Sistemas Informáticos
Curso 2007/2008

Sistema de navegación para el robot del Museo García Santesmases. Utilización de una Webcam para posicionamiento. "PosCam"

Pablo Iglesias Marcos.
Pablo Martos Rodríguez.
Felipe M. Polo Ruiz.

Dirigido por:
Dtor. José A. López Orozco.
Dpto. Arquitectura de Computadores y Automática.

Facultad de Informática
Universidad Complutense de Madrid

Agradecimientos

En primer lugar, queremos agradecer a algunos antiguos alumnos como Jorge Sánchez y José Manuel Maldonado su ayuda y colaboración en momentos puntuales de nuestro proyecto. Muy especialmente agradecemos, además, la valiosa ayuda que nos prestaron tanto Carlos León como Pablo Lanillos en los momentos más críticos.

Gracias también a todos los trabajadores del Laboratorio de Arquitectura de Computadores y Automática por prestarnos sus instalaciones y por las molestias que seguro les hemos causado.

Queremos agradecer a los profesores de la Facultad de Informática José Jaime Ruz y Gonzalo Pajares sus consejos e ideas sobre visión artificial al principio de nuestro trabajo.

Gracias también a Google por hacernos la vida más fácil.

Por último, gracias a nuestro director de proyecto, profesor Dr. D. José Antonio López Orozco, por su paciencia, sus consejos y su humanidad.

Índice

Índice de imágenes

Resumen del proyecto

Resumen

Este proyecto se basa en un trabajo previo desarrollado por anteriores grupos de Sistemas Informáticos, los cuales crearon un sistema que permitiría la navegación de robots móviles en dos dimensiones. Este sistema fue utilizado en el robot guía del museo García–Santesmases de la facultad de Informática de la Universidad Complutense de Madrid.

El trabajo de este año consistió en dotar al robot guía de la capacidad de analizar su posición real mediante técnicas de visión artificial utilizando una webcam. Para este propósito se desarrolló un sistema autónomo de localización. Asimismo, se creó un sistema que permite comunicarse con el robot mediante una aplicación remota. Por último, se ha llevado a cabo el cambio de comunicación serie a comunicación USB, permitiendo así una mayor robustez en el intercambio de información entre el robot, los sensores y los motores.

Abstract

This project is based on a previous work developed by some SSII groups in last years, that created a system which would allow the navigation of mobile autonomous robots on flat surfaces. This system was used by the robot guide of the Garcia- Santesmases museum in the Faculty of Computer Sciences of the Complutense University of Madrid.

Our work at this year consisted on providing the robot guide with the ability of analysing its real position by means of some artificial vision techniques using a webcam. For this purpose, an autonomous location system was created. Likewise we created a system that would allow to communicate to the robot through a remote application. Finally, there has been a modification to change the whole system from the actual serial communication to a new one based on USB, allowing a bigger strength in the information interexchange between the robot, the sensors and the motors.

Palabras clave

Robot, WiFi, XML, visión, artificial, remoto, modular, PIC, webcam, USB, API, firmware, software libre, frame, servicio Web.

Autorización

Autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Pablo Iglesias Marcos

Pablo Martos Rodríguez

Felipe M. Polo Ruiz

1 Introducción

1.1 Introducción

Esta sección ofrece una primera toma de contacto con el proyecto, explicando de la manera más simple y concisa posible los primeros detalles generales que hemos considerado de interés para el conocimiento de "PosCam".

1.2 Punto de partida

El proyecto llevado a cabo se enmarca en un trabajo previamente realizado. Fueron dos los grupos de Sistemas Informáticos que desarrollaron en cursos pasados el sistema que nos encontramos inicialmente: uno creó los controladores de hardware a **nivel medio** entre el software y los componentes físicos del robot (sensores, motores...), y otro realizó el control del robot a **alto nivel** mediante software, para permitir dotar al robot de inteligencia artificial. Es en este contexto de alto nivel donde nuestro proyecto debería incluirse en un principio. Esto supuso la dificultad añadida de realizar un análisis exhaustivo de todo el trabajo previo para comprender y familiarizarnos con el entorno de nuestro proyecto.

1.3 Objetivos

El objetivo principal del proyecto era dotar al robot guía de la Facultad de Informática de un sistema de visión artificial que permitiera localizarse al robot utilizando una webcam como sensor de posicionamiento incremental ayudándose del reconocimiento de balizas absolutas. Esto se hacía necesario, ya que las mediciones de los encoders en el sistema desarrollado en cursos anteriores estaban sujetas a variaciones dependientes del terreno. Esto provocaba una acumulación de error que hacía que el robot pudiera perderse con el paso del tiempo. Es reseñable que este sistema de visión artificial es **autónomo e independiente,** habiéndose integrado a posteriori en todo el software que controla al robot.

Otro objetivo fue la consecución del control del robot mediante red WiFi. El robot podría ser controlado a través de un terminal remoto, conectado al mismo mediante una conexión inalámbrica.

Finalmente se decidió cambiar la comunicación con los PIC. Previamente existía una comunicación serie, y el objetivo era cambiarla a USB para hacerla más robusta, ya que los resultados hasta el momento no eran del todo satisfactorios.

La idea fundamental ha sido añadir capas de funcionalidad a un sistema hardware, haciéndolo a la vez más robusto. Esto implicaba también el trabajo extra de analizar concienzudamente la documentación y los diseños previos, lo que nos llevó a crea guías de desarrollo y uso del sistema.

Mediante la implementación, además, hemos procurado desarrollar un sistema eficiente y sólido, a la vez que dotado de gran generalidad, para que el flujo de producción de la aplicación y posteriores modificaciones fuesen lo más cómodas y sencillas posibles.

1.4 Organización de la memoria

La memoria está organizada en 5 grandes apartados, que a continuación pasamos a describir:

Introducción. Se trata del apartado actual, en el que se pretende dar una visión general del trabajo realizado.

Especificación y diseño. En este apartado se explican con detalle la especificación y el diseño para cada uno de los distintos objetivos perseguidos.

Implementación. Se exponen detalladamente ciertos aspectos concretos de implementación y desarrollo para cada uno de los objetivos.

Integración. Se indican los aspectos más reseñables de la integración de cada una de las partes del proyecto en el contexto global.

Resultados del proyecto. En este apartado se muestran distintas pruebas y prototipos realizados en cada uno de los objetivos.

El sistema en funcionamiento. Se ofrecen guías y ejemplos de funcionamiento del sistema para poder ejecutarlo de manera sencilla.

Posteriormente se muestran distintas conclusiones, además de varios apéndices complementarios.

1.5 Gestión del proyecto

1.5.1 Introducción

En este apartado se van a exponer algunos detalles de cómo se ha logrado organizar al grupo de 3 personas que ha realizado el proyecto presente.

1.5.2 Gestión de configuración

La **gestión de configuración** consiste en garantizar que en todo momento está controlado:

- Cuál es la última copia buena de la aplicación
- Cuál es la anterior y qué cambios se le hicieron para llegar a la actual
- Dónde están y qué tenían cada una de las versiones anteriores
- Qué cambios están todavía pendientes de hacerse a la versión actual

Para conseguir una buena gestión de configuración, se han utilizado dos repositorios SVN de **Google**: uno para llevar el control de versiones para el servicio web que permite el control remoto del robot (https://robotserver.googlecode.com/svn/trunk/), y otro para el propio robot (https://robot-fdi.googlecode.com/svn). Con ambos repositorios se ha garantizado la seguridad del proyecto, a la vez que se ha permitido trabajar sobre él paralelamente.

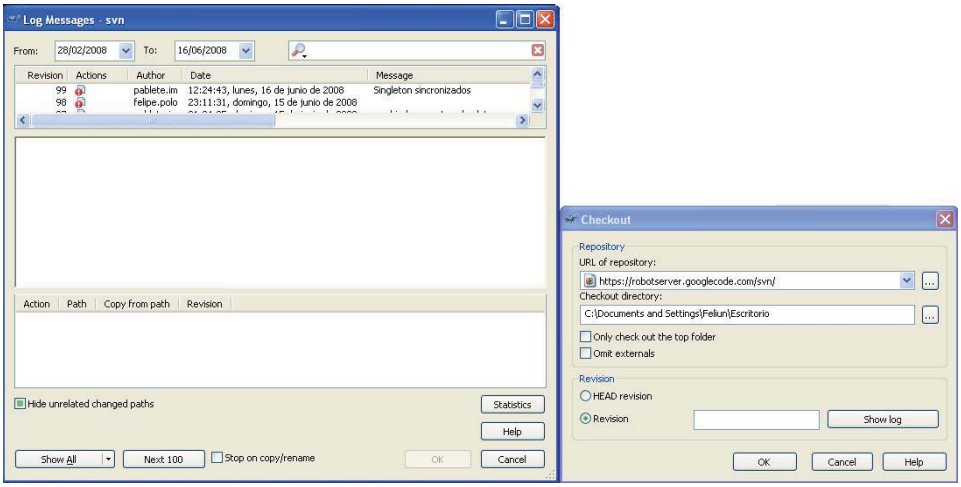


Figura 1 Aplicación SVN utilizada para gestión de configuración

1.5.3 Planificación: trabajo de prototipado

A lo largo de la realización del proyecto, se ha llevado a cabo un trabajo basado en el **prototipado exploratorio**; esto es, el prototipo inicial se refina progresivamente hasta convertirse en versión final.

En efecto, nuestro trabajo se ha desarrollado de esta forma: después de un proceso de investigación, el prototipo se ampliaba de manera incremental. Esto ha sido factible debido a que los objetivos perseguidos eran independientes entre sí, por lo que se han podido desarrollar todos los prototipos en paralelo, al no existir dependencias que pudieran retrasar el desarrollo.

Esto además ha permitido una mayor facilidad a la hora de organizar el trabajo entre los 3 componentes del grupo, los cuales han trabajado en todas las partes del proyecto al final del curso.

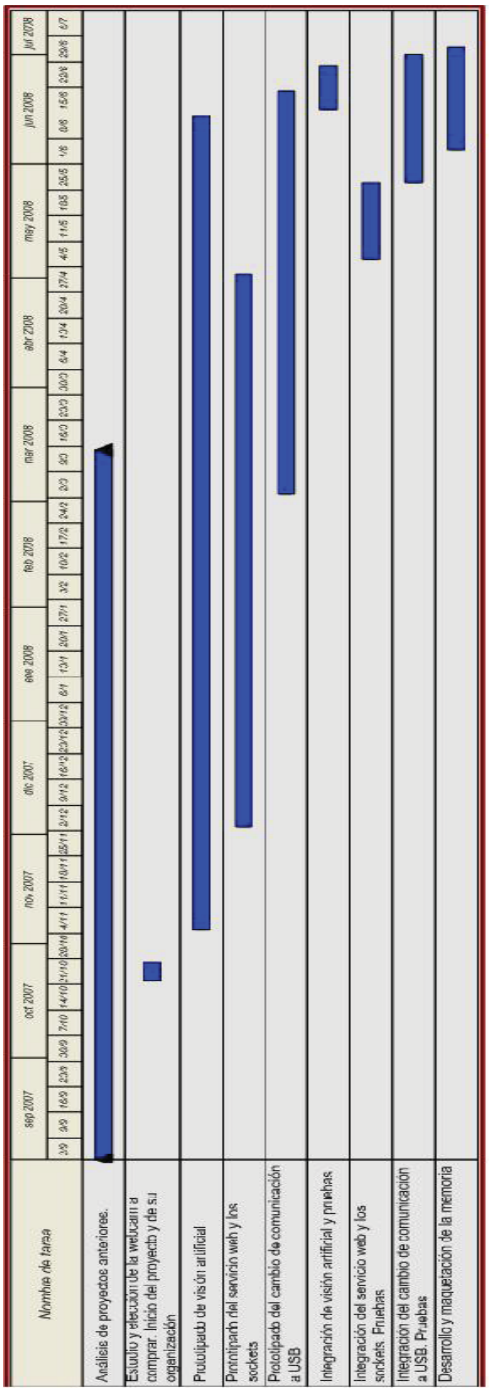


Figura 2 Planificación de las tareas a lo largo del curso

La figura 2 muestra la planificación que se ha seguido en cada una de las tareas realizadas a lo largo del curso. A continuación pasamos a enumerarlas, explicando brevemente algunos aspectos de su realización.

- **Análisis de proyectos anteriores**. Esta etapa era muy importante, y debía ser necesariamente anterior a la fase de diseño y por supuesto implementación; por ello, el análisis de estos proyectos comenzó incluso antes de empezar el curso, leyendo las memorias que nos proporcionaron.
- **Estudio y elección de la webcam a comprar. Inicio del proyecto y de su organización**. Esta etapa duró únicamente una semana, y se realizó próxima al comienzo del curso. Es la fase en la que el proyecto realmente arranca.
- **Prototipado de visión artificial**. Los prototipados fueron incrementales, como hemos comentado anteriormente. Éste, en concreto, fue el que más duró, debido a su complejidad y a que se trataba del objetivo principal del proyecto. Por ello, en cuanto se compró la webcam se comenzó a trabajar en él.
- **Prototipado del servicio web y los sockets.** La parte de este prototipado más complicada fue la conexión de la aplicación cliente – servidor con la aplicación del robot a través de sockets, por lo que fue la primera tarea en la que se trabajó dentro de este prototipo incremental, y también la que más tiempo requiso.
- **Prototipado del cambio de comunicación a USB.** Este objetivo fue el más tardío. Surgió aproximadamente a finales de febrero, cuando se intentó mover el robot y el PIC de comunicación serie falló, imposibilitando cualquier prueba sobre el robot. Por ello, en este punto nos planteamos el cambio a comunicación USB, que resultaría mucho más fiable y robusta.
- **Integraciones.** Todas las integraciones se realizaron en las semanas finales de los prototipados. En el caso del servicio web, se realizó antes de finalizar el prototipo, ya que desde el momento en el que se integró, se decidió ampliar su funcionalidad.
- **Desarrollo y maquetación de la memoria.** Este proceso se desarrolló a lo largo de las últimas semanas del proyecto, mientras se realizaban pruebas de los prototipos desarrollados.

1.5.4 Recursos utilizados para la organización

Para permitir una mejor comunicación entre los miembros del grupo y el tutor del proyecto, así como para tener un espacio donde poder albergar documentación e informes a lo largo de todo el trabajo, se creó un grupo en Google (http://groups.google.es/). Éste se usó como una lista de distribución, de forma que todos los mensajes escritos en el foro del grupo se reenviaban a las cuentas de correo de los componentes, incluido el tutor del proyecto.

Además se utilizó para almacenar informes, guías de desarrollo, gráficos y software útil.

1.6 Conocimientos previos

A pesar de haber sido un proyecto más cercano a la investigación que al desarrollo, nos ha permitido poner en práctica gran parte de los conocimientos adquiridos a lo largo de la carrera. Se ha seguido el sistema de trabajo basado en análisis, diseño y síntesis propio de la Ingeniería.

En cada uno de los objetivos del proyecto se ha usado esta metodología: el proceso de análisis ha sido fundamental a la hora de estudiar los anteriores proyectos sobre los que se iban a trabajar, de forma que obtuviéramos un marco adecuado sobre el que desarrollar nuestros objetivos.

Una vez esta fase de análisis se superó, pasamos a estudiar el diseño de nuestros sistemas. Sólo cuando esta segunda etapa finalizó, llegamos a la tercera y última fase de implementación y desarrollo.

Figura 3 Proceso de ingeniería seguido en el proyecto

2 Especificación y diseño

2.1 Introducción

En este apartado se pretende explicar detalladamente cada uno de los objetivos del proyecto, basándonos especialmente en la especificación de cada uno de ellos y el diseño obtenido.

2.2 Trabajo inicial.

Antes de explicar las especificaciones y diseños seguidos, procedemos a detallar el trabajo realizado inicialmente.

- Cada uno de los componentes del grupo estudió las memorias de los proyectos anteriores en los que se basaba el nuestro. Este punto era fundamental para entender el contexto en el que nos íbamos a mover. Se realizó un análisis a fondo de todo el trabajo anterior.
- Posteriormente, se creó una guía de desarrollo para instalar y utilizar todas las tecnologías que habían sido necesarias en los proyectos anteriores. Esto se hizo vital para poder trabajar sin problemas de configuraciones, ya que en las memorias de otros años no había ninguna guía ni explicación al respecto. Esta guía se puede consultar con detalle en el Apéndice B.
- Más adelante, cuando los objetivos del proyecto estaban más o menos claros, definidos y situados dentro del marco que habíamos heredado de estos proyectos pasados, pasamos a estudiar el problema de qué webcam debíamos comprar. El presupuesto era limitado, y necesitábamos un dispositivo que cumpliese con ciertos parámetros: resolución, frecuencia de captura, ángulo de visión, conectividad,… A continuación se puede ver una comparativa de las webcams que se ajustaban a nuestras necesidades:

WebCam	Sensor	Resolución	Resolución interpolada	Frecuencia de cuadros máxima	Ángulo de visión	Conectividad
PHILIPS SPCL 300 NC	CMOS	1,3 MP	6 MP	90 fps	80º	USB 2.0
Creative WebCam Live! Motion	CCD	640 x 480	1,3 MP	30 fps	76º	USB 2.0
Creative Live! Cam Notebook Ultra		1,3 MP	5 MP		80º	USB 2.0
Creative Live! Cam Voice		1,3 MP	5 MP	25 fps	85º	USB 2.0
Creative Live! Cam Optia Pro	CMOS	1,3 MP	5 MP		71º	USB 2.0
Creative Live! Cam Optia AF	CMOS	2 MP	8 MP		63º	USB 2.0
Logitech Quickcam Pro for Notebooks	CMOS	2 MP	8 MP	30 fps		USB 2.0
Logitech Quickcam Pro 9000	CMOS	2 MP	8 MP	30 fps		USB 2.0
Microsoft WebCam LifeCam VX-7000	CMOS	2 MP	8 MP		71º	USB 2.0

Al final seleccionamos las cuatro últimas, ya que destacaban en algunos aspectos sobre las demás y no superaban nuestro presupuesto. Como sus características son muy similares acabamos comprando la Logitech Quickcam Pro for Notebooks de 1,3 MP ya que fue la más fácil de encontrar en las tiendas de las cuatro elegidas.

