la imagen se realiza por petición de cada *frame* por parte del cliente a través de una llamada CORBA. Los datos se envían encapsulados en el protocolo CORBA sin comprimir, con codificación YUV que el cliente deberá convertir a RGB.

El mayor inconveniente de este módulo es que el ancho de banda necesario para enviar los datos sin comprimir es muy alto. No tiene importancia cuando el cliente y la cámara están en el mismo ordenador, pero cuando los datos han de enviarse inalámbricamente para cumplir con los requisitos del proyecto se consume toda la capacidad de la red existente, aumentando las latencias de otras llamadas CORBA, y siempre que se usen resoluciones bajas. Para las resoluciones más altas el módulo no es usable: latencias de varios segundos y refrescos muy bajos.

3.4. Descripción del sistema láser

El sistema de detección de obstáculos principal consiste en el sensor láser SICK LMS200, con capacidad para la lectura de distancias basado en el tiempo de retorno de un haz de luz láser, desde 8mm hasta 80m. Cuenta con tres parámetros principales configurables:

1. Resolución y distancia máxima. Se puede configurar entre una resolución de 1mm y distancias de hasta 8m o una precisión de 1cm y distancias de 80m.

2. Ángulo de barrido. Seleccionable entre 180°y 100°.

3. Resolución angular. Las opciones son 0,25°, 0,5°y 1°, siendo la primera usable únicamente para barridos de 100°.

Figura 3.3: Cámara estereoscópica

Figura 3.4: El sensor láser SICK LMS200

El último parámetro configurable, no relacionado con el funcionamiento del láser, es la velocidad de comunicación por el puerto serie. Puede seleccionarse entre 9600bps, 19200bps y 38400bps en el modo RS-232, y también 500000bps en el modo RS-422. La selección del modo se realiza empezando la comunicación a la velocidad de 9600bps, ya sea RS-232 o RS-422, y enviando el comando apropiado para cambiar la velocidad a la deseada. Si con los parámetros elegidos, también configurable mediante comandos, la velocidad del puerto serie no es suficiente para enviar todos los datos de las distancias, se pierde el último barrido.

El sensor dispone de dos entradas con protección contra polvo y agua, La primera de ellas es el suministro de alimentación a 24V y la segunda de ellas es la conexión del puerto serie, ya sea RS-232 o RS-422. Tal como queda indicado en el manual, la selección entre ambos protocolos se realiza cortocircuitando dos terminales en el interior de la caja de conexiones de la entrada del puerto serie.

El manual del láser y el protocolo que sigue se puede encontrar en el directorio del código fuente del módulo. Existe un módulo CORBA realizado en el proyecto fin de carrera [11] que hace uso de la librería ARIA, está limitado a las necesidades del citado proyecto y no está integrado con el resto del robot.

3.5. Descripción del sistema GPS diferencial

El sistema GPSd es uno de los menos desarrollados. Los componentes hardware fueron adquiridos en el año 2007 y se trata del model OEMV-2-RT2 compuesto por una estación base emisora de correcciones diferenciales y una estación móvil.

Al inicio del proyecto los componentes del GPS se encontraban almacenados en el maletín original y se hicieron algunas pruebas infructuosas para tratar de recibir la posición, usando para ello un terminal serie conectado al receptor y comandado mediante órdenes manuales sencillas. Está pendiente el desarrollo del módulo CORBA, el driver de conexión, pruebas de funcionamiento y la integración.

Figura 3.5: Componentes que forman el sistema GPS de la estación móvil

La estación base está compuesta por una caja de control en la que se procesan las señales recibidas de los satélites. Tiene dos conectores coaxiales, el primero de ellos e la entrada del oscilador de alta precisión que no se usará, y el segundo es donde debe conectarse la antena que irá situada en la parte más elevada del robot. La parte frontal consta de cuatro conectores para puerto serie, de los cuales el etiquetado como COM1 se usa como interfaz de conexión con el ordenador de a bordo y COM2 para recibir las correcciones diferenciales de la radio. La radio es el cilindro azul en el que se colocará la antena correspondiente. Consta de un solo conector del que parte un cable que separa la señal de datos y la alimentación. La radio está alimentada con una batería propia. Esta batería tiene además su propio cargador.

El sistema GPS de la estación base es muy parecido al de la estación móvil. Se compone de una caja de control con la electrónica, exactamente el mismo hardware pero con una

(a) (b) (c)

Figura 3.6: La antena, radio y caja de control de la estación móvil del GPS.

configuración software distinta, una antena de mayores dimensiones y una radio con una potencia de emisión de 2W. El resto de componentes son iguales.

Figura 3.7: Componentes de la estación base del sistema GPS

Existe una antena GPS instalada en el ático del departamento de automática. Su localización se puede ver en la figura 3.8. De él cae un cable coaxial por la fachada del edificio que permite instalar los componentes de la estación base en el interior del edificio.

3.6. Descripción de la tarjeta de adquisición de datos

El robot cuenta con diversos sensores y actuadores que no son directamente conectables a un puerto serie o cualquier otra interfaz de conexión con una computadora moderna. La tarjeta de adquisición de datos viene a suplir esta necesidad de conectar dispositivos sencillos con el ordenador de a bordo.

La funcionalidad de la tarjeta de adquisición de datos está dividida dos partes. La primera de ellas reside en el ordenador de a bordo y se encarga de mediar entre la interfaz CORBA existente y el puerto serie con el que se comunica con la tarjeta, y la segunda es la tarjeta de adquisición de datos y el firmware desarrollado en sincronía con el servant.

Figura 3.8: Localización de la antena GPS en el ático del departamento.

El servant CORBA está desarrollado en Java y dispone de una interfaz gráfica por la que se pueden visualizar todas las variables cuyo valor se reciba desde la tarjeta. Asimismo estas variables se pueden observar a través de la interfaz CORBA. La versión inicial de este módulo se desarrolló en [15].

La tarjeta de adquisición de datos es una tarjeta Arduino Mega que usa el microcontrolador Atmel Mega 1260. Se trata de una tarjeta debajo coste diseñada para desarrollos rápidos que incorpora su propio entorno de desarrollo basado en la herramienta *processing* e incluye librerías que evitan tener que programar en ensamblador para acceder a las funciones básicas de entrada y salida del microcontrolador.

A continuación se describen el acelerómetro y la brújula electrónica, que están pendientes de implantar.

3.6.1. Acelerómetro

El acelerómetro es una tarjeta con dos sensores de aceleración monoaxiales independientes, dispuestos perpendicularmente. Parten dos cables con alimentación y datos para cada acelerómetro, como se muestra en la figura 3.9. Su salida es analógica.

Es necesario probarlo e integrarlo en el conjunto del robot.

3.6.2. Brújula electrónica

La brújula electrónica usada en este proyecto es una pequeña tarjeta autocontenida con un microcontrolador Microchip que procesa la información recibida de dos sensores Philips unidireccionales de campo magnético situados perpendicularmente, tal que la salida de la tarjeta es una señal PWM proporcional al ángulo respecto al norte magnético. Cuenta con compensación del campo magnético creado por la corriente eléctrica del cableado de los edificios con opción de configurarlo a 50Hz o a 60Hz y de un terminal de inicialización que sólo

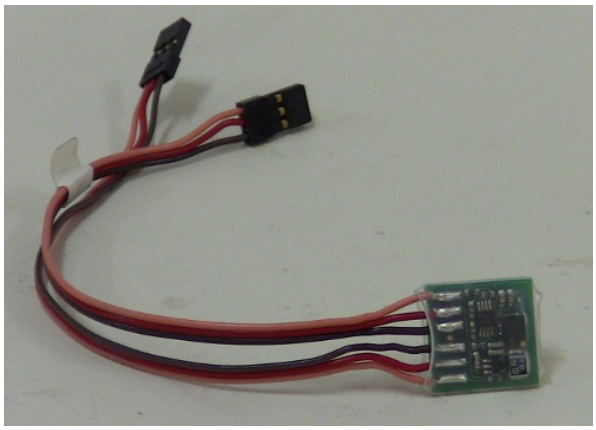


Figura 3.9: Detalle del acelerómetro.

ha de emplearse en el primer uso o cuando haya cambios de localización importantes, para ajustarse a los nuevos niveles de campo electromagnético terrestre.

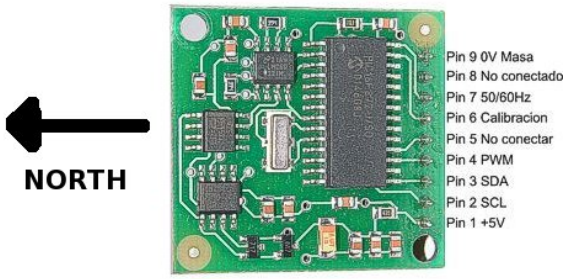


Figura 3.10: Detalle de la brújula electrónica.

3.7. Descripción del ordenador de a bordo

El ordenador de a bordo es un portátil marca Sony y modelo Vaio TX2HP. Sus caractarísticas son:

- Intel Pentium M a 1,1GHz.
- 512MB de memoria SDRAM.
- 80GB de disco duro a 4200rpm.
- Pantalla LCD de 11,1".
- DVD+RW

Los puertos de los que dispone son:

- 2xUSB
- FireWire

- Ethernet 100Mbps
- Wifi IEEE 802.11b/g
- Bluetooth

Tiene instalado un sistema operativo Fedora Linux desde el que se han hecho pruebas con algunos dispositivos pero sin llegar a funcionar autónomamente, es decir, que los servants y los programas se han ejecutado manualmente y el ordenador no se ha llegado a integrar adecuadamente con el resto de componentes del robot.

3.8. Descripción del entorno de red del laboratorio

El laboratorio dispone de una red propia con servidores de distinto tipo. Los servicios más importantes que se han usado a lo largo de este proyecto, junto con una breve descripción del uso que se ha hecho de ellos se expone a continuación:

NIS Identificación por contraseña en los ordenadores. El laboratorio está dispuesto tal que cualquier usuario puede usar cualquier ordenador y encontrarse en él la misma configuración con la que abrió sesión por última vez, aunque fuera en otro ordenador. El mantenimiento de las contraseñas se puede realizar así desde un único punto para todos los usuarios.

NFS Net File System, o sistema de archivos en red. Se trata de un disco duro virtual situado remotamente. Es un servicio complementario a NIS por el cual se pueden almacenar los archivos personales en el servidor principal. Permite abrir sesión con los archivos personales disponibles en cada ordenador. La tarea de realizar las copias de seguridad para todos los miembros del laboratorio se automatiza fácilmente y, al tener el servidor un sistema RAID con discos duros extraíbles en caliente se mejora la seguridad de los datos.

CVS Sistema de control de versiones. Utilizado inicialmente para ir almacenando las diferentes versiones del software desarrollado así como los archivos de diseño del hardware, en su evolución. También usado para compartir el código con el resto del laboratorio.

SVN Equivalente a CVS pero más moderno. Se describe en detalle en el anexo E.

SSH Secure SHell. Programa con interfaz en modo texto que abre consolas en otros ordenadores remotamente. Ha sido utilizado para coordinar cómodamente todos los ordenadores con los que se ha trabajado simultáneamente.

Existe un punto de acceso wifi con ESSID *aslab_wireless* necesario para las comunicaciones inalámbricas con el robot.

Capítulo 4

Diseño hardware

En este capítulo se describirá el proceso seguido para el diseño del hardware. No todo el robot ha sido diseñado, sino que una parte estaba ya montado y funcionando, aunque sin robustez ni integración adecuadas, otra parte son elementos adquiridos pero no montados sobre el robot, y algunos componentes han tenido que diseñarse desde cero.

4.1. Diseño de la tarjeta de alimentaciones

4.1.1. Análisis de los requisitos

En esta sección se va a analizar el requisito 1.2. Una de las condiciones que debe cumplir el robot móvil es, como ya se ha comentado en el capitulo 2 relativo al análisis de requisitos, es que pueda continuar operando con normalidad aunque no todos sistemas estén en funcionamiento. El caso típico sería aquel en el que el robot se pone en marcha con todos los sistemas operando correctamente y con el transcurso del tiempo y de la misión asignada, ciertos componentes dejen de responder bien sea por avería o por cualquier otra cosa. Por ejemplo, una misión en la que el robot móvil deba desplazarse por interiores y por exteriores necesita de sensores de proximidad, en este caso el láser, en el interior del edificio que le ayude a reconocer las habitaciones, mobiliario y obstáculos, y otro sensor totalmente distinto para exteriores, en este caso el GPS. En interiores el GPS no es funcional debido a la mala recepción de las señales electromagnéticas enviadas por los satélites, y en exteriores el sensor de proximidad únicamente valdría para la detección de obstáculos, no así la localización por mapas ya que en exteriores no hay paredes ni mobiliario y el sensor tiene un rango máximo de detección de objetos de unos ocho metros (u ochenta metros, dependiendo de la configuración). En ambos casos, exterior e interior, los sensores dejan de aportar información útil por lo que siempre habrá algún sensor que no pueda ser usado para recabar información del entorno y que sin embargo siguen consumiendo energía y reduciendo la carga de las baterías del robot. Seria conveniente poder apagar estos sensores para aumentar el tiempo de operación del robot sin necesidad de recargar.

Por lo tanto, los requisitos principales que debe reunir la tarjeta de alimentaciones es que sea capaz de gestionar la alimentación de cada uno de los sensores y actuadores por separado de dos maneras distintas. Una automática, controlada por ordenador, en la que se pueda apagar remotamente los sistemas para ahorrar energia independientemente de si en el punto remoto esta trabajando un operador humano o es una arquitectura cognitiva que decide que el

sistema en cuestión no es útil en ese momento, y otra manual que haga las veces de simulador de fallos en los sistemas.

Desconexión automática

Para el método automático, se necesitará un medio que conecte la tarjeta de alimentaciones con la tarjeta de adquisición de datos en la que cada una de las lineas controle digitalmente la tension de cada sistema, que se llevara a los pines digitales de entrada y salida generales de la tarjeta de adquisición de datos. Será responsabilidad de la tarjeta de adquisición de datos la que, a través de la conexión USB con el ordenador de a bordo, suministre las señales adecuadas para el correcto control del apagado automático de cada uno de los canales. El requisito 1.2.3 estaría resuelto.

Desconexión manual

Por otra parte también se desea simular la avería de los sensores para que el software de control que opere el robot pueda ser probado ante estas eventualidades sin tener que provocar una avería real sobre los sistemas. Muchas veces estas averías se dan de forma intermitente y el robot debe ser capaz de recuperarse de ellas una vez el sistema afectado vuelve a estar disponible. Tal es el caso en el que un cable defectuoso queda a circuito abierto, por problemas en el conector o incluso por problemas de los conductores del cable mismo, bajo determinadas condiciones, como por ejemplo cuando la temperatura ambiente supera un cierto umbral, digamos 30 ℃. La condición de fallo de este ejemplo se daría únicamente en verano y en exteriores. El robot, una vez han desaparecido las condiciones de fallo, debe restablecer el sistema. La mayor dificultad sería pues que el software del sistema tolerara estos fallos y tuviera una cierta capacidad de auto-curación, problema que se afrontará en el capítulo 6, y para poder probar este software se requiere poder simular los fallos. En este proyecto la simulación de estos fallos se va a limitar al apagado manual arbitrario de sistemas individuales, suponiendo que si un sistema está operativo entonces interactuará con el entorno de una manera correcta y fiable, cumpliendo con el requisito 1.2.2. Otro ejemplo sería el de un actuador que se sobrecalienta y se avería, se le ordenaría moverse pero los sensores no registrasen los datos que se esperaría si el actuador se hubiera realmente movido.

Otras consideraciones

Para el método manual se facilitará un interruptor por cada dispositivo. Adicionalmente habrá un interruptor general que controle la alimentación de todos los sistemas y reduzca efectivamente, con corriente consumida nula o despreciable, el gasto de carga de las baterías cuando el robot no esté en uso. Tambien será necesario instalar un conector de alimentación estándar “power jack” que cargue las baterías aunque el interruptor general esté apagado.

Algunos de los dispositivos funcionan a 24V, mientras que las baterias aportan una tension nominal de 12V, que en la practica es variable entre 10,5V y 13.5V. Existe un convertidor 12VDC/24VDC instalado en la parte frontal de la base del robot que tambien deberá ser controlado por la tarjeta de alimentaciones de modo que este apagado cuando no haya dispositivos de 24V en funcionamiento y así reducir el consumo ya que este convertidor tiene un consumo fijo de 6W incluso cuando no existe consumo eléctrico en los 24V.

Por tanto, cada sistema estará controlado por un canal con dos métodos distintos de apagado, digital y con un interruptor, configurados como una ’AND’ lógica, es decir, que el

Tension de bobina	12VDC
Corriente nominal máxima	5A
Configuración de los contactos	DPST-NO
Resistencia de contacto	30mΩ
Resistencia de bobina	480Ω
Referencia del fabricante	G6B-2214P-US 12DC

Cuadro 4.1: Características principales del relé usado en la tarjeta de alimentaciones

sistema estará encendido solamente si ambos métodos permiten simultáneamente el encendido del sistema. Existirán tantos canales como sistemas haya, más algunos de reserva para futuras ampliaciones del robot con nuevos dispositivos. Cada canal deberá contar con un indicador luminoso que indique la existencia de tensión en el conector de salida de la tarjeta de alimentaciones de cada canal y un indicador luminoso que facilite la comprobación de encendido de la tarjeta. Con este desarrollo se cumpliría también el requisito 1.2.1.

4.1.2. Diseño electrónico y selección de componentes

En esta sección se describe el proceso para la selección cualitativa de los componentes y los cálculos matemáticos para obtener los valores cuantitativos de los componentes que lo requieran.

Selección del interruptor electrónico

En la selección del dispositivo de interrupción electrónica se ha descartado el uso de transistores y otros componentes semiconductores debido a la caída de tensión entre sus terminales, de unos 0,6V, y que resulta inaceptable para muchos de los dispositivos cuando la carga de las baterías se reduce, al imponer una reducción en la tensión de la alimentación aun mas baja y restringiendo el mínimo de carga necesaria para que continúe operando.

Finalmente se optó por el uso de relés. Estos dispositivos electromecánicos ofrecen una caída de tensión despreciable aunque la velocidad de conmutación es baja, del orden de 10Hz. Esto no es un problema ya que la frecuencia de conmutación del relé debe estar por debajo del tiempo de inicialización de los dispositivos que alimenta. A modo de ejemplo, el sensor láser tarda de media unos 40 segundos en inicializarse, más el tiempo que dura la sincronización y la inicialización del software controlador. El relé debe ser capaz de soportar la corriente del dispositivo de mayor consumo, que se ha analizado en párrafos anteriores y corresponde a los 4A de la muñeca, la bobina debe funcionar a 12V y debe disponer de dos contactos normalmente abiertos, uno para controlar el dispositivo en si y otro para desconectar el convertidor 12VDC/24VDC en los canales de 24V. Se usará el mismo relé para todos los canales. Las características principales del relé seleccionado se muestran en la tabla 4.1. Se trata de un relé subminiaturizado de dos polos y un terminal por polo con contactos normalmente abiertos.

De la tabla de datos se deduce que la caída de tensión máxima que sufrirá el relé en el peor de los casos es de $4A * 30m\Omega = 0,12V$.

Diseño del circuito de control del relé

La bobina de activación del relé no es tan sensible a variaciones de tensión como lo son los dispositivos y es posible controlarlo con un transistor. Se ha usado un transistor de propósito general NPN en serie con la bobina. Ver la figura A.1. La base del circuito estará controlado por una fuente de corriente continua formada por la fuente de 12V y una resistencia que realice simultáneamente la función de resistencia *pull-up*, siendo el resto de componentes los que desvían la corriente de base del transistor hacia tierra sin pasar por éste, cuando se quiere tener el dispositivo apagado, bien sea con el método automático o con el manual. El método automático dispone de un diodo que evita que el microcontrolador que se encuentra al otro lado pueda suministrar la tensión de polarización al transistor, es decir, que sólo podrá forzar la base del transistor a 0V. Sucede que las caídas de tensión propias del microcontrolador y de este último diodo son mayores que la caída de tensión base-emisor del transistor, de ahí la colocación del diodo zéner de tensión de polarización inversa de 2,4V que invierta esta situación. El valor de la resistencia está diseñada teniendo en cuenta ésto y la β del transistor. Para el método de apagado manual se ha dispuesto un interruptor con un condensador en paralelo que elimina el efecto rebote del interruptor y ayuda a alargar la vida útil del relé. Finalmente el transistor está protegido contra sobretensiones con un diodo en paralelo con la bobina.

Por aplicación de la ley de ohm a las características del relé, la corriente necesaria para su activación, y sabiendo por las especificaciones que necesita un mínimo de tensión del 70 % del nominal que se va a cumplir incluso en condiciones de baja carga, es

$$\frac{12V}{480m\Omega} = 25mA \tag{4.1}$$

A ésta corriente de colector, la β del transistor está situada entre 60 y 100.

$$\frac{25mA}{60} = 0,41\widehat{6}mA \tag{4.2}$$

Aplicamos un margen de seguridad aproximando la corriente de base a 0,9mA, y la resistencia de *pull-up* limita la corriente con un valor de

$$\frac{12V - 2,4V - 0,7V}{0,0009A} = 9,\widehat{8}k\Omega \rightarrow 10k\Omega \tag{4.3}$$

Las resistencias limitadoras de corriente para los LED indicadores:

$$\frac{V_{cc} - V_{LED}}{I_{LED}} = \frac{12V - 2,5V}{10mA} \simeq 1k\Omega; \frac{24V - 2,5V}{10mA} \simeq 2,2k\Omega \tag{4.4}$$

Para terminar, la constante de tiempo en el condensador, que ha sido validado por estimación, queda:

$$\tau = 10k\Omega * 1,5\mu F = 15ms \tag{4.5}$$

más que suficiente para eliminar el efecto rebote.

Resto de componentes

El número de canales que se van a implementar son nueve: seis de ellos con tensión de alimentación a 12V y el resto a 24V. La distribución elegida para los dispositivos es la que se

Canal	Tensión	Dispositivo	Consumo (en A)
1	12V	Reserva	–
2	12V	Sensores[4]	0,3
3	12V	Servo	1,5
4	12V	Reserva	–
5	12V	Cámara	0,083
6	12V	GPS	0,5
7	24V	Muñeca	3
8	24V	Láser	0,7
9	24V	Reserva	

Cuadro 4.2: Distribución de dispositivos por canales en la tarjeta de alimentaciones

muestra en la tabla 4.2. Se ha resuelto dejar dos canales de reserva en el canal a 12V y uno en el que va a 24V.

En cuanto al diseño de las pistas, la separación mínima entre pistas viene fijado por la diferencia de potencial máxima que es de 24V,que se corresponde a una separación de 0,3mm. El grosor de las pistas viene condicionado por la corriente que transportan, siendo las mayores las de los conmutadores de los relés. Se acepta un incremento de temperatura admisible máxima de 30ºC. Para un grosor de cobre de 35μ, el ancho de pista que soporta una corriente de 1A es de 0,45mm. Se ha tenido en cuenta que no todos los dispositivos usan el máximo de corriente de 4A por canal, y que los dispositivos de 24V consumen el doble de corriente en la parte de 12V antes de la conversión a 24V. Un ancho de 1cm soporta así 22,7A que es aproximadamente la corriente máxima consumida por todos los dispositivos en funcionamiento.

La conexión con la tarjeta de adquisición de datos se realizará mediante cable plano, que reúna en un solo cable los 9 canales y la conexión a tierra. La tarjeta de adquisición de datos dispone de una serie de pines digitales de entrada/salida general con disposición idéntica a la de cable plano, circunstancia que se aprovechará para facilitar la conexión. Los esquemas electrónicos y la distribución de pistas, serigrafía y orificios se pueden encontrar en el anexo A.

4.1.3. Fabricación y complicaciones surgidas

Circuito impreso

El diseño finalizado se exportó a archivos *gerber*, que es el formato más aceptado entre los fabricantes de circuitos impresos. Estos archivos se pueden encontrar en formato comprimido dentro del directorio `Power_Board/OrCAD` del cedé del proyecto. Se solicitó presupuesto para la realización del prototipo a las empresas detalladas en la tabla 4.3.

Las ofertas detalladas de estos presupuestos remitidas por los fabricantes se encuentran en el cedé que acompaña este proyecto. Cabe destacar los siguientes factores a la hora de elegir el fabricante:

- El plazo de entrega debía ser lo más corto posible sin que se dispare el precio por urgencia. Tras recibir los presupuestos el plazo más razonable con todos los fabricantes resultó ser de 5 días.

Empresa fabricante	Presupuesto
FAST PCB	308,00€
2CI	202,87€
ELATE	280,88€
NOVATEK	251,76€
LAB CIRCUITS	335,00€

Cuadro 4.3: Listado de presupuestos para la fabricación del circuito impreso de la tarjeta de alimentaciones.

- El espesor de cobre base debe ser mayor que el estándar para soportar las altas corrientes que se esperan. El espesor base estándar es de 18μ, existiendo ofertas de 35μ e incluso 70μ.
- Se tomará en consideración el color del acabado de la tarjeta por motivos estéticos, siendo preferible el azul que es el color de la tarjeta de adquisición de datos. No obstante seguirán teniendo más peso los motivos económicos.
- El circuito impreso será de clase 3, que es la de menor calidad que comúnmente realizan los fabricantes.
- Los fabricantes exigen un mínimo de dos a cuatro tarjetas en la fabricación del prototipo. No se tendrá en cuenta el precio por tarjeta del presupuesto, sino el precio global ya que el prototipo será también el circuito impreso final que se usará en el robot.

Una vez analizados los presupuestos se optó por aceptar la oferta de 2CI. Los dos principales motivos fueron el coste y la calidad percibida. Las características más relevantes del circuito impreso se exponen a continuación:

- Grosor de cobre: 35μ.
- Color de máscara: Azul. Blanco para la serigrafía.
- Plazo de entrega 5 días
- Dos capas clase 3.
- Grosor del material base 1,6mm de material FR4 (fibra de vidrio con resina epoxi).

Montaje de componentes

Recibidos los tres prototipos del circuito impreso se procedió a montar y soldar los componentes previamente pedidos. La lista de componentes, *Bill Of Materials*, se muestra en el cuadro 4.4. Se ha añadido la referencia de la tienda online RS-Amidata para cada componente.

El montaje y soldadura de los componentes se realizó manualmente con herramientas pertenecientes al grupo de investigación. Durante el montaje se encontraron algunos problemas y fallos de diseño que pudieron resolverse sin necesidad de encargar más circuitos impresos, pero que habrá que tener en cuenta y rediseñar el circuito impreso si se da el caso de tener que fabricarlo de nuevo:

Cantidad	Referencia RS	Descripción
9	332-818	Right angle small switch
1	317-695	Right angle switch with lever
9	294-312	Low signal NPN transistor
12	670-5715	MiniFit connector, male PCB
9	544-3503	Zener diode 2V4
9	188-3036	Capacitor, 1,5uF electrolythic
1	542-8784	Ribbon connector 10way PCB header
10	466-4260	Right angle LED
9	369-781	DPST Relay
1	487-836	Power Jack connector, male PCB
3	131-299	Resistor 2K2
7	131-255	Resistor 1K
9	131-378	Resistor 10K
18	628-9029	1N2004 Diode
3	426-3722	Ribbon 10way plug
20	679-5776	MiniFit connector, female plug
1	172-9134	MiniFit connector, bag with 100 terminals

Cuadro 4.4: Listado de componentes de la tarjeta de alimentaciones.

1. Los orificios para los pines de los relés presentan una separación excesiva. Además, los dos pines correspondientes a la bobina están invertidos, fallo que no se encontró por situar casualmente los pines con la polaridad adecuada durante las pruebas iniciales y por no estar suficientemente documentado en la hoja de especificaciones del relé. Se solucionó separando manualmente los pines antes de montarlos sobre las tarjetas y extendiendo los pines de la bobina e intercambiándolos, usando pegamento termoadhesivo como aislante. Además, debido al debilitamiento de los pines con su manipulación, se reforzó la sujeción del relé a la tarjeta con adhesivo.

2. El conector de carga de la batería tiene desconectado el pin de tensión positiva. Se resolvió creando un puente de estaño con la pista adecuada.

3. Los orificios de los diodos 1N4004 son de tamaño insuficiente. Se aumentó el tamaño con una taladradora que destruyó la capa de cobre pasante, por lo que hubo que soldar estos componentes por ambas caras del circuito impreso.

4. La serigrafía del conector 24VIN es incorrecta y está marcado como 12VIN que queda duplicado. De ambos, es el conector más alejado del cable plano el que se marcó con tinta indeleble la marca correcta 24VIN.

El resultado final con la localización de conectores e interruptores se muestra en la figura 4.1.

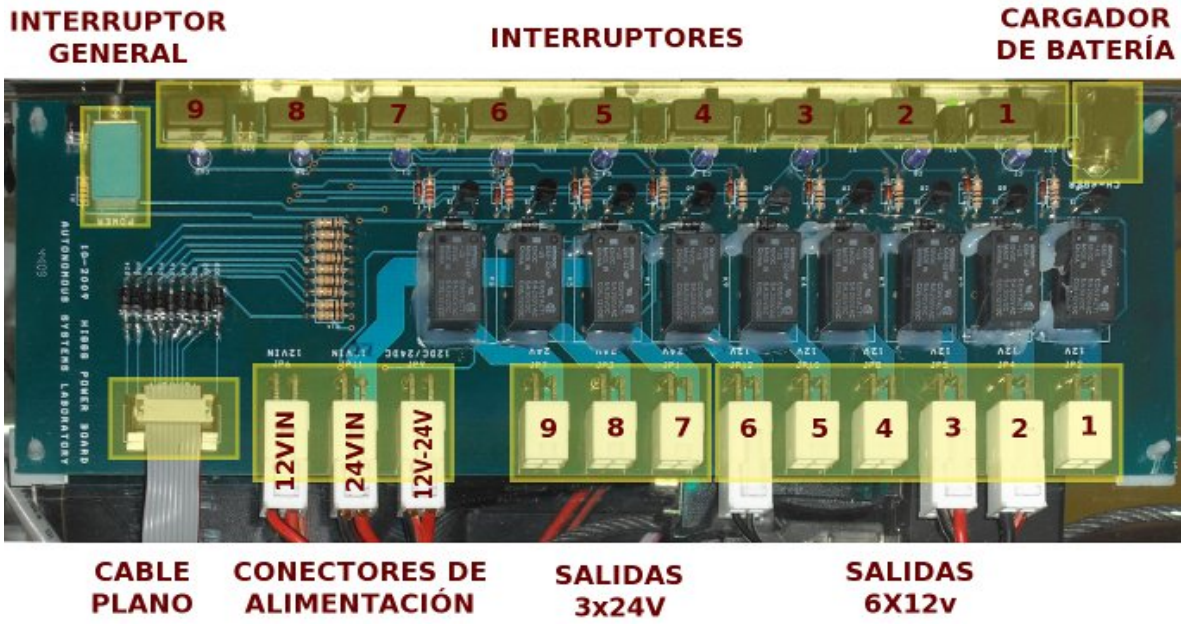


Figura 4.1: Fotografía de la tarjeta de alimentaciones montada sobre el robot.

4.2. Diseño del sensor de corriente y tensión

4.2.1. Análisis de requisitos

En general un robot cuenta con sensores que captan información del entorno pero también cuenta con otros sensores que reciben información sobre el estado del propio robot, de esta manera los algoritmos pueden adaptar sus resultados para obtener la mejor solución teniendo en cuenta no solamente el exterior sino las funciones internas, descalibraciones y otras desviaciones de la idealidad. El sensor de corriente y tensión está pensado para satisfacer esta necesidad en cuanto se refiere a la cantidad de energía que aún puede usar el robot y el consumo actual, tal como se ha indicado en el requisito 2.4.3.

Existen cuatro baterías independientes en el robot móvil, cada uno alimentando una serie de sistemas distintos.

Baterías de motores Alimentan el robot base, incluyendo los motores, el controlador interno y los sensores sónar.

Baterías de instrumentación Alimentan los dispositivos montados sobre el robot base. La distribución de las alimentaciones está gestionada por la tarjeta de alimentaciones. Incluye Láser, GPS, muñeca, etc.

Batería del ordenador de a bordo Está integrada en el ordenador de a bordo e incorpora su propio sistema de gestión de baterías, accesible desde el sistema operativo.

Batería de la radio del GPSd Alimenta el receptor de radio del GPS diferencial.

Se van a monitorizar las tres primeras baterías. La cuarta presenta las siguientes características que desaconsejan su monitorización, pues añadiría complejidad al sistema sin que sea una información suficientemente útil: Se trata de un sistema propietario cerrado con cables y conectores dedicados, la duración de la carga es de un orden de magnitud mayor que el resto de baterías en el robot y la pérdida de la radio del GPS implicaría únicamente la reducción en la precisión de la posición en exteriores.

La batería del ordenador de a bordo puede ser monitorizada por el núcleo del sistema operativo y no necesita ningún circuito adicional; Las baterías de los motores y de la instrumentación son muy parecidas diferenciándose en la potencia que deben aportar, siendo en el caso de los motores de hasta 17A y en el de la instrumentación de hasta 6A (ver cuadro 4.2). Ambas baterías son de plomo y ácido compuestas por tres células de $7A \cdot h$ cada una.

4.2.2. Diseño teórico

La técnica elegida ha sido colocar en serie una resistencia schunt a la salida de las baterías, de modo que se pueda medir la tensión de la batería antes de la resistencia y la corriente indirectamente leyendo la diferencia de tensión en la resistencia schunt. Esta configuración necesita un divisor de tensión que reduzca las tensiones a niveles con los que los componentes que se usarán puedan trabajar, a saber un amplificador operacional que convierta dos tensiones próximas en una tensión proporcional a la diferencia de tensiones y un microcontrolador. La figura 4.2 muestra el esquema del circuito electrónico diseñado, donde se aprecia los divisores de tensiones y el circuito diferencial, haciendo uso de uno de los divisores para una función doble: la lectura de la tensión por parte del microcontrolador y la adaptación de esa misma tensión al amplificador operacional.