- Una vez comprado y probado el dispositivo en distintos terminales, se creó una primera planificación con el objetivo de comenzar a desarrollar prototipos iniciales, tanto para el sistema de visión artificial como para el servicio web que permitiría conectar el sistema del robot con un cliente remoto.
- Se crearon los recursos necesarios para la organización y gestión del proyecto.
- A partir de aquí, una vez creada una plataforma sólida de organización sobre la que poder trabajar, comenzamos a abordar los objetivos que a continuación se explican.

2.3 Control remoto del robot

2.3.1 Introducción

El robot, al tratarse de un sistema autónomo y no poseer una consola de control integrada, ni un monitor visual, necesita un sistema con interfaz gráfica, que permita controlarlo remotamente.

Con el nuevo control remoto del robot se pretende mejorar la comunicación existente entre éste y el robot, para ello debíamos abordar los siguientes problemas:

- Perdida de paquetes de datos en la conexión y por lo tanto la perdida de envío de algunos comandos.
- Perdida frecuente de la conexión establecida.
- Tratamiento de la información enviada por el Robot al control remoto
- Posibilidad de control remoto multiplataforma, de modo que pueda ejecutarse tanto en PC's normales como en dispositivos móviles.

No debemos olvidar, que el control remoto no es más que un mero sistema de dirección, con lo cual, se pretende imitar la funcionalidad de un Joystick, pero siempre pendientes de la posibilidad de añadir nuevas funcionalidades, es decir, de hacer el control remoto extensible.

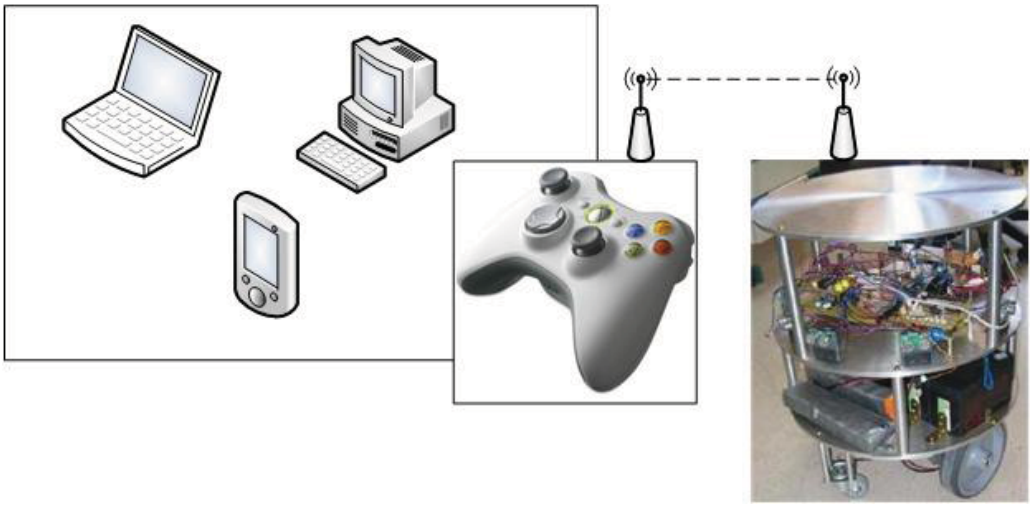


Figura 4 Control remoto del robot

En el proyecto heredado de anteriores cursos se había desarrollado un pequeño servidor de Sockets en C, pero no resultó ser realmente operativo, ya que la conexión se desconectaba con demasiada frecuencia. Por esto, decidimos realizar un sistema robusto a la vez que flexible en Java que permitiese conectar con el robot y ejecutar distintos servicios del mismo desde un terminal remoto.

Para llegar a este fin, surgieron dos nuevos objetivos: realizar una aplicación cliente – servidor en Java, y crear un módulo dentro de la arquitectura del robot que atendiese las peticiones del cliente.

Se barajaron varias posibilidades para el diseño e implementación del nuevo servicio. En la figura 5 se muestran dichas opciones:

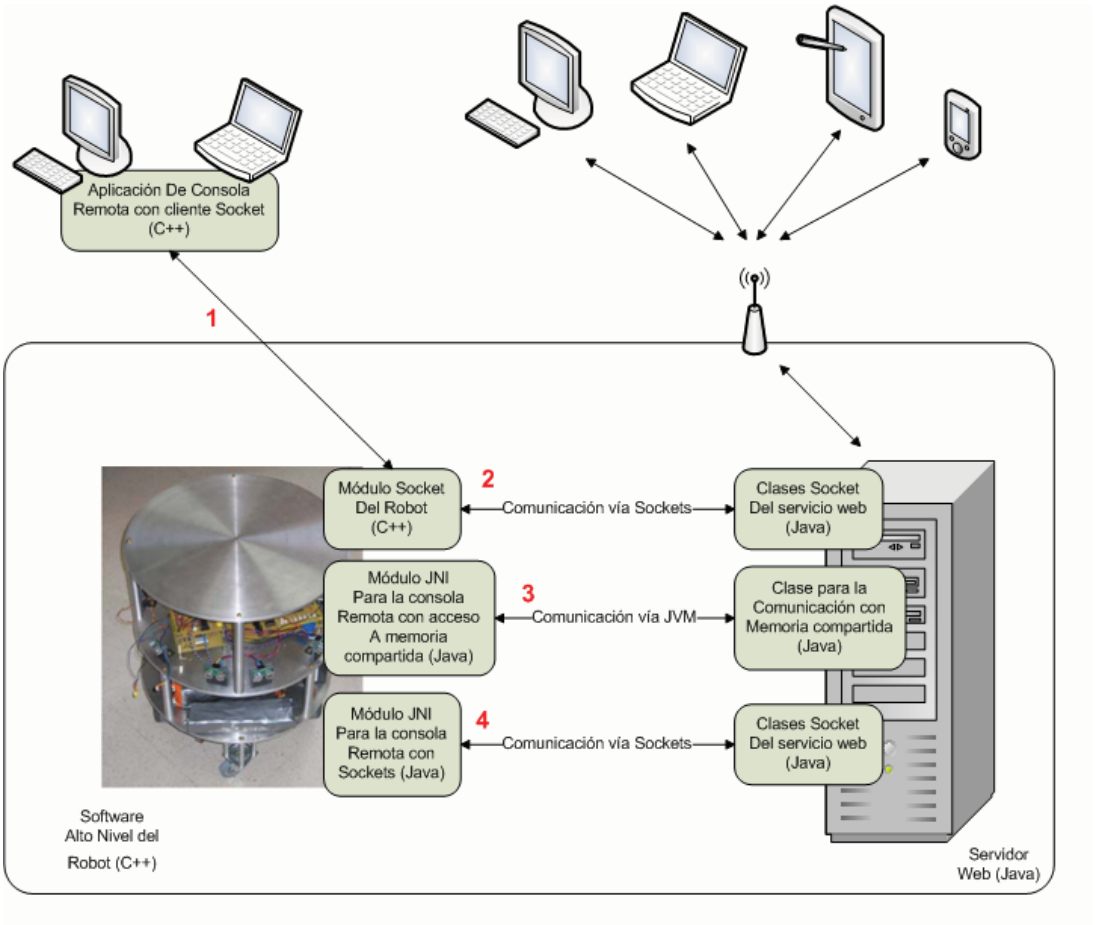


Figura 5 Opciones de diseño del servicio Web

Se partía de la opción número uno, ya que era el trabajo implementado anteriormente y que debíamos sustituir.

Las opciones restantes suponían la construcción de un servicio Web en Java, utilizando la tecnología AJAX. Este punto se tuvo claro desde el principio, ya que permitiría tener control absoluto del robot de una manera limpia mediante cualquier dispositivo conectado de manera inalámbrica con el robot: esto incluye tanto a un PC como a una PDA o cualquier dispositivo móvil con tecnología WiFi y un navegador Web integrados, siendo además Java independiente de la plataforma, pudiendo instalar el servicio web en cualquier sistema operativo. De este modo se continúa el requisito del proyecto de partida, en el que se especificaba que el sistema debía ser multiplataforma.

Tan sólo quedaba decidir cómo se interconectarían de forma local, la aplicación servidor Java y el software de alto nivel del robot en C++.

La opción dos era la más sencilla, ya que suponía realizar la comunicación vía Sockets de forma local entre un cliente Java y un servidor C++. Además minimizaba los riesgos tecnológicos, al ser una tecnología conocida por los integrantes del proyecto y realizarse de forma local, suponiendo una conexión más fiable que la primera opción.

La opción tres suponía la implementación de una aplicación intermedia en Java con conexión directa a un módulo del robot utilizando JNI (Java Native Interface) y la comunicación a través de la máquina virtual de java del servicio Web con dicha aplicación. Desconocíamos la posibilidad de comunicar dos aplicaciones Java mediante la máquina virtual utilizando memoria compartida, y la línea de investigación sobre esta posibilidad derivó a la comunicación de las dos aplicaciones Java con Sockets. De esta nueva idea surgió la cuarta opción.

Finalmente se eligió la segunda opción, ya que se obtenía una mayor independencia del código Java y C++, y evitábamos riesgos tecnológicos.

El diseño de esta arquitectura permite ampliar fácilmente la funcionalidad del servicio Web, dotando de una gran flexibilidad al uso del robot.

2.3.2 Aplicación cliente-servidor

Para la realización de esta aplicación, nos hemos apoyado en la arquitectura cliente-servidor, utilizando una metodología de diseño basada en capas. Los casos de uso se han ido desarrollando a medida que surgían nuevas necesidades o nuevas ideas para el uso de robot.

2.3.2.1 Arquitectura cliente-servidor

Esta arquitectura consiste básicamente en que un programa –el cliente- realiza peticiones a otro programa -el servidor- que le da respuesta. Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora, es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras, aumentando así el grado de distribución del sistema.

La **arquitectura cliente-servidor** sustituye a la arquitectura **monolítica** en la que no hay distribución, tanto a nivel físico como a nivel lógico.

2.3.2.2 Diseño basado en capas

La calidad tan especial de la arquitectura basada en capas consiste en aislar la lógica de la aplicación y en convertirla en una capa intermedia bien definida.

En el sistema del servicio Web se han definido tres capas claramente diferenciadas:

- Capa de presentación
- Capa de negocio
- Capa de datos

Se ha integrado la interfaz web y el modelo en un mismo servidor aunque conservando su independencia funcional. Ésta es la distribución en capas más común en las aplicaciones web.

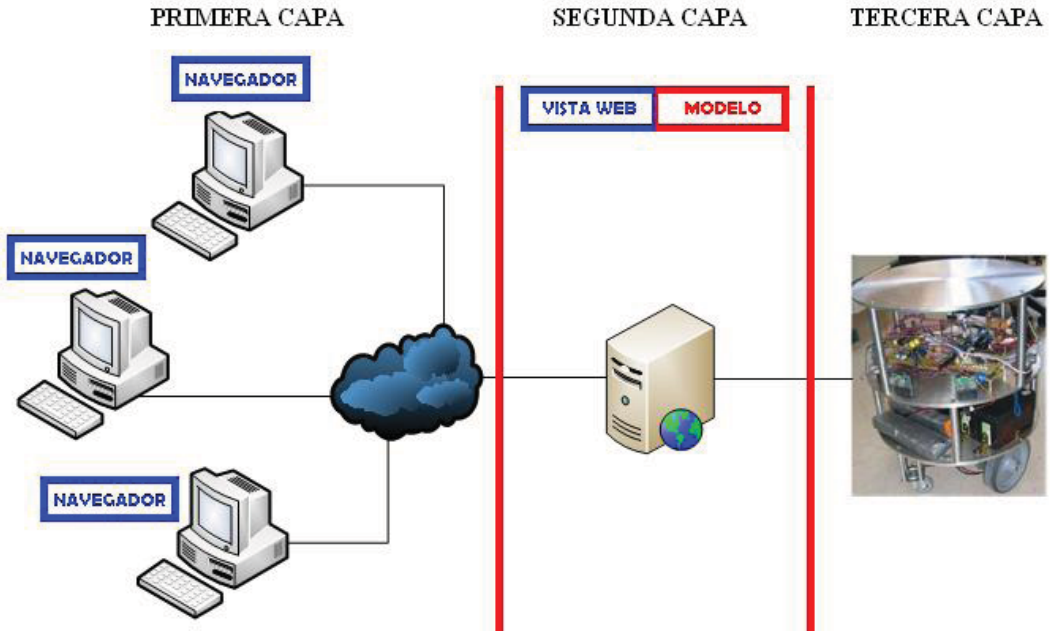


Figura 6 Diseño de capas del servicio Web

La capa de presentación, al tratarse de una aplicación con la utilización de la tecnología AJAX, aprovecha la potencia de procesamiento de las máquinas cliente, sin sobrecargar el servidor, haciendo peticiones asíncronas cuando necesita realizar cambios u obtener información del modelo de datos, a través de la interfaz de servicios asíncronos de la capa de negocio.

La capa de presentación por lo tanto, gestiona en la máquina del cliente todas las actualizaciones entre las diferentes vistas, sin necesidad de intervención de la capacidad de procesamiento del servidor.

La capa de datos en sí, puede parecer inexistente en nuestra arquitectura, al no tratarse de una base de datos con un modelo de persistencia en la capa de negocio. No obstante, se considera el software de alto nivel del robot como el "ente" que aporta la información al modelo a través del servicio de Sockets.

Decir también, que cada una de las funcionalidades añadidas al servicio web, tales como el control de carga de la batería, la funcionalidad de alertas, o la configuración con ficheros XML, aportan su propia capa de datos, ya sea remota, local o aportada por el propio sistema operativo, con lo cual, podemos hablar de una capa de datos múltiple.

Tal y como muestra la figura 6 del diseño basado en capas y como es característica habitual en un servicio Web, se permiten varias conexiones simultaneas, con lo cual en este caso, podría haber varios clientes remotos controlando el Robot en un momento determinado. Esto podría suponer un problema, ya que un usuario podría observar comportamientos anómalos provocados por el control de otra consola sin tener conocimiento de su existencia. No obstante la consola remota diseñada en nuestro proyecto, partía de la precondición de mantener tan sólo un cliente que debía estar en presencia del robot.

En el apartado de trabajo futuro, se muestran las ventajas, potencia y posibles ampliaciones de funcionalidad habiendo utilizado para el control remoto un servicio Web.

2.3.2.3 Casos de uso de la aplicación

Un **caso de uso** es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. Normalmente, en los casos de usos se evita el empleo de jergas técnicas, prefiriendo en su lugar un lenguaje más cercano al usuario final.

En otras palabras, un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema.

Los casos de uso para esta aplicación se han pensado por entero para interactuar con el robot guía, aunque algunos de ellos podrían ser ejecutados en un terminal cualquiera. Pasamos a continuación a ver la descripción de cada uno de los casos de uso contemplados.

Nombre: OBTENER EL ESTADO DE LA BATERÍA
Objetivo: Obtener un informe sobre el estado de la batería del robot
Entradas: Ninguna
Precondiciones: Ninguna
Salidas: Mensaje informativo con el estado de la batería, además de un sonido
Postcondición si éxito: Ninguna
Postcondición si fallo: Ninguna
Actores: Cliente y Servidor
Secuencia normal:

El actor selecciona la opción de consultar estado de batería Aparece un mensaje informando acerca del estado de la batería y un sonido
Secuencia alternativa: El actor selecciona la opción de consultar estado de batería Aparece un mensaje indicando que no ha sido posible conectar, en caso de error
Requisitos no funcionales: conectividad entre el cliente y el servidor. Además, la consulta sobre el estado de la batería debe ser compatible con el terminal donde esté alojado el servidor.

Nombre: COMUNICARSE CON EL TÉCNICO RESPONSABLE
Objetivo: Comunicarse con el técnico responsable, enviándole un mensaje vía e-mail y/o sms a su móvil
Entradas: Elección de envío del mensaje a través de email y/o sms, y elección del asunto del mensaje
Precondiciones: La cuenta de GMail del técnico debe haberse configurado correctamente en el fichero XML. Si desea poder recibir SMS a su número de móvil, debe configurar además su cuenta de GCalendar. Se dan detalles más concretos acerca de esto en 3.1.2.3.
Salidas: Mensaje para elegir la forma y el asunto de envío del mensaje
Postcondición si éxito: El técnico recibe el mensaje en forma de mail y/o sms
Postcondición si fallo: El técnico no recibe el mensaje
Actores: Cliente, Servidor y Técnico
Secuencia normal: El actor selecciona la opción de enviar un mensaje al técnico Aparece una ventana para seleccionar el tipo de mensaje Se pulsa el botón “Enviar” Aparecen ventanas confirmando los envíos junto a unos sonidos El técnico recibe los avisos
Secuencia alternativa: El actor selecciona la opción de enviar un mensaje al técnico Aparece una ventana para seleccionar el tipo de mensaje Se pulsa el botón “Enviar” Aparecen ventanas indicando que no se han podido realizar el envío, junto a unos sonidos El técnico no recibe el aviso
Requisitos no funcionales: conectividad entre cliente y servidor. El técnico debe tener su cuenta de correo de GMail configurada correctamente para recibir avisos por sms. Además esta cuenta debe aparecer en el XML de configuración.

Nombre: CAMBIAR EL MODO DE EJECUCIÓN DEL ROBOT
Objetivo: Alternar entre el modo Usuario y el modo Autónomo de ejecución del robot.
Entradas: Modo al que se desea cambiar.
Precondiciones: Ninguna
Salidas: Mensaje informativo con el resultado del cambio.
Postcondición si éxito: Se cambia al modo deseado.
Postcondición si fallo: Si no hay conexión WiFi, ninguna. Si hay conexión WiFi pero no Socket, se fuerza el modo autónomo.
Actores: Cliente, Servidor y Robot.
Secuencia normal: El Actor Cliente selecciona el modo de ejecución nuevo. Se envía en forma de comando el modo nuevo por Sockets al Robot. El Robot responde con el comando que se le envió y ejecuta la acción pertinente. El Servidor comprueba que la cadena recibida es igual a la que se envió y se muestra el mensaje en consecuencia. Se conmuta a la interfaz de usuario para el modo nuevo.
Secuencia alternativa: El Actor Cliente selecciona el modo de ejecución nuevo. Si no se puede comunicar el Servidor, se muestra un mensaje de error. Si se conecta con el Servidor pero no con el Robot, se supone que este no está en ejecución, y se fuerza la interfaz de usuario en modo automático mostrando un mensaje, ya que será el estado inicial del Robot al ser reiniciado.
Requisitos no funcionales: conectividad entre el cliente y el servidor y entre el servidor y el robot.

Nombre: ENVIAR COMANDOS EN MODO AUTÓNOMO AL ROBOT
Objetivo: Enviar los comandos de ejecución en modo automático al robot.
Entradas: Comando que se desea enviar. Este comando podrá ser: Terminar, Presentación, Guía, Posición u Orientación.
Precondiciones: El robot debe encontrarse en modo autónomo.
Salidas: Mensaje informativo con el resultado del envío del comando.
Postcondición si éxito: El Robot ejecuta el comando enviado.
Postcondición si fallo: Se muestra un mensaje de error en la consola.
Actores: Cliente, Servidor y Robot.
Secuencia normal: El actor Cliente selecciona el comando a enviar, rellenando la información pertinente. Se envía el comando por Sockets al Robot.

El Robot responde con el comando que se le envió y ejecuta la acción pertinente. El Servidor comprueba que la cadena recibida es igual a la que se envió y se muestra el mensaje en consecuencia.
Secuencia alternativa: El Actor Cliente selecciona el comando a enviar, rellenando la información pertinente Si no hay conexión con el servidor o con el robot, se muestra un mensaje de error.
Requisitos no funcionales: conectividad entre el cliente y el servidor y entre el servidor y el robot.

Nombre: ENVIAR COMANDOS EN MODO USUARIO AL ROBOT
Objetivo: Enviar los comandos de ejecución en modo usuario o manual al robot.
Entradas: Comando direccional que se desea enviar (Avanzar, Retroceder, Girar, Rotar, etc.).
Precondiciones: El robot debe encontrarse en modo usuario.
Salidas: Mensaje informativo con el resultado del envío del comando.
Postcondición si éxito: El Robot ejecuta el comando enviado.
Postcondición si fallo: Se muestra un mensaje de error en la consola.
Actores: Cliente, Servidor y Robot.
Secuencia normal: El Actor Cliente selecciona el comando a enviar. Se envía el comando por Sockets al Robot. El Robot responde con el comando que se le envió y ejecuta la acción pertinente. El Servidor comprueba que la cadena recibida es igual a la que se envió y se muestra el mensaje en consecuencia.
Secuencia alternativa: El Actor Cliente selecciona el comando a enviar. Si no hay conexión con el servidor se muestra un mensaje de error. Si se conecta con el Servidor pero no con el Robot, se supone que este no está en ejecución, y se fuerza la interfaz de usuario en modo automático mostrando un mensaje, ya que será el estado inicial del Robot al ser reiniciado.
Requisitos no funcionales: conectividad entre el cliente y el servidor y entre el servidor y el robot.

Nombre: CONOCER LA POSICIÓN U ORIENTACIÓN ACTUAL DEL ROBOT
Objetivo: Conocer la posición o la orientación actuales del robot.
Entradas: Comando especial que no requiere ACK, sino la información requerida del Robot.
Precondiciones: ninguna
Salidas: Mensaje informativo con la información requerida.

Postcondición si éxito: Se muestra la posición o la orientación en la consola.
Postcondición si fallo: Se muestra un mensaje de error en la consola.
Actores: Cliente, Servidor y Robot.
Secuencia normal: El Actor Cliente selecciona el comando a enviar (Orientación o Posición). Se envía el comando por Sockets al Robot. El Robot responde con la información pertinente dependiendo del comando enviado. El Servidor muestra por pantalla dicha información.
Secuencia alternativa: El Actor Cliente selecciona el comando a enviar (Orientación o Posición). Si no hay conexión con el servidor o con el robot, se muestra un mensaje de error.
Requisitos no funcionales: conectividad entre el cliente y el servidor y entre el servidor y el robot.

Nombre: REALIZAR UN TEST DE LAS CONEXIONES
Objetivo: Conocer el estado de las conexiones, tanto WiFi como Socket.
Entradas: Comando especial de test de conexión.
Precondiciones: ninguna
Salidas: Mensaje informativo la información del estado de cada una de las conexiones.
Postcondición si éxito: Se muestra un mensaje del estado de las conexiones OK.
Postcondición si fallo: Se muestra un mensaje con la causa de no funcionamiento del WiFi o los Sockets.
Actores: Cliente, Servidor y Robot.
Secuencia normal: El Actor Cliente selecciona el comando "*test*". Se envía el comando por Sockets al Robot. El Robot responde con un ACK igual al comando enviado. El Servidor Envía al cliente WiFi: OK; Sockets OK.
Secuencia alternativa: El Actor Cliente selecciona el comando "*test*". Si no hay conexión con el servidor se muestra un mensaje informativo de que no hay conexión WiFi. Si no hay conexión con el robot, se muestra "WiFi: OK" y se informa de la causa de error de la conexión Socket.
Requisitos no funcionales: conectividad entre el cliente y el servidor y entre el servidor y el robot.

Nombre: MOSTRAR AYUDA DEL SERVICIO WEB EN EL CLIENTE

Objetivo: Mostrar una breve ayuda sobre la utilización del servicio Web.
Entradas: ninguna
Precondiciones: ninguna
Salidas: Mensaje informativo en consola de los comandos de modo autónomo, así como un panel con la ayuda extendida.
Postcondición si éxito: Se muestra la ayuda en consola y en un panel externo.
Postcondición si fallo: ninguna
Actores: Cliente
Secuencia normal: El Actor Cliente pulsa el botón de ayuda. Se muestra un panel con la ayuda en HTML y un resumen de los comandos automáticos en la consola.
Secuencia alternativa: ninguna
Requisitos no funcionales: conectividad entre el cliente y el servidor.

Nombre: LIMPIAR LA CONSOLA DE MENSAJES
Objetivo: Resetear la consola de información del servicio Web, eliminando todos los mensajes antiguos.
Entradas: ninguna
Precondiciones: ninguna
Salidas: Consola vacía.
Postcondición si éxito: Se borra la consola.
Postcondición si fallo: ninguna
Actores: Cliente.
Secuencia normal: El Actor Cliente pulsa el botón de limpiar. Se resetea la consola, borrando todos los mensajes anteriores.
Secuencia alternativa: ninguna
Requisitos no funcionales: conectividad entre el cliente y el servidor.

2.3.2.4 La interfaz gráfica

El diseño de una interfaz gráfica debe conllevar un uso sencillo, amigable, intuitivo y accesible, a la par de ser usable e implementar toda la funcionalidad que se le requiere.

Para la capa de presentación del servicio Web, se ideó una interfaz dividida en paneles, con funcionalidades claramente diferenciadas.

Los casos de uso especifican el envío de comandos en modo autónomo o en modo usuario, la utilización de una consola con herramientas para su gestión y comandos que pueden enviarse en cualquier momento, como los controles de batería, la consulta de la posición, etc. Por lo tanto se ha diseñado una interfaz dividida en 4 partes:

La mitad izquierda contiene un panel de pestañas, en el que eligiendo una u otra, se alterna entre los modos "Autónomo" y "Usuario", cada uno con su funcionalidad específica.

La mitad derecha, queda dividida en tres partes. La parte superior, Se utiliza como consola de texto para "logging", la parte intermedia para los controles que manipulan directamente dicha consola, y la parte inferior para los comandos especiales.

Además, se incluye un titulo en la parte superior, y una barra de estado en la parte inferior de la interfaz.

Figura 7 Aspecto de la interfaz gráfica

Se ha obtenido un diseño amigable y accesible, gracias a la utilización de textos alternativos en cada uno de los paneles y componentes de la interfaz. Esta cualidad hace que la interfaz, puramente gráfica, pueda ser interpretada por lectores de pantalla y por consiguiente utilizada por personas con visibilidad reducida.

Figura 8 Botones intuitivos

La interfaz es usable gracias a la consola de texto y una correcta gestión de errores, ya que en ella se muestra toda la información acerca del estado de la interfaz, con lo cual el usuario en todo momento sabe lo que está sucediendo, quedando oculto todo detalle interno del servidor.

Otro aspecto muy importante de las interfaces gráficas, es una elección correcta de los colores, ya que debe ser igualmente usable y correctamente interpretada tanto en color, como en blanco y negro. Nuestra aplicación sería perfectamente usable sin color, ya que las ilustraciones con texto alternativo identifican perfectamente cada una de las funcionalidades.

Por último, se debe mencionar, que los controles que activan las funcionalidades más usadas, se encuentran en las zonas externas o alejadas del centro de la interfaz, ya que es en dichos lugares, donde los usuarios tienen más facilidad de dirigir el puntero del ratón. Si no es así, como en el caso del botón "Enviar Comando" del panel de controles del modo automático, se ha aumentado considerablemente el tamaño del control para un acceso más sencillo a dicho botón.

2.3.3 Aplicación para alertar al técnico en caso de nivel bajo de batería.

Además de realizar una aplicación cliente – servidor que hiciera las funciones de un mando a distancia para el robot, decidimos realizar una aplicación extra, que dota de más independencia a su gestión.

La aplicación comprueba constantemente y cada cierto tiempo el estado de la batería en el terminal, y cuando detecta que ésta presenta un nivel de carga inferior al permitido (parámetro configurable por XML), envía un SMS y un mail al técnico responsable, cuya dirección de correo electrónico es

también configurada por XML. Además, emite un sonido de alarma por si surgiera algún problema en el envío. La figura 9 representa un intuitivo gráfico que nos aclara el funcionamiento de esta aplicación de alertas:

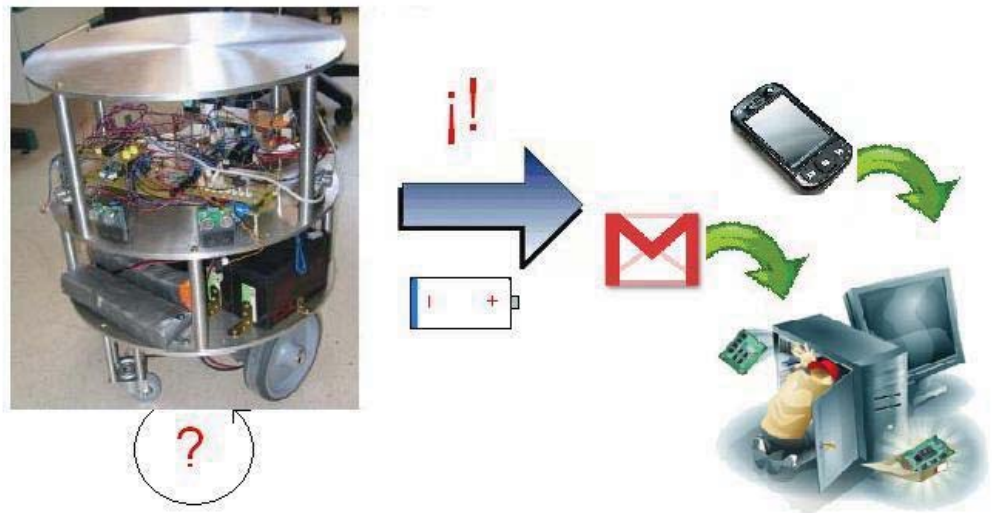


Figura 9 Comportamiento de la aplicación de alertas

Como se puede intuir, en esta aplicación se **reutilizan** algunos servicios de la aplicación web basada en la arquitectura cliente – servidor explicada en el punto anterior 2.3.2, como la consulta de batería disponible o el envío de mensaje. Esto ha sido posible debido al buen diseño realizado, basado en componentes y servicios claramente definidos a través de las distintas interfaces.

Es digno de mencionar, igualmente, que esta aplicación es perfectamente operativa en cualquier otro terminal. Un usuario normal puede utilizarla para su ordenador portátil, siendo notificado via mail y sms el estado de la batería cuando alcanza un límite, también configurado por XML. Todos estos detalles de configuraciones se detallan en el punto 3.1.2.3.

2.3.3.1 Componentes

Algunos criterios técnicos para un buen diseño son:

- Un diseño debe presentar una estructura arquitectónica que
  - Se haya creado mediante patrones de diseño reconocibles.
  - La integren **componentes** que exhiban buenas características de diseño.
  - Que pueda implementarse de manera evolutiva para que de estar forma facilite la implementación y las pruebas.
- Un diseño debe ser modular.
- Un diseño debe conducir a componentes que representan características funcionales independientes.

El diseño del proyecto en general, y de esta aplicación en particular, se ha basado en estas características. Este diseño basado en componentes nos permite, por un lado, realizar una implementación fácil de depurar y, por otro, al ser entidades funcionales independientes, reutilizarlos en otras aplicaciones (como ocurre en este caso). La figura 10 muestra un diagrama donde se muestra la relación entre los componentes de la aplicación:

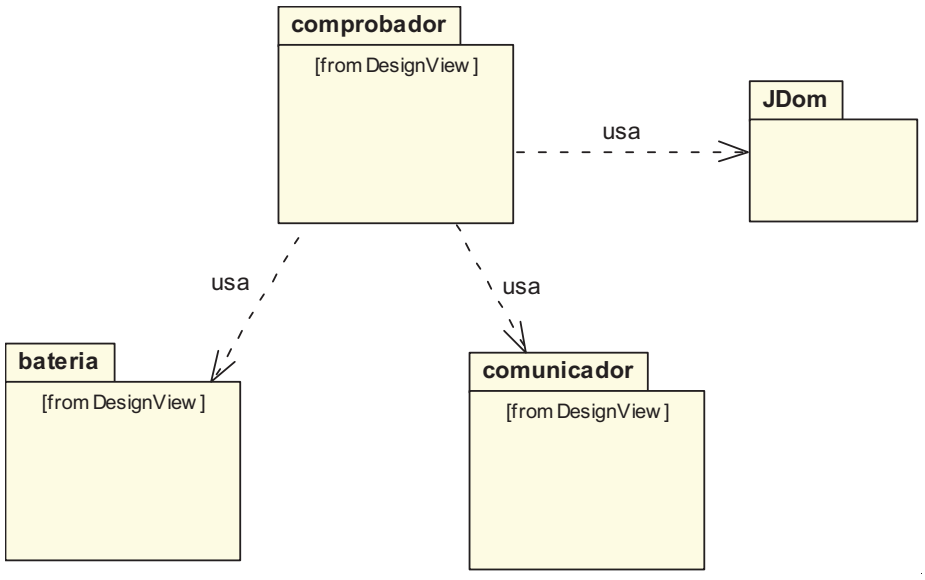


Figura 10 Componentes de la aplicación de alertas

Pasamos a continuación a describir el aspecto de los componentes reutilizados en esta aplicación, que a la vez se usan en el servicio web descrito en el apartado 2.3.2:

2.3.3.2 El componente "Batería"

- Aspecto del módulo

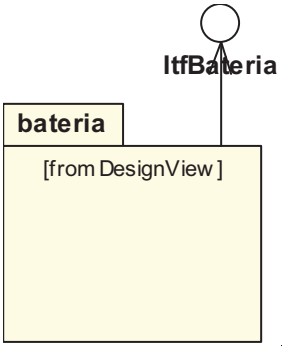


Figura 11 Componente "Batería"

Como se puede observar en la figura 11, el componente implementa una funcionalidad definida según la interfaz "ItfBateria".