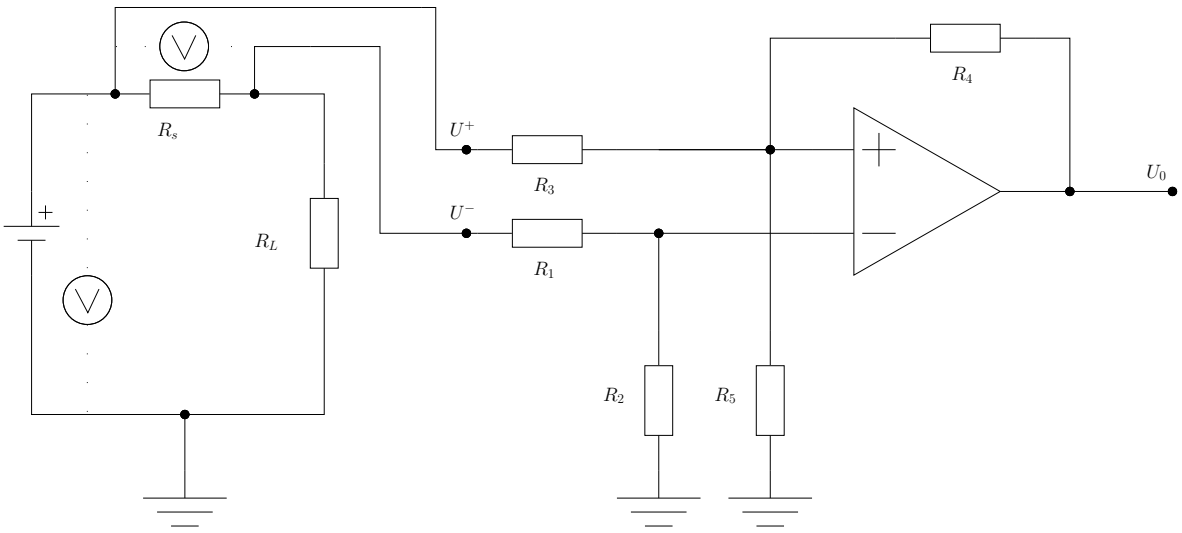


Figura 4.2: Esquema de referencia para el sensor tension-intensidad

Aplicando la primera ley de Kirchhoff a la entrada positiva del amplificador operacional,

$$\frac{U^+ - U_e}{R_3} = \frac{U_e}{R_2} \Rightarrow U_e = U^+ \cdot \frac{R_2}{R_1 + R2} \tag{4.6}$$

Aplicando la misma ley sobre la entrada negativa,

$$\frac{U^- - U_e}{R_3} = \frac{U_e - U_0}{R_4} + \frac{U_e}{R_5} \tag{4.7}$$

Sustituyendo (4.6) en (4.7)

$$U_0 = U^+ \cdot \frac{R_2 R_4}{R_1 + R_2} \cdot \left(\frac{1}{R_3} + \frac{1}{R_4} + \frac{1}{R_5} \right) - U^- \cdot \frac{R_4}{R_3} \tag{4.8}$$

Ahora, según esta última ecuación, U_0 depende tanto del diferencial entre las tensiones de entrada como del modo común. Eliminaremos el la desviación por el modo común buscando una ecuación de la forma $U_0 = K \cdot (U^+ - U^-)$. Para ello igualamos los factores por los que se multiplican las tensiones de entrada:

$$\frac{R_2 R_4}{R_1 + R_2} \cdot \left(\frac{1}{R_3} + \frac{1}{R_4} + \frac{1}{R_5} \right) = \frac{R_4}{R_3} \tag{4.9}$$

Y despejando R_1/R_2 en función de R_3, R_4 y R_5 queda:

$$\frac{R_1}{R_2} = \frac{R_3}{R_4 \parallel R_5} \tag{4.10}$$

Conservar esta relación de valores en las resistencias es importante de cara a mantener la linealidad del circuito. Al no existir condensadores ni bobinas, no hay polos ni ceros en la función de transferencia del circuito y por tanto el sistema es estable.

Cálculo de los valores de las resistencias

La tensión máxima de entrada al amplificador operacional y al microcontrolador está limitado a 3,5V superiormente y a 0V inferiormente según la hoja de especificaciones de ambos.

Los factores que se han de controlar son: La ganancia diferencial, el divisor de la tensión en el lado de la batería y la tensión de entrada máxima al amplificador operacional.

En ambos casos se ha seguido un procedimiento iterativo en el cual se empieza asignando valores razonables al divisor de tensión formado por R_1 y R_2, seguir diseñando la ganancia con R_3 y R_4 y terminar ajustando R_5 para que se cumpla la condicion (4.10). Se analizan los valores obtenidos y se comprueba estén dentro de unos márgenes prácticos entre 10kΩy 1MΩaproximadamente y que existan comercialmente. En caso de no cumplir se asignan nuevos valores y se repite el proceso hasta encontrar unos valores satisfactorios.

El valor elegido para la resistencia shunt de la parte de instrumentos es de 0,05Ω. Para un consumo máximo de 6A resulta en una caída de tensión máxima en la resistencia de $0,05\Omega \cdot 6A = 0,3V$ y una potencia de $0,3V \cdot 6A = 1,8W$. La parte de motores llevará una resistencia de 0,01Ωque tendrá una caída de tensión de $0,01\Omega \cdot 17A = 0,17V$. Al primero se le puede aplicar una ganancia de 10 y al segundo de 18 aproximadamente para adaptar esas tensiones diferenciales al rango 0V-3,5V de la entrada del convertidor analógico-digital del microcontrolador.

Los valores de diseño de las resistencias, la ganancia del diferencial y el factor de la caída de tensión están indicados en el cuadro 4.5.

Elemento	**Instrumentación**	**Motores**
$R1$	$33k\Omega$	$18,7k\Omega$
$R2$	$8,45k\Omega \parallel 330k\Omega$	$5,6k\Omega \parallel 330k\Omega$
$R3$	$33k\Omega$	$18,7k\Omega$
$R4$	$330k\Omega$	$330k\Omega$
$R5$	$8,45k\Omega$	$5,6k\Omega$
Ganancia	10	17,647
U/U^+	0,1998	0,2275

Cuadro 4.5: Valores obtenidos para el sensor I/V

4.2.3. Consideraciones prácticas y construcción física

El circuito tiene una alta sensibilidad a la entrada, tanto para desviaciones en las tensiones diferenciales, que es lo que se busca medir, como a variaciones producidas por las tolerancias de las resistencias. Analizando el efecto de la tolerancia de las resistencias comprobamos que pequeñas desviaciones tienen una gran repercusión en la tensión de salida del amplificador diferencial. Haciendo $R_1 = R_3$, la resistencia R_2 es ahora la más sensible, pues tiene que valer exactamente el paralelo entre R_4 y R_5. Sustituyamos su valor por otro un 1 % mayor $R_2' = R_2 \cdot 1,01$ e introduzcámos el nuevo valor en la ecuación (4.8) tomando por ejemplo $U^+ = 12V$ y $U^- = 11,7V$ y usando los valores del resto de resistencias los correspondientes a la parte de instrumentación:

$$U_0 = 10,07986173\Omega \cdot 12V - 10\Omega \cdot 11,7V = 3,96V \tag{4.11}$$

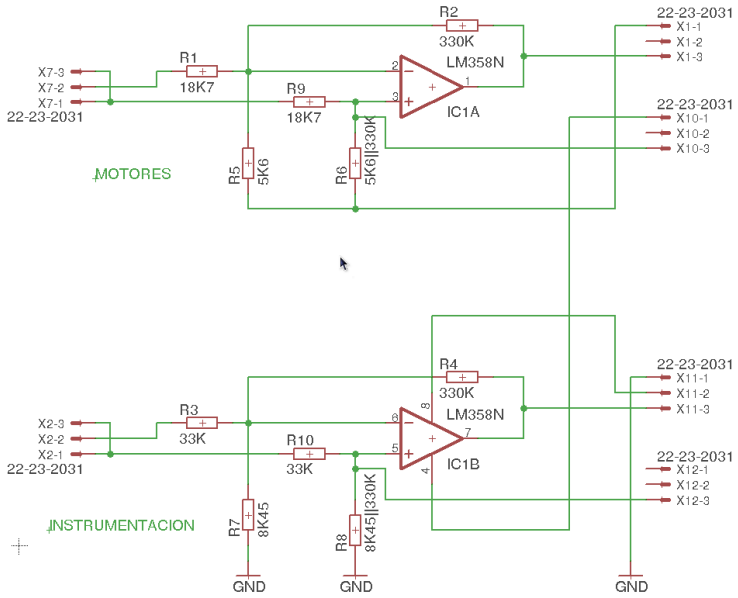


Figura 4.3: Esquema final de los sensores I/V

Comparándolo con el valor que daría sin el error en R_2, que sería de $U_0 = 10\Omega \cdot 12V - 10\Omega \cdot 11,7V = 3V$ se observa que un error de un 1 % en la resistencia R_2 produce desviaciones de casi 1V, es decir, errores un 33 % mayores. Dado que las resistencias comerciales tienen tolerancias de un 10 % o de un 5 % típicamente, no se puede confiar en que colocando una resistencia con el valor teórico exacto el circuito vaya a funcionar.

Para solventar este problema se ha recurrido al hecho de que las resistencias se fabrican en serie y que una resistencia fabricada inmediatamente posterior a otra sera muy parecida por haber tenido practicamente las mismas condiciones de temperatura, presion, etc. Cuando se necesiten coger dos resistencias iguales, se cogera del lote recibido de la empresa suministradora dos resistencias contiguas en la parte central y se comprobara mediante polimetro que tienen el mismo valor. De esta manera, aunque el valor de ambas resistencias tenga una tolerancia de 5 % o 10 % tendran un valor parecido cualquiera que sea este. Es el caso de las resistencias R_1 y R_3 correspondientes a los sensores de la bateria de instrumentacion y de los motores. Para las resistencias cuyo valor teorico se corresponde con el paralelo de dos resistencias se han montado uniendo fisicamente dos resistencias en paralelo obtenidas como se ha descrito en el procedimiento anterior, de esta manera la precision obtenida es la adecuada para las necesidades.

4.3. Montaje y cableado

En esta sección se describe cómo se han montado los componentes hardware que faltaban por integrar físicamente en el robot, al igual que la distribución esquemática del cableado tanto de la alimentación como de los datos.

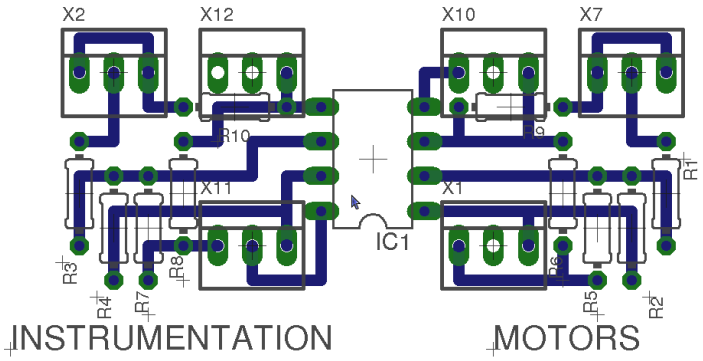


Figura 4.4: Distribución de pistas del sensor I/V

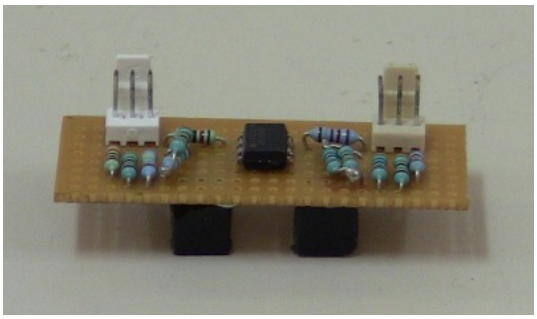


Figura 4.5: Fotografía del convertidor de señal del sensor de tensión e intensidad.

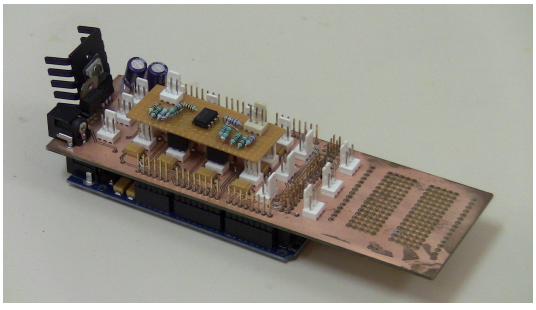


Figura 4.6: Fotografía del sensor i/v acoplado a la tarjeta de adquisición de datos.

4.3.1. Diseño del pórtico

Se ha construido un pórtico en aluminio que proporciona una plataforma elevada donde apoyar la antena receptora de señal GPS y la brújula electrónica (requisitos 2.4.1 y 2.5.3).

El material elegido ha sido el aluminio por sus características mecánicas y sus propiedades electromagnéticas.El peso calculado de toda la estructura aporta únicamente 0,34Kg a todo el conjunto del robot. El aluminio es paramagnético y no interfiere significativamente en el campo magnético del globo terráqueo por lo que la brújula no se ve interferida al usar este material.

Se ha utilizado un perfil en L porque es el el perfil que mejor se ajusta a los elementos ya presentes en el robot. Con este perfil es posible fijar el pórtico a la estructura de metacrilato. esta unión se ha realizado con tornillos. Puesto que el metacrilato es un material frágil, se ha utilizado arandelas de nylon que se adapta a las rugosidades superficiales del metacrilato y el aluminio y distribuye las presiones por toda la superficie con la que está en contacto, reduciendo el riesgo de concentración de tensiones y rotura del metacrilato. También se han usado tuercas con nylon que evitan el fallo accidental de la unión por vibraciones.

Los datos del pórtico se presentan en el cuadro 4.6

Altura	502mm
Anchura	335mm
Espesor de perfil	3,3mm
Anchura del perfil	19,2mm
Peso del conjunto	0,34Kg
Tornillería	M5x25
Material de las arandelas	Nylon
Tipo de tuerca	Con retención de nylon

Cuadro 4.6: Datos de diseño del pórtico

4.3.2. Montaje de módulos

GPS

La estación móvil del GPS se montó de forma definitiva sobre el robot. Se adaptó la superficie del robot base para la fijación de la radio en el lado derecho del láser. La antena se atornilló al pórtico. La caja de control y la batería de la radio se fijaron al interior de la estructura de metacrilato posterior.

La estación base no se ha instalado de forma definitiva, sino que ha de prepararse en una ubicación temporal con cada uso. Esta preparación está descrita en el manual del usuario, anexo C.

Sensores de la tarjeta de adquisición de datos

La brújula electrónica se montó sobre el pórtico de la manera que ya se ha descrito. Se conectó a la tarjeta de adquisición de datos con un cable plano de tres conductores portando alimentación y la señal PWM con la codificación de la orientación del sensor.

Los acelerómetros se colocaron detrás de la tarjeta de extensión de conectores del arduino, entre ésta y la estructura de metacrilato que soporta el láse, donde encuentra protección física y a la vez se puede observar su correcta colocación, que por su tamaño y bajo peso se ha realizado con velcro. Junto al servant del arduino desarrollado en el capítulo 6, se realizará el requisito 2.4.2.

Las resistenacias shunt se han montado:

1. La primera, que controla las baterías de dispositivo a la entrada de la tarjeta de alimentaciones, al aire con la sujeción de los cables.

2. La segunda, que controla las baterías de la base del robot, sobre la puerta de acceso a las baterías, por el lado interior. Se accede a él desmontando la base del robot y la tarjeta electrónica de control de los motores.

4.3.3. Cableado de datos

Se ha tratado de reducir al máximo el impacto visual del cableado sin que reduzca la accesibilidad a éstos para mantenimiento, reparaciones o reconfiguraciones. El diagrama simplificado del cableado de datos se muestra en la figura 4.7. Las flechas señalan el flujo de información útil entre dispositivos. También se ha incluido la transmisión de datos inalámbrica del sistema GPSd, no así las conexiones inalámbricas con la red del laboratorio.

4.3.4. Cableado de alimentaciones

El cableado de alimentaciones está centrado alrededor de la tarjeta de alimentaciones. En la figura 4.8 se mueestra un esquema del cableado de alimentaciones. Cada bloque representa el conjunto de dispositivos que alimenta cada una de las baterías presentes en el robot. Los elementos hexagonales representan las baterías y los elementos cuadrados, los dispositivos que necesitan ser alimentados. La tarjeta de adquisición de datos, o tarjeta de adquisición de datos, se ha representado entre bloques pertenecientes a baterías distintas porque puede ser alimentado desde USB y usando la batería de dispositivos.

Algunos dispositivos usan el mismo cable simultáneamente para la transimisión de datos y para la alimentación. Estos cables se han indicado con linea discontinua.

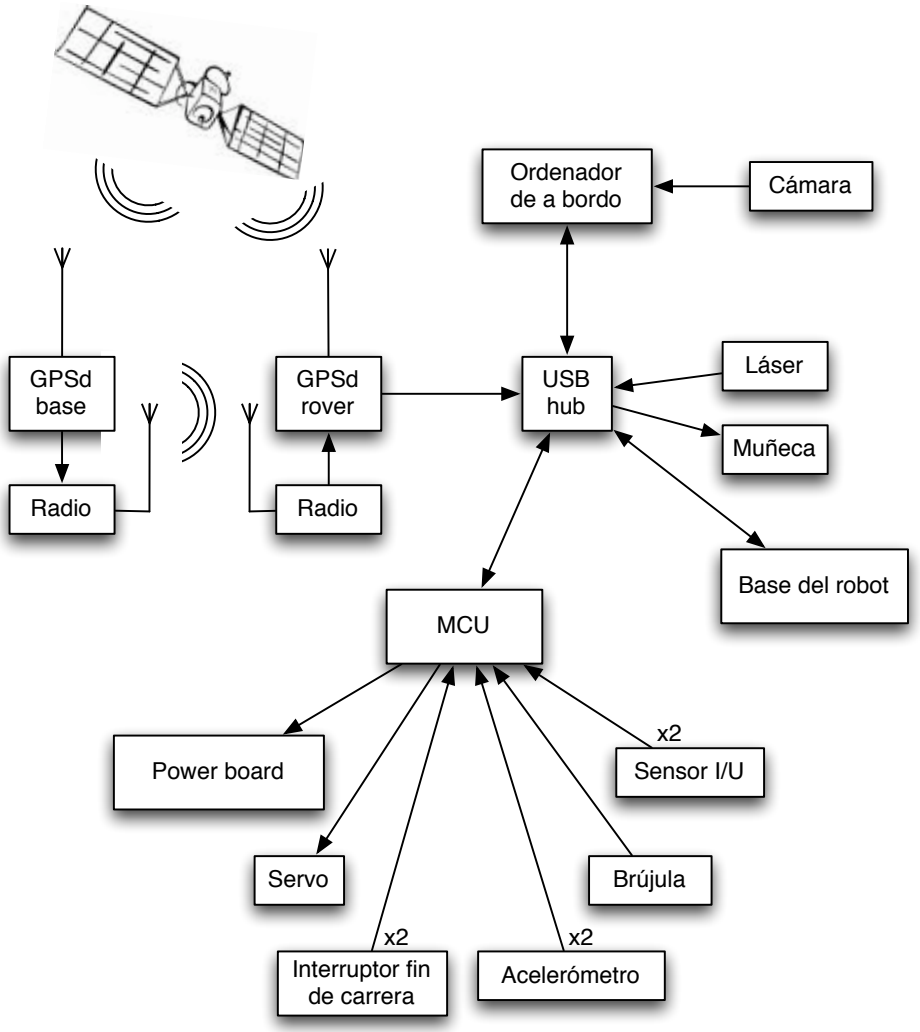


Figura 4.7: Diagrama de conexiones de datos.

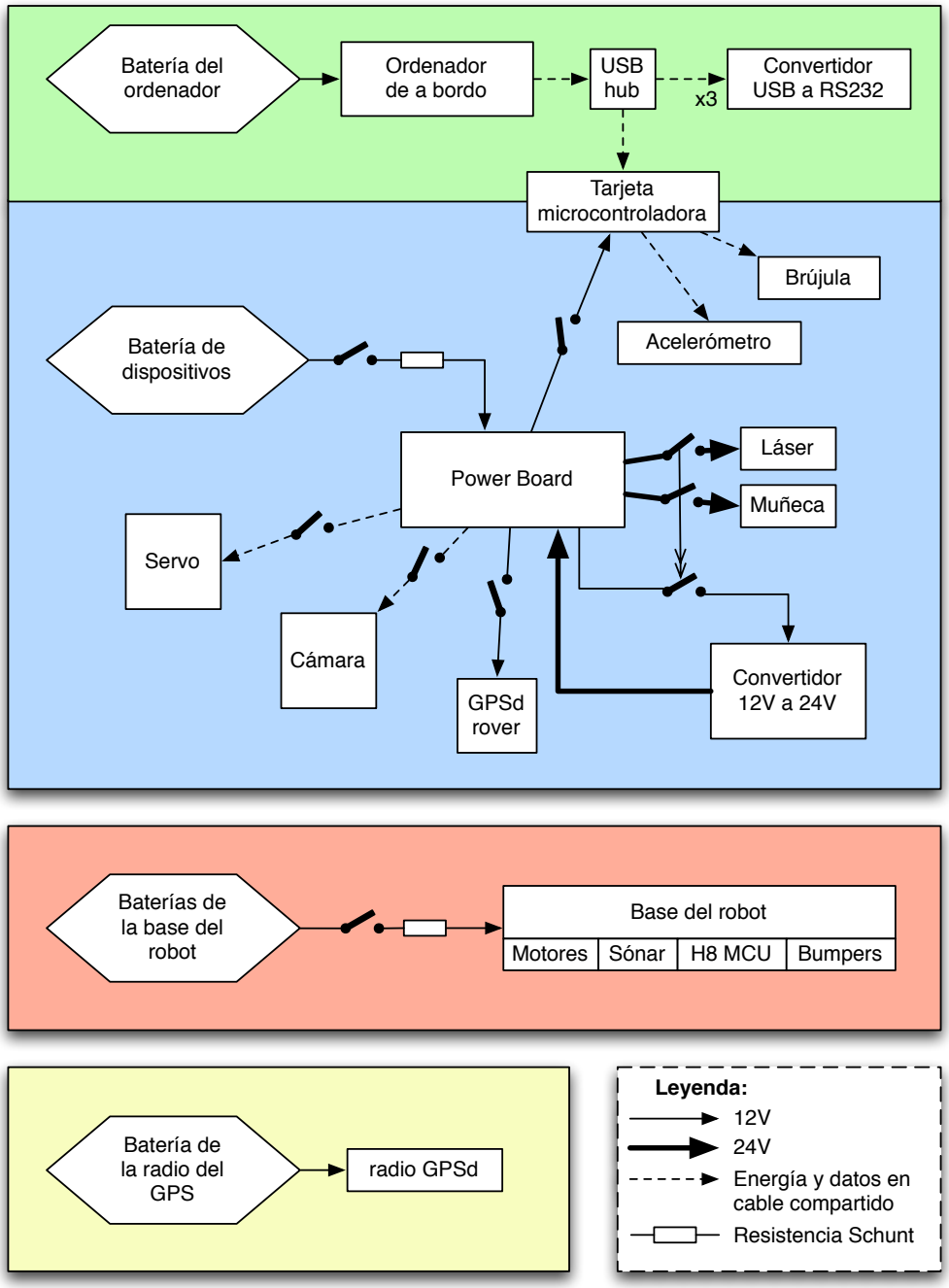


Figura 4.8: Diagrama de conexiones de alimentaciones.

Capítulo 5

Entorno software de trabajo

Dado el número de módulos con los que se ha trabajado y la similitud entre ellos, ha sido necesario convenir la organización de los módulos. El código fuente se ha almacenado en el repositorio SVN (ver anexo E), y ha sido también utilizado para el intercambio y la actualización del código. Al no ser una herramienta de comunicación, la publicación de nuevas versiones, cuando así se ha considerado oportuno, se ha realizado personalmente, si sólo estaba involucrada una persona, o por correo electrónico cuando esa persona no estaba presente o había más personas involucradas. Ésto ha sido contemplado en el requisito 1.

5.1. Subversion

Una de las primeras recomendaciones que se planteó fue mudar el repositorio de software de CVS, o Concurrent Versioning System, a SVN, abreviatura de Subversion. Los motivos que llevaron a este cambio fueron:

- Posibilidad de versionar directorios.
- Posibilidad de mover, borrar y copiar archivos sin renunciar al historial de cambios.
- Cada cambio que se hace define un nuevo árbol completo en el repositorio, lo que evita actualizaciones incompletas.
- Mejor integración general y robustez.
- Mayor flexibilidad en la configuración del servidor y la administración del repositorio.

Todo ello llevó a migrar el código, la documentación y el resto de archivos contenidos en CVS a SVN, tal y como hicieron en su día la mayoría de proyectos que usaban CVS como sistema de control de versiones. No obstante se ha mantenido el servidor CVS con el repositorio completo por motivos de archivado histórico.

5.2. Sistema de compilación CMake

CMake es una herramienta que automatiza la tarea de construir, probar y empaquetar software. Es multiplataforma y puede generar los archivos de compilación necesarios para una multitud de entornos de desarrollo, tales como Makefiles, Eclipse y entornos Windows. El

proceso de compilación del software se controla mediante archivos de configuración textuales sencillos. CMake es el sistema de compilación de elección para los grandes proyectos de código abierto y su adopción es cada vez mayor.

Antes de la implantación de este sistema de compilación los módulos se compilaban directamente desde la consola, con Makefiles generados manualmente o usando el ya obsoleto GNU/Autoconf y GNU/Automake, con todos los problemas que acarrea. Se decidió cambiar el sistema de compilación en todos los módulos para unificar y simplificar la tarea de trabajar con los módulos, facilitando el desarrollo y el mantenimiento del código al tener que trabajar con una única herramienta de compilación. El nuevo sistema mejorará el cumplimiento de los requisitos 1 y 3.

Igualmente CORBA plantea una serie de dificultades en cuanto a la generación del código de los *stubs* y los *skeletons*. A continuación se presenta una extensión que se ha hecho en el propio lenguaje de archivos de configuración de CMake que permite a esta herramienta compilar los archivos IDL al lenguaje C++ y controlar las modificaciones:

```
1  MACRO (MacroGenerateIDL)
2    IF(NOT IDL_COMPILER)
3      SET(IDL_COMPILER tao_idl)
4    ENDIF(NOT IDL_COMPILER)
5    IF(NOT IDL_DIR)
6      SET(IDL_DIR ../../idl)
7    ENDIF(NOT IDL_DIR)
8    FOREACH(_in_FILE ${ARGN})
9      GET_FILENAME_COMPONENT(_base_FILE ${_in_FILE} NAME_WE)
10     ADD_CUSTOM_COMMAND(
11       OUTPUT ${_base_FILE}S.h ${_base_FILE}S.cpp ${_base_FILE}S.inl
             \
12               ${_base_FILE}C.h ${_base_FILE}C.cpp ${_base_FILE}C.
                 inl
13       MAIN_DEPENDENCY ${IDL_DIR}/${_base_FILE}.idl
14       COMMAND ${IDL_COMPILER} -I${IDL_DIR} ${IDL_DIR}/${_base_FILE
             }.idl
15       )
16   ENDFOREACH(_in_FILE ${ARGN})
17 ENDMACRO (MacroGenerateIDL)
```

Ahora es posible crear nuevos módulos y que la tarea de crear las dependencias y compilar el código CORBA sea casi inmediato. A continuación se presenta un archivo típico de configuración de CMake:

```
1  # Example configuration file for CMAKE.
2  # Substitute names beginning with _
3  cmake_minimum_required(VERSION 2.6)
4  project(_module)
5
6  set(CORBA_LIBS TAO_RTCORBA TAO_RTPortableServer TAO_PortableServer
       TAO_CosNaming TAO ACE TAO_AnyTypeCode)
7  # Remove these two lines if not using ARIA.
```

```
8  include_directories(/usr/local/Aria/include/)
9  link_directories(/usr/local/Aria/lib/)
10
11 set(CMAKE_CXX_FLAGS "-Wall -fPIC")
12 set(CMAKE_BUILD_TYPE Debug)
13
14 # CORBA related sources.
15 INCLUDE(../../lib/IDL_command.cmake)
16 MacroGenerateIDL(CosNaming _module)
17 set(CORBA_SOURCES _moduleC.cpp _moduleS.cpp CosNamingC.cpp)
18
19 # Client.
20 add_executable (_module_client _module_client.cc ${CORBA_SOURCES})
21 target_link_libraries (_module_client ${CORBA_LIBS})
22
23 # Servant.
24 add_executable (_module_server _module_server.cc ${CORBA_SOURCES})
25 target_link_libraries (_module_server ${CORBA_LIBS} Aria pthread dl
       )
26
27 # Installation of servant on onboard computer.
28 set(CMAKE_INSTALL_RPATH_USE_LINK_PATH TRUE)
29 install_programs(/../../../../../../etc/higgs/servants
       _module_server.sh)
30 install_targets(/bin _module_server)
```

CMake también se ha usado para la generación de documentación de código con Doxygen (Anexo E).

5.3. Librerías comunes

Los módulos incorporan elementos comunes en su programación que han sido recogidos e implementados en un único lugar. Las dos ventajas principales de compartir código son, primero, que se reduce el tiempo de programación en cada módulo, evitando reinventar la rueda y reduciendo el número de líneas, que facilita la revisión del código. Segundo, que si se hace necesario modificar este código común con realizarlo una vez basta y los cambios se propagan a todos los módulos cuando se compilen, o si la modificación es en la configuración, cuando se reinicien. Aunque la necesidad de ésta librería ha surgido después del incio del proyecto, se enmarca dentro de los requisitos 1 y 3.

El código fuente de esta librería se puede encontrar en el cedé que acompaña al proyecto dentro del archivo

`code/lib/CORBA_utils.h`

A continuación se lista el código de un hipotético servant que se ha tomado como referencia para construir los módulos.

```
1 // A typical servant executable would be something like:
```

```
2
3  #include "implementation.h"
4  #include "CosNamingC.h"
5  #include "../../lib/CORBA_utils.h"
6
7  int main(int argc, char* argv[])
8  {
9      CORBA_BEGIN_SERVER(argc, argv);
10
11     implementation_t impl();
12     higgs::implementation_t_var implvar = impl._this();
13
14     CORBA_REGISTER_REFERENCE(implvar, "IMPL");
15     CORBA_END_SERVER;
16     return 0;
17 }
```

La implementación CORBA que se ha usado obliga a pasar como argumentos del ejecutable la dirección del servidor de nombres. Los servants CORBA existentes al inicio del proyecto resolvían esta situación ejecutando en lugar del ejecutable un script conteniendo la llamada al ejecutable del cliente o servant y la dirección del servidor de nombres como argumento, más el puerto de escucha para el caso de los servants. Usando esta librería se eliminó la necesidad de este script, duplicado por cada ejecutable creado, sin necesidad de modificar el código de ningún módulo y pudiendo realizar cambios en la configuración de todos los servants con sólo modificar un archivo (ver sección 5.5).

5.4. Sistemas operativos en el ordenador de a bordo

Se probaron distintos sistemas operativos antes de decidir el más adecuado para cumplir el requisito 1.1. Los requerimientos por los que se selección el sistema operativo fueron:

- Posibilidad de núcleo en tiempo real (requisito 1.1.1).
- Amplia biblioteca de librerías.
- Facilidad de instalación y uso.
- Compatibilidad con el código ya desarrollado.

A continuación se muestra una lista con los sistemas operativos probados junto con las ventajas y desventajas de cada uno de ellos.

Windows XP Es el sistema operativo por defecto que venía incluido con el ordenador. Se incluye licencia. Windows XP no trae características de tiempo real en la distribución estándar y sus librerías son propietarias. No es compatible con gran parte del código ya desarrollado por estar hecho para sistemas basados en UNIX. Configurar este sistema operativo para tareas empotradas es dificultoso, sin embargo existen muchas implementaciones CORBA, la mayoría de ellas propietarias.

WindRiver VxWorks Éste es el sistema operativo usado hasta la fecha. Está basado en UNIX (cumple POSIX), tiene un buen soporte para tiempo real y está diseñado para ser usado en sistemas empotrados. Su mayor inconveniente reside en la dificultad para su instalación, por la que hay que crear un entorno de desarrollo cruzado para compilar las herramientas y servicios desarrollados. Es necesario solicitar licencia a la casa comercializadora.

Fedora Linux Se trata de un sistema operativo de fácil instalación y uso para usuarios que no necesariamente tienen altos conocimientos en administración de sistemas UNIX. Está enfocado hacia un entorno gráfico con una carga alta sobre el procesador y la memoria, lo que lo hace demasiado complejo para sistemas empotrados. No es tiempo real.

Orquestra Orquestra Control Engine es un sistema operativo basado en Ubuntu Linux de libre distribución, preparado para ser utilizado en aplicaciones industriales y de control de robots. Es de fácil instalación, incluye un núcleo de tiempo real y múltiples librerías y utilidades de control de robots, creación de entornos virtuales, cálculo cinemático inverso y filtros. Además está siendo usado por otros grupos de investgación dentro del departamento.

Ubuntu Linux + RTAI Se trata de la distribución Ubuntu Linux, una de las más usadas en ordenadores de esritorio y portátiles, a la que se le añade un núcleo de tiempo real. Hay mucha documentación y tutoriales que facilitan la modificación de la distribución para hacerlo en tiempo real.

Finalmente se decidió instalar Ubuntu Linux 10.04 LTS, con soporte oficial durante tres años, hasta el 2013, usando como núcleo el kernel de linux con los parches RTAI. El proceso seguido para descargar, configurar, compilar e instalar el núcleo se describe en el manual del desarrollador, anexo D.

5.5. Configuración de los módulos CORBA

Ésta sección resuelve las necesidades planteadas por el requisito 1.1.2.

5.5.1. Parámetros de los servants

Se ha creado un árbol de directorios donde se han introducido los archivos de configuración necesarios para iniciar los servants. Éste directorio debe ir ubicado en

```
/etc/higgs
```

en todos los ordenadores que vayan a ejecutar componentes CORBA, ya sean clientes o servants. Los clientes únicamente necesita el archivo con la dirección del servicio de nombres, mientras que en el ordenador de a bordo se han necesitado todos ellos. El árbol resultante es el siguiente:

```
higgs
|-- devices
```

```
|   |-- arduino_dev -> /dev/ttyUSB2
|   |-- higgs_dev -> /dev/ttyUSB0
|   |-- wrist_dev -> /dev/ttyUSB3
|   `-- laser_dev -> /dev/ttyUSB1
|-- listen\_endpoint.ip
|-- modules
|   |-- ftdi\_sio.ko
|   `-- pl2303.ko
`-- nameservice.ip
```

A continuación se hace una breve explicación de cada entrada:

devices Subdirectorio que contiene enlaces simbólicos a los archivos de dispositivos en `/dev`. Los servants abren estos archivos en lugar del dispositivo real para no tener que recompilarlo cada vez que el dispositivo cambia de nombre, hecho muy frecuente con los convertidores USB a RS-232.

listen_endpoint.ip Dirección IP de la interfaz de red por el que se desea que se pongan a la escucha los servants.

modules En este caso *modules* se refiere a módulos recargables del núcleo de linux. Los drivers de los convertidores USB a RS-232 cargan en orden arbitrario cada vez que se reinicia el ordenador de a bordo, lo que provoca cambios en los nombres de los archivos de dispositivo. Para evitarlo se han movido los drivers a este directorio, evitando que el núcleo los cargue automáticamente. Se ha añadido una instrucción de carga manual de estos drivers en la secuenca

nameservice.ip Dirección IP del servicio de nombres. Único archivo obligatorio para los ordenadores que solo ejecuten clientes. de arranque del sistema operativo, de este modo los drivers se cargan siempre en el mismo orden.