- Funcionalidad

El componente posee la funcionalidad especificada por su interfaz "itfBateria", la cual posee las siguientes operaciones:

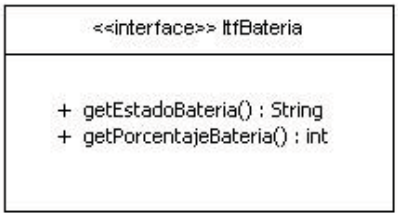


Figura 12 Interfaz "ItfBateria"

Ambas operaciones son necesarias para controlar en todo momento el estado de la batería del terminal que las invoca. Este componente es perfectamente reutilizable, ampliable e integrable en cualquier arquitectura modular.

2.3.3.3 El componente "Comunicador"

- Aspecto del módulo

El aspecto de este módulo es muy similar al del anterior:

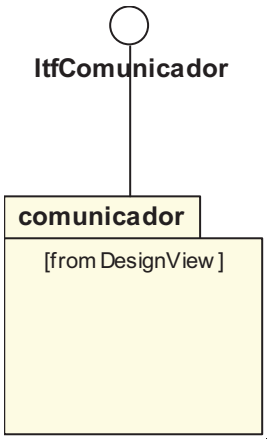


Figura 13 Componente "Comunicador"

La diferencia principal radica en la interfaz implementada ItfComunicador.

- Funcionalidad

La funcionalidad de este módulo permite enviar un correo electrónico o un mensaje de texto según los parámetros requeridos. Los detalles de implementación se facilitarán en el apartado 3. Las operaciones ofrecidas por la interfaz son:

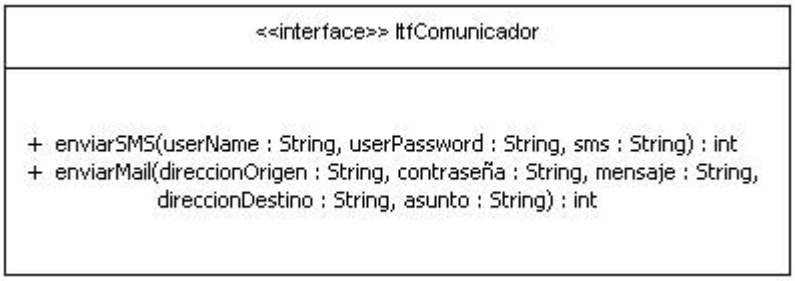


Figura 14 Interfaz "ItfComunicador"

El diseño realizado en este módulo también ha sido pensado para que pueda ser reutilizado, ampliado o perfectamente integrado en cualquier arquitectura, como ocurre en este mismo proyecto.

2.3.1 Sockets

El diseño de los Sockets, incluye tanto el diseño de un cliente en la parte del servicio Web como el rediseño y la implementación de un servidor de Sockets en el sistema de alto nivel del robot.

2.3.1.1 El componente servidor Socket en el Robot

El servidor existente y que debíamos mejorar, utilizaba dos servidores de Sockets, uno de recepción y uno de envío, y mantenía una conexión durante todo el ciclo de ejecución con la consola remota que se utilizaba.

En nuestro caso, dados los requisitos funcionales y los casos de uso que se desprenden de ellos, tan sólo era necesaria una conexión entrante con respuestas atómicas a las peticiones, y dado el caracter esporádico de los comandos que se recibían, no resultaba interesante mantener una conexión activa.

Una conexión activa permanente es más robusta y eficiente, pero no se adaptaba a los requerimientos de nuestros casos de uso, además de suponer un inconveniente la recuperación de la conexión, si ocurre algún problema en alguno de los componentes del sistema.

Desde otro punto de vista, una conexión activa también permitiría enviar datos de forma asíncrona del servidor de Sockets del Robot al cliente de Sockets de Java y por lo tanto a los diferentes clientes web que pudieran estar conectados al servicio. Esto sería interesante de cara a un control remoto desatendido, es decir, sin estar personado ante el propio robot. No obstante el objetivo del proyecto era mejorar la comunicación de la consola remota existente, implementándola de modo que la conexión fuera más fiable.

Además, con la implementación actual no sería necesario un envío de datos asíncrono por parte del Robot, ya que al estar en presencia del robot, siempre se debería conocer donde está. A pesar de ello, se han integrado dos opciones en el Servicio Web para conocer la orientación y posición del Robot en cada momento.

En el apartado de trabajo futuro se explican detalladamente las ventajas de un servicio de Sockets orientado a conexión.

Con el diseño seguido, si termina la ejecución del Servicio Web que contiene el cliente de Sockets, no supone ningún problema para el ciclo de ejecución del Robot.

Con lo cual, el servidor esta diseñado para, una vez activado, quedar a la espera de comandos entrantes. Una vez llega un comando, se crea una conexión, se procesa, se envía la respuesta y se desconecta, quedando a la espera de nuevos comandos.

Existen dos comandos que requieren respuesta especial, tales como la consulta de posición y orientación actuales del robot. Para ello, se ha hecho utilización de la arquitectura del Robot para crear una conexión con el módulo "Robot Posición" con el fin de mantener actualizada en varias variables la información requerida por dichos comandos.

2.3.1.2 El componente cliente Socket en el Servicio Web

El cliente de Sockets, debe ser acorde al diseño del servidor de Sockets, aunque con el nuevo diseño de este, son prácticamente independientes, pudiendo sustituir fácilmente tanto el cliente como el módulo servidor en el robot, siempre que los comandos enviados sean los mismos.

En el diseño heredado eran mucho más dependientes los módulos cliente y servidor entre si, ya que debían mantener una conexión activa con dos clientes Socket y dos Servidores Socket (uno de recepción y uno de envío) respectivamente.

De forma análoga al Servidor, el cliente conecta con el servidor, envía un comando, recoge la respuesta y cierra la conexión. Si el programa del Robot finaliza por cualquier causa y se intentan enviar comandos, al no poder establecer la conexión con el servidor de Sockets, se muestra por consola el mensaje de error pertinente, con lo que el usuario podrá avisar a un técnico de la anomalía en el funcionamiento del sistema, utilizando por ejemplo, la funcionalidad de alertas por correo electrónico o SMS.

2.4 Localización mediante visión artificial

2.4.1 Introducción

Uno de los defectos que presentaba el robot en su versión inicial era que acumulaba un error en el cálculo de la posición mediante los encoders. Esto suponía que cuando llevaba un tiempo en movimiento terminaba

perdiéndose, ya que la posición que calculaba no se correspondía con la posición en la que se encontraba realmente.

Para solucionar este problema existían diversas alternativas. Entre ellas se encuentra la que desarrollamos en nuestro proyecto y que explicamos a continuación.

Se pensó en incorporar una webcam al sistema que fuese complementaria con los encoders en la obtención de la posición y la orientación del robot en cada momento.

Aprovechamos que el techo de las plantas de la facultad y, por tanto, el de la planta que alberga el museo y donde se ubicará en un futuro próximo nuestro robot está formado por placas que forman rectángulos, como podemos ver en la figura 15:

Figura 15 Fotografías del techo de la FDI

Fijándonos en las líneas que delimitan las distintas placas es posible calcular la posición y la orientación en la que se encuentra el robot en cada momento a partir de una posición y orientación iniciales de referencia.

En la actualidad existen multitud de robots móviles que están dotados de visión artificial. Sin embargo, la idea de localizarse con la ayuda de líneas en el techo es innovadora, lo que hizo que tuviésemos que llevar a cabo un gran trabajo de investigación y desarrollo.

Pensamos que la mejor opción para la elaboración de esta parte del proyecto era diseñar e implementar un sistema autónomo de navegación. El hecho de que el sistema sea autónomo tiene la ventaja de que se puede utilizar de forma independiente y puede ser integrado en otras arquitecturas en las que resulte útil.

Además, las distintas funciones que utiliza este sistema se ofrecen en forma de API para poder ser usadas de forma independiente por cualquiera que lo considere oportuno. De esta forma no se tiene que utilizar obligatoriamente la aplicación completa, sino que puede utilizarse sólo la parte que sea necesaria. Esto hace que este sistema sea bastante versátil y adaptable a distintos entornos con distintas finalidades.

Una vez que tuvimos claro cuales eran las funcionalidades que debía ofrecer este sistema de localización, el siguiente paso fue pensar que lenguaje utilizaríamos para resolverlo. La elección fue C/C++, ya que es uno de los lenguajes más potentes en la actualidad, además de ser un lenguaje con el que se pueden llevar a cabo tareas de bajo nivel, algo que nos vendrá muy bien para el acceso a la webcam. Otra de las razones para elegir este lenguaje, fue que los anteriores proyectos realizados sobre el robot también lo utilizaban.

Después sentar unas bases iniciales, ya podíamos comenzar con el diseño del sistema. Para mayor claridad y sencillez, dividimos el problema en tres partes bien diferenciadas:

- Captación de la imagen
- Preprocesamiento e interpretación de la imagen
- Cálculo de la posición y la orientación

En los apartados siguientes se especifican con mayor detalle cada una de ellas.

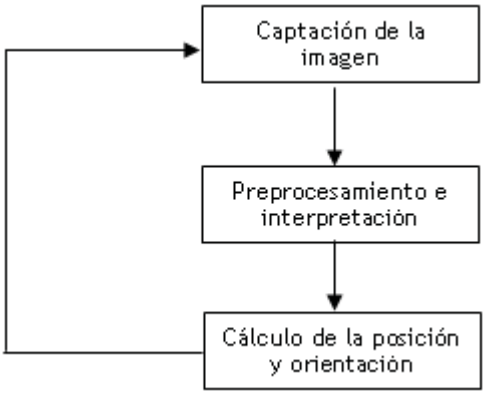


Figura 16 Fases para la obtención de la posición y orientación del robot

2.4.2 Captación de imágenes

Esta es la tarea de más bajo nivel dentro de la visión por computador. Consiste en capturar imágenes con la cámara y almacenarlas en memoria con una determinada estructura (digitalización) para que posteriormente puedan ser tratadas por los algoritmos necesarios.

Figura 17 Obtención de una fotografía a partir de la webcam

En nuestro caso, el objetivo es obtener una imagen cada cierto periodo de tiempo a través de la webcam. Este tiempo no debe ser demasiado pequeño, ya que se sobrecargaría demasiado el sistema. Sin embargo, tampoco puede ser excesivo, porque se podría llevar a perder alguna línea y que el cálculo de la posición y la orientación fuese incorrecto. Por ello, pensamos que la mejor forma de resolverlo, es que este periodo fuese configurable mediante un archivo XML. Así, el sistema es más flexible y robusto y dependiendo de la utilidad que se le quiera dar puede ser configurado de una u otra forma.

El siguiente paso es almacenar la imagen en memoria con una estructura adecuada para que pueda ser procesada. Es importante no olvidarse de eliminarla cuando ya no sea necesaria, pues de lo contrario se podría sobrepasar la capacidad de almacenamiento de la memoria y podrían aparecer problemas.

Una vez tenemos la imagen en memoria el siguiente paso es la aplicación de diversas técnicas para tratar la imagen y conseguir nuestro objetivo. Lo explicamos en el siguiente apartado.

2.4.3 Preprocesamiento e interpretación de imágenes

Como ya comentamos anteriormente, nuestro sistema se basa en localizar las líneas que se encuentran en el techo para, a partir de ellas, calcular la posición y la orientación en que nos encontramos. Para conseguir nuestro objetivo es necesario aplicar a las imágenes técnicas de procesamiento tales como filtros, detectores de bordes, cálculos de histogramas,…

Al final de esta etapa, tenemos que tener una estructura donde se almacenen todas las líneas rectas que aparecen en la imagen para que en etapas posteriores seamos capaces de deducir la posición y la orientación actuales.

La visión por computador y más concretamente el tratamiento de imágenes digitales es un campo muy amplio y poco tratado a lo largo de la titulación. Es por ello que tuvimos que llevar a cabo una gran labor investigadora de recopilación de información para poder llegar a proponer una solución a este problema.

En un principio, cuando tuvimos unos conocimientos suficientes para diseñar una primera solución al problema decidimos que para llegar a ella teníamos que seguir los siguientes pasos:

- Convertir la imagen a escala de grises. Este paso es necesario porque las imágenes en color utilizan 3 matrices para su representación (R, G, B). Cómo en nuestro caso, nos es indiferente el color, trataremos con una imagen en escala de grises que ocupa menos espacio y es más sencillo su tratamiento.
- Obtener el histograma de la imagen y la varianza asociada a éste (para obtener más información sobre el concepto de histograma consultar Apéndice A) para distinguir entre tipos de imágenes
- Clasificar la imagen según su varianza. Dependiendo del tipo de imagen, los parámetros elegidos para llamar a las siguientes funciones (umbral de borde, tamaño de matriz de suavizado,…) variarán para así poder ajustarse mejor a las propiedades de la imagen. Después de realizar un

estudio exhaustivo con multitud de imágenes tomadas en techos que reunían las características que requiere nuestro sistema llegamos a la conclusión de que existían cuatro tipos principales de imágenes según la varianza de su histograma:

- Varianza <= 250: Imágenes con poco contraste y que al ser tomadas no les afecta la luz, es decir, no están debajo de ninguna fuente de luz ni lo suficientemente cerca como para que ésta pudiera ser tenida en cuenta.

Figura 18 Fotografía con varianza <= 250

- 250 < Varianza <= 2000: Imágenes tomadas cerca de una ventana o de alguna fuente de luz que hace que aparezcan partes de la imagen (en los extremos) muy claras debido a ello. Esto hace que la varianza sea mayor.

Figura 19 Fotografía con 250 < varianza <= 2000

- 2000 < Varianza <= 6900: Imágenes tomadas casi debajo de alguna fuente de luz.

Figura 20 Fotografía con 2000 < varianza <= 6900

- Varianza > 6900: Imágenes tomadas debajo de una fuente intensa de luz.

Figura 21 Fotografía con varianza > 6900

- Aplicar algoritmos de suavizado a la imagen para reducir el ruido y así facilitar la labor del algoritmo detector de bordes.
- Modificar el brillo y el contraste de la imagen si fuese necesario.
- Aplicar un algoritmo de detección de bordes.
- Detectar y almacenar las líneas rectas que se encuentran en la imagen mediante la transformada de Hough (se puede encontrar más información sobre la transformada de Hough en el Apéndice A).

Este primer diseño fue abandonado al cabo de un tiempo por los problemas que nos causaba en la implementación. El principal problema fue que era un diseño demasiado dependiente de las características de la imagen por lo que había que distinguir multitud de casos y en función de ellos tratar a cada imagen de una forma u otra. Para ello, había que ser muy fino en la

clasificación de las imágenes porque existen multitud de casos minoritarios que teníamos que tener en cuenta. Además, muchas imágenes presentaban características tan similares que complicaban su distinción. Por eso pensamos retocar el diseño anterior para hacerlo más robusto y menos dependiente de las características concretas de cada imagen. De esta forma el sistema será más sencillo, por lo que si fuese necesaria su ampliación por personas ajenas al proyecto, éstas no tendrían excesivas dificultades en su comprensión. Esto también ayuda a la localización de errores en caso de que se produjesen, sin olvidarnos de que hacen que la aplicación sea más flexible y versátil.

El nuevo diseño se divide en las siguientes partes:

- Convertir la imagen a escala de grises
- Obtener el histograma de la imagen y la varianza asociada a éste para distinguir entre tipos de imágenes. Este paso tiene menos relevancia en este nuevo diseño que en el anterior. Se realiza simplemente para distinguir casos concretos como por ejemplo que nos encontramos debajo de una fuente de luz que podría actuar como baliza absoluta a la hora de calcular la posición y la orientación
- Aplicar algoritmos de suavizado a la imagen para reducir el ruido
- Aplicar un algoritmo de detección de bordes. Para este segundo diseño buscamos un algoritmo que fuese menos dependiente de la imagen concreta y donde no hubiese que modificar apenas sus umbrales para distinguir los bordes de forma correcta. Finalmente nos decantamos por el detector de bordes de Canny (puede conocerse más a fondo el funcionamiento de este algoritmo en el Apéndice A)
- Detectar y almacenar las líneas rectas que se encuentran en la imagen mediante la transformada de Hough. Este paso también se ve influenciado con el nuevo diseño y se hace más sencillo, pues el umbral utilizado será prácticamente el mismo para todas las imágenes.

Asimismo, como ya explicamos en la introducción, para conseguir que nuestro sistema sea lo más independiente posible y que en un futuro pueda ser ampliado o integrado dentro de otros sistemas mayores pensamos en desarrollar un conjunto de funciones básicas para el procesamiento de la imagen que especificamos aquí en forma de API:

- funHistograma (imagen) » Calcula el histograma de una imagen y su varianza asociada.
- funUmbrales (imagen) » Clasifica la imagen en función de su varianza y adapta los valores de los umbrales utilizados en funciones posteriores a la imagen en cuestión.

- funBordes (imagen, umbralCanny1, umbralCanny2) » Forma una imagen con los bordes extraídos de otra mediante el detector de Canny con los umbrales pasados por parámetro.
- funHough (imagen, umbralHough) » Extrae las ecuaciones de las rectas de la imagen mediante la transformada de Hough y las almacena en una estructura.
- funDibujarLineas (imagen, rectas) » Dibuja sobre la imagen las rectas extraídas por la función de Hough.
- funOrientacion (rectas, orientación anterior) » Calcula la orientación en base a las rectas y a la orientación anterior.
- funPosicion (rectas, referenciaAnterior, rectasAnteriores, orientacionActual) » Devuelve una posición calculada a partir de un punto de referencia y una orientación anteriores y las rectas de la imagen actual.
- lineaMasCercanaRobot (rectas, rectaReferencia) » Devuelve la línea mas cercana al robot. Opcionalmente se puede pasar una referencia a otra recta, con lo cual se devolverá la siguiente recta más cercana al robot a partir de esa referencia, formando con ella un ángulo de aproximadamente 90º.
- obtenerRectaSimilar (rectas, rectaReferencia, error) » Devuelve una recta similar a la de referencia, teniendo en cuenta el error pasado por parámetro.
- obtenerCoeficientesRecta (recta) » Devuelve los coeficientes de la recta de la forma Ax+By+C=0, a partir de los coeficientes polares θ y ρ.
- bajoFoco (focos, X actual, Y actual) » Devuelve un booleano que indica si estamos bajo un foco o no, y las coordenadas absolutas del foco de luz más cercano al robot.
- setPosicion (X conocida, Y conocida) » Configura la posición actual del robot a las coordenadas pasadas en los parámetros, reseteando la fiabilidad del algoritmo del cálculo de la posición al 100%.
- funBrilloContrast (imagen, brillo, contraste) » Modifica el brillo y el contraste de una imagen.
- distancia_euclidea(punto1, punto2) » Calcula la distancia Euclídea entre dos puntos del plano.

2.4.4 Cálculo de la posición y la orientación

Los objetivos de este apartado son calcular la orientación y la posición en la que nos encontramos respecto a la posición inicial de la que partimos. Para conseguirlo tenemos que analizar las líneas rectas que hemos obtenido como resultado en el apartado anterior.

Esta es la función principal de nuestro sistema. Todos los pasos anteriores eran necesarios para que esta fase se pudiese llevar a cabo con garantías.

2.4.4.1 Cálculo de la orientación

La orientación es la forma en la que conocemos el espacio que nos rodea, guiándonos por unos puntos ya conocidos que actúan como referencia.

Es muy importante conocer la orientación en un sistema de navegación. En nuestro caso particular, utilizaremos un conjunto de líneas rectas extraídas de una imagen para realizar este cálculo. Este conjunto de líneas rectas las obtenemos del modo indicado en el apartado previo. También es indispensable conocer la orientación anterior, para poder calcular la nueva orientación en referencia a ésta.

La orientación devuelta por este sistema, siempre utiliza una orientación de referencia inicial y es que el sistema se encontrará en orientación 0º nada más iniciarse, a no ser que se le introduzca otro valor. De esta forma, la orientación del sistema va a estar siempre referida a esta orientación inicial. Si el móvil que utilice nuestro sistema gira hacia la derecha, y por lo tanto las líneas hacia la izquierda, la orientación irá disminuyendo hasta alcanzar el mínimo de -180º y si gira hacia la izquierda, girando las líneas hacia la derecha, la orientación aumentará hasta alcanzar un máximo de 180º. En vez de esto, podíamos haber decidido que la orientación estuviese comprendida entre 0º y 360º, pero decidimos hacerlo del modo anterior.

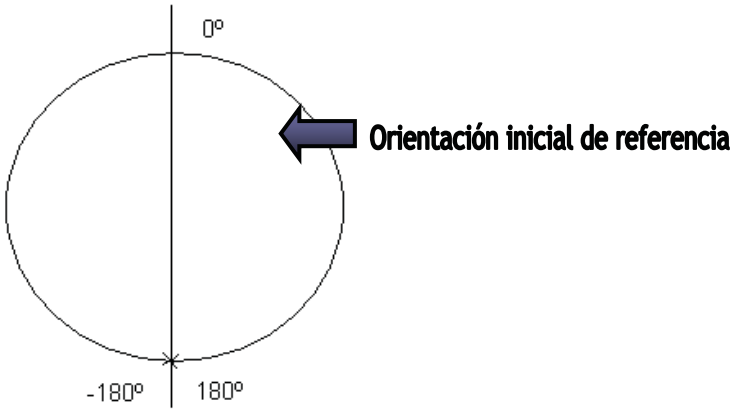


Figura 22 Orientación inicial de referencia

Aprovechamos que la transformada de Hough devuelve la ecuación de las rectas en forma paramétrica (Apéndice A) y utilizamos θ para realizar nuestro cálculo.

Hay que tener en cuenta, que cada recta tiene dos posibles orientaciones con respecto a la orientación inicial, una positiva (θ) y otra negativa -(180º - θ).

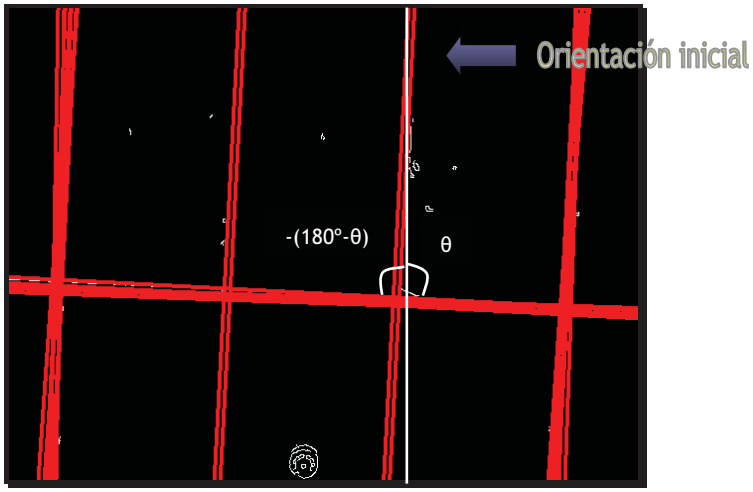


Figura 23 Orientación inicial con transformada de Hough

Por tanto, para calcular la orientación con respecto a la orientación anterior, se recorren todas las líneas devueltas por la transformada de Hough y se calcula la diferencia entre el valor absoluto de la orientación de

cada línea y el valor absoluto de la orientación anterior. La orientación de la línea que minimice esta distancia será la escogida como nueva orientación. Esta es una buena solución siempre que el móvil que usa este sistema no gire a una velocidad excesiva, ya que entonces la orientación actual no será la que esté a una distancia mínima con respecto a la orientación anterior. Si esto ocurriese, se podría aumentar la frecuencia con la que son tomadas las capturas con la cámara quedando resuelto el problema. Hay que tener cuidado con el paso de orientación negativa a positiva y viceversa. Para ello es imprescindible darnos cuenta de que existen tres casos fundamentales:

- -10 <= Orientación anterior <= 10 ó 170 <= Orientación anterior <= -170: En este caso, la orientación podría cambiar de signo, por lo que es necesario comprobar la diferencia de la orientación anterior con θ y con (180º - θ)
- Orientación anterior no se encuentra en el caso anterior y además es > 0: En este caso, sabemos que la nueva orientación va a seguir siendo positiva, por lo que únicamente se realiza la diferencia entre θ y la orientación anterior
- Orientación anterior no se encuentra en el primer caso y además es < 0: En este caso, sabemos que la nueva orientación va a seguir siendo negativa, por lo que únicamente se realiza la diferencia entre (180º – θ) y la orientación anterior

La razón fundamental por la que decidimos que la orientación estuviese comprendida entre 0º y +/-180º en vez de entre 0º y 360º, es que resultaba más sencillo realizar el paso por 0º. Esto es debido a que la diferencia entre los ángulos próximos a 360º con los ángulos próximos a 0º es muy elevada y el método de calcular la orientación en función de la distancia mínima a la orientación anterior no funcionaría directamente, sino que habría que realizar algún tipo de conversión.

2.4.4.2 Cálculo de la posición

Como podemos observar, la cámara toma capturas del techo de la habitación en la que se encuentre el robot. La visión de alto nivel de nuestro sistema se ha diseñado de modo que se aproveche la cuadricula de líneas obtenidas con la transformada de Hough como si fuera la cuadricula formada por unos ejes de coordenadas.

De este modo, tomando puntos de referencia y sumando incrementos en X y en Y, podremos tener las coordenadas absolutas de la habitación, en las que se encuentra el Robot en cada momento.

Es necesario por lo tanto, definir un sistema de coordenadas fijas en el mundo en que el móvil se desenvuelve. La representación del mapa que se hacía en el proyecto anterior ya definía un sistema de coordenadas. Como vemos en la figura 24, el sistema de coordenadas del mapa que almacena el Robot es "levógiro", es decir, si giramos desde X hacia Y, el ángulo de giro disminuye.

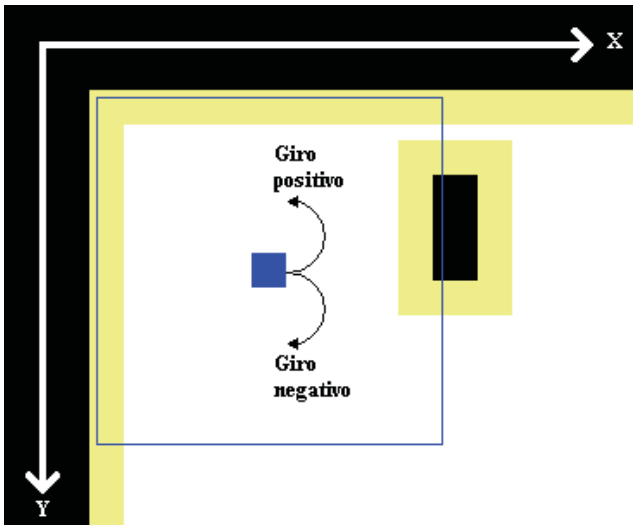


Figura 24 Sistema de coordenadas absoluto

Debemos tener en cuenta también, que el Robot tiene su propio sistema de coordenadas que normalmente no coincidirá con el eje de coordenadas fijo. La figura 25 muestra la posible captura de una imagen en la cuadrícula del techo en un mapa, y como los ejes móviles o del Robot (en rojo), se encuentran girados con respecto a los ejes fijos (en negro). En este caso el robot se encuentra con orientación absoluta 0º.

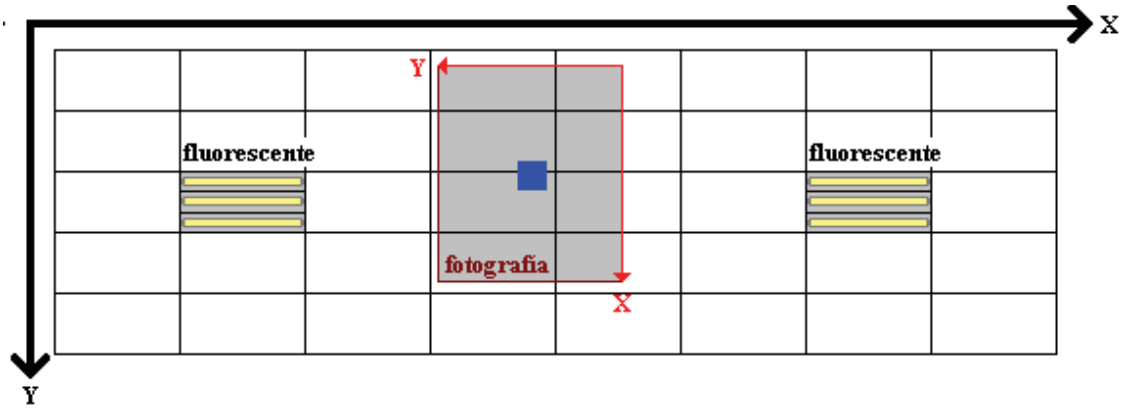


Figura 25 Sistema de coordenadas relativo al robot

La figura 26 muestra una fotografía real, tomada durante la realización de las pruebas:

Figura 26 Fotografía real con coordenadas incluidas

El sistema de posición se basa en las coordenadas absolutas del mapa, pero las coordenadas que se pueden obtener de la fotografía son relativas a la posición y orientación del robot. Para ello debemos aplicar una matriz de transformación a las coordenadas relativas, para pasarlas a los ejes fijos.

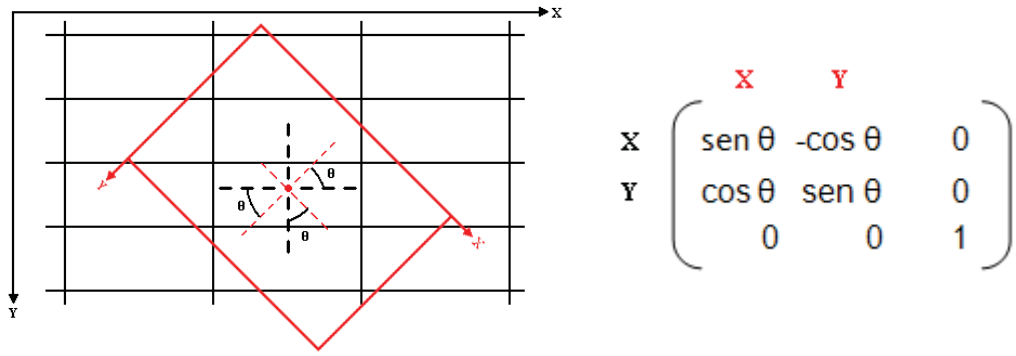


Figura 27 Matriz de transformación de coordenadas

Se hace evidente la importancia de la orientación "θ", calculada previamente mediante el diseño del apartado anterior.

Como comentábamos en el primer párrafo de este apartado, es esencial obtener una referencia inicial. En la primera captura tomada, se escoge un punto de referencia, que será el punto de corte entre dos rectas, más cercano al centro de la foto, ya que suponemos que es donde se encontrará el robot.

En la figura 28, se buscará el mismo punto que se tomo como referencia anteriormente, para así calcular el incremento que se ha producido en la posición, pudiendo así sumarlo a la posición absoluta.

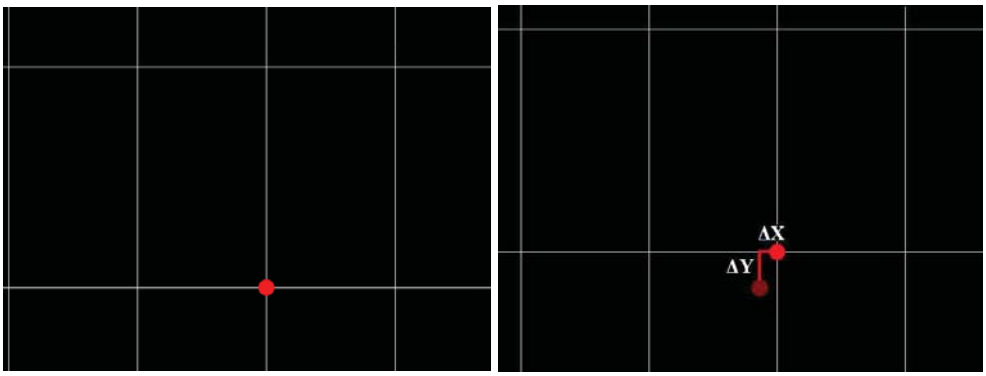


Figura 28 Incrementos en la posición

Como el Robot avanza en el espacio, puede darse la situación en la que no se pueda realizar el seguimiento de las rectas que produjeron el punto de corte anterior, ya que pueden encontrarse fuera de la fotografía. Llegados a este punto debemos renovar el punto de referencia. Para ello buscamos

de nuevo el punto de corte de dos rectas más cercano al Robot, y despreciamos el incremento que se haya podido provocar en el cambio del punto de corte.

Al despreciar este incremento cuando cambiamos el punto de referencia, se comete un error, con lo cual la fiabilidad de la medición disminuye. Nos vemos obligados por lo tanto a llevar una medición de fiabilidad.

Puede ocurrir, que la transformada de Hough no consiga extraer rectas de una imagen debido a unas condiciones de iluminación peculiares, y que por lo tanto, no existan líneas con las que calcular una nueva referencia, o hacer un seguimiento de una referencia anterior. En este caso la fiabilidad disminuirá en mayor medida que cuando se pierde un punto de referencia, por no aparecer las rectas de las que surgió en la imagen.

Si se da una situación de error como las comentadas en los párrafos anteriores, no se puede sumar ningún incremento a la posición calculada por el algoritmo, y esto puede ocurrir durante varios segundos. Cuando se reanuda el cálculo de la posición tras solucionarse esta situación, se debe leer la posición de una fuente externa para partir de un nuevo punto inicial más fiable y no acumular el error cometido por la pérdida de incrementos. En nuestro caso, se obtendrá la posición calculada por los encoders y la brújula del Robot.

Hemos visto que los errores de cálculo debidos a la pérdida del punto de referencia, pueden hacer que la fiabilidad disminuya hasta ser igual a cero.