5.5.2. Ejecución de los servants

El requisito 2.3 requiere que los servants se recuperen automáticamente tras un fallo. Los servants se encargan de abortarse a sí mismos, pero el arranque debe realizarse a más bajo nivel, desde el sistema operativo. Se ha usado la utilidad de arranque de los servicios del sistema operativo Ubuntu, Upstart. Ésta utilidad es un reemplazo del antiguo sistema Init V. Se usan archivos de configuración en lugar de entradas en el árbol de directorios de arranque del sistema y de esta manera lograr mayor flexibilidad en las opciones de arranque de los demonios mientras se mejor el control de los mismos. Estos archivos se guardan en el directorio

`/etc/init`

Típicamente el archivo de configuración de ejecución de los módulos será como el mostrado a continuación del láser que se toma como ejemplo. Nótese que se redirecciona la salida estándar y de error al archivo de log, pero no la dirección del servicio de nombres de CORBA:

```
description "Upstart config file for the arduino servant"
author "Francisco J. Arjonilla García"
#start on started Naming_Service
respawn
script
  sleep 5
  date >> /var/log/higgs/laser.log
  su -l -c /usr/local/bin/laser_server higgs >> /var/log/higgs/laser.log 2>&1
end script
```

El servant java de la tarjeta de adquisición de datos no usa las macros C++ que automatizan la configuración del broker de CORBA por lo que hay que especificarlo manualmente en el archivo de configuración del servant:

```
description "Upstart config file for the arduino servant"
author "Francisco J. Arjonilla García"
start on started Naming_Service
respawn
script
  sleep 5
  date >> /var/log/higgs/arduino.log
  su -l -c "java -jar /usr/local/bin/arduino.jar \
 -ORBInitRef Naming_Service=corbaloc::$(cat /etc/higgs/nameservice.ip)\
/NameService" higgs >> /var/log/higgs/arduino.log 2>&1
end script
```

El servidor de nombres de CORBA necesita tener fijados la dirección y el puerto por los que escuchar. El archivo de configuración es:

```
description  "CORBA Naming Service"
start on net-device-up
respawn
exec Naming_Service -ORBEndPoint iiop://higgs2.disam.etsii.upm.es:9876
```

Capítulo 6

Módulos software

En este capítulo se describe la parte software de los módulos. Todos ellos están encapsulados en un objeto CORBA que implementa las interfaces IDL creadas como consecuencia del requisito 2.1.

6.1. Muñeca

El desarrollo del módulo de la muñeca viene a cumplir con el requisito 2.5.1.2 y se ha centrado en el desarrollo del servant. Al inicio del proyecto se encontraba instalada sobre la estructura de metacrilato que soporta el láser pero preparada para ser alimentada externamente y controlada desde ordenadores externos. Se terminó de integrar al robot adaptándolo a la tarjeta de alimentaciones y a los convertidores USB a RS-232.

6.1.1. Servant

El desarrollo del módulo se inició a partir de un código que lograba conectar con la muñeca y aplicar comandos sencillos de movimiento. La interfaz de este código no era suficientemente buena para trabajar con ella, por lo que se empezó reorganizando y reestructurando el código. Tras unos días de trabajo se decidió que sería más sencillo comenzarlo desde cero usando la documentación para la elaboración del protocolo y tomando como referencia de código en funcionamiento el código antiguo. Se inició analizando el protocolo y diseñando unas clases, aún sin la complejidad de CORBA, que facilitaran su implementación. En cuanto se consiguió la comunicación con la misma secuencia de envío de bytes por el puerto serie, la extensión del protocolo a la funcionalidad completa de la muñeca se aceleró, gracias a la nueva estructura de clases, que se puede observar en la figura 6.1.

Conseguida la comunicación con la muñeca se escribió la interfaz IDL y se implementó el servant usando el código descrito en el párrafo anterior.

6.1.2. Tolerancia a fallos

Se ha logrado una muy buena tolerancia a fallos en este servant. Los fallos que se tienen en cuenta son:

1. Tensión de alimentación fuera del rango admitido.

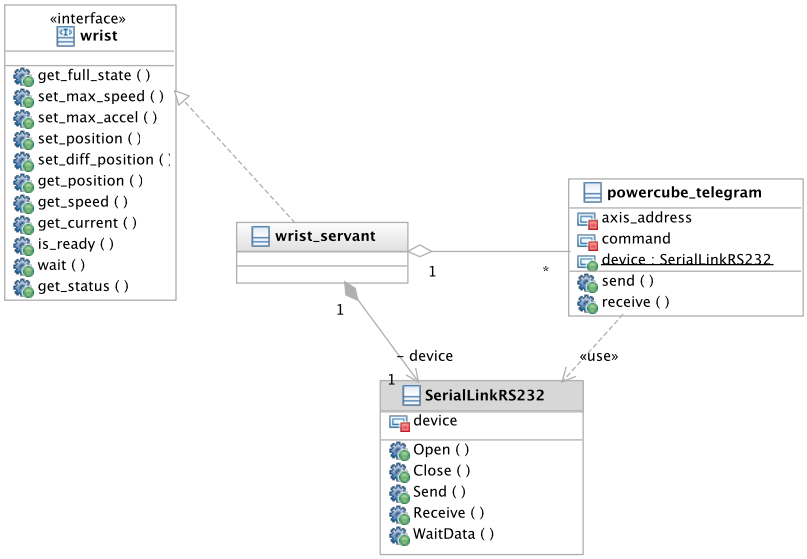


Figura 6.1: Diagrama de clases de la muñeca.

2. Sobrecorrientes.

3. Dispositivo apagado.

En los tres casos el servant detectará la condición de error y abortará la ejecución, a la espera de que se reinicie y vuelva a intentar abrir la comunicación.

6.2. Láser

A continuación se expone las tareas desarrolladas en el ámbito del requisito 2.5.2

6.2.1. Selección del código base

El código base desde el que se partió está basado en la librería ARIA[1] y usa algunas clases con funciones avanzadas de cálculo estadístico para la determinación de segmentos rectos en los datos de entrada, en concreto `ArSick` y `ArLineFinder`. Éste código se ha usado satisfactoriamente en [11] en el cual no es necesario capacidades de curación del sistema.

La integración de este módulo CORBA ya realizado tropezó con la necesidad de recuperarse de un fallo o de una desconexión del sensor, por lo que no pudo integrarse sin previa adaptación, lo que constituiría la primera opción para tener un módulo láser funcional. Por otra parte se disponía del código creado a medida por los compañeros del laboratorio de robótica móvil consistente en dos unidades de compilación. La tercera opción consistió en programar un nuevo driver a medida de las necesidades del robot. A continuación se describen brevemente las ventajas y desventajas de cada opción.

- Librería ARIA. Como ventaja fundamental se puede citar que ya está integrado dentro de un servant CORBA probado y en funcionamiento y que dispone de funciones para

[1] Se puede descargar más información sobre esta librería en `http://robots.mobilerobots.com/wiki/ARIA`

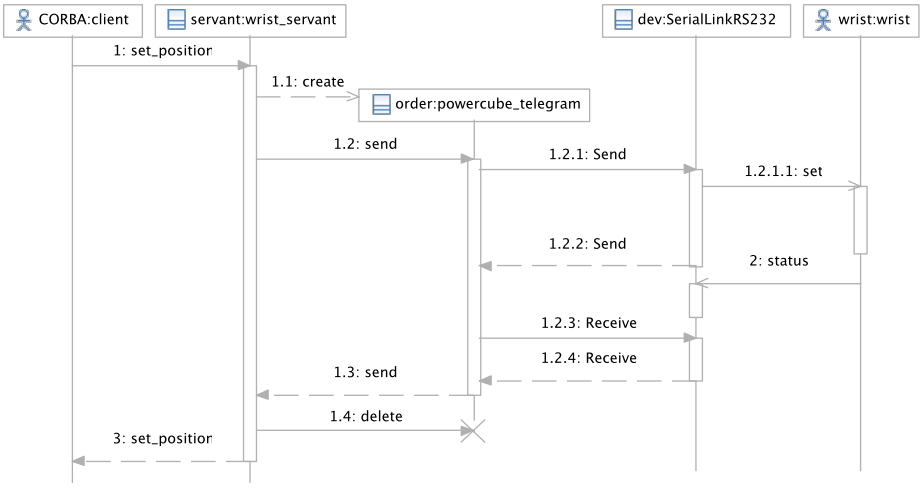


Figura 6.2: Diagrama de secuencia típica para una llamada al servant de la muñeca.

la determinación de segmentos rectos. En contra, que no dispone de algoritmos para resincronizar el protocolo en caso de fallo del hardware o del servant, que depende de una librería de gran tamaño poco manejable y que el esfuerzo de adaptación requiere estudiar el código fuente de la librería. Además, habría que actualizar la librería con cada nueva versión que apareciese y llevar registro de los cambios. Adicionalmente, esta librería filtra algunos mensajes perdiendo parte de la información de las distancias en cada lectura y no está preparado para funcionar a la máxima velocidad que permite el sensor. Tampoco proporciona métodos para controlar condiciones de error en el protocolo sino que simplemente ofrece la última lectura tomada como válida por antigua que sea.

- Driver proporcionado por compañeros de robótica móvil. Dispone de la funcionalidad completa que ofrece el sensor pero no puede recuperar el sincronismo. No tiene dependencias con librerías externas (es autocontenido) y está preparado para ser enlazado estáticamente.

- Empezar un nuevo driver desde cero. Se pierde la ventaja de partir de un sistema ya probado y en funcionamiento para repetir un trabajo de programación que ya está realizado. A favor, que el código resultante se ajustaría perfectamente a los requerimientos del proyecto.

En primer lugar se programó una serie de funciones capaces de conectar con el láser y recuperar el sincronismo independientemente de su estado inicial, con idea de usar este código tras la selección del código base. Se decidió abandonar la librería ARIA en favor de un código más controlable y compacto donde la robustez del protocolo quedara garantizada, no solo en la ejecución sino en la instalación y facilidad de mantenimiento del código. Las funciones desarrolladas se introdujeron en las rutinas de inicialización del driver proporcionado por los compañeros de róbotica móvil aprovechando así la posibilidad de cambiar la frecuencia del puerto serie y los parámetros de sensado del láser.

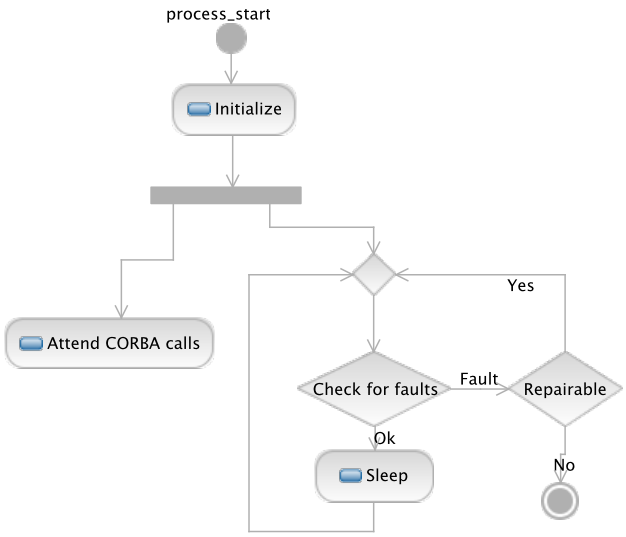


Figura 6.3: Gestión de errores en la muñeca.

6.2.2. Servant

Finalmente se creó el servant CORBA usando este driver y el cliente de prueba. El servant adapta los valores de entrada a la convención matemática usada a lo largo del proyecto: Radianes para referirse a los ángulos en sentido positivo con referencia vertical hacia arriba y origen mirando al frente, y las distancias en metros. Existe un hilo específico para manejar el protocolo del láser y otro para atender las llamadas de los clientes CORBA, teniendo la implementación del servant un buffer intermedio en el que los dos hilos se transmiten la información del sensor.

A continuación se presenta la parte desarrollada del driver capaz de resincronizarse bajo todas las frecuencias del protocolo y en cualquier punto de éste:

```
1   char cadena[100];
2   int baud[]={9600,38400,500000};
3   //  int baud[]={9600,38400};
4   // int baud[]={9600};
5   int i=0;
6   int size=sizeof(baud)/sizeof(int);
7   for(i=0;i<=size;i++)
8     {
9   int bps;
10  if (i == size) // In case a Reset() returns the laser to 9600
      bps.
11      bps = baud[0];
12  else
13      bps = baud[i];
14  port.SetBaud(bps);
15  printf("Trying to connect at %d baud.\n",bps);
```

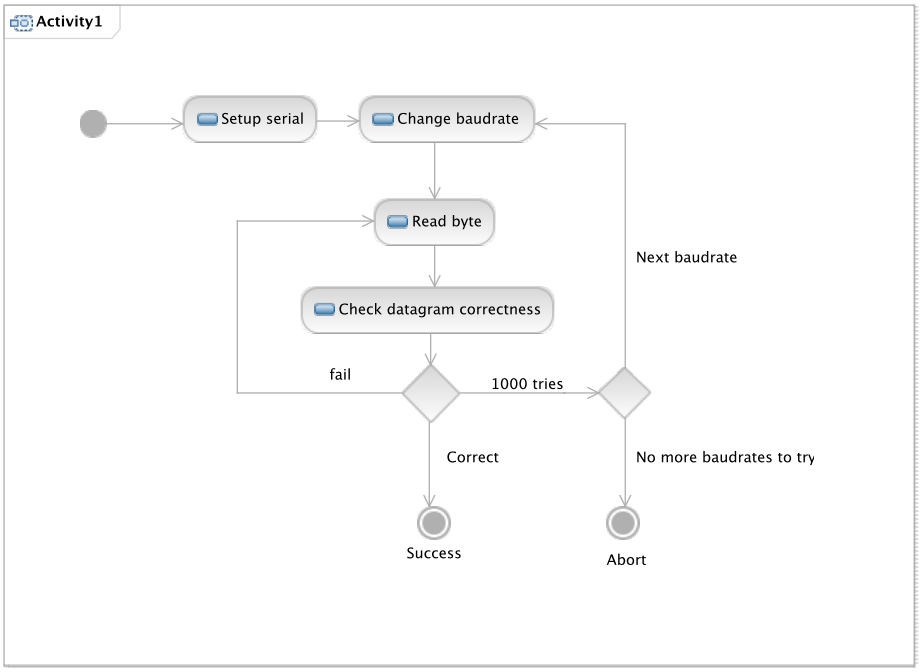


Figura 6.4: Procedimiento de inicialización del protocolo del sensor láser.

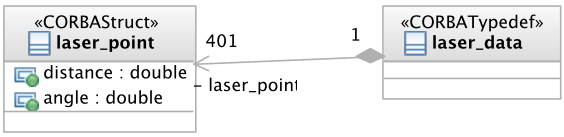


Figura 6.5: Estructuras de datos definidas en la interfaz IDL del láser.

```
16      // Try to syncronize from a previous program crash.
17      unsigned char c[5];
18      memset(c, 0, 5);
19      int j;
20      for (j = 0; j < 1000; j++)
21        {
22          for (int k = 0; k < 4; k++)
23          c[k] = c[k + 1];
24          if (!port.Receive(1, c+4))
25            {
26          cout << "No data received. Synchronization failed." << endl
                ;
27          break; // Receive failed. We try to set up the laser.
28            }
```

```
29          int msg_length = 0;
30          if (c[0] == 0x02 && c[1] == 0x80 && c[4] == 0xb0)
31            {
32          msg_length += c[2] + 256*c[3] - 1 + 2;
33          unsigned char dump[1000];
34          cout << "Telegram detected with length " << msg_length <<
35              ". Waited for " << j << " characters." << endl;
36          if (msg_length >= 1000 || port.Receive(msg_length, dump) !=
                msg_length)
37            {
38              cout << "Telegram not fully received. Synchronization
                    failed."
39             << endl;
40              break; // Receive failed. We try to set up the laser.
41            }
42          Stop();
43          break; // Receive successful. We stop and reset the
                configuration.
44            }
45        }
46      if (j == 1000)
47        {
48          cout << "Data stream received, but no header found. Trying
                to reset."
49         << endl;
50          Reset();
51        }
52      int pru=LMSType(cadena);
53      if(pru!=PROT_LMS_OK)
54        {
55          printf("Connection at %d baud failed. Reason: %d\n",bps,pru
                );
56        }
57      else
58        {
59          printf("Connection at %d succesful\n",bps);
60          break;
61        }
62        }
63      if(i>=size)return PROT_LMS_ERROR;
```

6.3. GPSd

El desarrollo del módulo GPS está motivado por el requisito 2.5.3 y 2.5.3.1 y se ha dividido en tres partes: Instalación mecánica, análisis del protocolo y desarrollo del servant. La instalación mecánica se ha contemplado en el capítulo 4 En primer lugar se descifró el

protocolo de comunicación, estudiando los manuales del GPS, conectando por el terminal usando la utilidad estándar de UNIX *minicom* y realizando pruebas hasta conseguir capturar la posición.

6.3.1. Servant

El desarrollo del servant se realizó como el resto de módulos. Se utlizó el sistema CMake para el sistema de compilación y se aprovecharon las librerías comunes para el desarrollo del servant. La interfaz incluye métodos para las siguientes funciones:

1. Posición en coordenadas geográficas.
2. Velocidad en m/s.
3. Desviación estándar de la posición.
4. Número de satélites usados en la solución.
5. Tipo de solución (Ver tipos en figura 6.6).

El funcionamiento interno del módulo es similar al de la tarjeta de adquisición de datos y se basa en la existencia de dos hilos. El primero de ellos está bajo control del broker de CORBA y atiende llamadas de objetos remotos volcando la información de variables intermedias en los valores de retorno. El segundo hilo es el más complejo y es el que inicializa la caja de control del GPS para después quedarse a la escucha y decodificar los mensajes del GPS, que son enviados cada segundo.

Un tercer hilo asegura que las comunicaciones entre el servant y la caja de control siguen activas. Su funcionamiento es similar al de un *watchdog*: Se habilita un temporizador que con cada mensaje recibido se reinicia. Si el contador pasa un límite, que en el caso de este módulo está establecido en 5 segundos, fuerza la finalización del proceso y consecuentemente el reinicio, si el archivo de configuración de *upstart* está correctamente instalado.

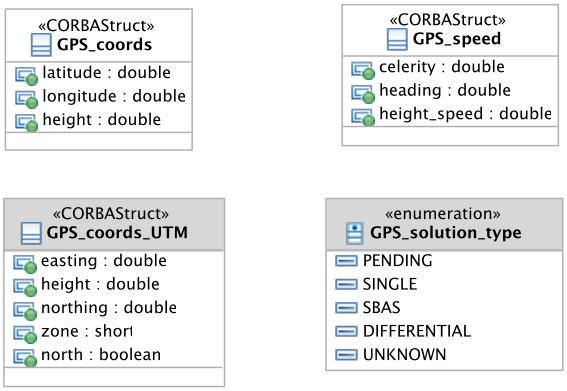


Figura 6.6: Tipos de datos retornados por llamadas al servant del GPS.

6.4. Tarjeta de adquisición de datos

La tarjeta de adquisición de datos presenta una serie de desventajas que hace necesario su modificación. Se ha tenido que estudiar en detalle el sistema de partida, realizando ingeniería inversa, debido a la escasa documentación que acompaña el código.

1. La interfaz gráfica impide la ejecución del servant en ordenadores empotrados, sin sistema gráfico o con recursos limitados.
2. Algunas de las órdenes enviadas a la tarjeta son ignoradas.
3. El código del firmware no es lo suficientemente estable como para asegurar un funcionamiento continuado.
4. La interfaz IDL es mejorable.
5. La configuración e inicialización del servant es difícil y laboriosa.
6. El conjunto no es robusto y necesita monitorización constante por parte del operador.

Éstos defectos vienen motivados de la naturaleza del desarrollo anterior que se limitaba a una fusión sensorial de algunos dispositivos del robot. El desarrollo realizado en este proyecto mejora este módulo y lo amplía, habilitando algunos sensores que no existían a la hora de realizar el proyecto.

Los sensores y actuadores que se han añadido son:

1. Brújula electrónica.
2. Acelerómetros.
3. Apagado remoto de dispositivos, ver sección 4.1.
4. Sensores de medición de corriente e intensidad de las baterías, ver sección 4.2.

Para ello se ha modificado tanto el firmware como el servant.

6.4.1. Firmware

El firmware ha necesitado un repaso general de las funciones que implementa. En primer lugar se eliminó el código irrelevante que únicamente reducía el rendimiento del procesamiento. En segundo lugar se añadió la captura de los nuevos sensores, habilitando nuevos tipos de datos de entrada. Se añadió la captura de tiempos para el PWM enviado por la brújula electrónica, añadiendo esta captura al ciclo de lectura de los registros correspondientes a cada sensor. Finalmente se adaptaron los comandos a las nuevas funcionalidades y se eliminaron funcionalidades obsoletas no aplicables al conjunto del robot.

Además durante todo el proceso se fueron corrigiendo pequeños errores así como mejorando la estructura para facilitar el mantenimiento futuro del código. En el manual del desarrollador, anexo D se ha documentado en detalle los pasos a seguir para realizar actualizaciones del firmware.

6.4.2. Servant

En primer lugar se ha adaptado el código para poder ejecutarse en modo demonio, es decir, sin interfaces con las que interaccionar. Esto permite iniciar el servant con el arranque del sistema operativo y es necesario para lograr mayor autonomía del módulo, limitando las interacciones a la interfaz CORBA ofrecida y el registro de los mensajes en los archivos de log.

Se ha realizado un estudio de las necesidades del grupo ASLab con la funcionalidad que debe ofrecer este módulo y se ha modificado la interfaz IDL acorde a los métodos requeridos. Ello debe ir acompañado por la modificación correspondiente en el código del servant, que está escrito en Java. Una implementación ideal usaría un lenguaje que no necesitara una máquina virtual y así aprovechar el tiempo de procesamiento de la CPU, que es escaso debido a los requisitos de tiempo real y de procesamiento de los servants del resto de módulos.

Se buscó el motivo por el cual se ignoraban algunos mensajes y el diagnóstico resulto ser que la estructura de datos usada para comunicar los hilos del servant y de escucha del puerto serie era sobreescrita antes de ser enviada a la tarjeta. Se resolvió sustituyendo la estructura de datos por una cola de órdenes a enviar. Esta cola crece con cada solicitud recibida por el hilo del servant y se reduce cada vez que el hilo de escucha del puerto serie envía la orden correspondiente a la tarjeta (figura 6.8). La nueva relación de clases se puede ver en la figura 6.7.

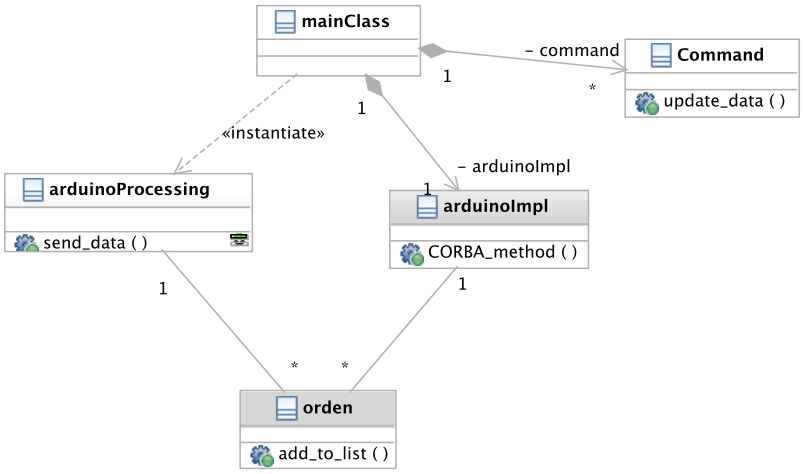


Figura 6.7: Diagrama de clases del servant de la tarjeta de adqusición de datos.

Dado que el sistema CMake no soporta la compilación de código Java, se ha hecho una excepción con este módulo y se ha creado un archivo Makefile específico generado manualmente y que incluye instrucciones para las siguientes operaciones:

- Instalación de las librerías requeridas, que se encuentran dentro del árbol del código fuente del módulo.
- Generación del código a partir de las IDLs.
- Compilación del servant.

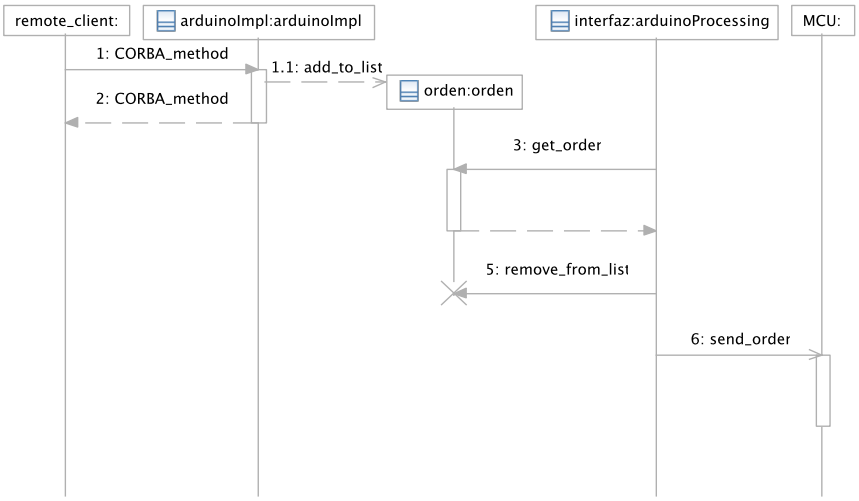


Figura 6.8: Diagrama de secuencia del envío de mensajes.

- Instalación en el ordenador de a bordo.

6.5. Executive

libExecutive es el nombre de una librería desarrollada para facilitar la comunicación entre módulos que deben trabajar con datos que se actualizan síncronamente. Surgió la necesidad de contar con una utilidad que, funcionando por ciclos, cogiera en primer lugar el valor de ciertos datos en determinados objetos, los distribuyera entre todos aquellos objetos que hiciesen uso de ese dato y cerrase el ciclo solicitando a los objetos que hiciesen los cálculos necesarios con los datos actualizados para comenzar un nuevo ciclo, que se representa en la figura 6.10

Figura 6.9: Diagrama de clases de libExecutive.

Los datos son atributos miembro de objetos C++. Los métodos que deben llamarse son tres y también son miembros de alguna clase, por lo que junto a la referencia del puntero a método miembro hay que almacenar el objeto, de modo que la llamada al método sea efectiva tal como se produciría con un objeto. Ha sido necesario el uso de plantillas debido a que los objetos pueden ser de cualquier tipo, combinando las características más avanzadas

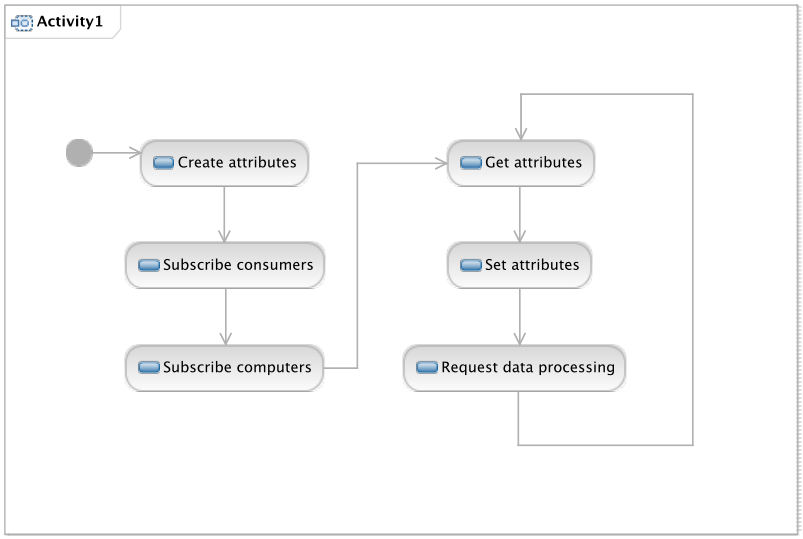


Figura 6.10: Diagrama de actividades de libExecutive.

del lenguaje C++ en su versión C++03. Ver [7] para una de las mejores referencias C++ existentes.

Es similar al servicio de eventos de CORBA, con la diferencia de que el servicio de eventos espera a que los objetos suministren el dato, en lugar de recogerlos él mismo, y libExecutive gestiona los ciclos y el momento en que cada objeto puede realizar los cálculos, manteniendo en sincronismo los datos.

Debido a la naturaleza distribuida del proyecto, se ha preparado esta librería para trabajar con objetos CORBA, estando soportados la mayoría de tipos de datos, habiéndose excluido los datos de longitud variable. También se ha optimizado el uso de los recursos habilitando un *hilo* por cada llamada que se realiza, de modo que éstas llamadas se hacen simultáneamente y, si los objetos distribuidos se encuentran en máquinas remotas o la CPU está compuesto por varios núcleos, se mejora el uso los recursos de computación existentes al no tener que esperar a que cada objeto realice los cálculos en su máquina, es decir, que el tiempo de cálculo no depende del número de objetos/llamadas a realizar, sino del objeto que tarde más en realizar el cómputo.

El código fuente de esta librería se ha incluido en el cedé que acompaña el proyeto y la documentación generada ha sido extraída con Doxygen (ver anexo E y manual de uso en [9]) y se incluye en el anexo B. También se ha incluido un ejemplo de uso que ha servido asimismo para probar y validar el código, del cual se pueden encontrar más detalles en el capítulo 7.

A continuación se incluye algunos de los métodos implementados en la cabecera de la librería. Este código no es funcional, pero sirve para tener una ideal del funcionamiento interno del código desarrollado. Se han omitido tipos de datos, atributos y comentarios. Éstos últimos ya están incluidos en la documentación y se han omitido para evitar duplicidades; nótese que el cuerpo de los métodos es suficientemente sencillo para ser autoexplicativo y no necesitar más comentarios.

```
1 template <class T>
2 class executive_attribute
```

```
3  {
4  public:
5      template <class O>
6          executive_attribute(executive & ex, O * obj, T (O::*get)()
               const)
7            {
8              // Register the supplier.
9              supplier_entity.obj = (phony *)obj;
10             supplier_entity.get = (T (phony::*)()const)get;
11             supplier_entity.get_type = NORMAL;
12             supplier_entity.value = &value;
13
14             // Register to the executive.
15             add_this_to_executive(ex);
16           }
17
18     ~executive_attribute()
19       {
20         for (std::vector<executive::callbacks>::iterator i =
21                   exec->callbacks_pool.begin();
22                 i != exec->callbacks_pool.end();)
23           {
24             if (i->obj == (phony *)this)
25                 exec->callbacks_pool.erase(i);
26             else
27                 i++;
28           }
29       }
30      template <class O>
31         void subscribe_consumer(O * obj, void (O::*set)(T))
32           {
33             consumer c;
34             c.obj = (phony *)obj;
35             c.set = (void (phony::*)(T))set;
36             c.value = &value;
37             c.set_type = NORMAL;
38             consumer_pool.push_back(c);
39           }
40     void add_this_to_executive(executive & ex)
41       {
42             executive::callbacks ev;
43             ev.obj =                (phony *)this;
44             ev.async_get =
45                  (void (phony::*)())&executive_attribute<T>::
                        async_get;
46             ev.async_set =
47                  (void (phony::*)())&executive_attribute<T>::
```

```
                    async_set;
48              ev.async_compute =
49                   (void (phony::*)())&executive_attribute<T>::
                        async_compute;
50              exec = &ex;
51              exec->callbacks_pool.push_back(ev);
52          }
53      void async_set()
54          {
55            for (size_t i = 0; i < consumer_pool.size(); i++)
56              {
57                pthread_t thread;
58                pthread_create(&thread, 0, threaded_set, &consumer_pool
                      [i]);
59                exec->thread_pool.push_back(thread);
60              }
61          }
62       static void * threaded_set(void * void_data)
63          {
64            consumer * data = (consumer *)void_data;
65            T * value_p;
66            switch (data->set_type)
67              {
68                case NORMAL:
69                    (data->obj->*data->set)(*data->value);
70                    break;
71                case POINTER:
72                    value_p = new T(*data->value);
73                    (data->obj->*data->set_p)(value_p);
74                    break;
75                case REFERENCE:
76                    (data->obj->*data->set_r)(*data->value);
77                    break;
78              }
79            return 0;
80          }
81  };
```

6.6. Prevención de impactos

Se ha desarrollado un algoritmo para extender el módulo del robot base y que protegerá contra posibles impactos, reduciendo la velocidad del robot de manera progresiva a medida que se aproxima a un obstáculo. El algoritmo calcula por una parte la velocidad máxima en dirección del eje del sensor que puede tener el robot en función de la distancia al obstáculo detectado, si hay alguno, con los sensores ultrasónicos de la base robótica, y por otra parte la velocidad solicitada desplazada a cada sensor y proyectada sobre su eje. Si

la velocidad calculada supera la velocidad máxima, se acota la primera a la segunda. Este algoritmo permite desplazar el robot cerca de obstáculos y paredes siempre que no se acerque a ellos. Puede pasar por pasillos estrechos sin limitar la velocidad de avance y evitando choques laterales y también evitar choques frontales sin reducir la capacidad de giro ante paredes. Los movimientos que el robot hace son previsibles, pues no modifica la dirección ni el sentido del desplazamiento solicitado, únicamente limita la velocidad de avance, incluso deteniendo el robot, en caso de choque inminente. Tampoco limita la velocidad cuando se solicitan movimientos suaves y precisos en espacios pequeños, cumpliendo con los requisitos solicitados.