En este punto sería interesante la localización de balizas absolutas, con las que renovar la posición real, y por tanto el 100% de fiabilidad para los siguientes cálculos de posición.

El pasillo del museo, tiene fluorescentes repartidos de forma uniforme y conocida. Además sabemos cuándo nos encontramos justo debajo de uno de ellos, ya que la varianza de la imagen es mayor que 6900.

Utilizando la función "*bajoFoco*" podremos conocer si nos encontramos o no justo debajo de un foco, y la posición del foco más cercano.

2.5 Adaptación de comunicación con microcontroladores a USB

2.5.1 Introducción

La arquitectura del proyecto heredado incluía un módulo de nivel intermedio ("Pic_comm") que conectaba un módulo de alto nivel de los motores ("Motores") con el microcontrolador PIC, que llevaba a cabo las órdenes. El esquema es el siguiente:

Figura 29 Conexiones de "Pic_comm" de motores

Además, el mismo módulo "Pic_comm" se comunicaba con varios módulos de sensores:

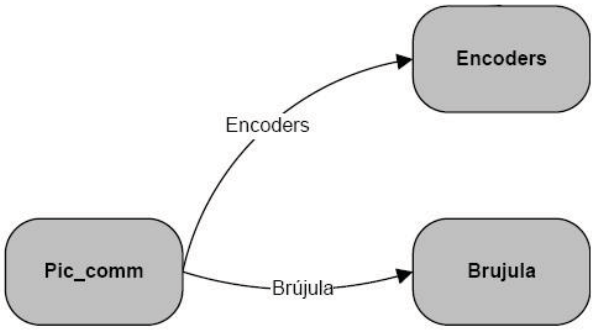


Figura 30 Conexión de "Pic_comm" de sensores

De esta forma, "Pic_comm" realizaba consultas al PIC para obtener información acerca del estado de los sensores, y propagaba esta información a los módulos adecuados mediante los puertos de conexión.

Por lo tanto, el módulo "Pic_comm" utiliza una comunicación bidireccional: por un lado **proporciona** información al PIC para hacer mover

los motores, y por otro **obtiene** datos del PIC sobre los sensores. Esto se aclara con el siguiente gráfico:

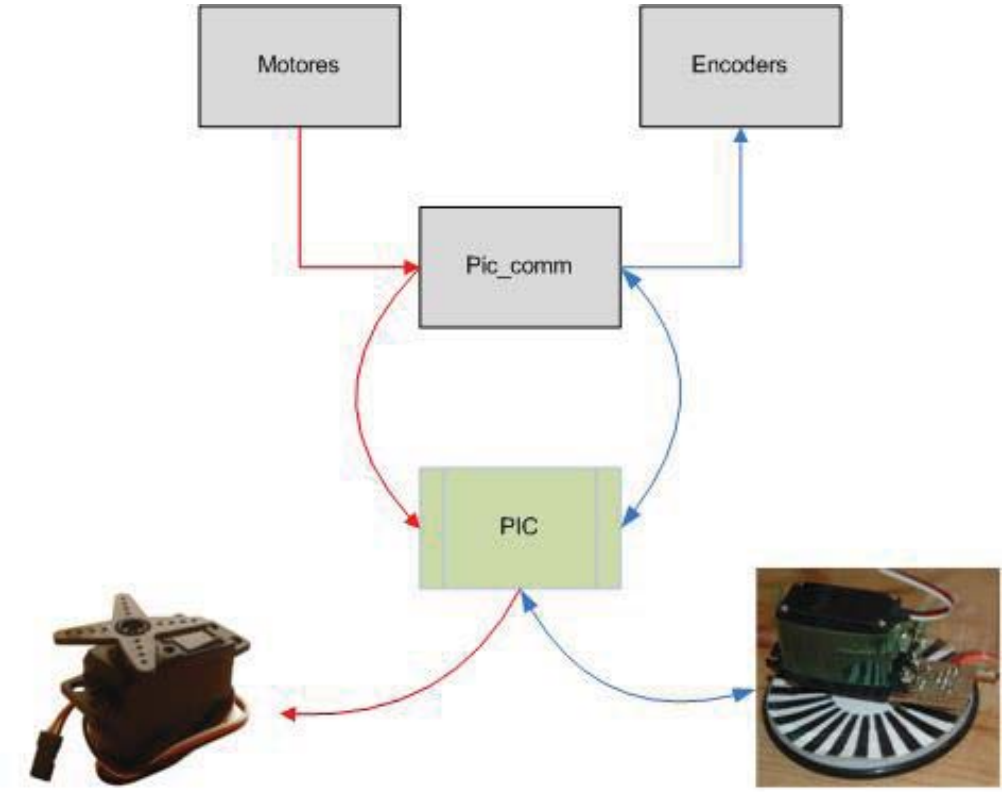


Figura 31 Responsabilidades del módulo "Pic_comm"

Como vemos, del módulo "Motores" surge una orden que, a través del módulo "Pic_comm", llega al PIC, el cual procesa las acciones y las envía al motor (en rojo). Por otro lado, el "Pic_comm" pregunta cuando sea necesario el estado a los encoders a través del PIC. Estos datos se propagan hacia el módulo superior "Encoders" (en azul).

El problema surgía en la comunicación entre el "Pic_comm" y el propio PIC. Al comienzo de nuestro trabajo, esta comunicación se realizaba en serie, y no era demasiado fiable. Por ello, uno de los objetivos potenciales sería cambiar esta comunicación a USB, mucho más robusta.

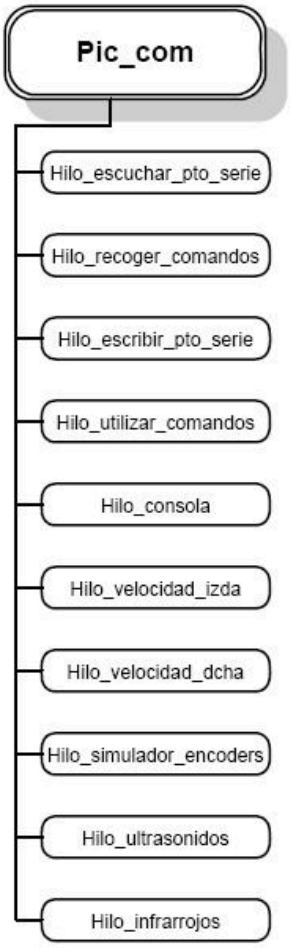
Este posible objetivo se hizo realidad cuando, de forma desconocida, el PIC dejó de funcionar, y como consecuencia el robot dejó de andar. Este problema ajeno a nosotros hizo que nos pusiéramos a trabajar junto a otras

personas en la adaptación a USB para conseguir que el robot caminara de nuevo.

Nuestro trabajo principal se enfocó en modificar el módulo "Pic_comm" y todas las configuraciones necesarias del proyecto del robot para adaptar todo a este nuevo tipo de comunicación. Esto supuso un análisis profundo del complejo funcionamiento de este componente concurrente, que pasamos a describir a continuación:

2.5.2 El módulo "Pic_comm"

2.5.2.1 Diseño de hilos



Gran parte del trabajo en este módulo se realiza a través de hilos. A continuación mostramos la estructura de los hilos concurrentes que realizan las distintas tareas:

Hilo_escuchar_pto_serie: Hilo que escucha el puerto serie para controlar el overlapping y almacenar en el buffer los caracteres recibidos.

Hilo_recoger_comandos: Hilo que toma los caracteres del buffer cuando han sido completamente leídos del puerto y los traduce a un comando.

Hilo_escribir_pto_serie: Hilo que escribe en el puerto serie los comandos a enviar al PIC.

Hilo_utilizar_comandos: Hilo que recoge de la cola de comandos recibidos del PIC y los ejecuta enviando los datos a los módulos correspondientes.

Hilo_consola: Hilo que carga una consola interactiva para enviar comandos al PIC

Hilo_velocidad_izda: Hilo que recoge peticiones de cambio de velocidad en la rueda izquierda procedentes del módulo Motores y los guarda en la cola de comandos a enviar al PIC.

Hilo_velocidad_dcha: Hilo que recoge peticiones de cambio de velocidad en la rueda derecha procedentes del módulo Motores y los guarda en la cola de comandos a enviar al PIC.

Hilo_simulador_encoders: Hilo que simula el avance del robot mediante pulsos de encoders lo que permite hacer pruebas de movimiento sin necesidad de tener el

robot conectado al PC.

Hilo_ultrasonidos: Hilo que recoge peticiones de barrido de ultrasonidos y los almacena en la cola de comandos a enviar al PIC.

Hilo_infrarrojos: Hilo que recoge peticiones de barrido de infrarrojos y los almacena en la cola de comandos a enviar al PIC.

Sin entrar en más detalle sobre todos estos hilos (se puede obtener más información consultando la memoria del proyecto de SSII que referimos en la Bibliografía), pasamos a hablar de su funcionalidad.

2.5.2.2 Funcionalidad

Este apartado se creó tras la fase de análisis del módulo, en la que se estudió con detenimiento su comportamiento y su utilidad. De esta forma acotaríamos el problema del cambio de comunicación, sabiendo dónde realizar los cambios pretendidos. Las conclusiones del análisis se escriben a continuación:

Este módulo es el encargado de hacer llegar las órdenes enviadas desde los módulos de más alto nivel hacia el PIC, que es el que en última instancia realiza las operaciones con los sensores y los motores.

Para ello debe conocer en qué puerto se encuentra cada PIC, ya que al disponer de varios microcontroladores, cada uno ocupándose de una tarea, necesitamos varios puertos y conocer unívocamente qué PIC se encuentra conectado a cada uno. La conexión hacia los PICs se realizaba en el proyecto heredado a través de un **intérprete serie** grabado en el firmware de los PICs de forma que, aunque la conexión física al ordenador es a través de USB, el intérprete hacía creer al sistema operativo que en realidad se trata de un puerto serie (al cual hubo que proporcionarle unos drivers previamente modificados).

Este sistema es el que tratamos de modificar en el proyecto actual, ya que **la comunicación serie a veces se perdía**, y había que reenviar los mismos datos varias veces para asegurar la consistencia. Además, otro de los inconvenientes que presenta dicho interprete es que **sólo permitía el envío de cadenas de caracteres** a través del puerto virtual, de forma que, si queríamos enviar datos numéricos, había que hacer una conversión a caracteres, enviarla a través del puerto, y en el receptor volver a realizar la conversión a datos numéricos.

Por otro lado, este módulo también se encarga de transmitir a los módulos de más alto nivel los datos proporcionados por el PIC, de modo que la comunicación es bidireccional; por ejemplo, si el PIC envía los datos registrados por un ultrasonido, este módulo de comunicación con el PC (Pic_comm), sabe que dicho valor ha de reenviarlo al módulo de más alto nivel encargado de controlar los sensores. Al igual que realiza transmisiones, también se encarga de recibir todos los datos procedentes del PIC; es decir, la información de los encóders, información de los ultrasonidos, infrarrojos y brújula, así como comandos informativos para saber si todavía existe comunicación con el PIC y saber si el comando enviado anteriormente ha sido recibido en éste. En este caso se recibiría un comando ACK confirmando la recepción.

En el proyecto heredado, los comandos recibidos por parte de todos los sensores del robot se denominaron *comandos con dato*, por lo que su tratamiento era diferente al de los comandos informativos. En el caso de los comandos con dato, se hacía una división del comando recibido, tomando los 8 primeros caracteres como identificador del comando, mientras que el resto de caracteres eran tratados de manera que serían convertidos a un número entero para su posterior procesamiento en módulos superiores.

Todos estos problemas pasados se pretendieron solucionar mediante el **cambio a comunicación USB**, además de realizando una **modificación importante en la representación de los datos enviados y recibidos**. Estos cambios los pasamos a describir a continuación:

2.5.3 Modificaciones realizadas

2.5.3.1 Proceso de ingeniería

Tras el análisis del problema (diseño e implementación del módulo "Pic_comm" y la comunicación utilizada por el mismo), pasamos a crear un diseño propio a partir de la especificación deseada, para poder así llevar a cabo el cambio propuesto:

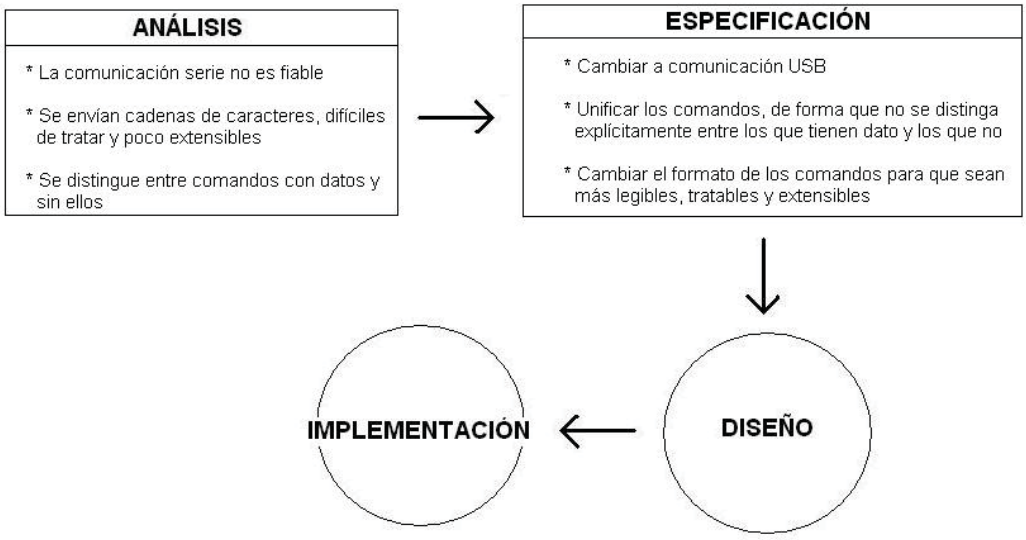


Figura 32 Proceso de ingeniería seguido para modificar el módulo "Pic_comm"

2.5.3.2 Diseño

En este apartado pasamos a dar algunos detalles de alto nivel acerca de las soluciones tomadas en la especificación (ver diagrama del apartado anterior) para llevar a cabo los cambios pertinentes.

2.5.3.2.1 Cambio en la comunicación

La misión principal era cambiar todas aquellas funciones que se comunicaran con el PIC, leyendo o escribiendo de su puerto, para adaptarlas a la nueva interfaz de lectura/escritura USB con la que había sido modificado el firmware del microcontrolador, tanto de motores como de sensores. La figura 33 muestra un diagrama básico de este cambio, en el que podemos observar que la mayor modificación consiste en el uso de la nueva interfaz para comunicación por USB, con la consiguiente adaptación de las funciones y los hilos del módulo "Pic_comm". Los detalles de implementación se ofrecen en el apartado 3.3.1.

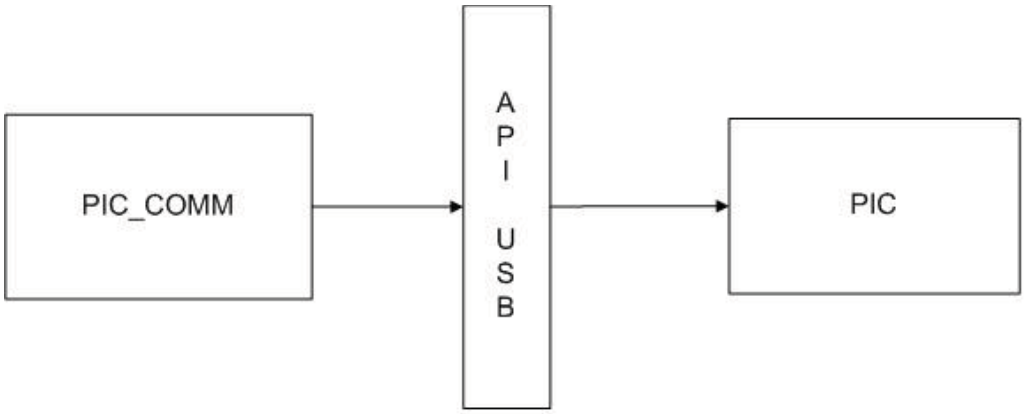


Figura 33 Adaptación a la nueva interfaz USB

2.5.3.2.2 Cambio en la representación de los comandos

Se realizó un cambio importante en la **representación** de la información transmitida del robot al PIC y viceversa. Como el intérprete serie que había en el trabajo anterior sólo permitía la transmisión de caracteres, la mayoría de las órdenes que se enviaban al PIC se hacían en forma de códigos de comando, indicando un carácter de control de comando (@ en caso de tratarse de comandos de control de algún dispositivo) y un código que determinaba el comando.

Por ejemplo, si queríamos recibir la lectura del ultrasonido 1, se enviaba a través del puerto serie el comando " @80|UltraSonido1\0", tras lo cual, el PIC determinaba la acción a realizar, y una vez obtenido el resultado, enviaba el valor registrado a través del puerto al PC.