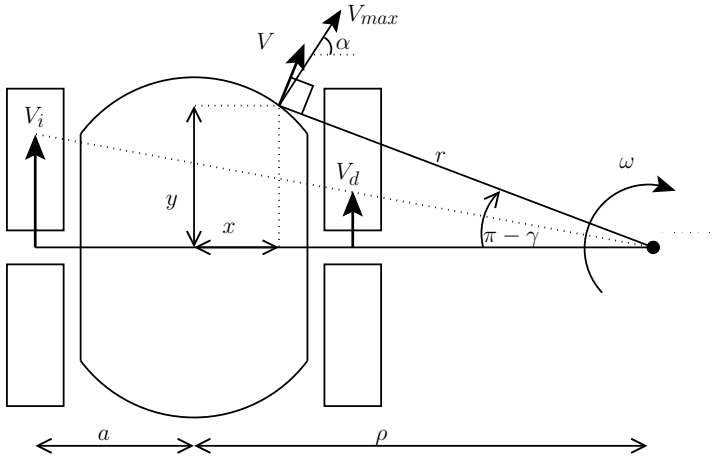


Figura 6.11: Representación de las variables usadas en la prevención de impactos.

Las velocidades de las ruedas izquierda, derecha, del punto central del robot y del punto representado son, respectivamente,

$$V_i = -(\rho - a) \cdot \omega; \qquad V_d = -(\rho + a) \cdot \omega; \tag{6.1}$$
$$V_\rho - \rho \cdot \omega \qquad V = -r \cdot \omega$$

Los datos conocidos, obtenidos por la odometría y por las especificaciones geométricas del robot son V_i, V_d y a. Las incógnitas para obtener el centro instantáneo de rotación son ρ y ω. Despejando en 6.1,

$$\omega = \frac{V_d - V_i}{2 \cdot a} \qquad \rho = \frac{V_i + V_d}{V_i - V_d} \cdot a \tag{6.2}$$

Por relaciones geométricas,

$$r = \sqrt{y^2 + (\rho - x)^2} \qquad \pi - \gamma = arctg(\frac{y}{\rho - x}) \tag{6.3}$$

Y tomando $\theta = \gamma - \alpha + \pi/2$, resulta que la velocidad se excede cuando

$$V \cdot cos(\theta) > V_{max} \tag{6.4}$$

Y se debe limitar la velocidad a

$$|V| = \frac{|V_{max}|}{cos(\theta)} \tag{6.5}$$

Sensor #	α	**x(m)**	**y(m)**	d_{min}**(m)**
1	0	0.136	0.147	0.12
2	40	0.119	0.193	0.17
3	60	0.079	0.227	0.11
4	80	0.027	0.245	0.09
5	100	-0.027	0.245	0.09
6	120	-0.079	0.227	0.11
7	140	-0.119	0.193	0.17
8	180	-0.136	0.147	0.12
9	180	-0.136	-0.144	0.12
10	220	-0.119	-0.189	0.17
11	240	-0.19	-0.223	0.11
12	260	-0.027	-0.241	0.08
13	280	0.027	-0.241	0.08
14	300	0.079	-0.223	0.11
15	320	0.119	-0.189	0.17
16	360	0.136	-0.144	0.12

Cuadro 6.1: Parámetros de los sensores y distancias mínimas en la prevención de impactos.

La velocidad máxima V_{max} se calcula a partir de la distancia del sensor al obstáculo, siguiendo la curva de la figura 6.12. En esta figura la velocidad admmisible está acotada superiormente por tres rectas:

1. Distancia de seguridad, que incluye la distancia del sensor hacia el interior del contorno en planta del robot y la distancia mínima del sensor.

2. Rampa de velocidades incrementales. Se permite mayor velocidad cuanto más lejos esté el obstáculo,

3. La velocidad máxima V_{motor} que son capaces de alcanzar los motores.

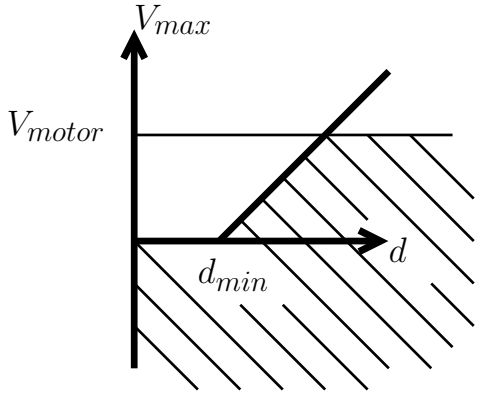


Figura 6.12: Función de limitación de la velocidad máxima respecto de la distancia.

En cada actualización de la lectura de los sensores ultrasónicos y cada vez que se recibe un nuevo comando de movimiento se recalculan las velocidades máximas V_{max} permitidas por cada sensor y la transformación espacial de la velocidad solicitada a la posición de cada sensor y proyectadas sobre su dirección usando los datos de la tabla 6.1, limitando la velocidad absoluta solicitada al sensor que más limite la velocidad, es decir, el mínimo de velocidades máximas.

Capítulo 7

Validación y pruebas

7.1. Pruebas

A lo largo del proyecto se han ido realizando pruebas sobre los módulos en desarrollo, comprobando el correcto funcionamiento de los componentes de forma individual. Las pruebas se han centrado en la verificación funcional de los componentes.

La tarjeta de adquisición de datos sufre de un problema en la interacción con la tarjeta de alimentaciones. Se trata de que si está alimentado únicamente por USB, los pines de salida digitales se configuran a cero con una resistencia, lo que hace que apague ocasionalmente algunos de los dispositivos. Se ha resuelto asegurándose que la tarjeta de alimentaciones tiene el interruptor correspondiente a la tarjeta de adquisición de datos encendido.

Se han realizado pruebas en exterior que contemplan el funcionamiento simultáneo de varios servants. Éstos son el robot base, la tarjeta de adquisición de datos, el GPS y el sensor láser. Ésto implica que otros módulos desarrollados, como la tarjeta de alimentaciones y el sensor i/v, han sido correctamente integrados, ya que el sistema de control usado controló el robot a través de las interfaces IDL proporcionadas.

7.2. Validación de los requisitos

Se hará un repaso breve de la verificación efectuada sobre cada requisito.

Requisito 1: Establecer base de desarrollo. Este requisito ha sido validado a lo largo del desarrollo de los módulos. El hecho de existir nuevos módulos funcionando sobre la plataforma da por válido la verificación del requisito.

Requisito 1.1: Integración de un ordenador de a bordo. Se han abierto constantemente consolas remotamente desde la instalación del sistema operativo para configuración, instalación, etc.

Requisito 1.1.1: Sistema operativo en tiempo real. Se han seguido los pasos para la isntalación de Linux RTAI en el sistema. El último paso consiste en comprobar que funciona. Los comandos críticos para esta verificación son:

```
# cd /usr/realtime/testsuite/modules
# insmod rtai_hal.ko
# cd /usr/realtime/testsuite/kern/cd latency
# ./run
```

Ésto comenzará la ejecución de la prueba. Se ha probado con el procesador en reposo y ejecutando programas convencionales que ocupan el 100 % de la CPU. El resultado es una salida del programa en forma de listado cuya última columna indica el número de *overruns*. En todos los casos ha sido de 0, validando el requisito.

Requisito 1.1.2: Elaboración de un sistema de configuración e inicialización de servants. Se puede comprobar cuando se arranca el ordenador de a bordo que los servants se inicializan con la configuración de los archivos que se encuentran en `/etc/higgs`.

Requisito 1.2: Desarrollo de tarjeta de alimentaciones. Es inmediato encender un dispositivo con los interruptores de la placa de alimentación. No es necesario tener el ordenador de a bordo encendido, por ello no se prestará el servicio remoto aunque en este caso es irrelevante; se está comprobando únicamente que se controla la alimentación a los dispositivos.

Requisito 1.2.1: Los dispositivos se controlarán individualmente. Como el anterior, solo que se accionan los interruptores específicos del canal que se quiera controlar. Debajo de cada interruptor está indicado con una etiqueta el dispositivo que controla.

Requisito 1.2.2: Apagado manual de dispositivos. La verificación coincide con la del requisito anterior.

Requisito 1.2.3: Apagado remoto de dispositivos. Similar a la verificación del requisito 1.2.1 salvo que en lugar de accionar los interruptores manualmente, se dejan todos encendidos y mediante el cliente de pruebas desarrollado para la tarjeta de adquisición de datos y que apaga y vuelve a encender cada dispositivo aproximadamente cada segundo, queda verificado el requisito.

Requisito 2: Integración de sensores y actuadores. En los siguientes párrafos se encuentran multitud de ejemplos de control de dispositivos remotos.

Requisito 2.1: Uso de la tecnología CORBA. No hay más que conectarse a alguno de los servants. Muchos de los requisitos se han validado simultáneamente ya que entran en funcionamiento muchos elementos que trabajan a la vez.

Requisito 2.2: Los objetos CORBA estarán disponibles cuando el dispositivo lo esté. Se puede comprobar que si se apaga la alimentación de cada dispositivo, las referencias a objetos CORBA dejan de ser válidas. Cuando se llama a un método remoto en estas condiciones se recibe una excepción TRANSIENT. Cada módulo ha sido diseñado prestando mucha atención a éste requerimiento. Se han añadido hilos de ejecución a modo de *watchdog* y se capturan los errores en la comunicación.

Requisito 2.3: Recuperar funcionalidades automáticamente. Continuación de la verificación anterior. Durante el arranque del ordenador de a bordo o después de la caída de un dispositivo, se puede observar en los logs como los scripts de inicio tratan de ejecutar los servants continuamente hasta que es fructífera por estar el dispositivo físico nuevamente operativo. Los servants se han desarrollado cuidadosamente para ofrecer este comportamiento.

Requisito 2.4: Integración de sensores propioceptivos.
Requisito 2.4.1: Integración de una brújula electrónica.
Requisito 2.4.2: Integración de acelerómetros.
Requisito 2.4.3: Integración de sensores de tensión e intensidad. Estos cuatro requisitos se validan con llamadas remotas al servant del Arduino. Cada uno tiene un método en la interfaz IDL. Basta con llamar a los tres métodos[1] en secuencia para efectuar la validación.

Requisito 2.5: Integración de sensores exteroceptivos. Validando cualquiera de los requisitos derivados se valida éste.

Requisito 2.5.1: Integración de una cámara direccionable. Validando los dos requisitos derivados se valida éste.

Requisito 2.5.1.1: Integración de la cámara estereoscópica existente. Validado parcialmente. La conexión CORBA funciona correctamente cuando el servant y el cliente están en el mismo computador, pero no así cuando corren en computadores distintos porque aparecen retardos mayores al segundo y el refresco es también demasiado bajo. Ésto es así por el ancho de banda requerido para transmitir los datos de vídeo, que no están optimizados para *streaming*.

Requisito 2.5.1.2: Integración de la muñeca. Al igual que módulos anteriores, es posible controlar remotamente la muñeca usando los dos clientes CORBA desarrollados. El primer cliente realiza un bucle que envía comandos de vaivén alternativos de amplitud creciente. A continuación se muestra este código como ejemplo representativo para los demás módulos y en el que se observa la sencillez con la que se pueden escribir clientes:

```
1  #include <iostream>
2  #include "wristC.h"
3  #include "CosNamingC.h"
4  #include "../../../lib/CORBA_utils.h"
5  int main(int argc, char* argv[]){
6      CORBA_BEGIN_CLIENT(argc, argv);
7      CORBA_GET_REFERENCE(higgs::wrist, wrist, "wrist");
8
9      wrist->set_max_speed(6.F, higgs::wrist::PITCH);
10     wrist->set_max_accel(5.F, higgs::wrist::PITCH);
11     wrist->set_max_accel(2.F, higgs::wrist::ROLL);
12     wrist->set_max_speed(4.F, higgs::wrist::ROLL);
13     float f = .1F;
```

[1] En el manual del usuario, anexo C, se dan detalles para llevarlo a cabo.

```
14      for (int i = 0; i < 9; i++)
15        {
16          wrist->set_position(0.3*f, higgs::wrist::PITCH);
17          wrist->set_position(0.3*f, higgs::wrist::ROLL);
18          while (!wrist->is_ready(higgs::wrist::PITCH))
19            {
20              std::cout << wrist->get_current(higgs::wrist::PITCH) <<
                    std::endl;
21            }
22          wrist->wait(higgs::wrist::BOTH);
23          f += (f > 0) ? 0.1 : -0.1;
24          f *= -1;
25        }
26      wrist->set_position(0.F, higgs::wrist::BOTH);
27      wrist->wait(higgs::wrist::BOTH);
28
29      CORBA_END_CLIENT;
30      return 0;
31  }
```

El segundo cliente consiste en un control de la muñeca con el ratón del ordenador donde reside el cliente. Como nota curiosa, con esta aplicación se ha conseguido engañar a algunos visitantes haciéndoles pensar que el robot realizaba reconocimiento y seguimiento de rostros[2].

Requisito 2.5.2: Integración del dispositivo láser. El cliente de pruebas desarrollado lista por pantalla las distancias medidas por el sensor láser. Este cliente ha sido ejecutado remotamente, validando el requisito.

Requisito 2.5.3: Integración del receptor GPS. Debido a la pérdida de cobertura de la conexión wifi en exteriores, no ha sido posible verificar este requisito tal y como se redactó. Sin embargo, sí se ha logrado conectar con el servant y recibir los datos de posición y velocidad con un cliente corriendo localmente. No hay motivos para pensar que con una conexión adecuada se pueda acceder remotamente a estos datos.

Por otra parte, se han hecho pruebas usando el receptor GPS situado en el ático. Como no se puede considerar que esta configuración de los componentes del GPS esté integrada en el robot tampoco es válido para verificar el requisito.

Requisito 2.5.3.1: Integración del sistema de correción diferencial del GPS. La corrección diferencial sufre de los mismos problemas que el GPS sin correcciones. Adicionalmente la estación base tiene que estar operativa y se deben recibir las correcciones vía radio.

Tan sólo en una ocasión y durante unos breves instantes se consiguió leer la posición del robot en exteriores con un error de menos de medio metro. Se logró en el centro del campo de futbito. Lamentablemente, se estaban haciendo pruebas de protocolo y el servant aún no

[2]El objetivo de este proyecto es proporcionar la base para lograr estas capacidades con sistemas de control que usen los servants del robot.

estaba desarrollado, por lo que la comprobación se hizo visualmente a través del terminal. Se llegó a una desviación estándar de 0.2m.

Al parecer existen sistemas de interferencia de la señal de radio en el entorno del campus universitario provenientes de edificios gubernamentales cercanos. Se ha especulado que éste es el motivo de no recibir la corrección diferencial.

Requisito 3: Uso de estándares y optimización en el desarrollo. Todos los servants desarrollados son conformes a C++ estándar salvo el módulo de la tarjeta de adquisición de datos que está escrito en Java. El software propietario se ha limitado a estos casos:

1. Diagramas convencionales con Omnigraffle.
2. Diagramas UML con RSA.
3. Primera verificación de funcionamiento con el software de control del GPS incluido en el paquete.
4. Diseño de las tarjetas con OrCAD.

Capítulo 8

Aspectos de la dirección de proyectos

8.1. Presupuesto

Concepto	**Precio**
Robot Pioneer 2AT-8	7.434,00€
Cámara STH-MDCS-C	876,00€
Sony Vaio 11¨	1.505,41€
Brújula CMPS03	39,90€
Acelerómetro 300019	42,20€
Powercube 2 GDL	3.400,00€
Láser SICK LMS200	1.230,00€
Sistema GPSd	18.500,00€
Arduino Mega	46,95€
Total	**33.074,46€**

Cuadro 8.1: Presupuesto del sistema de partida.

Concepto	**Precio**
Pedido de componentes 13/10/2009	63,11€
Fabricación de PCB	235,33€
Pedido de componentes 27/10/2009	140,25€
Pedido de componentes 18/01/2010	122,71€
Pedido de componentes 11/03/2010	106,80€
Pedido de componentes 27/10/2010	60,07€
Cables y convertidores USB-RS232	85,10€
Proyectante (23 meses, 900€/mes)	20.700,00€
Total	**21.513,37€**

Cuadro 8.2: Presupuesto del proyecto.

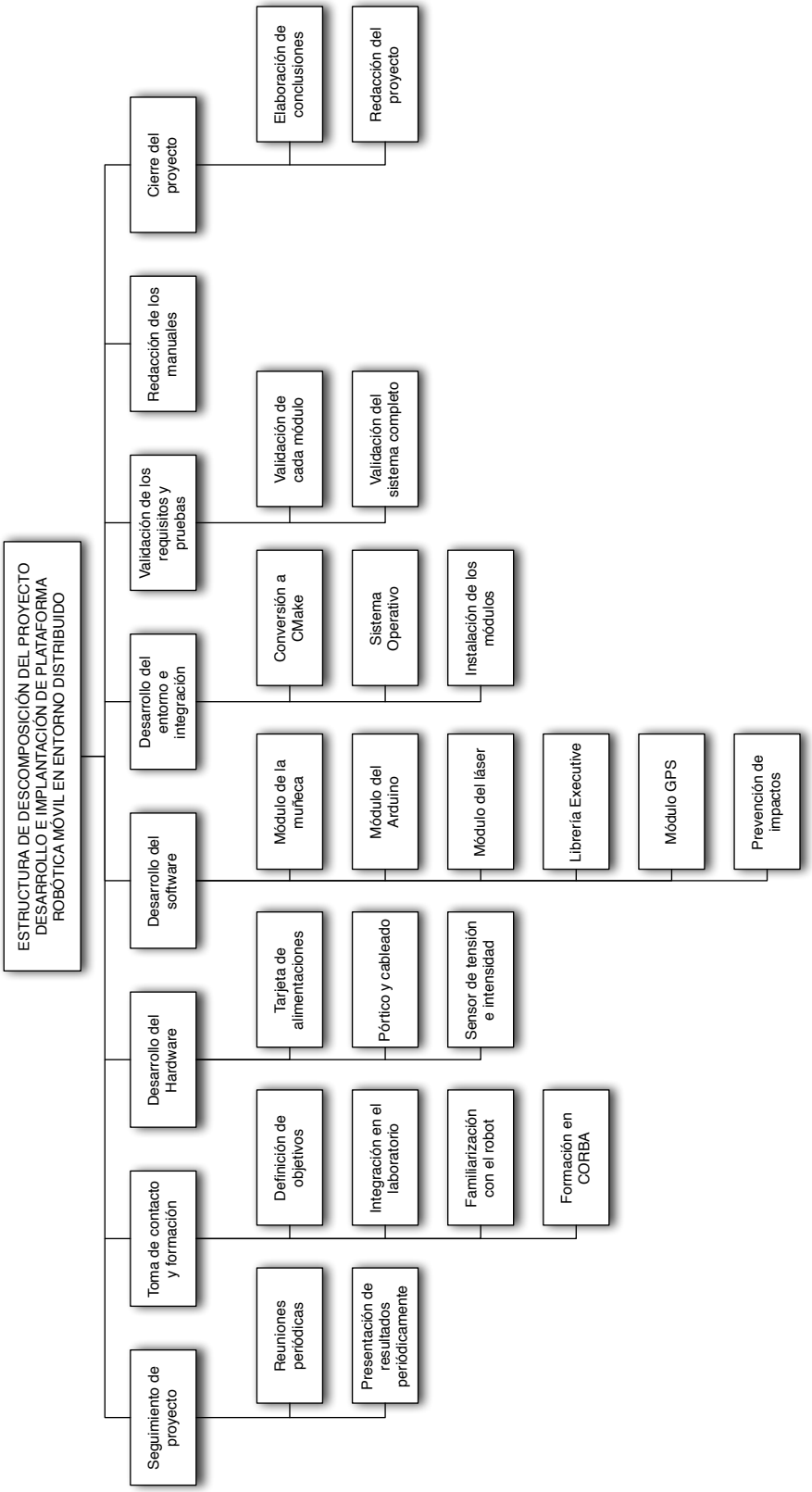
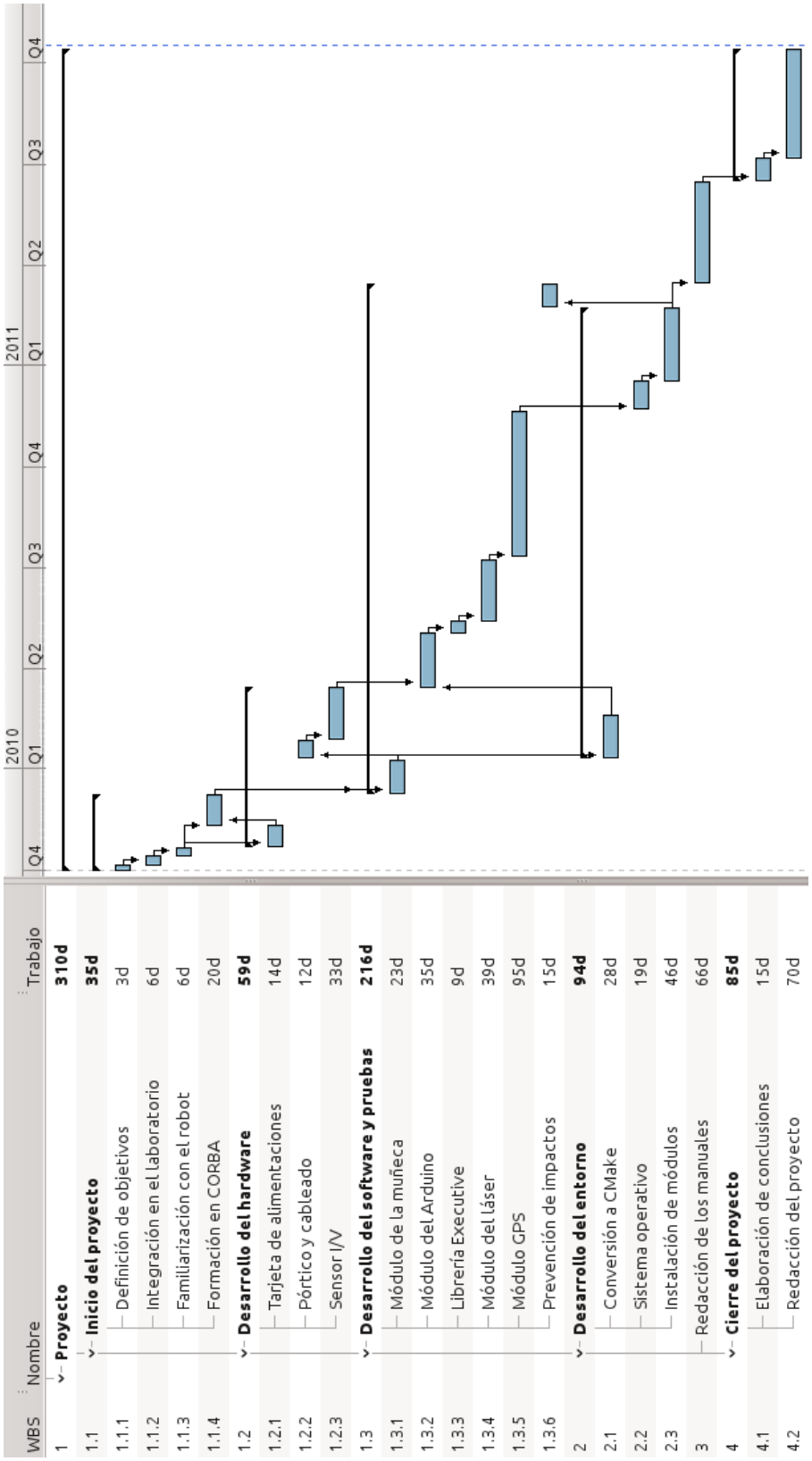


Figura 8.1: Estructura de Descomposición del Proyecto.



WBS	Nombre	Trabajo
1	**Proyecto**	**310d**
1.1	**Inicio del proyecto**	**35d**
1.1.1	Definición de objetivos	3d
1.1.2	Integración en el laboratorio	6d
1.1.3	Familiarización con el robot	6d
1.1.4	Formación en CORBA	20d
1.2	**Desarrollo del hardware**	**59d**
1.2.1	Tarjeta de alimentaciones	14d
1.2.2	Pórtico y cableado	12d
1.2.3	Sensor I/V	33d
1.3	**Desarrollo del software y pruebas**	**216d**
1.3.1	Módulo de la muñeca	23d
1.3.2	Módulo del Arduino	35d
1.3.3	Librería Executive	9d
1.3.4	Módulo del láser	39d
1.3.5	Módulo GPS	95d
1.3.6	Prevención de impactos	15d
2	**Desarrollo del entorno**	**94d**
2.1	Conversión a CMake	28d
2.2	Sistema operativo	19d
2.3	Instalación de módulos	46d
3	Redacción de los manuales	66d
4	**Cierre del proyecto**	**85d**
4.1	Elaboración de conclusiones	15d
4.2	Redacción del proyecto	70d

Figura 8.2: Diagrama de Gantt.

Capítulo 9

Conclusiones y trabajo futuro

En este capítulo se hará un balance global del proyecto, repasando el trabajo realizado, algunos incidentes y logros y las observaciones destacables que han ido surgiendo durante su desarrollo. Se seguirá comentando algunos de los desarrollos actuales sobre el robot que escapan del alcance de este proyecto y se plantean trabajos futuros que resuelven problemas o mejoras propuestas durante el desarrollo de este proyecto y que no estaban contempladas en los requisitos iniciales.

9.1. Conclusiones

Se han alcanzado los objetivos principales planteados al inicio del proyecto. Se ha logrado el funcionamiento remoto de todos los sistemas y se ha facilitado documentación que permita el uso y la continuidad del proyecto.

La complejidad y su extensión hacen de éste el mayor proyecto realizado por el autor hasta la fecha.

Algunos de los módulos desarrollados no tienen nada que envidiar a sus homólogos profesionales. Se podría comercializar algunos de estos desarrollos y así sacar rentabilidad a los productos provenientes de investigación.

En el mercado existen multitud de sensores que resuelven parte de la problemática que existe con el robot. Por ejemplo, existen cámaras direccionables comercialazadas como un paquete cerrado y con drivers estándar. Descartar la cámara y la muñeca en beneficio de uno de estos paquetes habría ahorrado tiempo de desarrollo y dinero, pues el coste de tener una persona desarrollando estos módulos durante varios meses es alto.

Colaboración con otros grupos

La comunicación con el resto de personal del departamento ha sido continua y ha permitido la colaboración con otros grupos de investigación dentro de la universidad, en concreto con el grupo de investigación en Control Inteligente. Las funciones programadas que realizan la sincronización del protocolo del sensor láser fueron integradas en el código fuente del driver desarrollado por ellos mismos, probadas y devueltas con las modificaciones realizadas. Esta colaboración se ha traducido en una reducción del tiempo de desarrollo del driver del láser al tener ya implementado y probado las funciones más importantes suministradas por el sensor láser. También ha aportado mayor fiabilidad y robustez a los robots de ambos grupos al no

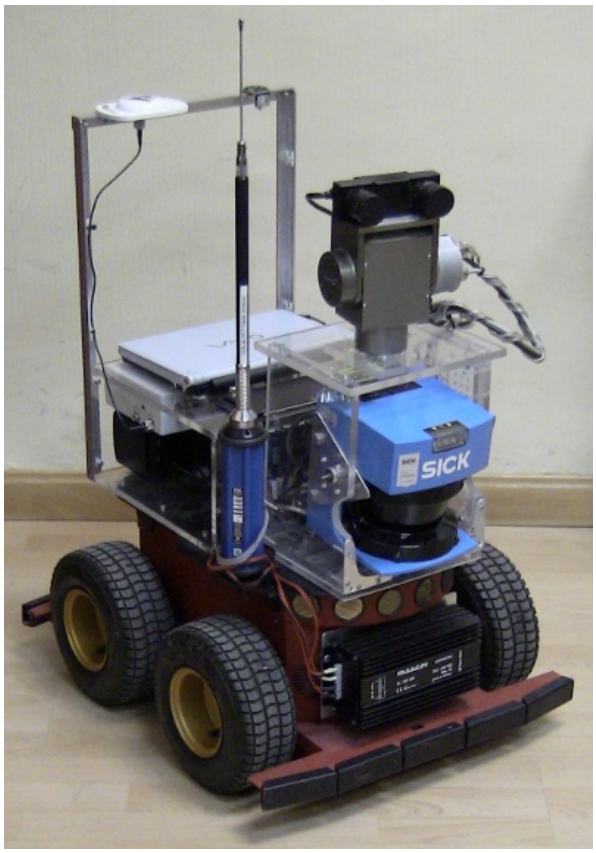


Figura 9.1: La plataforma robótica móvil, bautizada como Higgs, en todo su esplendor.

hacerse necesario reiniciar el software de control cada vez que surge algún problema en la comunicación, como la pérdida en la transmisión de algún byte.

Calidad y robustez

Durante una de las pruebas se produjo un accidente por el cual el cable de carga de las baterías quedó aprisionado por las ruedas del robot. Al avanzar éste, tiró del cable produciendo su rotura en el interior del conector `jack` y formándose un cortocircuito en la alimentación, tanto del cargador de baterías como de las baterías. El cortocircuito se cerró formando un circuito en serie formado por la batería, la resistencia shunt, la placa de alimentaciones y el conector del cargador con el fallo. La buena elección en los componentes de la placa de alimentaciones y el grosor de las pistas hizo que ésta no presentara ningún desperfecto tras el fallo, no así la resistencia shunt, que superó los valores máximos de funcionamiento y produjo una pequeña explosión por calentamiento repentino por efecto Joule, haciendo las veces de fusible improvisado del circuito. Gracias a ello sólo hubo que cambiar la resistencia shunt y reparar el cable del cargador para que el robot recuperara la funcionalidad. De esta manera se comprobó la calidad de los componentes desarrollados afectados en este accidente.

Ingeniería aplicada

El desarrollo de este proyecto ha incluido partes en las que se ha tenido que hacer uso de los conocimientos teóricos aprendidos a lo largo de la carrera y aplicarlos en la vida real. Las dificultades que aparecen son de otra naturaleza y se ha tenido que lidiar con ellas, como ejemplo más claro está el sensor i/v en el que las resistencias disponibles en el mercado sobrepasaban las tolerancias admisibles en el circuito diseñado, y aun así se ha podido realizar gracias al conocimiento de los métodos de producción en serie de las resistencias en las que artículos de un mismo lote son mucho más parecidos que artículos de lotes distintos.

9.2. Líneas de desarrollo actualmente abiertas

La plataforma robótica móvil es un proyecto en continuo desarrollo, por ello antes de la finalización de este proyecto fin de carrera se realizaron unas modificaciones no contempladas en los requisitos iniciales. Estas modificaciones son principalmente tres: La actualización de la cámara estereoscópica, la integración del sensor de profundidad y vídeo Kinect y el uso de la librería de robótica ROS.

9.2.1. Webcam

La cámara con la que se ha trabajado en este proyecto presenta una serie de inconvenientes que han llevado a la adquisición de otra cámara. Éstos inconvenientes son:

- Viñeteado muy acusado.
- Enfoque y apertura del diafragma manual.
- Distorsión de los colores hacia el verde.

Algunos de estos defectos pueden estar debidos al driver de la cámara. c La nueva cámara no solamente presenta mejor calidad en el color, sino que carece del efecto de viñeteado de la anterior y las imágenes en movimiento quedan más nítidas. Usa el protocolo relativamente moderno UVC. Sin embargo las imágenes de ambas cámaras no están sincronizados lo que puede dar lugar a cálculo incorrecto de la profundidad de objetos en movimiento. En realidad se trata de dos webcam independientes con un hub USB para compartir la conexión, por lo que además no se puede usar ambas cámaras a su máxima resolución simultáneamente.

9.2.2. ROS

ROS (Robot Operating System) suministra librerías y utilidades para ayudar a desarrolladores de software crear aplicaciones robóticas. Suministra abstracción de hardware y drivers para una gran variedad de dispositivos usados en robótica, librerías, visualizadores, paso de mensages y gestión de paquetes, entre otros. Puede plantearse como un reemplazo de CORBA específico para robótica, aunque no tiene la flexibilidad ni la capacidad de tiempo real.

Su funcionamiento se basa en la publicación de mensajes a un broker central y envío de estos mensajes a los nodos que lo soliciten. Los nodos pueden ser drivers de dispositivo, transformadores de datos, visualizadores, etc.

Se han realizado pruebas satisfactoriamente para el manejo simultáneo del láser, el kinect, el robot base Pioneer y un mando de control Wiimote en aplicaciones de navegación en interior.

Es la línea de trabajo con la que actualmente se está trabajando debido a la facilidad y la velocidad de desarrollo permitidos y a la disponibilidad de una amplia variedad de librerías enfocadas hacia robótica.

9.2.3. Kinect

Kinect es el sensor de Microsoft para la consola XBox 360, aparecido en el mercado en noviembre de 2010. Debido al gran éxito que ha tenido no solo comercialmente sino también en los laboratorios de investigación, en junio de 2011 Microsoft desarrolló una librería para el uso del sensor desde computadores personales.

Figura 9.2: El sensor Kinect de Microsoft

La característica fundamental de este sensor reside en la capacidad de detectar profundidades como si de una cámara se tratara, pudiéndose realizar mapas 3d del entorno con facilidad. También incorpora una cámara convencional en la misma dirección que el sensor de profundidad, un micrófono y un motor que inclina el sensor verticalmente. Puede substituir las cámaras estereoscópicas y reduce la carga de procesamiento que imponen éstas al no tener que calcular la profundidad a partir de dos imágenes estereoscópicas.

La librería ROS (sección 9.2.2) incluye un paquete para controlar este sensor, con el que se han realizado satisfactoriamente algunas pruebas de navegación.

9.3. Trabajos futuros

La plataforma robótica, al estar en continuo desarrollo, admite una gran cantidad de ampliaciones y el trabajo que se realice en nuevas funcionalidades está limitado únicamente por la imaginación.

9.3.1. Estación de carga

Para que un robot sea completamente autónomo además de comprobar su nivel de energía debe restaurarlo cuando se reduce. Se propone la realización de una estación de carga a la que el robot pudiera acoplarse sin intervención humana y recargarse, de modo que pudiera completar la tarea a pesar de necesitar varias cargas de baterías para completarla.

9.3.2. Optimización del envío de vídeo

Uno de los inconvenientes que presenta actualmente la interfaz CORBA para la cámara estereoscópica es el bajo rendimiento en el envío de datos. La solución que se propone, a incorporar en el futuro, es modificar la interfaz de manera que se use CORBA para configurar un sistema de streaming de video usando comunicaciones asíncronas UDP/IP para lo cual existen infinidad de paquetes software en Internet. Así el vídeo se enviaría comprimido y sin la pérdida de ancho de banda necesario en comunicaciones síncronas y del protocolo IIOP de CORBA, dejando parte de la conexión wi-fi libre para otras comunicaciones más críticas, aunque se perdería tiempo de CPU del ordenador de a bordo en la compresión del vídeo.

9.3.3. Uso de baterías de litio

Las baterías de plomo son pesadas y sufren de efecto memoria si se las descarga demasiado. Las baterías de litio son ligeras y no sufren del efecto memoria, sin embargo necesitan un circuito electrónico que controle las cargas ya que una carga en exceso puede calentarlas en exceso o incluso pueden llegar a explotar, y descargarlas al máximo las daña irreversiblemente.

9.3.4. Panel solar

Se ha propuesto la instalación de un panel solar que cubra toda la parte superior del robot para aumentar el tiempo de autonomía de las baterías en exteriores, y si la potencia es suficiente, eliminar la necesidad de baterías y así reducir el peso total del robot.

Bibliografía

[1] Leslie Lamport, *LaTeX: A Document Preparation System.* Addison Wesley, Massachusetts, 2a Edición, 1994.

[2] Rainer Granados, José Javier y Galán López, Ramón (2010) *URBANO: A Tour-Guide Robot Learning to Make Better Speeches.*

[3] I. Navarro, *Collective Movement in Robotic Swarms*, Tesis Ph.D., UPM, Madrid, 2010.

[4] Siciliano y Kathib. *Springer Handbook of Robotics.* s.l.: Springer, 2008.

[5] José Luis Sánchez López. *Sistema de control para el seguimiento de trayectorias de un UGV no holonómico tipo Ackermann*, PFC, UPM, Madrid, 2010.

[6] Michi Henning y Steve Vinoski. *Advanced CORBA Programming with C++.* Ed. Addison-Wesley, 1999.

[7] Bjarne Stroustrup, *The C++ Programming Language*, Tercera Edición, Ed. Addison-Wesley, 1997.

[8] Andreas Vogel, Bhaskar Vasudevan, Maira Benjamin y Ted Villalba. *C++ Programming with CORBA.* Ed. Wiley, 1999.

[9] Dimitri van Heesch. *Doxygen: Manual for version 1.3.2.* 2003.

[10] Grady Booch, James Rumbaugh y Ivar Jacobson. *The Unified Modeling Language User Guide*, Segunda edición, Addison-Wesley

[11] José Alberto Arcos. *Sistema de navegación y modelado del entorno para un robot móvil.* PFC, UPM, Madrid, 2009.

[12] Antonio Barrientos, Luis Felipe Peñín, Carlos Balaguer, Rafael Aracil. *Fundamentos de robótica*, McGraw-Hill, 1997

[13] Ricardo Carelli. *Control de robots móviles.* Conferencia ETSII UPM. 2009.

[14] Crcosi, I. Ortiz y Alejo, F.J. Sánchez. *El Proyecto Fin de Carrera. Normas de realización, presentación y defensa.* Sección de puclicaciones de la ETSII UPM. 2005.

[15] Marcos Salom García. *Fusión Sensorial en Plataforma Robótica Móvil.* PFC, UPM, Madrid, 2009.

[16] Adolfo Hernando Marcos. *Versión RT-CORBA del servidor Pioneer 2AT-8.* PFC, UPM, Madrid, 2005.

Apéndice A

Esquemas de la placa de alimentaciones

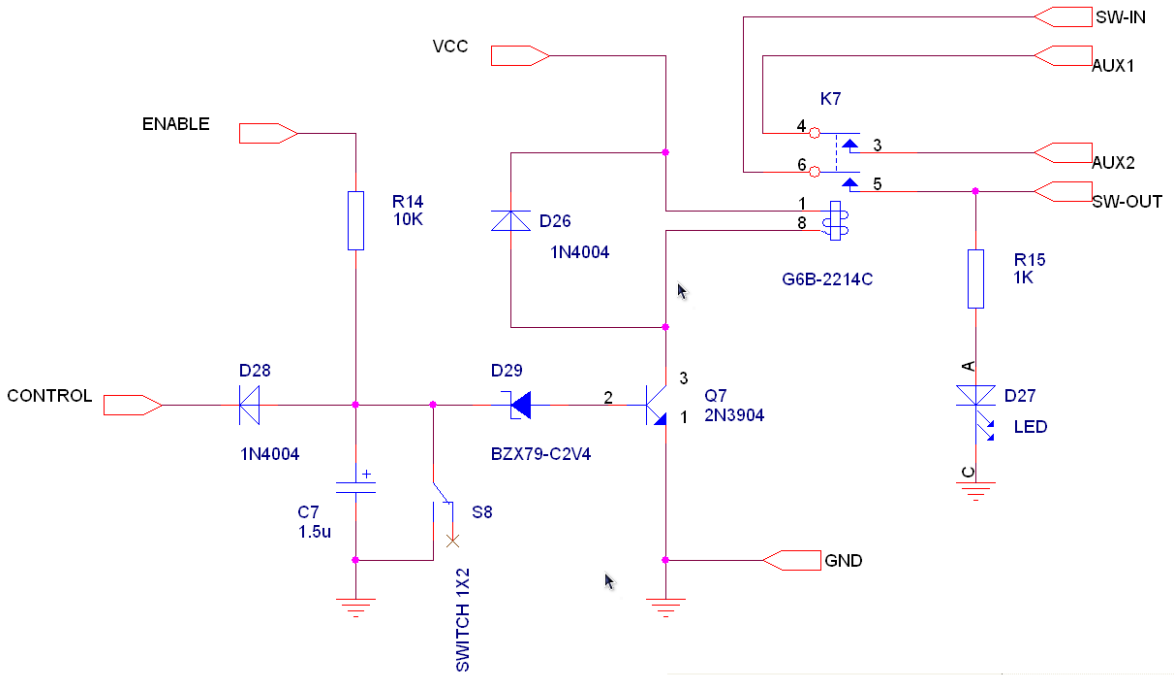


Figura A.1: Esquema del canal de 12V de la placa de alimentaciones

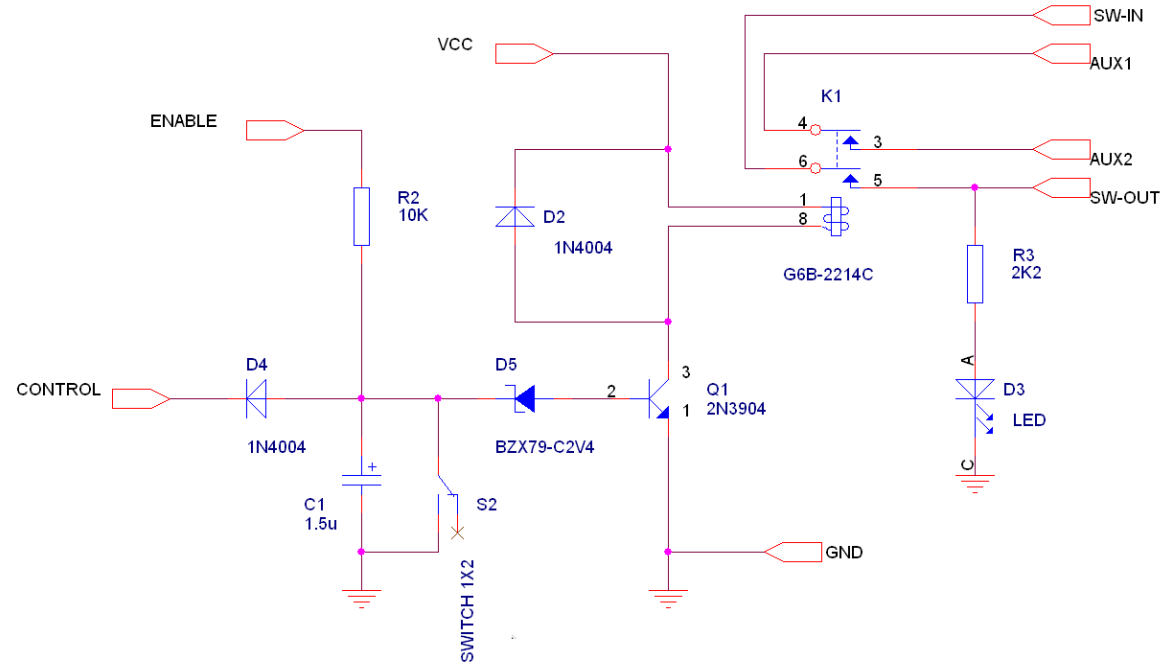


Figura A.2: Esquema del canal de 24V de la placa de alimentaciones

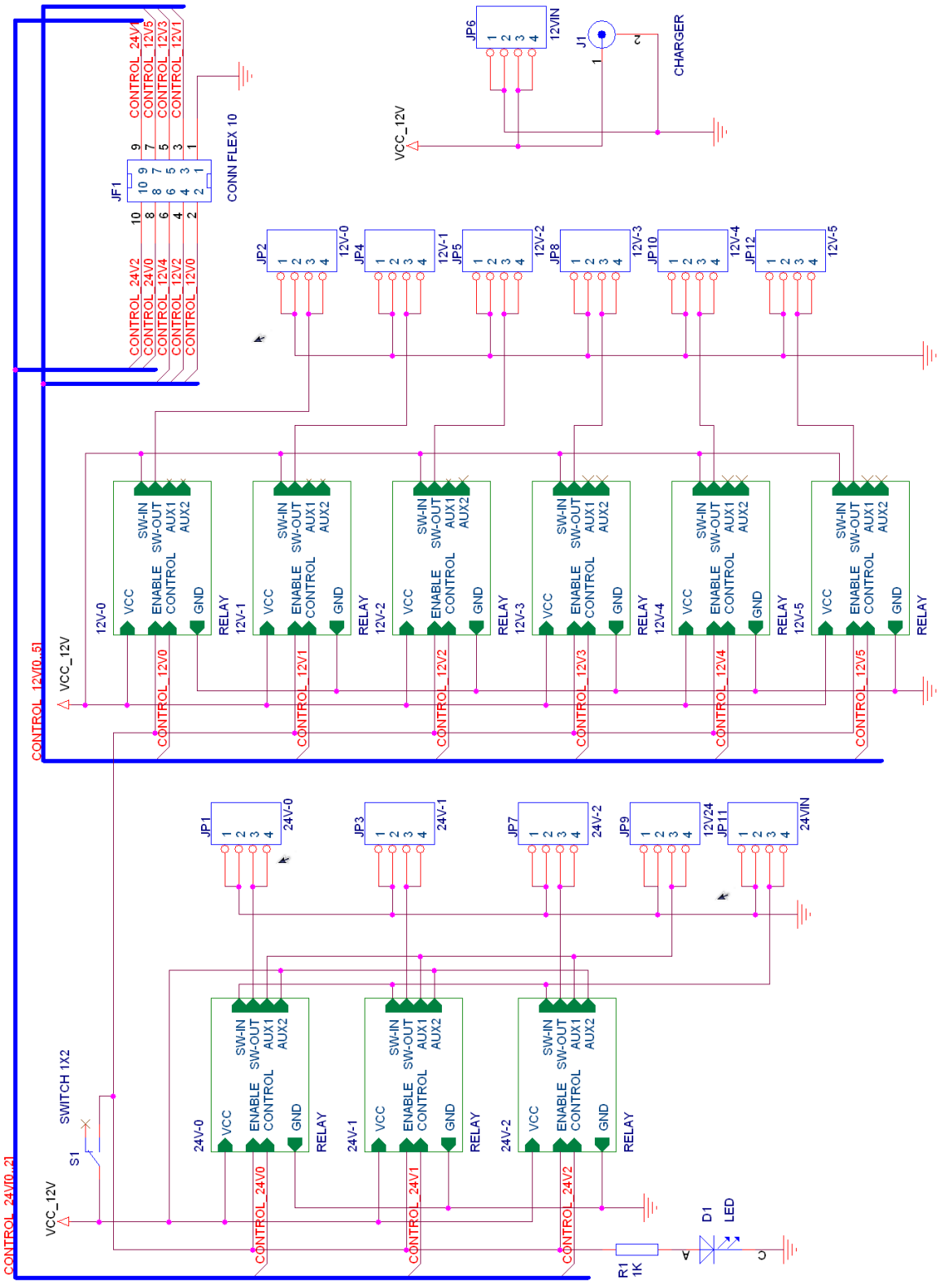


Figura A.3: Esquema general de la placa de alimentaciones

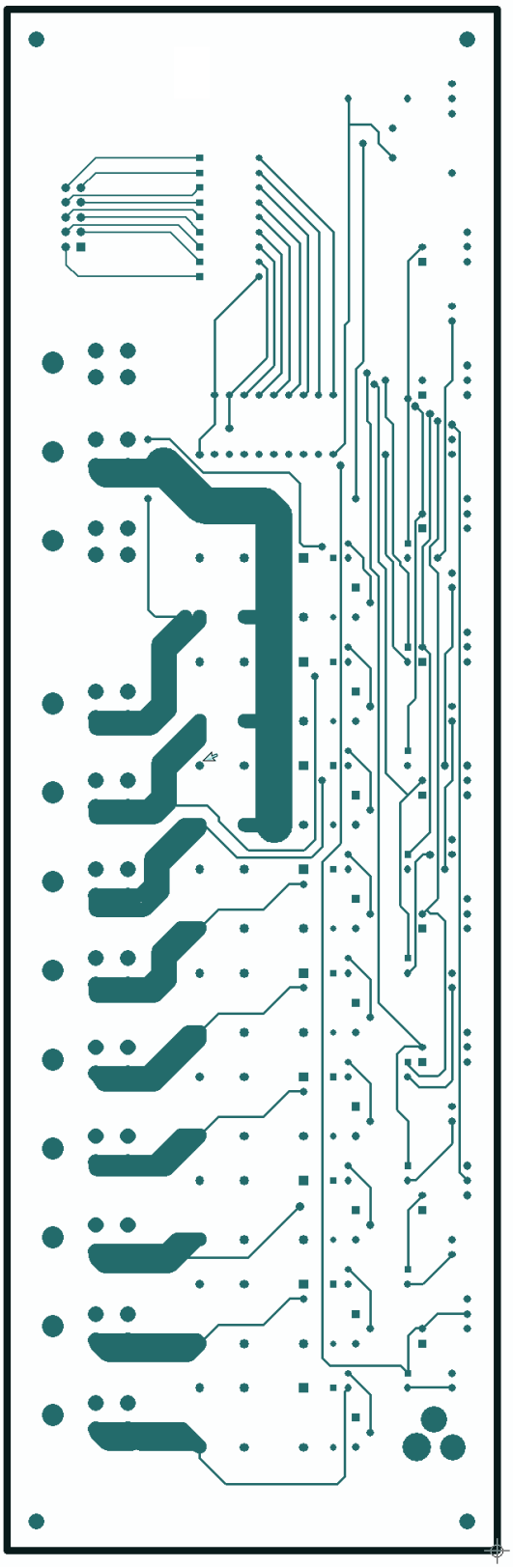


Figura A.4: Distribución de pistas en la capa superior de la placa de alimentaciones

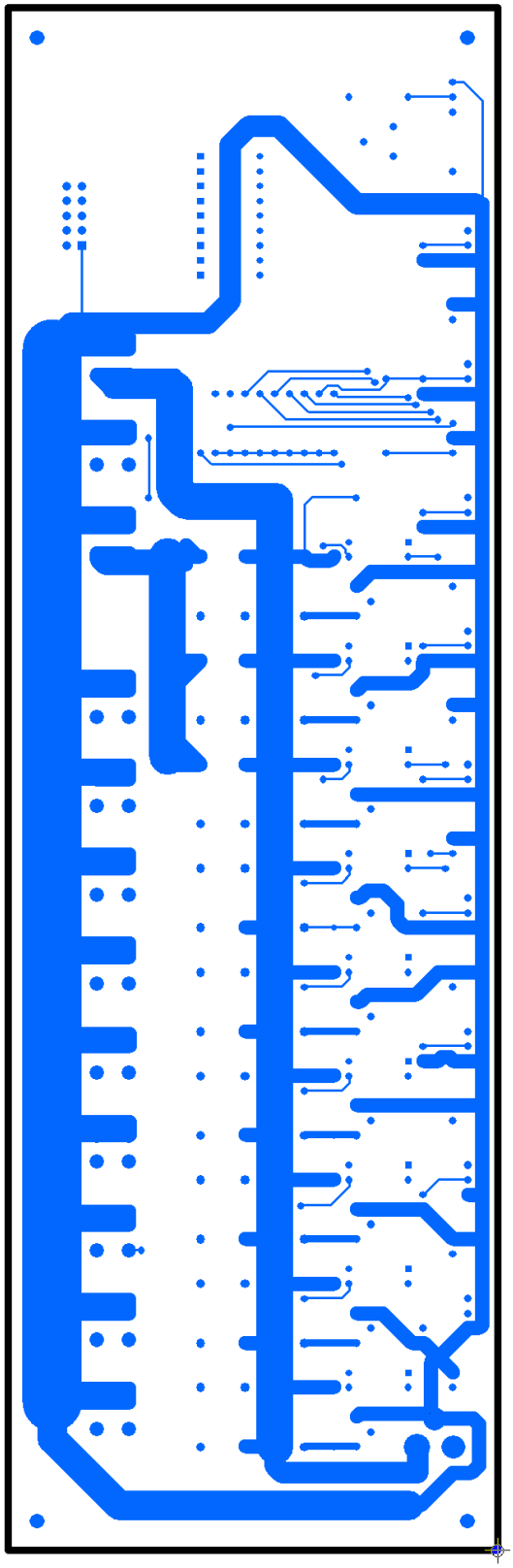


Figura A.5: Distribución de pistas en la capa inferior de la placa de alimentaciones

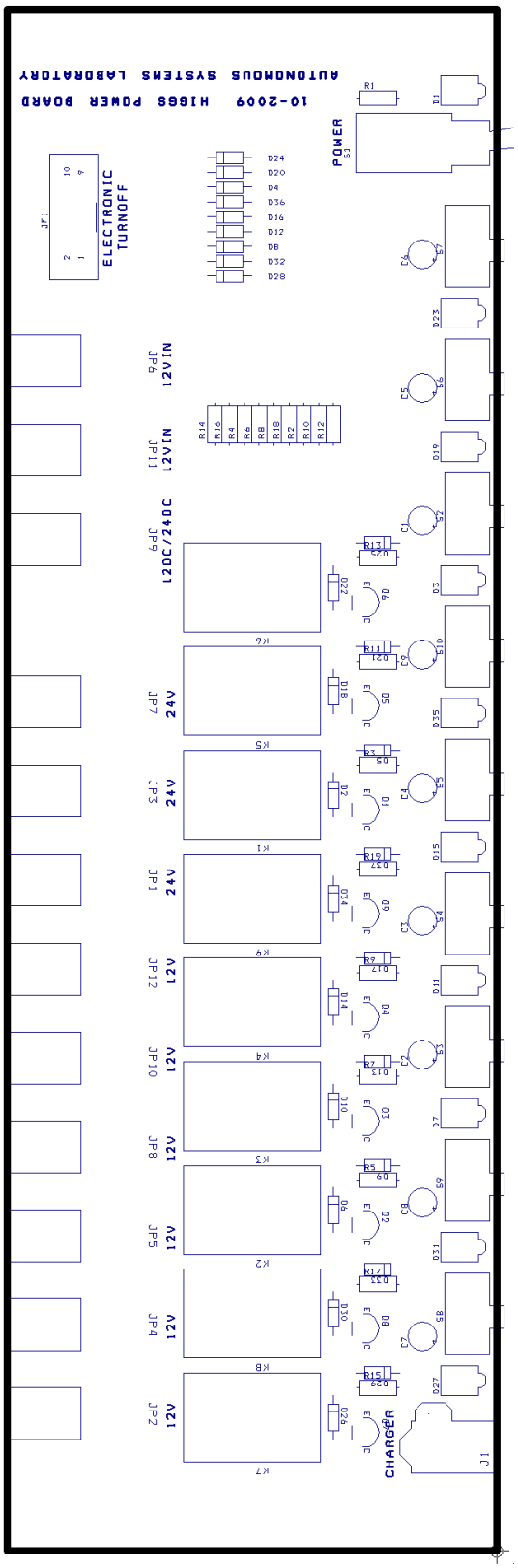


Figura A.6: Serigrafía de la placa de alimentaciones

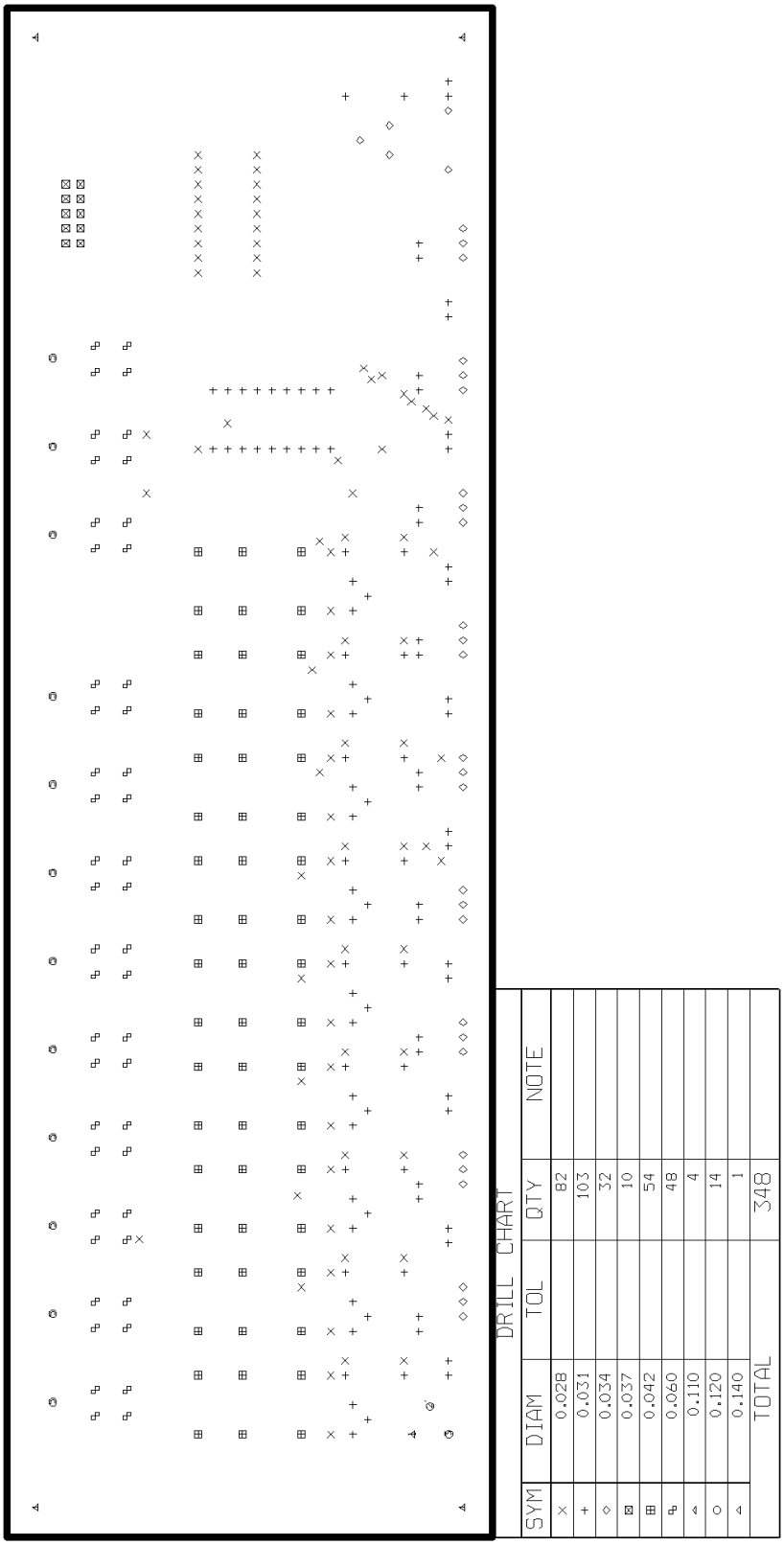


Figura A.7: Distribución y diámetro de los orificios de la placa de alimentaciones

Apéndice B

Manual de referencia de la librería executive

libexecutive

Generated by Doxygen 1.6.1

Tue Sep 27 12:20:45 2011

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 executive::callbacks Struct Reference

Do not use. Internal use.

`#include <executive.h>`

Public Attributes

- phony * **obj**
- void(phony::* **async_get**)()
- void(phony::* **async_set**)()
- void(phony::* **async_compute**)()

2.1.1 Detailed Description

Do not use. Internal use.

The documentation for this struct was generated from the following file:

- executive.h

2.2 executive Class Reference

Asynchronous event dispatcher with the hybrid Push/Pull model.

`#include <executive.h>`

Classes

- struct callbacks

  Do not use. Internal use.

Public Member Functions

- void get_all ()
- void set_all ()
- void compute_all ()
- void wait ()

  Waits for all threads to terminate.

- void cycle ()

  Gets and waits. Sets and waits. Computes and waits.

Public Attributes

- std::vector< callbacks > **callbacks_pool**
- std::list< pthread_t > **thread_pool**

2.2.1 Detailed Description

Asynchronous event dispatcher with the hybrid Push/Pull model.

Author:

Francisco J. Arjonilla García

Date:

April 2010

Each instance saves a data value to be distributed amongst the consumers. Do not modify the data in `executive`, just call the methods.

USAGE: Create an executive. Then create the attributes passing the executive just created. When creating the attributes the object and pointer to method must point to the supplier. Afterwards subscribe the consumers and the compute methods and finally `while(1) ev_mgr.cycle();`

Create one instance of `executive_attribute` for each data value to be tracked. The interface for the callback methods must be like: attribute_t get(); set(attribute_t); compute();

Note:

Include the pthread library when linking (-lpthread).

Warning:

Although it creates and uses threads, the interface is not itself thread safe.

2.2.2 Member Function Documentation

2.2.2.1 void executive::compute_all ()

Creates a thread for each computer for each attribute and asynchronously calls them.

2.2.2.2 void executive::get_all ()

Creates a thread for each supplier and asynchronously gets the value for each attribute by calling the suppliers.

2.2.2.3 void executive::set_all ()

Creates a thread for each consumer and asynchronously passes the value for each attribute and in each consumer by calling asynchronously all the consumers for that attribute.

2.2.2.4 void executive::wait ()

Waits for all threads to terminate. These threads are those created with `get_all()`, `set_all()` and `compute_all()`.

The documentation for this class was generated from the following files:

- executive.h
- executive.cc

2.3 executive_attribute< T > Class Template Reference

Stores an attribute and manages the supplier, consumers and computers associated with that attribute.

`#include <executive.h>`

Classes

- struct **computer**
- struct **consumer**
- struct **supplier**

Public Member Functions

- template<class O >
  executive_attribute (executive &ex, O *obj, T(O::*get)() const)
- template<class O >
  executive_attribute (executive &ex, O *obj, T *(O::*get)())
- template<class O >
  executive_attribute (executive &ex, O *obj, T &(O::*get)())
- template<class O >
  void subscribe_consumer (O *obj, void(O::*set)(T))
- template<class O >
  void subscribe_consumer (O *obj, void(O::*set)(T *))
- template<class O >
  void subscribe_consumer (O *obj, void(O::*set)(const T &))
- template<class O >
  void subscribe_compute (O *obj, void(O::*compute)())

Public Attributes

- T value

2.3.1 Detailed Description

template<class T> class executive_attribute< T >

Stores an attribute and manages the supplier, consumers and computers associated with that attribute. Subscribe the consumers and computers before calling the `executive` 's methods. TODO: Unsubscribing not implemented, you must delete the executive_attribute instance and create it again without the desired callbacks.

2.3.2 Constructor & Destructor Documentation

2.3.2.1 template<class T > template<class O > executive_attribute< T >::executive_attribute (executive & *ex*, O * *obj*, T(O::*)() const *get*) [inline]

Parameters:

ex The executive that will keep track of the value associated with this attribute.

obj The object to which the get call applies.

get Pointer to member method that supplies the attribute by polling it. It must be of the form `attribute_t get_t::get_method();`

Retrieve the pointer to member method with `&get_t::get_method` .

2.3.2.2 template<class T > template<class O > executive_attribute< T >::executive_attribute (executive & *ex*, O * *obj*, T *(O::*)() *get*) [inline]

The same as `executive_attribute(executive & ex, O * obj, T (O::*get)() const)` when the supplier returns a pointer to the attribute and executive_attribute must take care of that memory.

Parameters:

ex The executive that will keep track of the value associated with this attribute.

obj The object to which the get call applies.

get Pointer to member method that supplies a pointer to the attribute by polling it. The caller is responsible for deallocating the pointer, so the supplier must return a pointer to a memory it will not manage after returning. `get` must be of the form `attribute_t * get_t::get_method();`

Retrieve the pointer to member method with `&get_t::get_method` . TODO: Not tried with arrays. Probably will not work, as it needs an additional argument indicating the number of elements in the array.

Warning:

Uses `delete` to dealloc the memory, which is not portable when using CORBA.

2.3.2.3 template<class T > template<class O > executive_attribute< T >::executive_attribute (executive & *ex*, O * *obj*, T &(O::*)() *get*) [inline]

The same as `executive_attribute(executive & ex, O * obj, T (O::*get)() const)` when the supplier returns a reference to the attribute.

2.3.3 Member Function Documentation

2.3.3.1 template<class T > template<class O > void executive_attribute< T >::subscribe_compute (O * *obj*, void(O::*)() *compute*) [inline]

Parameters:

obj The object to which the compute method should be called with.

compute Pointer to member method that makes computations based on the attribute. It must be of the form `void compute_t::compute_method();` Retrieve this pointer with `&consumer_t::consumer_method` .

2.3.3.2 template<class T > template<class O > void executive_attribute< T >::subscribe_consumer (O * *obj*, void(O::*)(const T &) *set*) [inline]

The same as `void subscribe_consumer(O * obj, void (O::*set)(T))` when the consumer receives a reference to the attribute.

Warning:

`set` has `const` in the parameter by reference so you cannot modify the attribute.

2.3.3.3 template<class T > template<class O > void executive_attribute< T >::subscribe_consumer (O * *obj*, void(O::*)(T *) *set*) [inline]

The same as `void subscribe_consumer(O * obj, void (O::*set)(T))` when the consumer receives a pointer to the attribute.

Parameters:

obj The object to which the set (consumer) method should be called with.

set Pointer to member method that needs to read the attribute. It must be of the form `void consumer_t::consumer_method(attribute_t *);` Retrieve this pointer with `&consumer_t::consumer_method` .

Warning:

Do not forget to deallocate the pointer in the consumer method.

2.3.3.4 template<class T > template<class O > void executive_attribute< T >::subscribe_consumer (O * *obj*, void(O::*)(T) *set*) [inline]

Parameters:

obj The object to which the set (consumer) method should be called with.

set Pointer to member method that needs to read the attribute. It must be of the form `void consumer_t::consumer_method(attribute_t);` Retrieve this pointer with `&consumer_t::consumer_method` .

2.3.4 Member Data Documentation

2.3.4.1 template<class T > T executive_attribute< T >::value

The attribute that is managed. Set it instead of calling the supplier the first time to have a default value. Read it between calls to `cycle` for statistics.

The documentation for this class was generated from the following file:

- executive.h

2.4 phony Class Reference

Auxiliary class that helps circumvent C++ type safety. Please ignore.

`#include <executive.h>`

2.4.1 Detailed Description

Auxiliary class that helps circumvent C++ type safety. Please ignore.

The documentation for this class was generated from the following file:

- executive.h

Apéndice C

Manual de usuario

Chapter 4

User Manual

4.1 Functionality description

The Robotic Control Testbed is a Mobile Robot built from a commercial robot base with added funcionality. The robot base consists of a four wheel vehicle with position and speed control embedded, integrated power and control electronics and 16 ultrasonic sensors that give readings on the distance to close obstacles. Additionally front and rear bumpers have been installed for safeguarding the robot while operating it in very constrained areas and for manual stopping.

The Laser range reader can measure distances from 8mm up to 80m, depending on configuration. Typically it will function with a resolution of 1mm and a range of 8mm to 8m. It is the main sensor along with the odometer readings in SLAM navigation. It can be rotated to make three dimensional maps with the servo. The mechanism operates in open loop mode, which gives a low precision angular rotation, given the mechanical hystheresys and the thermal and age deviations of set.

Next to the laser the I/O board, or data acquisition board, is a general purpose input output board that connects to a compass, a two axis accelerometer, the laser servo that controls its tilt movement, and two current and voltage sensors one for monitoring the consumption of the robot base and the other for the power board and related devices. These sensors and actuators are controlled through the I/O board module.

For outside applications there is a global absolute position sensor which consists of a GPS antenna, Radio receiver and battery for receiving differential corrections from a base station and a GPS receiver that processes the data and converts it to an easier format readable by the onboard computer.

Visual applications can make use of the binocular camera, which in combination with the wrist, that gives pan and tilt movement to the camera with respect to the robot base and is located on top of the laser, can direct the focus of the attention in almost any direction. The wrist is internally controlled in position, speed and acceleration, which gives precise and accurate motion control to the camera.

The onboard computer runs a full capable operating system and is the communications hub for controlling remotely all the devices on the robot. Software controlling the devices may be installed onboard or in a remote computer.

4.1.1 Visual identification of devices

Figure 4.1: General view of Higgs.

Figure 4.2: Robotic base Pioneer 2AT.

Figure 4.3: On board computer: VAIO laptop.

Figure 4.4: Laser sensor.

Figure 4.5: Wrist.

Figure 4.6: Camera.

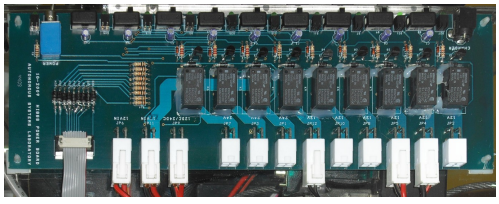


Figure 4.7: Power board.

Figure 4.8: 12V to 24V converter.

Figure 4.9: Differential GPS set. Rover station parts.

Figure 4.10: Differential GPS set. Base station parts.