De igual modo, también se disponía de códigos informativos, como por ejemplo, "@99|ACK\0" que indicaba que el último comando enviado se recibió y ejecutó con éxito, y muchos otros, que determinaban si el comando no era reconocido por el PIC al que se envió, comandos de pruebas ("@55|Testing\0") etc.…

Realizar el cambio a comunicación USB nos permitiría transmitir datos de manera más libre. Por ello, lo más cómodo fue realizar la transmisión de **datos hexadecimales.** Éstos se codificaron de manera que la flexibilidad a la hora de crear o modificar comandos de control era inmensa. Esto, unido a la facilidad de manejo de un solo dato hexadecimal más que de una cadena de caracteres que requería “trocearla” para obtener toda la información que contenía, hacía de esta nueva representación la ideal.

Además, de esta manera unificábamos los comandos que portaban datos con aquellos que no. Vemos este cambio con un gráfico que muestra la diferencia entre representaciones de datos:

Figura 34 Cambio en la representación de los comandos

3 Implementación.

3.1 Control remoto del robot

3.1.1 Aplicación cliente – servidor

3.1.1.1 Servidor de Sockets en el Robot

La implementación del servidor de Sockets, se ha realizado siguiendo la arquitectura modular de la aplicación del Robot.

Para ello se ha creado un módulo nuevo "Servidor" heredando la clase "Modulo" e implementando los métodos virtuales pertinentes.

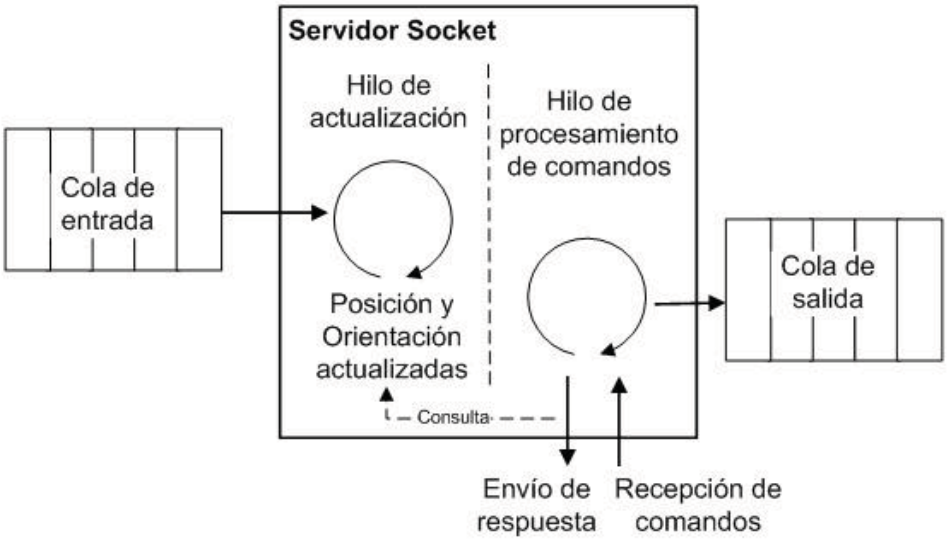


Figura 35 Aspecto del Servidor Socket

El módulo tiene una conexión entrante y una conexión saliente, por lo tanto, una cola de entrada y otra de salida respectivamente.

La cola de salida envía los comandos recibidos del servicio Web al módulo "Interprete WiFi", que interpreta los comandos y los envía al cerebro.

La cola de entrada sirve para que el módulo "Robot Posición" escriba la posición y orientación en cada momento, y el modulo "Servidor" mantenga actualizadas las variables de clase asociadas.

El módulo "Servidor" contiene dos hilos activos en todo el ciclo de ejecución. Uno de ellos se encarga de monitorizar el servidor de Sockets, recogiendo los comandos que llegan del servicio Web y escribiéndolos en la cola de salida. El otro hilo recoge los datos del puerto de entrada y se encarga de mantener actualizadas en el "Servidor" la posición y orientación.

Para gestionar la comunicación, se ha hecho uso de la librería de Sockets que se utilizó en el proyecto anterior, pero ampliando su funcionalidad para recibir y enviar cadenas de texto con una longitud determinada.

Se ha realizado esta ampliación de la librería, ya que se debía tratar la incompatibilidad de tipos de datos entre Java y C++.

A la librería, se han incorporado los siguientes métodos:

- int Escribe_Socket (char *Datos, int Longitud)

  Escribe en el socket una cadena de caracteres "Datos" de longitud "Longitud".

- int Lee_Socket (int fd, char *Datos, int Longitud)

  Lee del socket una cadena de caracteres "Datos" de longitude "Longitud".

La explicación de la incompatibilidad de los tipos de datos entre Java y C++, es sencilla. El código C++ compilado se ejecuta directamente sobre el procesador (Intel en nuestro caso), y su organización de memoria es "*Little endian*" (los bytes menos significativos ocupan la direcciones más bajas de memoria). A diferencia de Intel, la máquina virtual de Java utiliza una organización de memoria "*Big endian*" (los bytes menos significativos ocupan la direcciones más altas de memoria).

Por lo tanto, se debe traducir la representación de los datos recibidos o enviados a un formato común utilizando el formato estándar de red.

El formato estándar de red utiliza la organización "Big Endian", al igual que Java, con lo cual, en el servidor de Sockets de C++ se deben invertir los bytes antes de ser recibidos o enviados.

Afortunadamente, tanto en C de Linux como de Windows, tenemos la familia de funciones **htonl()**.

- htonl() pasa un entero de formato hardware a formato red (Hardware TO Network).
- ntohl() pasa un entero de formato red a formato hardware.

El tamaño de los tipos de datos de C++ no difiere del estándar de Red, pero en Java, la representación de caracteres con UNICODE, de dos bytes, difiere del estándar de red (un byte). El tratamiento de este problema lo explicaremos en la parte cliente.

3.1.1.2 Servicio Web Java

3.1.1.2.1 Introducción

La arquitectura que se definió (Cliente – Servidor), y el carácter esporádico de los datos y peticiones que debían enviarse al Robot, hizo que nos decidiéramos por la utilización del compendio de tecnologías conocido como AJAX (Asynchronous JavaScript And XML). Para conocer más sobre esta tecnología, diríjase al apéndice E.

Resulta un tanto engorroso programar y depurar código JavaScript directamente, ya que no existen buenos depuradores en la actualidad, no obstante, Google ha desarrollado un framework para aplicaciones Web con tecnología AJAX denominado "GWT" (Google Web Toolkit). Este framework, permite generar aplicaciones con la arquitectura Cliente – Servidor, enteramente escritas en Java. Durante la construcción del proyecto, GWT preprocesa el código de la parte cliente traduciéndolo a JavaScript.

GWT provee a la parte servidor de una interfaz de llamadas asíncronas, para la utilización de sus servicios por parte de los diferentes clientes.

Este hecho, unido a la facilidad de depuración de la parte cliente, ya que queda enteramente escrita en Java, nos hizo decidirnos por la utilización de esta tecnología para la implementación del servicio Web, por lo tanto, se implementaron las capas que comentamos a continuación.

3.1.1.2.2 Parte Cliente – Capa de Presentación

Para la parte cliente, se ha hecho utilización enteramente de la arquitectura provista por GWT.

Toda la interfaz está basada en contenedores y widgets, con lo cual se ha hecho utilización de un patrón de diseño "*Composite*".

Se ha creído conveniente la separación de los oyentes de la interfaz, ya que estos principalmente son quienes realizan las peticiones asíncronas al servidor, es decir, actúan de capa intermedia entre la interfaz gráfica y el servidor, con lo cual podríamos hablar también de un MVC con mediador, quedando el modelo encapsulado por el servidor, siendo la vista la interfaz basada en paneles y el controlador – mediador los oyentes agrupados en el paquete "listeners".

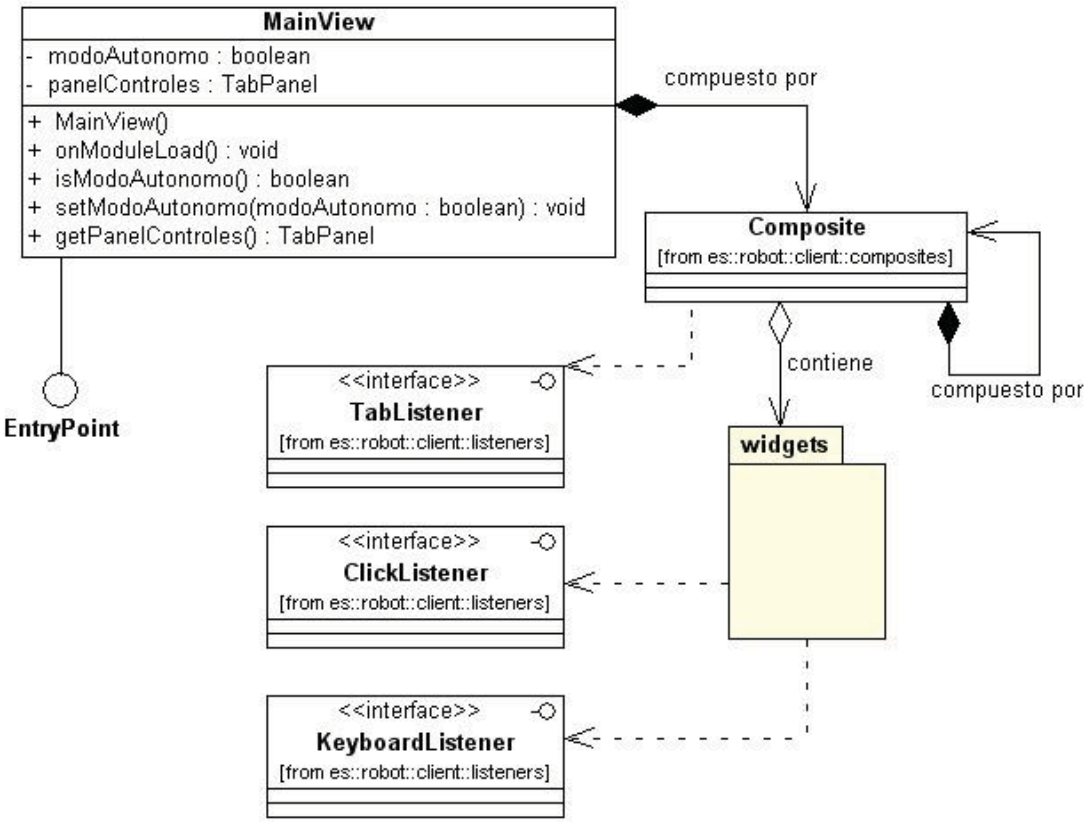


Figura 36 UML de la parte Cliente

Cabe destacar la importancia de la clase principal "MainView", que implementa la interfaz "*EntryPoint*", pilar de la arquitectura de la parte cliente en GWT, ya que es el contenedor raíz.

Una parte muy importante de la parte cliente es la generación de comandos para ser enviados de forma asíncrona al servidor, para que este a su vez los envíe vía Socket al Robot y más tarde devuelva una respuesta a la Interfaz. Ya que para solicitar una misma acción, siempre se debe enviar un

mismo comando y hay una cantidad limitada y fija de estos, se ha implementado una factoría de prototipos de comandos.

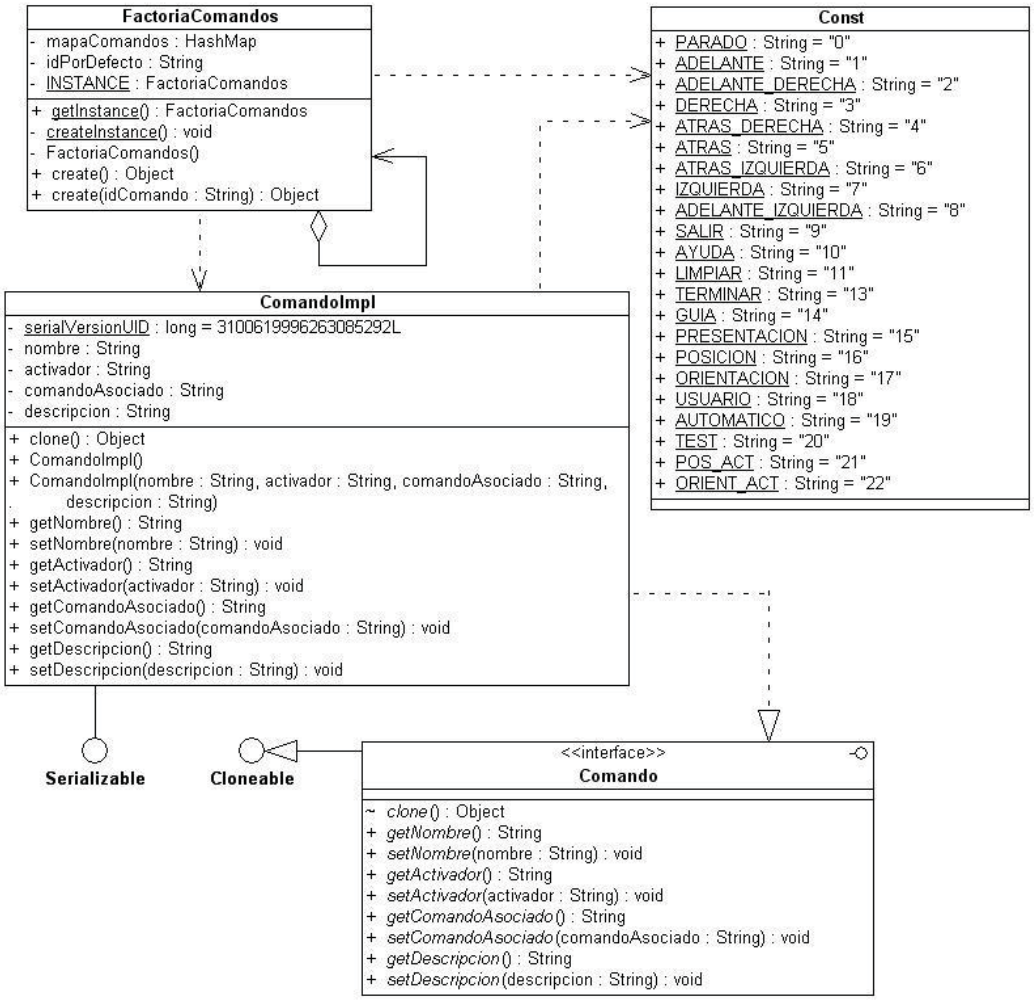


Figura 37 UML de Factoría de Comandos

Por último comentar que se ha utilizado una clase que implementa un patrón "Singleton" para gestionar posibles errores en la comunicación, y traducir estos a la interfaz, cargando el panel de controles del modo autónomo por defecto, ya que es este modo en el que se inicia la aplicación del robot. Se ha implementado así ya que son varios los oyentes que

realizan peticiones asíncronas, y resulta cómodo el acceso a una clase única que gestione los errores.

Al ser la clase "*ForzarAutonomo*" un Singleton, se ha sincronizado la constructora y el uso de los métodos públicos, con el fin de evitar instanciación y utilización múltiple de los recursos de la clase, ya que la interfaz no se bloquea al realizar las peticiones, al ser estas asíncronas.

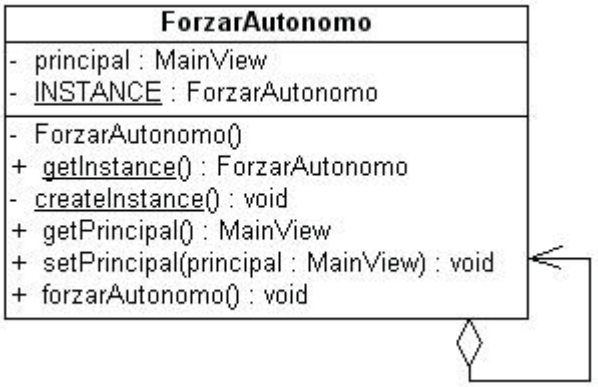


Figura 38 Singleton "ForzarAutonomo"

Siendo la responsabilidad de esta clase la de gestionar los errores de comunicación de los sockets reportados por el servicio Web, es usada por los distintos oyentes de la interfaz que realizan la comunicación asíncrona con el servicio.

3.1.1.2.3 Parte Servidor – Capa de Negocio y Acceso a Datos

En este apartado, podemos hablar de cuatro subcomponentes principales, aparte de la implementación de la interfaz para el servicio asíncrono proviso a los clientes, que gestiona las llamadas del cliente a dichos subcomponentes.