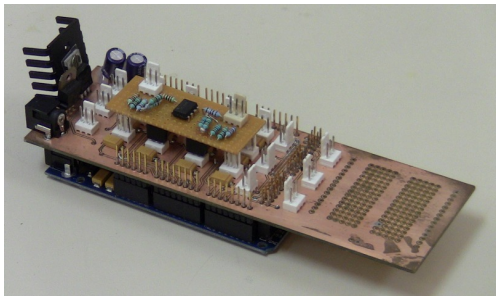


Figure 4.11: Input/Output board: Arduino.

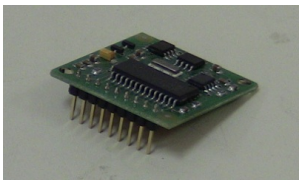


Figure 4.12: Compass.

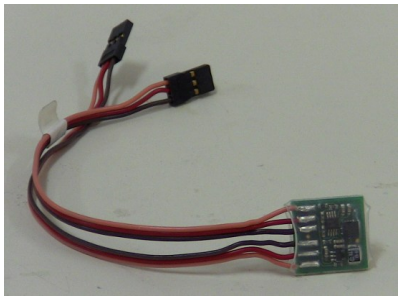


Figure 4.13: Accelerometer.

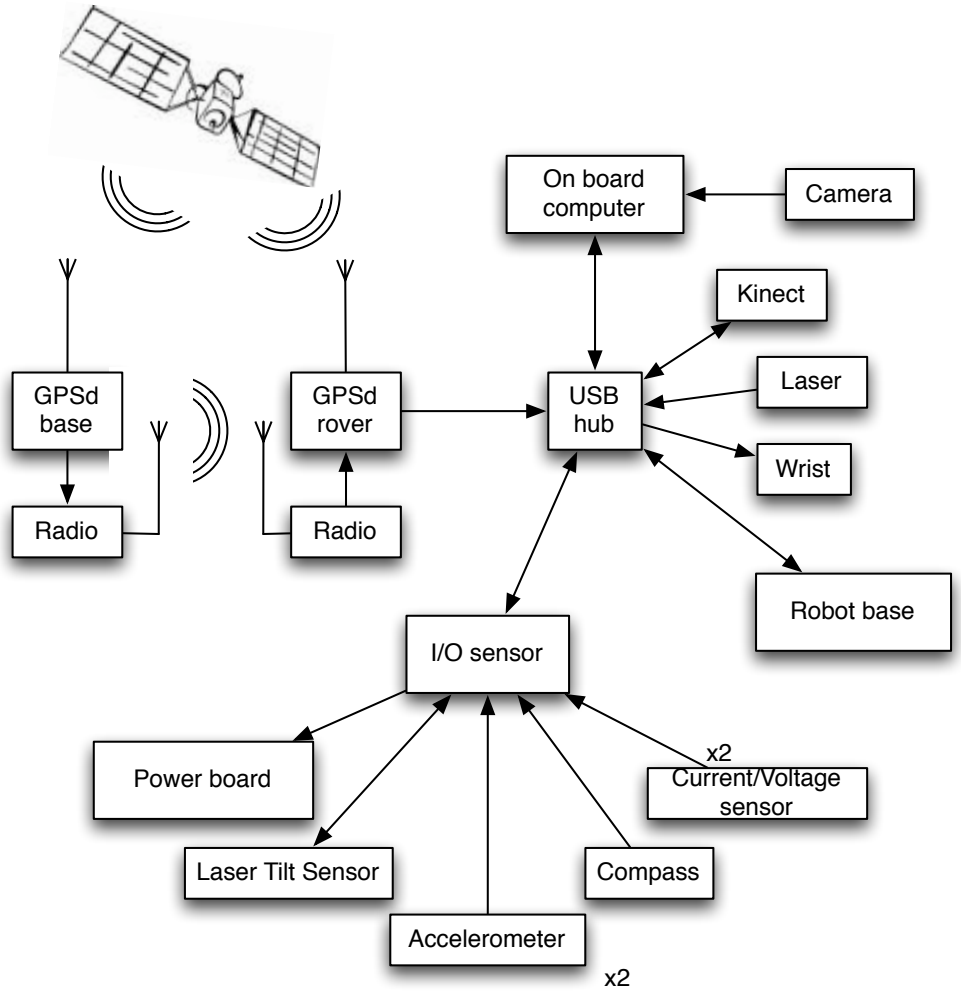


Figure 4.14: General data connection diagram

4.1.2 Data connection diagram

Arrows indicate the direction of valuable information flow.

4.2 Setting up the system

Preparing the robot for operation is easy. Power it on and all systems will start automatically. On next sections it will be discussed how to command it.

4.2.1 Powering the robot

There are three switches that must be turned on to enable the robot with full capabilities. The first one is at the back of the robot, next to the wheels. This switch powers the Pioneer2AT8 robot base and the power board. The power board has a general switch for all devices and one more for each device for

manual disabling. The power board is placed between the two big methacrylate structures. Each switch has a label with the device that enables. The last device to power on is the VAIO laptop.

Note: The devices powered by the power board can be also disabled automatically by the Arduino. When the Arduino is not correctly powered on, the voltage levels at its pins are undefined, and may disable the power to some devices. Be sure to power the arduino if you are going to use any other device from the power board.

Only the laptop is mandatory to power on. The other switches may stay off if you are not going to use the device associated to that switch. However you will have to start at least one device to make it useful.

The Arduino does not disable any device by default. Both the manual switches and the Arduino may force the shutdown of any device, so to use a device, be sure that none of them disables it. The automatic switches are remotely controlled through the Arduino module.

4.3 Basic maintenance

There is little maintenance to do with the robot. The batteries are the most important matter to be aware, followed by the wheels.

4.3.1 Wheels

Once every two months or so, the wheels will loose pressure and they must be inflated evenly, this way the odometry will not loose precision. It can be noticed when the wheels have deinflated by looking to the tread pattern: Two separate bands of moist will indicate underinflation, one on the middle overinflation, and correct inflation when moist is evenly distributed. There is a manual pump with manometer in the cabinet.

4.3.2 Battery management

The robot has 3 battery packages. One of them is inside the Pioneer2AT8, the other one powers the radio receiver for the differential GPS and the last one is embedded in the laptop.

Robot base lead batteries

There are three lead battery packages inside the Pioneer2AT8 that powers the motors, the internal electronics and the external power board. To access them, lift the small black lever on the back side of the robot and turn it to the left. This will loosen the battery door. Open it to 135°. You may have to pull the door up to get the door over the bump sensors. Once it is open,

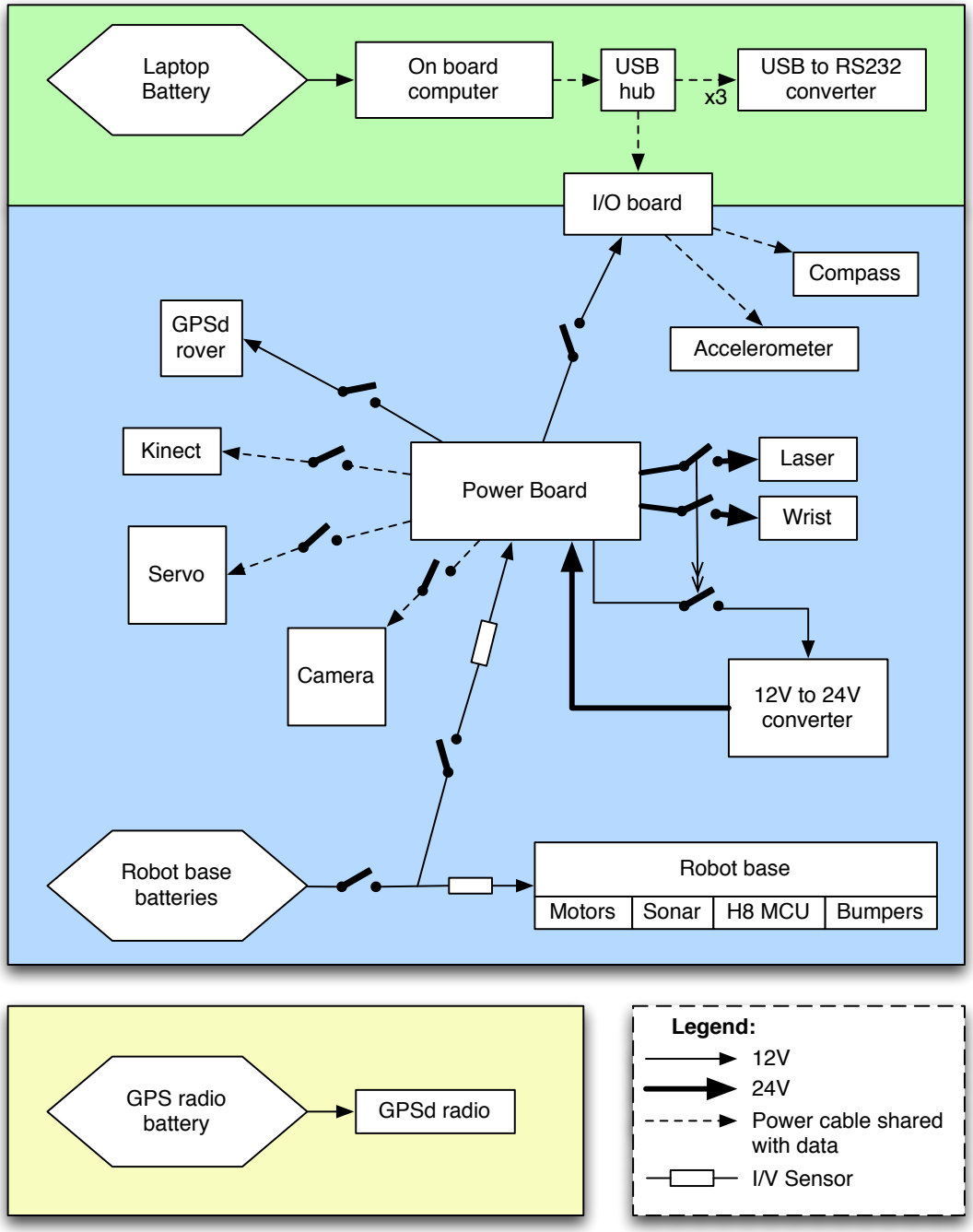


Figure 4.15: General power connection diagram

pull the batteries with the black sucker, or carefully with the hands do not get yourself pinched. When inserting the new batteries, remember that **the battery terminals go last**. Failure to introduce them correctly will cause short circuits and damage the electronics.

Figure 4.16: Robot base charger.

Recommended voltage when using the batteries is between 11.5V and 12.5V. If the batteries are too low the robot will indicate it with continuous short beeps. You can check the battery status through the battery LED in the Pioneer2AT8 panel: Green when fully charge through orange downto 11.5V, finally red. To keep the batteries in good conditions, do not discharge them completely. Doing so will decrease the charge capacity. It is better to fully charge them after each use.

To charge the batteries, connect the charger to the robot through the power jack connector to the left of the battery door. Its charger is the big black one, model PSC-124000A. You may leave the charger connected for as long as you want, but remember to **switch off the robot every night**, as electricity in the laboratory is turned off and the robot will fully discharge the batteries in this time, damaging them.

GPSD radio battery

There are two radio batteries, one for the GPS base station and the other for the GPS mobile station. Both batteries are interchangeable, and are 8x9x23cm in size with a black leather cover. Ni-MH batteries have memory effect, so

you must discharge them completely before charging them again. There are two chargers. you may found them in the yellow suitcase in the “F room” or in a white long box next to the suitcase. Remember that the “F room” is the name for the ASLab cabinet in the computer rack room. To charge it, open the battery tab and connect the charger. Press the button for discharging it fully, then wait until the charger finishes the operation. Detailed instructions are printed on the charger.

Figure 4.17: GPSD radio charger.

VAIO laptop battery

The VAIO laptop manages automatically its own battery. You may check its status with the command `acpi -b`. On linux RTAI, ACPI is not supported by the kernel and it will not work so this command is unavailable.

Figure 4.18: VAIO laptop charger.

4.3.3 Diagnostics

Power

There are several indicators of malfunction that can ve verified in case the robot does not work properly. In the first place, look for correct power in each device. This is how you can check it:

Pioneer2AT There is a red LED inside the Pioneer when it is powered, but can only be verified indirectly through reflections in the methacrylate structure that covers the top hole of the base. Alternatively, There is a dedicated red LED for this purpose in the Control Panel with the label *PWR*.

Laptop The VAIO laptop lights a green LED when it is on.

Laser When it is powered on, a green LED or two orange and red LEDs will be lighted in the front of the laser device. Green means powered and prepared, orange and red means powered and initializing.

Arduino When it is powered a little orange LED can be find behind the USB connector. A fast sanity check is to power the arduino and the servo. If everything is OK, the Laser will rotate to look upwards in a small angle.

Wrist There is no way to verify this device externally. However, it is quite robust and will normally work as supposed to. If everything is running ok, it will make a little calibration movement on startup.

Camera Depends on the camera used. The black stereoscopic one has a red LED that blinks when the driver is reading it and the Minoru3D lights up in white when the driver is reading it.

GPS There are two LEDs on the side. One of them indicates that power is OK and the other lights when enough satellites for position calculation are being tracked.

Power Board There is a green LED for the general switch and one for each device.

4.4 Operating the robot manually

4.4.1 Booting the onboard computer and choosing the OS

The onboard computer has several operating systems installed:

WindRiver OS `/dev/sda1` (1GB). This was once used as a testbench for the WindRiver OS. It is several years old and is not used any more.

Windows XP Professional `/dev/sda2` (25GB). The original Windows OS prepackaged with the laptop. The NovAtel GPSD utilities have been installed here for quick GPS diagnostics.

Ubuntu 10.04 Long Term Service `/dev/sda3` (14GB). This is the current working environment, with Real Time Application Interface (RTAI) kernel.

Fedora 13 `/dev/sda5 (29GB)`. Has the previous working environment, with CORBA modules and depending on the nameserver in the old onboard computer. Has a custom driver for supporting the FireWire camera.

Swap partition /dev/sda6 (100MB).

On bootup there is a selection of the operating system to start. It includes Ubuntu with and without real time kernel, a RAM memory test and the other three OSs. The default OS is Ubuntu with the RTAI kernel. Note that only the current working OS, that is, the Ubuntu with RTAI kernel, is documented here. The other operating systems are old but kept for backup and compatibility with old software.

Shutting down.

The correct procedure to shut down the onboard computer is to log in as root and request shutdown:

```
root@higgs2:/root# halt
```

Forcing instant shutdown using the power button is not recommended, however, normally there has been no problem with the OS afterwards.

4.4.2 Loggin in to the onboard computer through SSH

The Secure SHell server is run in the onboard computer by default and open to everyone that has a user account or the root password. The root password is the same as the root password for all the computers in the laboratory. Ask your tutor for a local account on higgs or the root password. Access can be obtained by opening a terminal and executing the command

```
your-pc:/$ ssh account_in_higgs@higgs2
```

and root access by changing `account_in_higgs` with `root`. If you want to execute programs that use a graphical interface, create a ssh tunnel for X with the option `-X`:

```
your-pc:/$ ssh -X account_in_higgs@higgs2
```

This way the program will execute in the onboard computer and display in your local screen remotely. In case you get this error while trying to connect:

```
 ssh: Could not resolve hostname higgs2:\
Name or service not known
```

then you have a problem with the name resolution on your computer. Copy these lines to `/etc/hosts` as root to solve it:

```
127.0.0.1 localhost
```

```
138.100.76.251  sagan.aslab.upm.es sagan

138.100.76.20   noe.aslab.upm.es noe
138.100.76.26   churchill.disam.etsii.upm.es churchill
138.100.76.250  quark.disam.etsii.upm.es quark
138.100.76.239  parker.disam.etsii.upm.es parker
138.100.76.252  aldiss.disam.etsii.upm.es aldiss
138.100.76.244  corea.disam.etsii.upm.es corea
138.100.76.243  mingus.disam.etsii.upm.es mingus
138.100.76.217  verne.disam.etsii.upm.es verne
138.100.76.241  gibson.disam.etsii.upm.es gibson
138.100.76.204  pohl.disam.etsii.upm.es pohl

138.100.76.15 ende.disam.etsii.upm.es ende

# 138.100.76.196  luna.disam.etsii.upm.es luna
138.100.76.196  arturo-desktop.disam.etsii.upm.es arturo-desktop

138.100.76.247 higgs.disam.etsii.upm.es higgs
138.100.76.246 higgs2.disam.etsii.upm.es higgs2
```

Now your computer knows how to translate the hostnames of the laboratory to IP's. Alternatively, if in a rush substitute `higgs2` with its IP, as in

```
your-pc:/$ ssh local_account@138.100.76.246
```

The machine with name `higgs` corresponds to the old on board computer which was embedded inside the robot base.

Once logged in to higgs2, applications can be run as in a standard linux distribution. Linux RTAI runs a normal kernel with standard functionality ontop of the realtime features. For using realtime, the programs must be loaded directly into the kernel as modules. All other programs run in soft real time.

On bootup, the onboard computer will connect automatically to the wireless accesspoint `aslab_wireless`. This accesspoint is in the same network as the other computers. The connection is configured using the ESSID of the wi-fi hotspot and the MAC address too.

4.5 Testing the modules

The CORBA servants access the NameService to publish their services[1]. The table 4.1 shows the "`id`"'s used by each module, being the "`kind`" parameter empty for all of them.

[1] If appropriate module is installed and running.

Description	IDL	Name Service ID
Camera	Camera.idl	CAMERA
I/O Board	Arduino.idl	Arduino
Robot base	Pioneer2AT.idl	PIONEER
Wrist	wrist.idl	wrist
Battery Model	BatteryModel.idl	BatteryModel
Current Monitor	BatteryModel.idl	CurrentAverage
Laser	Laser.idl	LASCOR
GPS	gps.idl	GPS

Table 4.1: Names of the CORBA objects registered in the NameServer.

4.5.1 Subversion.

The device modules have test programs that can be run either locally from the onboard computer or remotely without needing to log in. This section and the next one are a quick guide for reconfiguring and troubleshooting easy problems with the devices. Any problem not solved here requires further understanding of the robot software mechanisms and are described in the developer manual.

The first thing to do is download the source code. Supposing you already have installed the necessary programs and libraries, type

```
svn co svn+ssh://sagan/home/svn_repositories/Higgs
```

to check out the source. Again, replace `sagan` with 138.100.76.251 if your hosts.conf is not correctly configured and prepend it with your user name and an @ if your server user is not the same as the local user.

There is a second repository with older modules and code that have not yet been ported to the new CMake - subversion schema.

```
svn co svn+ssh://sagan/home/svn_root[/Higgs]
```

An even older CVS repository exists too.

4.5.2 Subversion directory hierarchy

Once finished, three directories will be available:

`trunk` The latest code available using the technology currently in development in the robot, which is ROS at the time of this writing.

`branches` Alternative code using other technologies, which currently is only CORBA.

`docs` The source code of this document.

This is the resumed directory tree inside `branches/CORBA`:

```
code
|-- LowLevelControl
|-- WorldModel
|-- batteries
|-- control_libraries
|-- devices
|   |-- arduino
|   |-- camera
|   |-- gps
|   |-- laser
|   |-- vaio_tools
|   `-- wrist
|-- idl
`-- lib
```

The directories to consider when programming new clients are:

`code/idl` Contains the IDL definition files of all the modules necesary for controlling the robot. You may open it for accessing the documentation for the interfaces and you will have to link to it for generating the stubs and skeletons.

`lib` C++ macro and CMake files (See section 4.6.2) for fast client development.

4.5.3 Checking the environment for starting test programs.

Configuration files

The next sections describe how to run the test programs contained in the CORBA branch for testing the devices.

Before running the test clients the configuration files must have been installed for proper operation. Clients only need one configuration file:

```
/etc/higgs/nameservice.ip
```

containing the address and port of the Naming Service that the servants are using for publishing themselves:

```
higgs2:9876
```

Configuring serial port links

On the server side there are a few more entries inside `/etc/higgs`, the most important one for configuration issues being `devices`. This directory con-

tains soft links to the character devices that represent the USB to RS-232 converters in `/dev`. The servants open these files instead of the real devices so they can be reconfigured without having to recompile. Usually you will modify these links when swapping the serial cables of the devices or changing the arrangement of the USB to serial converters. Knowing which device file goes with which device is a matter of trial and error, starting the servants and testing wether they started or not. See 4.5.3 for more information on servant logging.

Starting and stopping the CORBA servants

The servants use upstart, the standard utility in Ubuntu for booting the system and running the daemons. You may start or stop the servants in case you need the correspondant devices, i.e. the laser and the robot base in ROS, or they are not automatically started on boot.

To start a servant use:

```
$ start higgs_device
```

and to stop it,

```
$ stop higgs_device
```

where `device` is one of laser, wrist, gps, arduino, pioneer or any other available CORBA servant.

There is one more daemon, the CORBA Naming Service, that opens the port 9876, is always running and does not interfere with the devices.

Logs and troubleshooting

Servants print their output to a log file in the on board computer placed at `/var/log/higgs`. You will have to consult these for debugging problems with the servants such as not starting, setting the device file and checking overall status.

4.5.4 I/O board

The next procedure is standard on all modules. Enter the directory `$(HIGGS_ROOT)/branches/CORBA/code/devices/arduino/client`. Be sure to have the complete source tree, at least the code subdirectory, as many methods rely on files situated back in the tree. Run:

```
cmake .; make
```

This will generate the CORBA stubs and headers and then compile the client code. The binary `arduino_client` will appear. Run it to read the parameters of the devices attached to the IO board and setreset some of the devices.

4.5.5 GPSd

You may want to test the full GPSd equipment with differential corrections or only the rover part. The differential readings may not work in the campus because of interferences in the environment that blocks radio communications between the base station and the rover.

Rover

the serial cables are correctly installed. The USB to RS232 converter should be attached to COM1. Turn on the GPS switch. Now to the software part: Go to
`$(HIGGS_ROOT)/branches/CORBA/code/devices/gps/src` and run `cmake .; make`. Run `gps_client`. A command line menu will be printed form where you can check the satellites used in the solution, the current position and speed, the standard deviation for the position and the type of differential corrections used, if any.

Base station

The software part is the same as in the Rover part, with the standard deviation reducing to 0.02m or so if the differential correction is working. Preparing the base station:

Go to the ASLab closet and locate a yellow bag. Take out the black leather battery, the blue radio transmitter with the antenna, the white box housing the electronics and the cables. The GPS antenna is located on the roof with the coaxial cable hanging down the facade to the back of the room where Higgs lives. Open a window and take it inside. Beware of your workmates in winter! Connect it to the electronics box. Power the electronics box with 12V, for example using Higgs charger, and the radio transmitter to the battery pack using the appropriate cable. Finally connect the electronics box to the battery pack with the serial terminal attached to COM2. The battery pack is internally wired to connect the serial data and the power to the cable attached to the serial transmitter. Press power in the serial transmitter. After a few minutes, the Tx led should start blinking every second. This is the indication that the differential readings are being transmitted.

In the rover part, attach the three terminal cable to the blue radio receiver, the battery pack and the COM2 port in the rover GPS electronics box. Press the power button in the radio receiver and proceed as in the rover section.

4.5.6 Laser

Procedure is similar to the I/O board one. Go to
`$(HIGGS_ROOT)/branches/CORBA/code/devices/laser/src` and `cmake .; make`. `laser_client` will be generated. Power the laser in the

robot, wait for it to go green and some more seconds for the laser servant to start up, and run the client. A list with the distances of the latest reading will be printed to the console.

4.5.7 Wrist

Again, the procedure is similar, only in this case the device is not read but told the position to move to. Turn on the switch for the wrist and wait for a little calibration movement that indicates that it is ready. It will move to the reset position if not there before the calibration. Go to `$(HIGGS_ROOT)/branches/CORBA/code/devices/wrist/src`. Two clients will appear. The first of them, `wrist_client`, will move the two axis with incremental span around ten times then stop. The second one, `wrist_client_mouse_grab`, will let you control the two axis with the mouse. Once running, take the pointer to the center of the window to move the wrist to the starting position, then slowly move the pointer and the wrist will follow it. The range of movements is limited by the servant for secure operation whatever the input is. However, the batteries and converter may not be able to provide the required power if fast enough movements are requested.

4.5.8 Robot base

TODO

4.6 Developing a client

4.6.1 ASLab client utility functions for CORBA

The methods for initializing the CORBA infrastructure and getting the object references is quite complicated and always the same. The file `code/lib/CORBA_utils.h` contains C++ macros that ease the procedures for setting up a client, managing the errors, reading the object references from the nameserver and configuring itself for reaching that nameserver.

The typical client would be something like:

```
#include <iostream>
#include "implementationC.h"
#include "CosNamingC.h"
#include "Higgs/branches/CORBA/code/lib/CORBA_utils.h"

int main(int argc, char* argv[])
{
    CORBA_BEGIN_CLIENT(argc, argv);
    CORBA_GET_REFERENCE(module::implementation, impl, "IMPL");
```

```
    impl->do_things();

    CORBA_END_CLIENT;
    return 0;
}
```

With the correct route to `CORBA_utils.h`. The arguments for `CORBA_GET_REFERENCE()` are:

1. `module::implementation` The type of the object to fetch.
2. `impl` The identificator desired for the reference. It should not be defined nor declared, it is done inside the macro. Remember that references are used as pointers.
3. `"IMPL"` String with the name that the object is registered in the NameServer.

4.6.2 CMake

It is encouraged to use CMake as the tool for managing the compilation of the proyects. There is a sample CMakeLists.txt in `Higgs/branches/CORBA/code/lib` prepared for linking the CORBA libraries and generating the necessary dependencies for compiling. Substitute `_module` with the name of your module. The file `IDL_command.cmake` defines a macro for generating and managing the sources of the IDL interfaces. Include all interfaces that you need when calling the macro `MacroGenerateIDL`. Use the command

```
SET(IDL_DIR path/to/idl)
```

pointing to higgs' idl definition files to tell the macro where to find them.

Apéndice D

Manual del desarrollador

Chapter 5

Developer Manual

The RCT testbed has been designed and developed with the tools and libraries that are currently, as the time of this writing, the state of the art. These tools are usually replaced by easier and more powerful ones with time, and Higgs can benefit from these. When such a change is performed, normally it will invalidate some of the documentation herein. Remember to write down the procedures and descriptions to match the new devices and/or software. When replacing a broken part, this chapter can be used as a guide for installing the new components. The tools and libraries in use are defined, as well as how they have been installed and configured, and a detailed description of the modules, both hardware and software.

5.1 Mobile base

5.1.1 Disassembling the robot

The main reason for disassembling the robot is to access the inside of the mobile base for repairs and updates. The remaining components can be accessed without major disassembling. The robot base has two black plates screwed to the red chassis with a joint between them. This allows to access different parts of the robot without fully disassembling it. The front part of the robot is where the motors are placed and the inner on board computer can be fitted. No computer is currently installed inside so it will rarely need servicing. See section 5.1.1 to access it. The back part houses all power and control electronics and the batteries and can be reached disassembling the back methacrylate structure with the alluminium gantry. This is achieved by unscrewing all six bolts fixing it to the robot base. Be sure to unplug all necessary cords when taking it out. The robot base will power up but will not work correctly if the multi-coloured ribbon cable is not properly connected to the black instrument panel of figure 5.1.

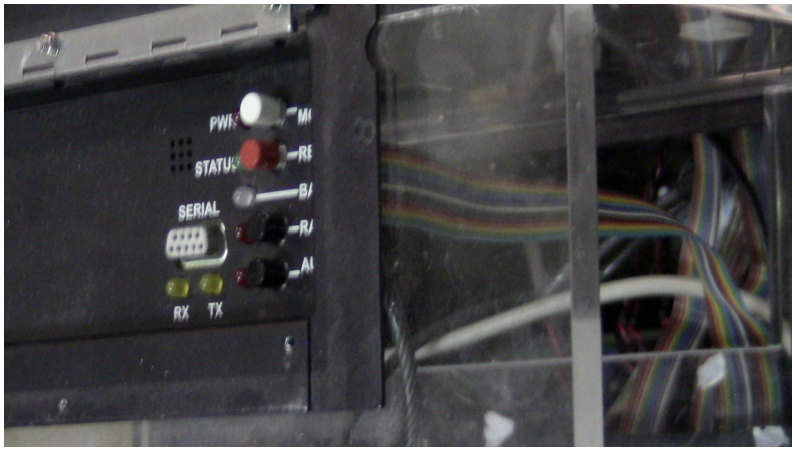


Figure 5.1: Robot base instrument panel.

Disassembling the laser

There are two ways to disassemble the laser. The first one removes the block composed of the laser sensor and the metacrylate structure that supports it. Get a number 5 allen wrench and unscrew the bolt placed under the laser, tiliting the latter upwards to reveal it. Then remove the two similar bolts on the other border of the alluminium plate that the first bolt was also holding. The methacrylate structure will be loose but do not take it off yet. Unplug the cords attached to the arduino board and the two data and power cords of the laser. Once all cables are released the structure can be lifted. Be careful not to damage the accelerometers when reassembling as its placement has been carefully crafted to fit between the laser structure and the arduino connection board.

The second way allows for easier but more tedious disassembling. Remove the radio receiver of the GPS turning it as it was a huge bolt. Then unscrew the four bolts holding the upper methacrylate plate of the laser structure. Be careful with the weight of the devices fixed to this plate. This will leave the two walls standing to the sides of the laser. Looking from a robot point of view, the right wall can slide outwards releasing the laser from its side constraints. Take it out after unplugging the power and serial data cords. If further disassembling is required proceed as in the previous paragraph.

Reaching the inside of the mobile robot.

Once the top part of the mobile base is free from devices and structures both front and back inside parts of the robot can be reached unscrewing the littlest black bolts on the black plates. Additionally you can also remove the sonar sensors for reaching deeper inside the robot. There are four little vertical bolts inside the structure that holds each sonar array. Remove the ribbon cable prior to disassembling it.

In the back part of the robot reside the electronics in two layers. To access

the bottom boards the top ones must be removed unscrewing the four small bolts situated behind the back wheels at the side of the robot and removing the sonar array so it can be slipped out.

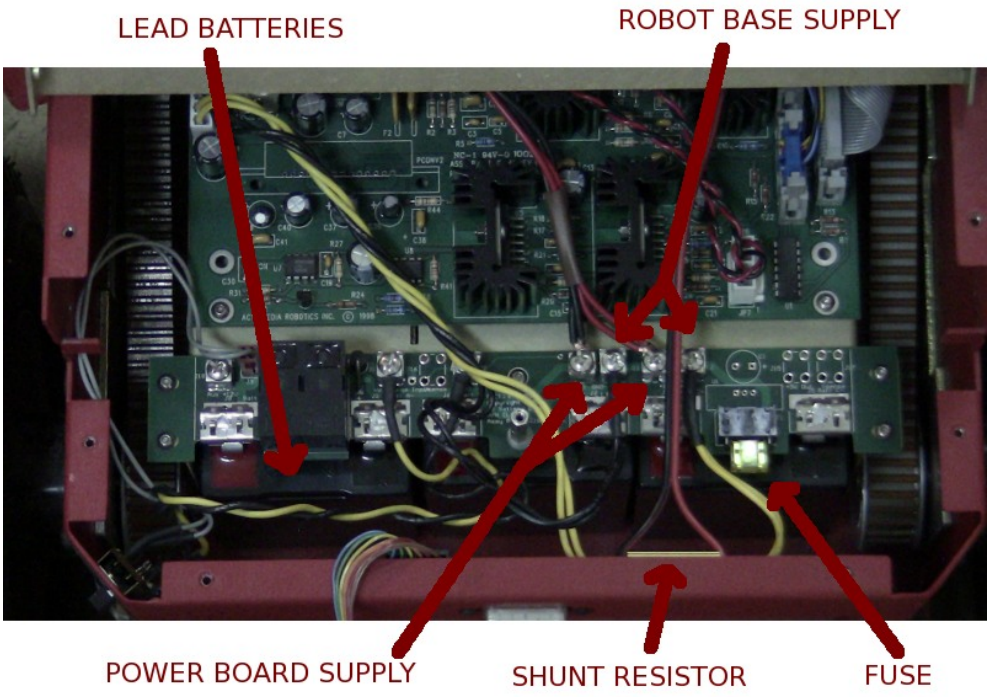


Figure 5.2: Robot base front part disassembled.

5.1.2 Firmware

It is possible to update the firmware of the Mobile base, uploading it to the nonvolatile memory of the Hitachi H8 microcontroller. Check the Pioneer 2 H8-Series Operations Manual. It is also possible to update the tick count of the odometry.

5.2 On-board computer

The onboard computer is the sony VAIO laptop running tUbuntu linux 10.04 with a RTAI kernel. The older onboard computer is a GENE board that was placed inside the mobile base with its own dc/dc regulator running WindRiver. See figure 5.3

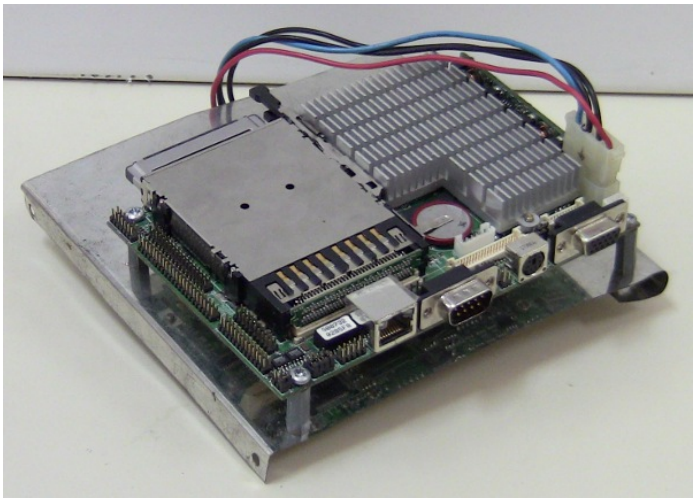


Figure 5.3: The old onboard computer GENE

5.2.1 Hubs and converters

The onboard computer has two USB ports and one FireWire port among others. The FireWire port is used with the binocular camera. The arduino, GPS, laser, wrist and robot base all use serial RS-232 communication which is achieved using USB to RS-232 converters. There is a hub behind the power board with 4 USB ports to where the converters are plugged, either directly (two of them), with a short USB cable (i.e. the GPS) or has the converter chip embedded inside the device (i.e. the data acquisiton board). There is a lack of one serial converter for having all five devices working at the same time. Be sure to reconfigure the serial ports at `/etc/higgs/devices` when changing the connected devices, see 4.5.3.

5.2.2 Real Time Operating System

This section describes the steps that have been made for building and installing the Ubuntu RTAI operating system.

1. Download and burn Ubuntu 10.04 LTS Desktop Edition, then install it like a standard Ubuntu installation. The Long Term Support surname guarantees that this version will be supported for 3 years.

2. Install these packages: `libncurses-dev build-essential kernel-package linux-source` They are needed for building the kernel and generating a debian package.

3. Download and uncompress the vanilla linux kernel 2.6.32.11. The version must much exactly so the RTAI patch can be applied smoothly. The linux kernel sources can be downloaded from `http://www.kernel.org/`.

4. Get the RTAI patch form `http://www.rtai.org/` As the time of this writing, the latest version was 3.8.1.

5. Apply the patch.

```
cd /pathtolinuxkernel2.6.32.11/
patch -p1 < /pathtortai/rtai-3.8.1/base/arch/x86/\
    patches/hal-linux-2.6.32.11-x86-2.6-0.3.patch
```

   Be sure to use the patch for the exact kernel vanilla version, in this case 2.6.32.11, or errors and warnings will arise. The `-p1` option removes the base directory form the path of the files inside the patch.

6. Configure the kernel. First, copy the ubuntu kernel config

```
cp /lib/modules/`uname -r`/build/.config \
    /path_to_kernel-2.6.32.11
```

   and run `make menuconfig` inside the root directory of the downloaded kernel sources. Look for the following options and change them to the appropriate values: local version-append to -rtai.3.8.1-1; number of CPUs to 1; ACPI to no; all power management features to no, module versioning support to yes, interrupt pipeline to yes. The RTAI patch needs ACPI not to be supported by the kernel. As a consequence, no power management features will work and the kernel will not be able to run the HALT instruction on shutdown.

7. Compile with `make`. This can last many hours.

8. Generate the debian package.

```
fakeroot
make-kpkg --initrd kernel_image kernel_headers
```

9. Go to the upper directory, where the packages are created, and install them with `dpkg -i *.dev`.

Once the kernel is installed, reboot and start with the new kernel to check it con boot. Now it is time to finish the installation compiling the realtime kernel modules and utilities.

1. Go to the root directory of the RTAI source code and run `make menuconfig`. Select the number of target cpu's with the same number as the kernel and write the path to the kernel sources.

2. `make` and as root `make install`.

3. Configure the libraries as root.

```
echo /usr/realtime/lib > /etc/ld.so.conf.d/rtai.conf
ldconfig
```

4. Add `/usr/realtime/bin` to the path of all users' .profile files.

   ```
   echo "PATH=$PATH:/usr/realtime/bin" > ~/.profile
   ```

5. Finally, check that the realtime extensions are working correctly. Go to `/usr/realtime/testsuite/kern/latency` and run `./run`. If realtime is correctly installed, the last column should be all zeroes after 2-3 minutes.

6. Optionally, if space is low remove the packages and the source for the kernel and patch, or only the object files with `make clean`.

5.2.3 OS Tweaks

There are some modifications to be done to the Operating System to have Higgs' modules running flawlessly.

There must be a username called higgs and it must be a member of the dialout group. Note also that by default the iptables firewall is enabled on many distributions and it must be configured or disabled before CORBA clients can make calls to the servants.

Kernel modules

The linux kernel loads the serial modules for the converters at startup by default, but the order in which it does it is not the same on each bootup, so the serial device links must be checked on each bootup with this setup. The modules are the FTDI driver `ftdi_sio.ko` and the pl2303 driver `pl2303.ko`. This inconvenience has been avoided by moving the kernel file objects from their standard placement at `/lib/modules/` to `/etc/higgs/modules`. The kernel tries to load the modules but fails because it can not find them where it expects them to be. The modules are loaded with the init scripts in a determined order. The following commands as root can be used to set up this behaviour on new installations:

```
cd /etc/init.d
echo "#!/bin/sh" > load_serial_modules
chmod a+x load_serial_modules
echo "insmod /etc/higgs/modules/pl2303.ko" \
 >> load_serial_modules
echo "insmod /etc/higgs/modules/ftdi_sio.ko" \
 >> load_serial_modules
ln -s /etc/init.d/load_serial_modules \
 /etc/rc3.d/S60load_serial_modules
```

Replace `rc3.d` with the runlevel the OS starts with. You may instead prefer to modify the init script skeleton and give more functionality such as removal of the modules using the standard init V procedure such that loading and unloading the modules is done with `./load_serial_modules start/stop`.

Daemon scripts

The onboard portable computer is running many of the CORBA modules under Linux. The base operating system has been modified slightly to run in an unattended fashion.

Modifications made to the onboard computer:

- `/usr/bin/gnome_power_manager` has been renamed to `/usr/bin/gnome_power_manager.disabled`. This allows[1] for the computer to run while it is on batteries and the screen is down. Otherwise the Vaio goes into suspend mode and no process can execute. To read the battery life do
  `cat /proc/acpi/battery/BAT1/state` or `acpi -b`

- `/etc/init.d` The servants are configured to be managed by upstart. The config files determine which modules to load, when to restart and where to redirect the output (logs). See section 5.3.2 for detailed description.

- *restart_servants* It is possible[2] to restart the servants by pressing the eject button on the Vaio, even with the lid down. This button will generate an ACPI event that will be processed by `acpid` using the configuration file `restart_servants.conf` which in turn will call `restart_servants.sh` that will force the termination of the servants, which will be restarted by upstart. The configuration and script files go respectively under `/etc/acpi/events` and `/etc/acpi/actions`.

- The output of the commands are redirected to `/var/log/higgs/$PROGRAM.log` Any problem associated to the execution of the modules may be diagnosed inspecting the logs in `/var/log/higgs/`, where all the output from the programs is registered. These files may grow big, so it has been created a logrotate config file for them (`higgslog`) placed in `/etc/logrotate.d/`.

- MODULES: Removed pl2303 and ftdi_sio kernel modules. They do not load with the same order on each bootup, so they are manually loaded by the inits scripts as described in 5.2.3. The binaries have been moved from the original location to `/etc/higgs/modules`.

5.3 Common libraries and module considerations

The RCT testbed may be controlled remotely by means of procedure calls to CORBA objects. All of Higgs devices including the base platform have a

[1] This is not valid in Ubuntu RTAI as the ACPI subsystem is not functional. It is documented here in case other OS is used.

[2] This is not available when running the RTAI kernel.

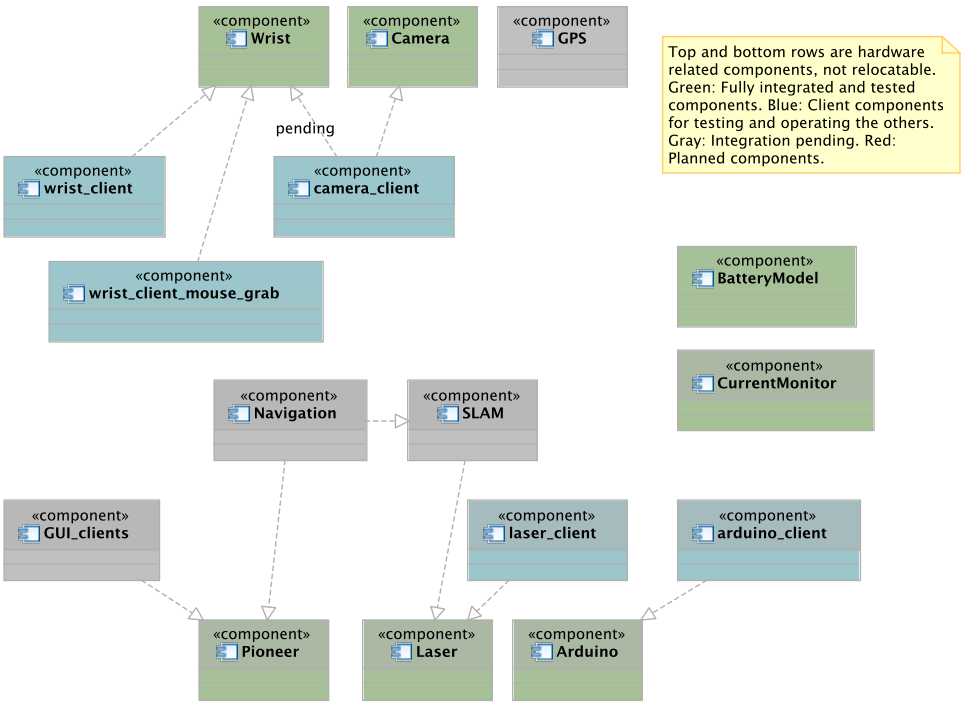


Figure 5.4: General view of the CORBA modules.

CORBA interface definition that must be used in order to remotely or automatically operate the robot. In this section these interfaces are described in detail.

There are additional CORBA objects running in the on board computers that whilst not having direct control on Higgs' physical devices, they do use the devices and provide useful, well-known and proved services.

The Java language requires that all CORBA interfaces are defined inside a module, so all Higgs interfaces are defined inside the module `higgs`.

5.3.1 ASLab servant utility functions for CORBA

The same file `code/lib/CORBA_utils.h` for easy creation of clients has also macros for servants. A typical servant executable would be something like:

```
#include "implementation.h"
#include "CosNamingC.h"
#include "../../lib/CORBA_utils.h"

int main(int argc, char* argv[]) {
    CORBA_BEGIN_SERVER(argc, argv);
```

```
    implementation_t impl();
    higgs::implementation_t_var implvar = impl._this();

    CORBA_REGISTER_REFERENCE(implvar, "IMPL");
    CORBA_END_SERVER;
    return 0;
}
```

It is possible to use also `CORBA_GET_REFERENCE` in case other objects are needed.

5.3.2 Module installation and configuration files

The CORBA macros for the servants make use of the file

```
/etc/higgs/listen_endpoint.ip
```

to configure the ip address where they should listen to, additionally to `nameservice.ip`. Typically will contain the IPv4 address of the computer it is running in, in this case,

```
138.100.76.246
```

Other config files are the device links at

```
/etc/higgs/devices
```

each module has this directory and the device file it uses hard coded.

Finally, they need an upstart config file. The upstart config file for the laser follows as example:

```
description "Upstart config file for the arduino servant"
author "Francisco J. Arjonilla Garcia"
start on started Naming_Service
respawn
script
sleep 5
date >> /var/log/higgs/laser.log
su -l -c /usr/local/bin/laser_server higgs\
            >> /var/log/higgs/laser.log 2>&1
end script
```

Other modules have different upstart config files. See each module source tree. The Naming Service should start automatically too after installing it. If not, an upstart config file must also be created for it. In either case, the endpoint must be specified in the parameters, i.e. manual execution:

```
./Naming_Service -ORBEndPoint\
    iiop://higgs2.disam.etsii.upm.es:9876
```

The JAVA servant does not use the automatic NameService resolution mechanism provided by the C++ macros in `CORBA_utils.h` and needs to have it specified as arguments when running it. Also, the next two lines must exist in the `.profile` file in the home directory of the user running the servant:

```
export PATH=/opt/jdk1.6.0_26/bin:${PATH}
export LD_LIBRARY_PATH=/usr/local/lib/
```

Replace paths apropriately.

5.4 I/O board

The I/O board manages the sensors and actuators that do not have a specific interface for connecting them to a computer. It has been implemented with an Arduino Mega board. The devices controlled by the I/O board are:

- Compass
- Accelerometers
- Battery sensors
- Laser pitch
- Power board

These devices are connected to the Arduino Mega commercial board through a custom made extension board. The connector layout is shown in Figure 5.5, and the connectors correspondence to the devices in Table 5.1. The ribbon cable connector is between the digital inputs. The position is marked on the board and pin 0 (ground) is next to Pin22.

5.4.1 Tilt mechanism for the laser

The laser sensor is housed inside a methacrylate structure with a joint that enables pitch movements on the laser. They are actuated by a servo placed just behind the laser, and by two end of stroke switches that limit the movements to safe values. To the left side of the laser, coaxial with the axis of rotation, there is a potentiometer that closes the loop for precise control of the rotation angle. The servo, switches and potentiometers are all controlled by the I/O board and has a PID programmed within its firmware. The details can be found in the PFC from Marcos Salom.

Currently the PID feature has been disabled and the servo operates in an open loop fashion, using the end of stroke switches as a reference for rotation limits during initialization. The factors that motivated this decision are:

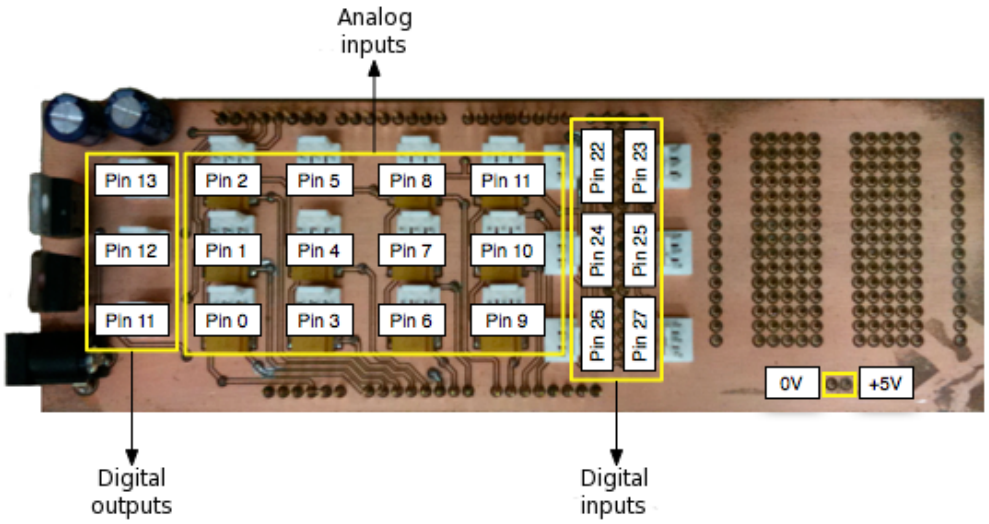


Figure 5.5: Connection diagram for compass board

Number	Device
Pin0	Not used
Pin1	Not used
Pin2	Not used
Pin3	Instrumentation current
Pin4	Instrumentation voltage
Pin5	Laser pitch potentiometer
Pin6	Motor current
Pin7	Motor voltage
Pin8	Accelerometer, X axis
Pin9	Not used
Pin10	Not used
Pin11(Analog)	Accelerometer, Y axis
Pin11(Digital)	Not used
Pin12	Not used
Pin13	Laser pitch servo
Pin22	Min pitch switch
Pin23	Max pitch switch
Pin24	Not used
Pin25	Compass
Pin26	Not used
Pin27	Not used
Ribbon	Power board

Table 5.1: Connector correspondence of the Arduino extension board with the devices.

1. The potentiometer is prone to errors due to mechanical hystheresis, worn out and temperature effects.

2. There is a lot of functionlity that can be more useful than controlling the pitch of the laser. The effort has been so displaced elsewhere.

3. A better pitch angle sensor, such as an optic encoder, would justify buying a whole controlled actuator with integrated sensing and electronics.

4. The kinect sensor can partially substitute the laser.

5.4.2 Power board

The I/O board and the power board have been connected with a ribbon cable that plugs to the doble row of pins of the extension board in the side of the I/O board. The ground connection has been stripped as the correspondent pin is not connected to ground in these pins. Common ground with the rest of the robot is achieved through the power connections that come from the batteries (Figure 5.2).

5.4.3 Compass

The compass is a solid state magnetism sensor placed next to the GPS antenna on the aluminum bridge. It was bought at www.superrobotica.com product reference CMPS03 S320160. The connections are as follows, from bottom to top as shown in figure 5.6.

1. VDD. To Arduino supply.

2. SCL. To 5V with 47Ω resistor.

3. SDA. To 5V with 47Ω resistor.

4. PWM. To Arduino digital input.

5. NC.

6. Calibrate. To button in compass connector.

7. 50Hz-60Hz. To GND.

8. NC.

9. GND. To Arduino supply.

All connections including the calibration button are done inside the IDT connector and protected by thermoadhesive. The button takes the calibration input to ground when pressed.

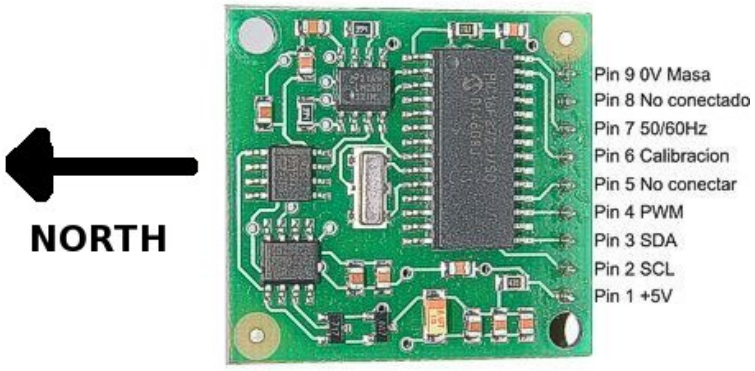


Figure 5.6: Connection diagram for compass board

Calibration instructions

As noted in the manual of the compass, to calibrate it you have to press the button with the board heading perfectly well once in each direction: North, West, South and East, no order required. It is already calibrated, so there is no need to do it again.

The compass gives an output pulse of 1ms to 37 ms VCC plus a fixed 65ms GND. 1ms corresponds to $0°$ and 37ms to $359°$.

The aluminum bridge does not interfere on the compass readings, but if the robot is moving near steel structures they may be perturbed.

5.4.4 Accelerometers

There are two accelerometers encased on the same electronic board attached horizontally with Velcro to the top cover of the robot base next to the I/O board. They give a standard analog 0V to 5V signal and are connected directly to the I/O board.

5.4.5 Intensity/Voltage sensor

There are two I/V sensors installed on board plus the battery status indicator in Vaio accessible through the command line. The two sensors can detect current and voltage of the Pioneer2AT batteries and the instrumentation batteries. One operational amplifier per battery has been used as a differential amplifier, as shown in Figure 5.7. This design allows to detect small variations in voltage of higher voltage than those of the working conditions of the operational amplifier, specifically lower than $3.5V$, with the LM2904P operational amplifier powered at $5V$. The voltage sensor is a scale down of the battery voltage, which is then divided by the scale factor by software to recover the true value. The gain is that of the differential amplifier. To read the true current value, both the gain and the sensing resistor value must be

considered. See Table 5.2 for the numerical values. The printed circuit board is designed to fit into connectors 3,4,6 and 7 of the Arduino extension board as an add-on module. The sensing resistors are placed one inside the Pioneer2AT glued to the chassis under the back sonars (figure 5.2) and the other one is in an aerial connection on the battery cable immediately before the power board.

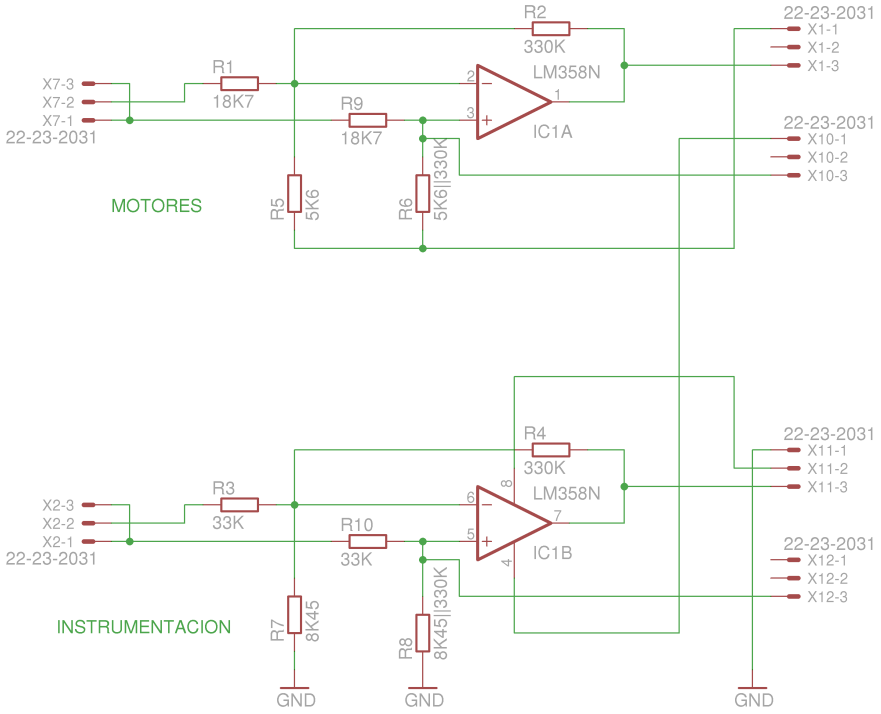


Figure 5.7: Battery sensor schematic

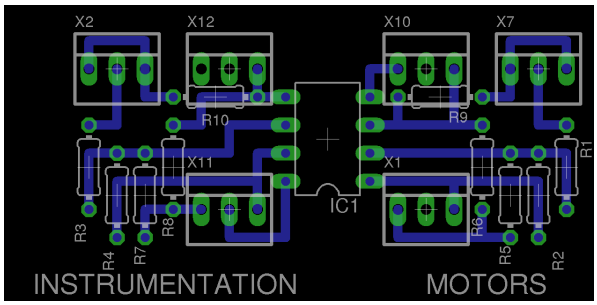


Figure 5.8: Battery sensor board layout

Supposing ideal components, the output voltage given by the current sensor is given by the formulae

$$U_0 = U^+ \frac{R_6 R_2}{R_9 + R_6}\left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_5}\right) - U^- \frac{R_2}{R_1} \tag{5.1}$$

The condition that must be met for both references to be scaled down by the

Parameter	Instrumentation	Motors
R1	$33k\Omega$	$18.7k\Omega$
R2	$330k\Omega$	$330k\Omega$
R5	$8.45k\Omega$	$5.6k\Omega$
R6	$8.45k\Omega//330k\Omega$	$5.6k\Omega//330k\Omega$
R9	$33k\Omega$	$18.7k\Omega$
Max sensing current	6A	17A
Max differential input	0.3V	0.17V
Gain	10	17.647
Voltage scale factor	0.1998	0.2275
Sensing resistor	0.05Ω	0.01Ω

Table 5.2: Characteristics of the differential amplifiers for the battery sensors.

same coefficient is that the relation of the resistor values between the positive input and the negative input is

$$\frac{R_9}{R_6} = \frac{R_1}{R_2 \parallel R_5} \tag{5.2}$$

This way,

$$U_0 = \frac{R_2}{R_1}(U^+ - U^-)$$

From this result we can observe that there is great sensibility in the operation of the differential amplifiers. The resistors from the negative input and the positive input must be of the same value, of the same brand and of the same batch to give enough precision: $R_9 = R1$ and R_6 physically being two resistors in parallel: $R_6 = R_2 \parallel R_5$.

The current sensors will not work correctly while the batteries are charging.

5.4.6 Servant and firmware

The CORBA module has been written in the JAVA language whilst the test client in C++. When the TurnOn and TurnOff methods are called, the appropriate device is turned on/off and simultaneously, for the gps, laser, wrist and camera, the servant that controls the device is killed from the arduino servant This only works if the device servant is installed and running on the same machine as the arduino servant, and will solve some issues on the protocol management that some of the servants have with their device. The servant will then be restarted by the vaio tools utility scripts.

Both the embedded program inside the I/O board and the Java servant communicate through a USB data connection that gets converted to serial RS-232 by a FTDI chip in the Arduino board. The I/O board starts the communication protocol by sending all the parameters and sensor readings to the servant, and then the servant optionally answers with the order.

The figures 5.9, 5.10, 5.11 and 5.12 show the UML model of the JAVA sources of the Arduino CORBA servant.

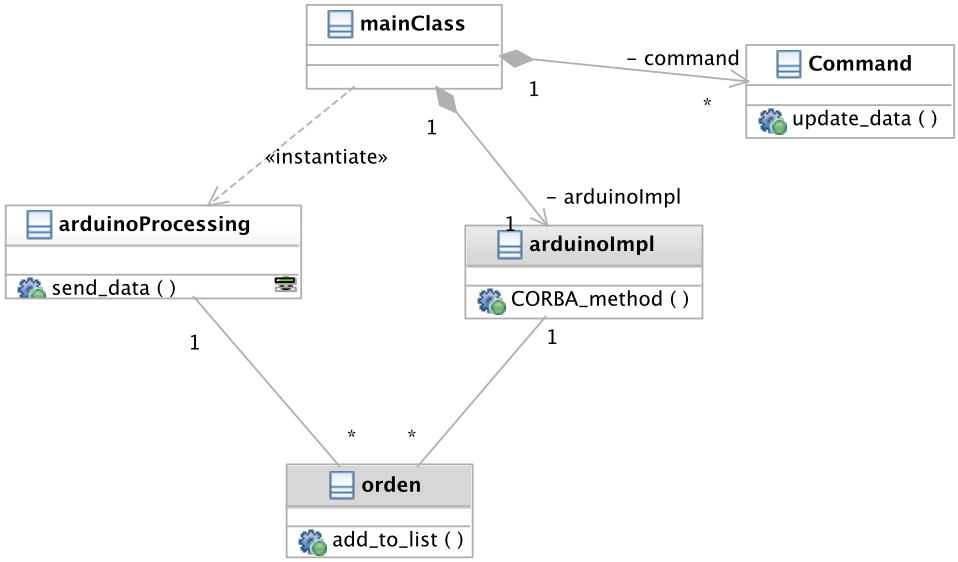


Figure 5.9: CORBA interface and classes used in the JAVA implementation.

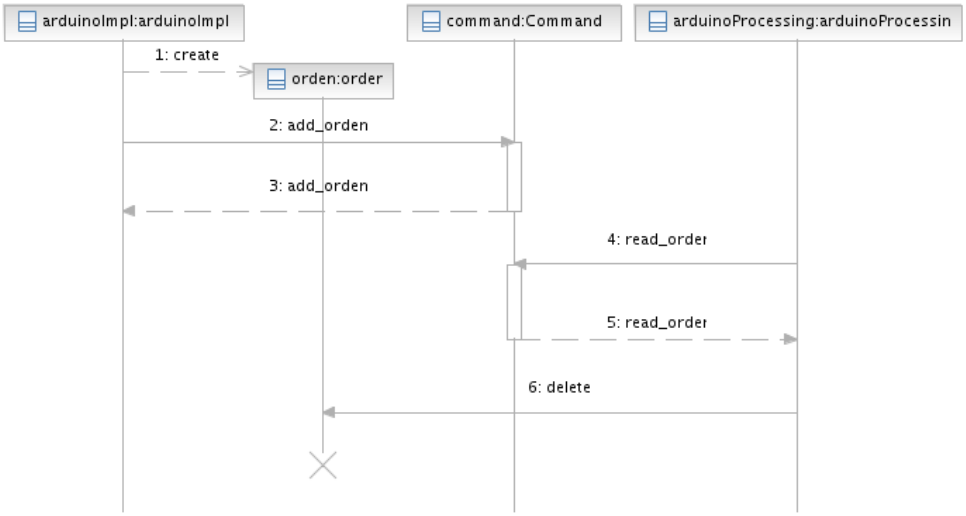


Figure 5.10: Interaction diagram for the Arduino CORBA module.

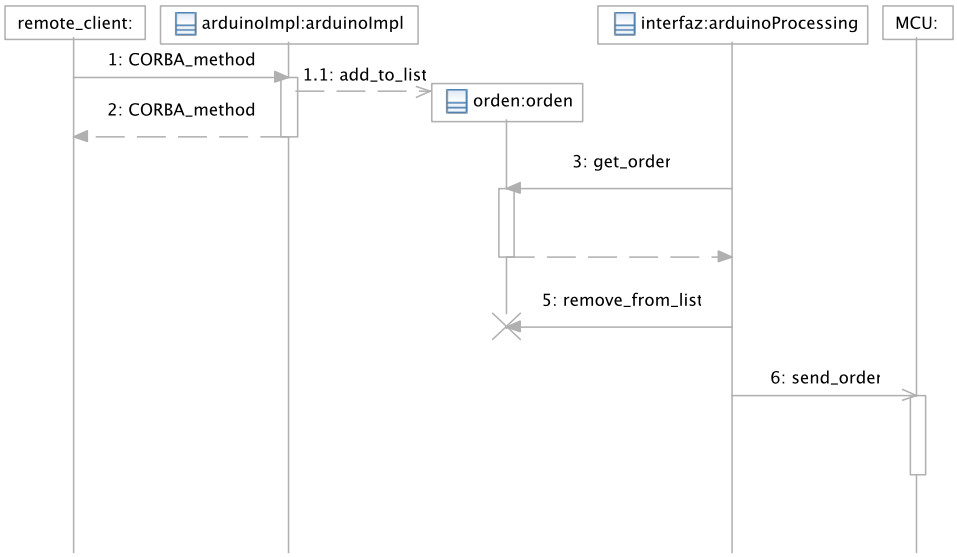


Figure 5.11: Sequence diagram for sending data to the i/o board.

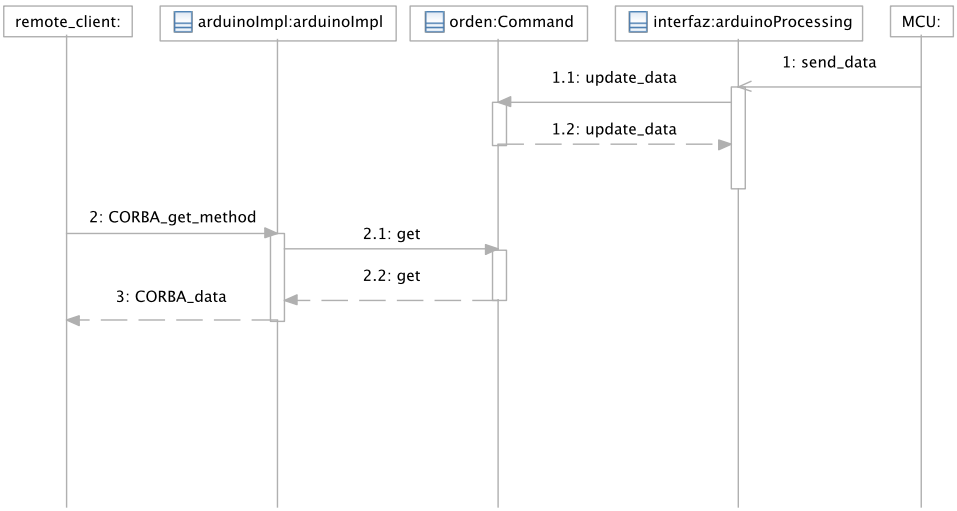


Figure 5.12: Sequence diagram for receiving data from the i/o board.

Modifying and uploading the embedded C code

The embedded program inside the Arduino has been developed using the official IDE based on the *processing* IDE and the libraries associated to it.

There are a few issues for remembering when installing this module on a fresh linux installation. The serial library RXTXcomm is included in the subversion directory
`svn+ssh://sagan/home/svnroot/Higgs/code/devices/arduino/lib`
The file librxtxSerial.so should be copied to
`/usr/$JAVA_BASE/jre/lib/i386` and the three jar files `core.jar RXTXcomm.jar` and `serial.jar` to
`/usr/$JAVA_BASE/jre/lib/ext`. Then make sure the user running the servant is in the groups `uucp`, `dialout` and `lock`, and that the package uucp is installed. Ensure the cross compilation environment for avr is installed. In Debian/Ubuntu these are the packages `gcc-avr`, `avr-libc` and `binutils-avr`.

Now check for the embedded C source in the subversion directory

```
svn+ssh://sagan/home/svnroot/Higgs/code/\
devices/arduino/arduino\_embedded
```

A copy of the Integrated Development Environment for the arduino is in

```
svn+ssh://sagan/home/svnroot/Higgs/code/devices/\
arduino/arduino-IDE
```

or you can get the latest version from the web at
`http://arduino.cc/en/Main/Software`. Start the IDE with `./arduino` and open the C source file `arduino_embedded.pde`. For the IDE to compile correctly, the name of the source file without the `.pde` extension must have the same name as the directory it is in. Select the correct Board and Serial Port [3] under the *Tools* menu, then Compile/Verify and Upload.

5.5 Power board

The Power Board is a custom printed circuit board designed specifically for the needs of the investigators at ASLab with which to test the algorithms in system auto-reprogramming with partial malfunction of a robot. The Power Board is in control of the power supply of up to nine devices in the robot. Each of these channels may be manually shut down by means of a switch or remotely/automatically using the ribbon cable connection to the Arduino board. There is a LED power indicator for each channel plus and a switch and LED indicator for all the board. Three of the channels, the ones nearer to the

[3] The MEGA Arduino board has an integrated USB to RS232 chip, so it will appear as a serial port `devttyUSBx` when connecting the USB cable.

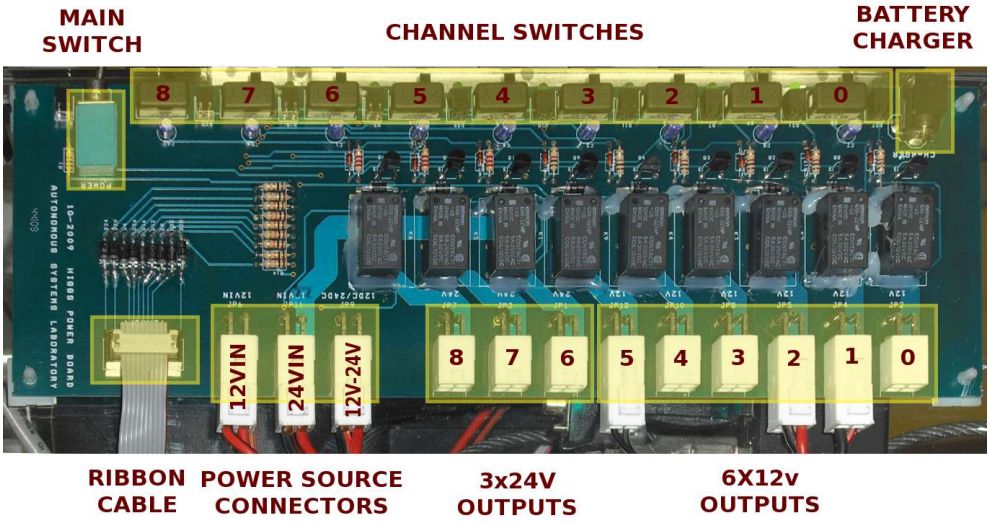


Figure 5.13: Photograph of the Power Board indicating each part.

general switch, can control 24V devices whilst the other 6 are for 12V devices. On the back side there are 12 connectors, 6x12V relay controlled outputs, 3x24V relay controlled outputs, 12V input, 24V input and a 12V output to be taken to the 12VDC to 24VDC converter that will be powered on whenever any 24V output is enabled. Additionally, there is a power jack connector for the battery charger[4]. Thus, the Power Board needs a 12V power source and if the 24V channels are used, it also needs a 12VDC to 24VDC converter.

5.5.1 Power board pinouts

Check figure 5.13 for a visual description of the connectors. The connector for the ribbon cable has the pinout indicated by table 5.3.

The connectors for the devices are Molex MiniFit, RS references 670-5717 (PCB male), 679-5776 (female), 172-9134 (terminals).

5.5.2 Electric interface

The ribbon cable connector has 10 pins. Starting from pin 0, these are ground, the six 12V channels and the three 24V channels. The channels are shut down writing a logical 0 (0V) to the corresponding pin, whilst a 1 (5V), a high impedance or a no connection will allow for the channel to be powered on. Both manual and remote switches must allow for the channel to be powered on for having that channel powered.

[4]Originally there was a lead battery pack for powering the devices on top of the robot base. This power jack connector is no longer neccessary as the robot base has its own power jack connector for charging the batteries.

Channel / Ribbon cable pin	Voltage	Device
0	—	Ground common
1	12V	Not used
2	12V	Arduino Extension Board (sensors)
3	12V	Servo
4	12V	Kinect
6	12V	GPS receiver
7	12V	Binocular Camera
8	24V	Laser
5	24V	Not used
9	24V	Wrist

Table 5.3: Distribution of devices in the channels.

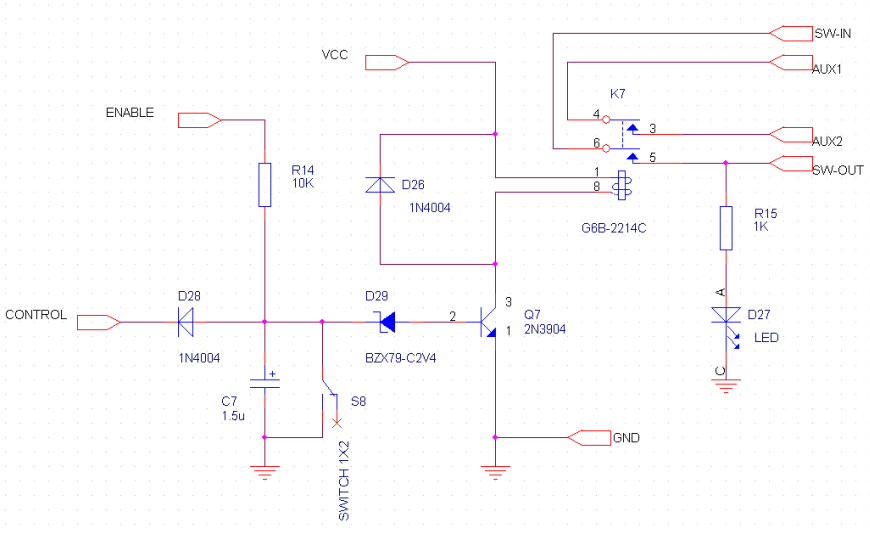


Figure 5.14: Schematic for each of the channels.

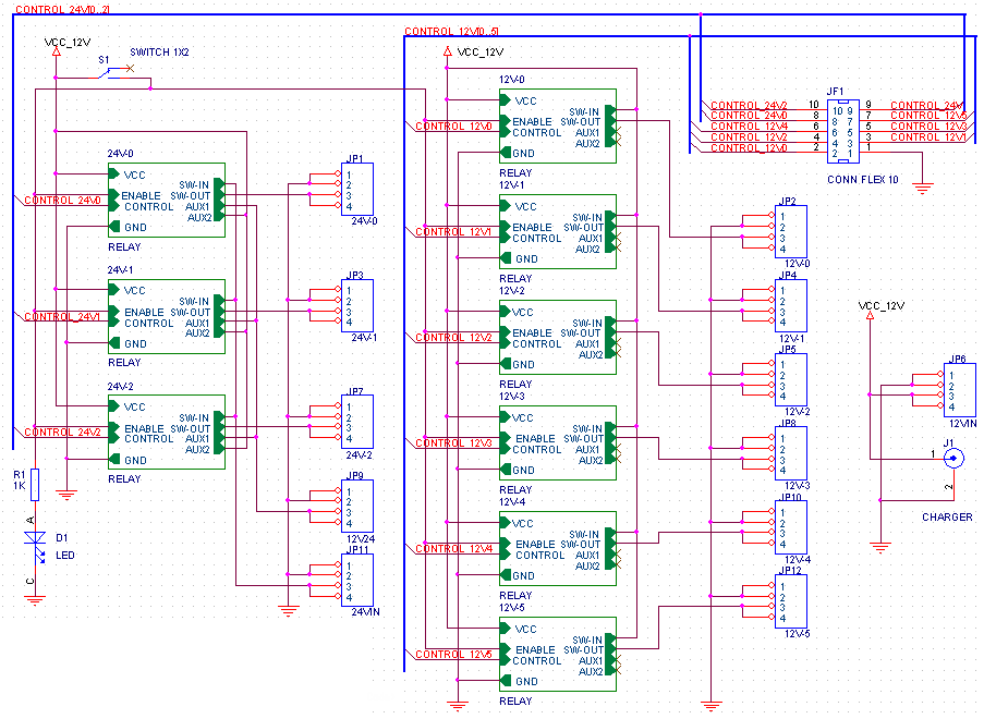


Figure 5.15: Schematic for the Power Board.

5.5.3 Power Board Bugs

During the design phase of the board there were some errors that passed the internal tests. As the board was to be manufactured once, it was not economically feasible to built it again and had to be repaired. These bugs should be revised before sending the design in case the power board should be manufactured again.

- The holes for the diodes 1N4004 are too small. Can be repaired by drilling bigger holes and solding the diodes on both sides.
- The power jack has the positive pin disconnected. Solved with a bit of tin covering both the connected pin hole and the positive pin hole.
- The silkscreen of the connector for the 24V input is wrong, it has a duplicated 12VIN instead of 24VIN. Solved with a marker-pen.
- The relay hole distribution has the two rows of pins too far apart. Moreover, the coils had positive and negative pins and was not well documented in the datasheet, so the coil pins are swapped. Solved by manually separating the pins of the relays and extending the coil pins and securing the relays with termoadhesive. A second better approach would have been to solder all components on the other side of the board.
- Sometimes the relays do not activate correctly and/or the external radio emitter for the DGPS interfere with them and turns them off. Even

though the calculations have been based on the components' specification, $R14$ and analogous have been reduced to $5K4\Omega$ for ensuring that enough current passes through the relay's coil.

5.6 Wrist

The wrist [5] is a two axis robotics kit module with pan and tilt movements manufactured by Schunk®. It was originally bought for use as the tilt mechanism for the laser, but as the center of gravity of the laser does not match the center of rotation of either axis of the wrist, it would be a very power hungry method for tilting the laser, given that the laser is heavy and the power comes from a portable battery system. Moreover, one of the axis from the wrist would be unused. It was finally decided to use the wrist for controlling the motion of the camera, even thought it could be achieved with a smaller controller with minor power requirements.

The manufacturer gives several interfaces for controlling the wrist: Profibus, CAN and serial. The serial RS-232 bus was chosen over the others because of the simplicity, the availability of drivers and the sufficient fulfillment of our requirements. It is connected to the on board computer via a custom made cable with an intermediate RS-2323 to USB converter. The wrist endpoint has industry standard serial closings.

5.6.1 Wrist servant

All source code for controlling the wrist is located under

```
\$(SVN\_ROOT)/Higgs/branches/CORBA/code/devices/wrist
```

The programs and utilities found there include:

- Low level library with direct access to serial port.
- Servant code for the CORBA object.
- Simple CORBA client to test the functionality and status.
- Graphical CORBA client for manually teleoperating the wrist with the mouse.
- Compressed file with the obsolete source code for the wrist, starting point but fully rewritten code for the current library.
- PDF file from the manufacturer describing the serial protocol to the wrist.

The sources are all under the `/src` directory and its documentation can be found inside the source files.

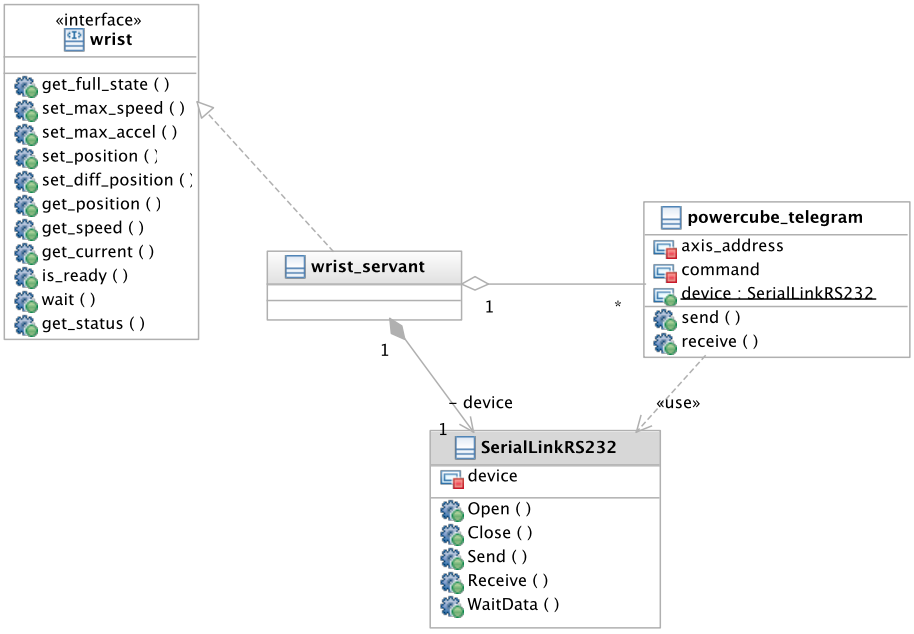


Figure 5.16: Class diagram for wrist module and CORBA interface

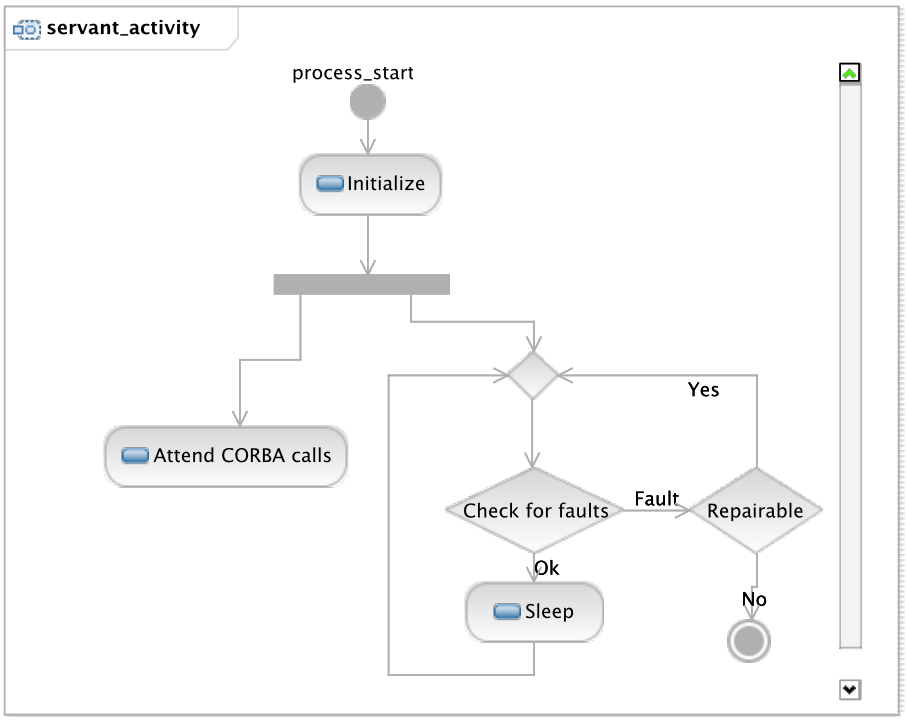


Figure 5.17: Activity diagram for wrist servant

When a CORBA client calls a method of this servant a new telegram is created with the parameters and format adequate to the call, passing the reference to the serial device already initialized. This telegram sends a message to the serial device and waits for the acknowledgement. This is valid for both directions of data flow. The telegram gets destroyed once the communication ends.

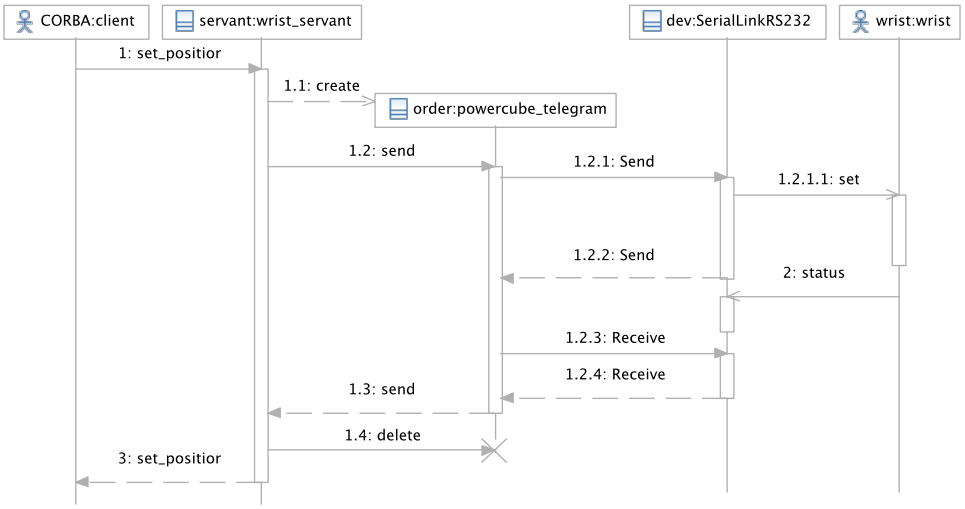


Figure 5.18: Sequence diagram for transmitting data to the wrist.

5.6.2 Error recovery

The wrist must be powered, as said in the official documentation, by a 24V power source, mobile or not. However, the fluctuations in voltage caused by battery charge and the instant consumption of the devices make it difficult to keep the voltage of the battery near this value. Because of this fact, the wrist controller has been designed for high error tolerance. A running thread in the servant polls periodically the wrist for error codes and takes the appropriate actions when a failure is detected, as explained in each considered fault. These are the possible faults that have been taken into consideration:

Battery voltage out of bounds

It has been detected empirically that the wrist will not function if the battery voltage is under 22V or if it is over 27V, so it will not work if the battery is either fully charged or next to empty. However, the control electronics are still available. The status command may be issued to detect this anomaly and a special status code will be sent by the wrist. The CORBA servant code is

[5]Note that even though the manufacturer calls this device a powercube, internally we call it wrist.

aware of this fault and will abort execution to allow to complete device reset, outputting a log and leaving the restart of the servant to the VAIO utility programs. This fault can be prevented by periodically reading the voltage sensor value from the I/O board, the Arduino CORBA servant.

Overcurrent

The internal current sensor will stop the axis when an overcurrent is detected or a low voltage condition is met if the 12V to 24V converter can not supply enough power. In this case, a reset and homing procedure is needed before continuing operating the unit. The maximum speeds and accelerations before the fault are saved and restored after the reset and homing procedure, but not the position.

Device not found

This error may arise if the device is powered off, if the serial port is not accessible or can not be found, or if the serial cable is not correctly connected. In this case the servant will die and get restarted automatically by the init scripts.

5.7 Laser

The data cable is a serial one prepared for RS-232 and RS-422 communications. There is a jumper on the end connector to select which of the protocols to use. With the jumper, RS-422. Without the jumper, RS-232. Note that the RS-422 has not been successfully tested, maybe because the jumper should be inside the laser connection black box instead of the serial cable endpoint.

5.7.1 Laser servant

The laser servant has been developed on top of the driver given by the people working in the mobile robotics lab. It has been modified to be able to resynchronize after a transmission error or a reboot of either the sensor or the servant. The original source code can be found in

```
$(SVN)/code/devices/laser/libreria_laser-paloma.zip
```

The documentation for the Sick Laser Sensor both hardwre and protocol definition can be found under the `doc` directory of the laser sources. Inside the `src` directory you will find the modified sources for the laser driver, the servant implementation and a test client, like in other modules.

5.8 Differential GPS

The GPS module is based on two OEMV-2-RT2 receptors with capacity for Satellite Based Augmentation System (SBAS) and Differential GPS (DGPS). Both receptors communicate through a radio based modem capable of transmitting in half-duplex mode through relatively long distances at 9600bps. The base radio has a transmitting power of 2W and the rover radio 0.5W, however the latter is only used for receiving data. Both radios must be set to work in the same channel, use the button with the channel label to change it. The electronics for each receptor have been encapsulated into plastic boxes with these connectors and indicators available:

- One double power LED indicator for checking 12V and 3.3V.
- One USB.
- Three RS-232 ports COM1 to COM3 with male DB9 connectors.
- One DB9 connector labelled CEXT.
- Power Jack connector, 12V nominal.
- Two TNC connectors.
- A reset button.

The connectors used on both base station and rover are the same. The antenna, either the small one on the rover or the bigger circular one on the base station is connected to the RFIN-labelled TNC connector. The other TNC connector is not used and its purpose is to have an external high precission oscillator for the GPS time measures. The power jack is used to power the unit. COM2 is connected to the radio emitter/receptor for sending/receiving the differential corrections from the base station to the rover. COM1 is connected to the VAIO laptop through a USB to RS232 adaptor so the GPS receiver can send the position information to the CORBA servant. On the base station, COM1 is only used when configuring the parameters of the receiver and saving them to its internal non-volatile memory.

5.8.1 Configuring the devices

These procedures are not to be used while on normal operation of the robot. They are only needed on first boot, then the configuration is saved permanently on the non-volatile memory of each device. There is a little booklet inside the yellow bag where the station base is stored with the commands and parameters used by the stations.

The `minicom` command line application is best used. You may have to configure it. Open `/etc/minirc.ttyS1` and insert these lines:

```
 # Machine-generated file - use "minicom -s" to change parameters.
pu port             /dev/ttyS1
pu baudrate         9600
pu bits             8
pu parity           N
pu stopbits         1
pu rtscts           No
```

Change the port as needed. Run

```
minicom ttyS1
```

Now you should have access to a command line where you can send commands and check the status of the device.

Base station

Set up the base station with the RF input to the external antenna situated in the roof of the department of automatics. Get a serial cable and plug your personal computer to COM1. Start the `minicom` program to start a new session with the GPS device. These are commands were used to configure it:

```
FRESET
FIX POSITION 40.4397076 -3.6881482 744
INTERFACEMODE COM2 NONE CMR OFF
LOG COM2 CMROBS ONTIME 1
LOG COM2 CMRREF ONTIME 10
LOG COM2 CMRDESC ONTIME 10 1
SAVECONFIG
```

The meaning of the commands is as follows:

```
FRESET
```

Clears the non-volatile memory and sets the configuration to the default values. `RESET` does the same thing without clearing the non-volatile memory.

```
FIX POSITION 40.4397076 -3.6881482 744
```

This is the position measured for the base station after a period of several hours (Latitude, Longitude, Height). The command indicates the base station that it should calculate the GPS position for the corrections. All position reading after this command returns the fixed coordinates given.

```
INTERFACEMODE COM2 NONE CMR OFF
LOG COM2 CMROBS ONTIME 1
LOG COM2 CMRREF ONTIME 10
LOG COM2 CMRDESC ONTIME 10 1
```

Sets the COM2 port for sending the corrections using the message types `CMROBS`, `CMRREF` and `CMRDESC`. See the GPS manual for more information. `ONTIME 1` means to repeat the command every second. Use the `UNLOG log_name` and `UNLOGALL` commands to stop transmitting automatic logs. Omit `COM2` for sending the data your terminal.

`SAVECONFIG`

Stores the configuration to the non-volatile memory.

Rover station

Proceed as for the base station, except that the serial connection may be already available from the on board computer. In this case you may need to stop the GPS servant first to gain control of the serial device port. These commands were used to configure it:

```
FRESET
SBASCONTROL ENABLE EGNOS 0 ZEROTOTWO
INTERFACEMODE COM2 CMR NONE OFF
SAVECONFIG
```

The `SBASCONTROL` command line configures the GPS mobile station to use the EGNOS/SBAS satellites for better precision than GPS alone if the differential readings are not available.

5.8.2 Servant

The GPS module uses the convention for other module sources: installation and execution procedure, code placement inside the subversion repository and CORBA utilities such as in other modules are used.

The servant code uses non standard commands to receive the data from the rover station. The commands

```
LOG BESTPOS
LOG BESTVEL
```

are sent on startup and used to retrieve the information about satellites, position, standard deviation and speed. This data is fetched by an independent thread to the CORBA one and stores it into a shared variable for the CORBA thread to read it each time a client asks for the data. There is also a timeout by which if no data is received after a fixed amount of time an error is issued.

5.8.3 Radio signals and ETSII-UPM

The ETSII-UPM is surrounded by many governmental sites. It is possible that radio transmissions with the differential corrections fail because from

interference from these sites. Other investigation groups have had the same problem. In this case the dGPS will not work. Official service has anyway recommended us to use this configuration in the rover if the problem persists:

```
UNDULATION USER 0.0
FIX NONE
COM COM2 9600 N 8 1 N OFF
LOG COM2 GPGGA ONTIME logperiod
INTERFACEMODE COM2 CMR NOVATEL OFF
RTKSOURCE type any (ANY)
```

5.9 Binocular camera

The binocular camera is attached to the top of the powercube, allowing for pan and tilt movements.

The image data is transmitted in raw yuv format between the CORBA servant and the clients.

When restarting the servants, you should kill the process and call `dc1394_reset_bus`, or else the servant will not succeed on starting again.

5.9.1 Compiling and running the CORBA module

For installing the CORBA module on a fresh fedora linux system, you must install these libraries which can be found in the repositories provided by the package `atrpms`: `glut glui libavcodec` and related. Inside the directory tree of the sources there are three scripts that help run the servant and test clients. Compile and run the server with

```
\$ ./1-server.sh
```

The test client with

```
\$ ./2-client.sh
```

or, for the old client,

```
\$ ./3-client.sh
```

The client runs best when configured to run with Xlib instead of OpenGL. The user that runs the servant, higgs when running unattended, needs to be in the video group.

The source structure has not been adapted to the new tree configuration nor cmake. `idl` Contains the idl interfaces. `generated` is where the skeleton and stub source files are generated to. `obj` the binary files are compiled to this directory. `text` contains code to test the camera without using CORBA.

5.10 BatteryModel and CurrentMonitor

Note: Porting to the newer SVN repository is pending.

These two CORBA servants have been implemented into one JAVA binary and made an unique module inside the VAIO on board computer. The BatteryModel is a component that calculates the parameters related to the charge in a battery and estimates the remaining life based on the power requirements. The CurrentMonitor is not used in any other module but has been an utility component for developing the BatteryModel. It periodically polls the current of the (currently instrumentation) batteries and reports the mean consumption since it was required to start monitoring.

Neither component reads or uses the peripherals on the computer. BatteryModel is exclusively a CORBA servant and the clients must pass the data to it in order to compute the estimations for the battery life, whilst CurrentMonitor is, besides a servant, a client to the Arduino CORBA servant for reading the current values of the batteries.

Installation of the module and automatic setup is achieved as in the other modules.

Apéndice E

Herramientas

En este capítulo se describen las herramientas software más utilizadas a lo largo del proyecto.

Descripción de la tecnología CORBA

CORBA (Common Object Reques Borker Arquitecture) es un estándar elaborado por OMG (Object Management Group) que define una plataforma de desarrollo para sistemas distribuidos que se basa en llamadas a métodos bajo un paradigma orientado a objetos. CORBA define protocolos de interoperabilidad, unas interfaces para los lenguajes de programación más comúnmente utilizados y unas librerías que ofrecen servicios adicionales.

CORBA está orientado hacia sistemas distribuidos heterogéneos, es decir, aquellos que están compuestos por distintos tipos de CPUs con distintos sistemas operativos y unidos mediante una red. Las arquitecturas, los sistemas operativos y los lenguajes de programación están limitado únicamente por las implementaciones disponibles que los soporte. Existen implementaciones para Ada, C, C++, C#, Smalltalk, Java, Python, Perl y Tcl, entre otros.

Todo ello hace de CORBA el *middleware* más flexible existente hoy en día y es utilizado por una infinidad de empresas, especialmente compañías de telecomunicaciones.

El mecanismo de funcionamiento de CORBA se basa en el empaquetamiento de las llamadas a funciones en cadenas de bytes que se envían codificadas de forma estándar y de ese modo el objeto remoto no necesita conocer más que el protocolo de comunicación. Para facilitar la labor del programador las herramientas de CORBA proporcionan la generación de código para ser usado tanto por los clientes, que son los usuarios de los servicios de los objetos CORBA, como los servidores, que implementan la funcionalidad del objeto CORBA. Estos servidores pueden estar en el mismo espacio de memoria que el cliente, en la misma máquina o en una máquina remota. La localización del servidor es transparente al cliente.

El Lenguaje de Definición de Interfaces (IDL) es un archivo de texto de sintaxis similar a C++ que actúa como pegamento que une los clientes con los servidores; con ellos se define la interfaz con la que se comunican y son los archivos a partir del cual se genera el código auxiliar en cualquier lenguaje, es decir, que es independiente de la máquina en la que se vaya a ejecutar el código y del lenguaje de programación.

Como ejemplo de uso, se describirá la generación de código en C++. La mayor parte de los lenguajes se aproximan a este método. Otros, como C, en los que no existe herencia de clases, se define la implementación de manera que facilite al máximo la labor del programador.

Figura E.1: Comunicación entre objetos de servicios CORBA.

Un cliente hará las llamadas a objetos remotos como si se tratara de un objeto residente en su espacio de memoria. La diferencia es que este objeto no es el auténtico sino un intermediario (proxy) que empaqueta los argumentos y los envía al objeto real en la máquina remota junto a información de la dirección donde debe enviar el valor de las variables de retorno y la función a la que se ha llamado. Este objeto intermediario se denomina *stub* y su código es el que se genera a partir de la IDL con las herramientas de la implementación CORBA elegida.

Los servidores por el contrario hacen uso de un mecanismo recíproco. El código generado resulta en una clase abstracta cuyos métodos virtuales son los definidos en la IDL. El broker de CORBA instanciará un objeto derivado de esta clase denominado *skeleton* y realizará las llamadas oportunas solicitadas por clientes, gestionará las llamadas simultáneas y decodificará los mensajes direccionándolos al objeto adecuado. La labor del programador se resume en heredar el *skeleton* y definir todos los métodos virtuales. Esta clase se denomina *servant* y es básicamente el software que se necesita desarrollar en este proyecto.

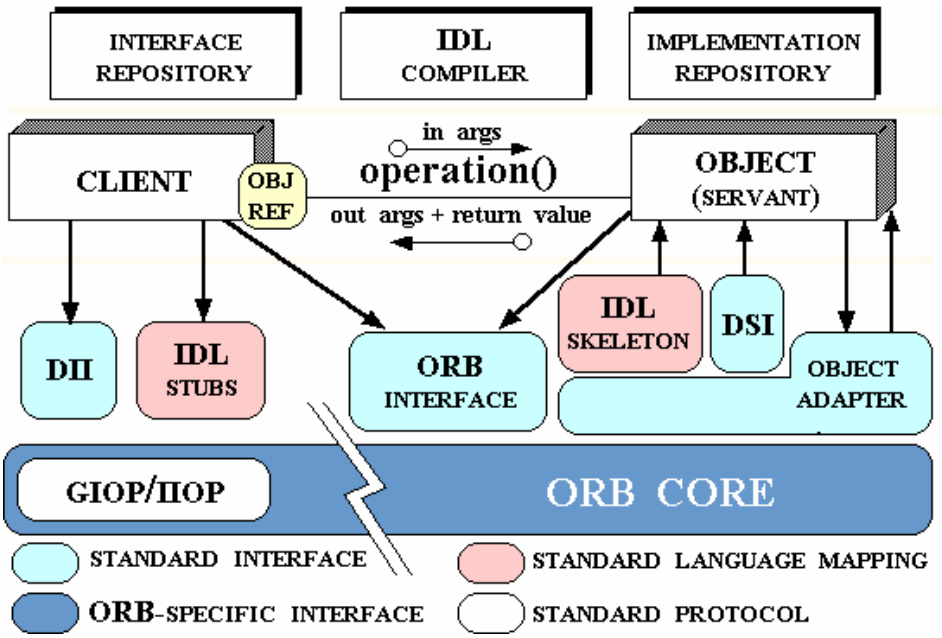


Figura E.2: Esquema de la arquitectura CORBA.

El concepto "Compilación de una interfaz IDL" hace referencia a la generación del código *stub* del cliente y *skeleton* del servidor.

CORBA es más que una definición de interoperabilidad entre objetos remotos. También define un conjunto de servicios que típicamente son usados en los entornos distribuidos. Los principales son el servicio de nombres, eventos, seguridad, etc.

Las últimas revisiones de CORBA deinen también una especificación para la tecnología de gestión de objetos distribuidos (DOM). Proporciona una interfaz de alto nivel para gestionar los servants, su ubicación y los métodos de comunicación. Facilita la labor de los analistas al permitir el diseño por componentes.

Dos buenas referencias para aprender a manejar esta tecnología y como referencia están en [6] y en [8].

Subversion

Figura E.3: Logo de subversion

Subversion, o más brevemente SVN, es un sistema de gestión de versiones ampliamente utilizado en el mundo del software libre. Llamado a ser el sucesor de CVS (Concurrent Verrsioning System), trabaja sobre una base de datos de versiones de los archivos fuente, aunque también trabaja con otros tipos de archivos como texto, imágenes, binarios, etc.

Una de las características más destacables es la posibilidad de trabajar con los mismos archivos por ditintos usuarios simultáneamente, actualizando automáticamente los archivos cada vez que se envía una nueva versión al repositorio. Se puede visualizar el historial de cambios de cada archivo y comprobar qué archivos se han modificado localmente, entre otras muchas características interesantes.

Se ha usado como almacenamiento de las distintas versiones del código fuente, como repositorio de copias de seguridad y como herramienta de intercambio de código fuente.

Doxygen

Doxygen es una herramienta de generación de documentación a partir del código fuente de un proyecto.

Figura E.4: Logo de Doxygen

Los comentarios del código fuente han de formatearse bajo una norma específica, aunque sencilla de seguir e intuitiva, para que sean capturados por el analizador de Doxygen. Además lee los nombres de las funciones y de otros elementos sintácticos para generar diagramas.

Un comentario típico se muestra a continuación (Se puede ver el resultado final en el anexo B):

```
1  /**
2   * \param ex The executive that will keep track of the value
        associated with this attribute.
3   * \param obj The object to which the get call applies.
4   * \param get Pointer to member method that supplies the attribute
        by polling it.
5   *   It must be of the form
6   *   <code> attribute_t get_t::get_method(); </code> <br>
7   *   Retrieve the pointer to member method with <code> &get_t::
        get_method </code>.
8   */
9  template <class O>
10 executive_attribute(executive & ex, O * obj, T (O::*get)()const)
11   {
12     ...
13   }
```

RSA

Entorno de desarrollo basado en Eclipse para el análisis y el diseño UML. Utilizado en la elaboración de los diagramas UML y para alguna funcionalidad típica de entornos de desarrollo, como visualizar el árbol SVN sin exportar el código.

VIM

Vi IMproved. Editor de textos basado en comandos que agiliza la programación. Ofrece una interfaz muy flexible pero con una curva de aprendizaje alta. Recomendado para todos aquellos técnicos que vayan a programar asiduamente.

Figura E.5: Logo de VIM

LaTeX

LaTeXes un sistema de documentos similar en función a Microsoft Word u OpenOffice. La diferencia con éstos es que LaTeXes un sistema no gráfico que se centra en la estructura, así el usuario puede centrarse en el contenido y dejar que el compilador LaTeXse ocupe de la maquetación, la generación de índices y la estética.

Toda la documentación del robot, al igual que este proyecto, han sido redactados en LaTeX.

OrCAD

OrCAD es una aplicación para Windows de diseño CAD de circuitos electrónicos. Puede exportar los proyectos al formato usado en las fábricas de circuitos impresos. Para más información consultar la web del fabricante `http://www.cadence.com`.

CÁTEDRA DE PROYECTOS	PROYECTO FIN DE CARRERA ANEXO III	ETS de Ing. Industriales UPM

TÍTULO DEL PROYECTO:
Desarrollo e implantación de plataforma robótica móvil en entorno distribuido.

ENTIDAD PROPONENTE:
Departamento de Automática,
Ingeniería Electrónica e Informática Industrial

Nº PROYECTO:

TUTOR ASIGNADO:
Ricardo Sanz Bravo

FECHA de COMIENZO:
1 de octubre de 2009

NOMBRE del ALUMNO:
Francisco Jesús Arjonilla García

Nº de MATRICULA:
00031

ESPECIALIDAD E INTENSIFICACIÓN:
Automática y electrónica

DESCRIPCIÓN DEL PROYECTO Y SUS OBJETIVOS PRINCIPALES

El proyecto consiste en el desarrollo de una plataforma robótica móvil para experimentación en robótica e inteligencia artificial. Para ello se desarrollarán, sobre la base de un robot móvil Pioneer 2AT8 los componentes físicos, la electrónica y el software necesarios para integrar de manera robusta y fiable, entre otros dispositivos, un escáner láser, una cámara binocular direccionable, un receptor GPSD y sensores propioceptivos. El proyecto se dividirá en dos bloques:

Plataforma física:
- Diseño de la plataforma para la integración física de todos los elementos.
- Selección e implantación de componentes y sensores que aumenten las capacidades del robot.
- Diseño del sistema de control de alimentaciones, con sensado del nivel de baterías.
- Integración de los canales de señales de los equipos.

Diseño del software de la plataforma:
- Despliegue de la tecnología CORBA.
- Integración del control remoto de los equipos con el desarrollo de módulos CORBA.
- Gestión de errores y tolerancia a fallos y desconexiones de los dispositivos.
- Herramientas de desarrollo y despliegue.
- Librerías que faciliten la explotación de la plataforma, junto con plantillas, documentación, etc.

OBSERVACIONES:

ENTIDAD PROPONENTE:

Nombre**:**

POR LA CÁTEDRA DE PROYECTOS

Nombre:

EL TUTOR:

Nombre: **Ricardo Sanz Bravo**

EL ALUMNO:

Nombre: **Fco. Jesús Arjonilla García**