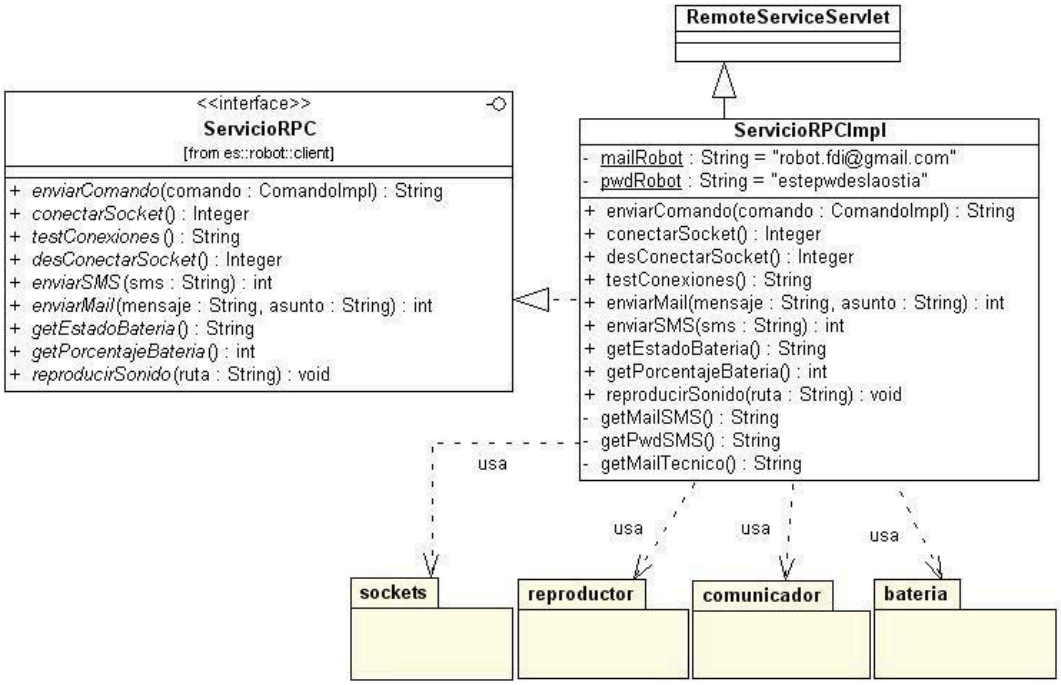


Figura 39 UML de la parte Servidor

Estos subcomponentes se detallan a continuación:

3.1.1.2.3.1 Cliente de Sockets

El Cliente de Sockets se compone de tan sólo dos clases:

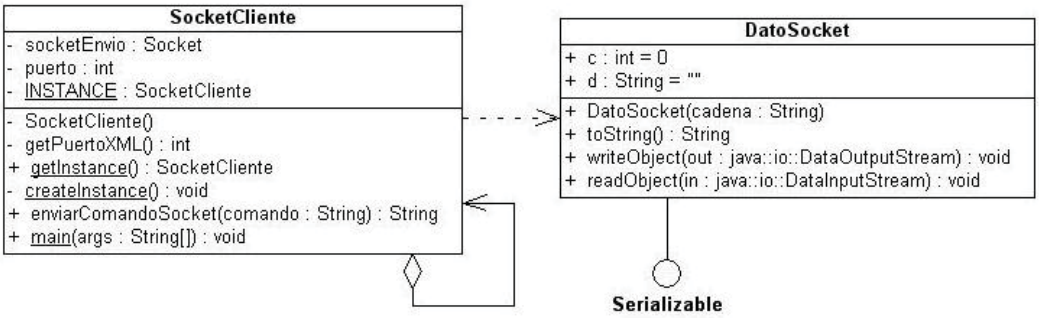


Figura 40 UML del cliente de Sockets

Ya que solo tenemos un cliente que debe ser "invocado" desde la interfaz de servicios asíncronos y por lo tanto ser accesible por varios clientes simultáneamente, se ha creído conveniente realizar la implementación con un patrón de diseño Singleton con constructora y métodos públicos sincronizados.

Cuando se desea enviar un comando, se debe utilizar el método "enviarComandoSocket (String comando)". Como podemos observar, se pide un parámetro de tipo "*String*", que es lo que finalmente se enviará al Robot. Este hecho hace que se pueda reutilizar el cliente en cualquier servicio Web o aplicación, ya que no depende de la implementación del sistema de comandos; tan solo se deben enviar comandos con el mismo formato.

En cuanto a la incompatibilidad del tipo de dato "*char*" de Java con el formato de red estándar, puesto que Java utiliza dos bytes, se ha optado por hacer que sea el cliente quien convierta esos caracteres a un único byte antes de enviar y los reconvierta a dos cuando los recibe. La clase String de Java tiene métodos que permiten realizar esta acción.

3.1.1.2.3.2 Control de la Batería

Para la implementación de esta funcionalidad, se reutilizó un componente diseñado precisamente para poder adaptarlo a cualquier arquitectura. Su especificación y diseño se explican en el punto 2.3.3.2, mientras que los detalles de implementación se ofrecen en el punto 3.1.2.1. Para consultar los detalles de **configuración por XML** en esta aplicación, es necesario leer el apartado 3.1.2.3.

3.1.1.2.3.3 Servicio de Comunicación

Para este servicio se utilizó el componente "Comunicador", diseñado de igual forma para poder reutilizarlo e integrarlo en cualquier sistema de manera fácil. Su especificación y diseño se puede consultar en el punto 2.3.3.3, mientras que los detalles de implementación se pueden ver en 3.1.2.2.

Para consultar los detalles de **configuración por XML** en esta aplicación, es necesario leer el apartado 3.1.2.3.

3.1.1.2.3.4 Reproducción de Sonidos

3.1.1.2.3.4.1 Funcionalidad

Este componente también es completamente reutilizable e integrable, y su funcionalidad viene dada por la implementación de la siguiente interfaz:

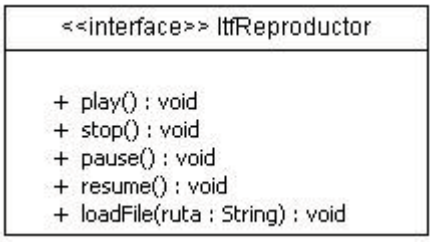


Figura 41 Interfaz "ItfReproductor"

Este componente se utiliza para reproducir sonidos en la interfaz gráfica al pulsar ciertos botones, lo que la hace más intuitiva.

A continuación pasamos a definir cada una de las operaciones:

- play
  - Entradas: ninguna
  - Salidas: ninguna
  - Efecto: reproduce el sonido que ha sido previamente cargado
- stop
  - Entradas: ninguna
  - Salidas: ninguna
  - Efecto: detiene completamente el sonido que ha sido previamente cargado
- pause
  - Entradas: ninguna
  - Salidas: ninguna
  - Efecto: detiene el sonido que ha sido previamente cargado en el instante de reproducción
- resume
  - Entradas: ninguna
  - Salidas: ninguna
  - Efecto: vuelve a reproducir el sonido, que ha sido previamente cargado y pausado, en el punto donde se detuvo

- loadFile
  - Entradas: un String con la ruta donde se halla el fichero mp3, wav o ogm
  - Salidas: ninguna
  - Efecto: carga el archivo de sonido en el componente, permitiendo la manipulación del mismo a través de las operaciones anteriores

3.1.1.2.3.4.2 Implementación

El diagrama de clases del diseño es muy básico: consta únicamente de una clase que implementa la interfaz recién definida, ayudándose de la API de una librería.

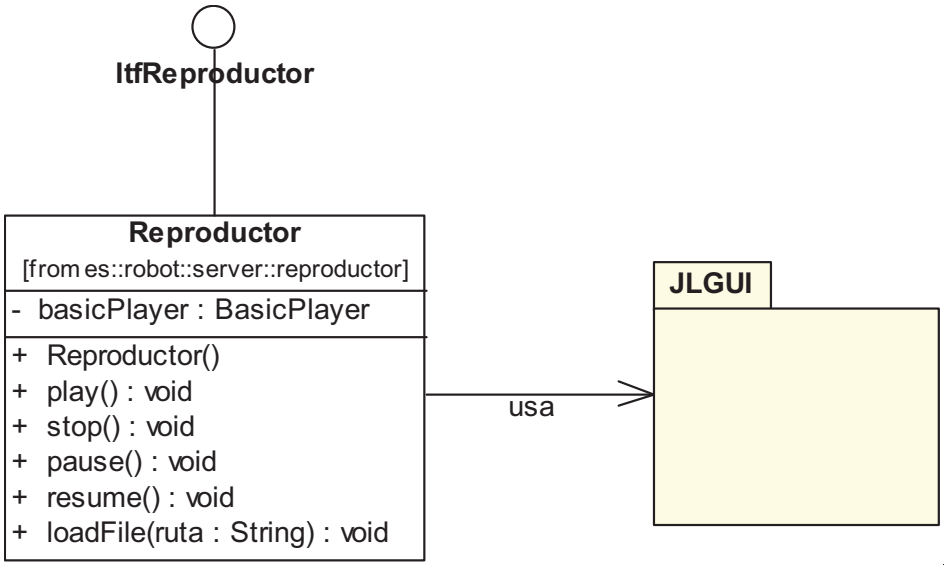


Figura 42 Implementación del componente "Reproductor"

3.1.1.2.3.4.3 Librerías utilizadas

Para este componente se ha usado la librería "JLGUI", que permite manipular sonidos a través de su clase "BasicPlayer". Se realizaron también diversos prototipos antes de crear el componente. Se puede encontrar información acerca de su API en http://www.javazoom.net/jlgui/api.html.

3.1.2 Aplicación para alertar al técnico en caso de nivel bajo de batería.

3.1.2.1 Componente “Batería”

3.1.2.1.1 Funcionalidad

Ya en el diseño hemos descrito la interfaz implementada por este componente. En este apartado vamos a describir las distintas operaciones:

- getEstadoBateria
  - Entradas: ninguna
  - Salidas: un String cuyo contenido indica en qué rango se encuentra el nivel de batería
  - Efecto: realiza una consulta sobre el nivel de la batería
- getPorcentajeBateria
  - Entradas: ninguna
  - Salidas: un entero entre [0,100] que indica el nivel de la batería en tanto por ciento
  - Efecto: realiza una consulta sobre el nivel de la batería

3.1.2.1.2 Implementación

El diagrama de clases que ilustra la implementación seguida para este componente es el siguiente:

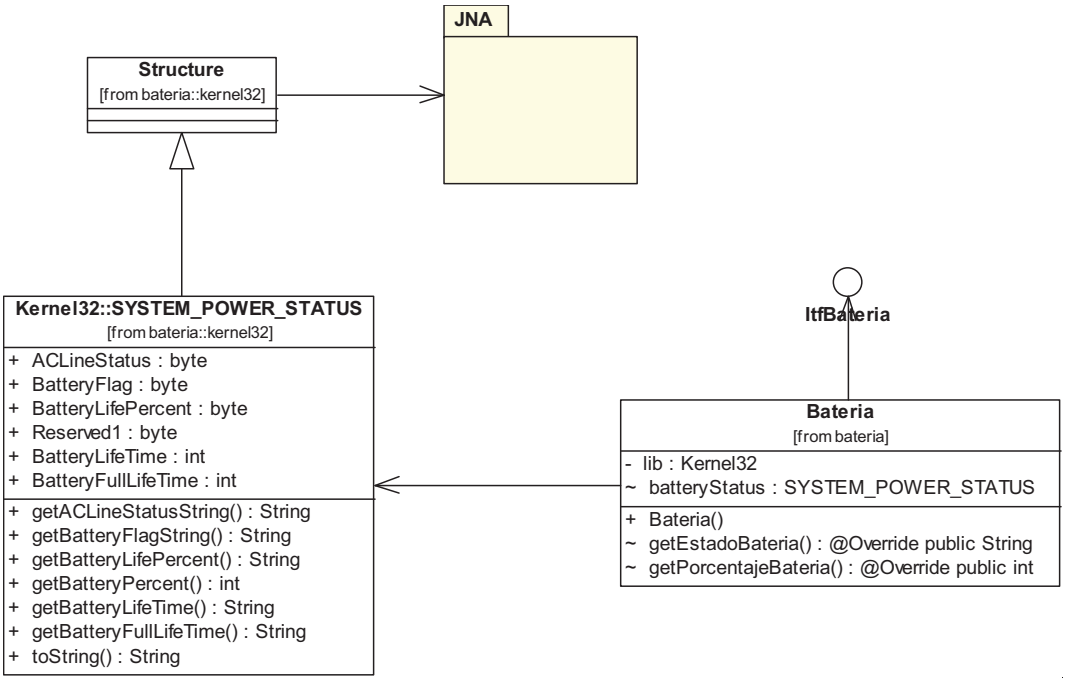


Figura 43 Implementación del componente "Batería"

La implementación se basa en el uso de una clase llamada "Kernel32" y una clase interna llamada "SYSTEM_POWER_STATUS". Básicamente la clase "Kernel32" maneja la librería dinámica con el mismo nombre, permitiendo de esta forma hacer llamadas a la dll (Dynamic Linking Library) mediante código nativo Java.

La clase "SYSTEM_POWER_STATUS" es la clave de este componente. Emula la estructura interna donde reside toda la información acerca de la batería y su estado:

```
public byte ACLineStatus;
public byte BatteryFlag;
public byte BatteryLifePercent;
public byte Reserved1;
public int BatteryLifeTime;
public int BatteryFullLifeTime;
```

Figura 44 Estructura de información sobre la batería

Todos estos campos poseen información sobre el estado del nivel de carga. Esta información es la que obtienen los métodos implementados por el componente "Batería". Por ejemplo, el atributo "BatteryLifePercent" posee la información en tanto por ciento de la capacidad restante para la batería.

3.1.2.1.3 Librerías utilizadas

Todo esto se realiza mediante la librería JNA (Java Native Access), que permite accesos a librerías **nativas,** pero manejando código puramente en Java. Es necesario decir que las pruebas realizadas en ordenadores portátiles han sido totalmente satisfactorias; sin embargo, en el PCBox del robot no ha sido así, debido seguramente a un distinto control de la placa base de este tipo de ordenador para obtener los datos sobre la batería disponible.

Más información al respecto de estas librerías en https://jna.dev.java.net/.

3.1.2.2 Componente "Comunicador"

3.1.2.2.1 Funcionalidad

Al haber visto el diseño en apartados anteriores, pasamos a describir detalladamente las operaciones:

- enviarSMS
    - Entradas: un String para el nombre de usuario, otro String para la contraseña y otro String para el mensaje corto (máximo 60 caracteres)
    - Salidas: un entero que indica si se envió correctamente el sms (0) o si hubo algún problema en el envío (-1)
    - Efecto: la aplicación se conecta con el servicio de Google, verifica el usuario y contraseña y realiza el envío al número de teléfono móvil configurado en Google Calendar

- enviarMail
  - Entradas: un String para la dirección de correo origen, otro String para la contraseña, otro String con el contenido del e-mail, otro String con la dirección de correo destino y un último String con el asunto del mensaje
  - Salidas: un entero que indica si se envió correctamente el sms (0) o si hubo algún problema en el envío (-1)
  - Efecto: la aplicación se conecta con el servidor adecuado, verifica el usuario y contraseña y realiza el envío a la dirección de correo especificada

3.1.2.2.2 Implementación

La figura 45 muestra el diagrama de clases para la implementación interna de este componente:

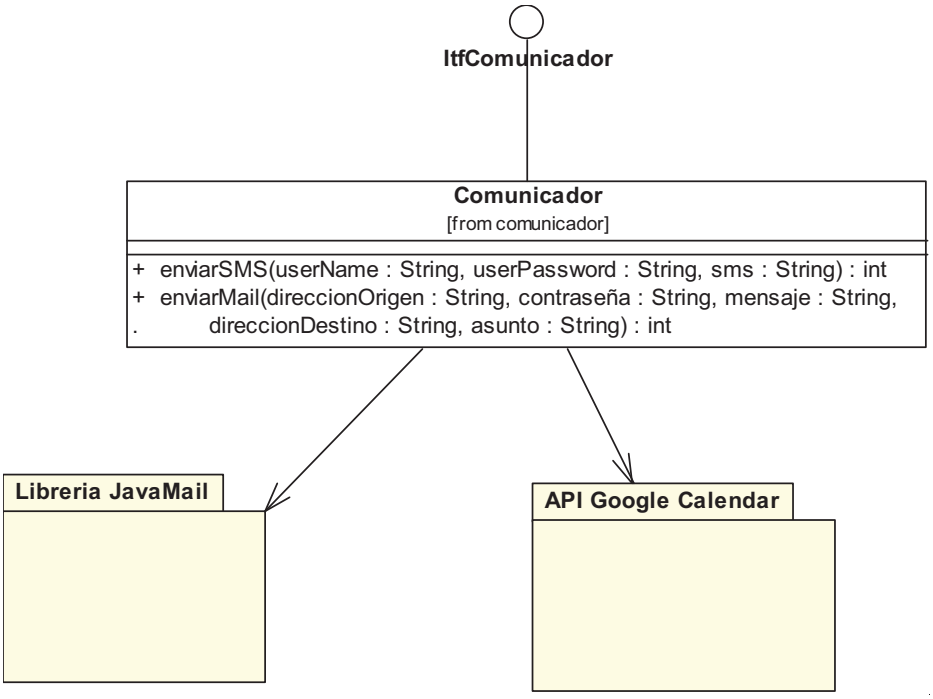


Figura 45 UML del componente "Comunicador"

Como se puede observar, la clase que implementa la interfaz descrita, define las operaciones de ésta utilizando dos APIs que se pasan a explicar a continuación.

3.1.2.2.3 Librerías utilizadas

En la implementación de este componente se hace uso de dos librerías:

- JavaMail: se trata de una librería que ofrece servicios para gestionar cuentas de correo. Por simplicidad, el único servidor disponible para enviar mails es el de GMail, aunque modificando la implementación esto se puede hacer extensible a otros servidores. Esto supone que la cuenta de correo del remitente, así como la contraseña asociada, deben aparecer en el XML de configuración con valores válidos para una cuenta de GMail. Más información sobre su API en http://java.sun.com/products/javamail/
- API Google: Google pone a disposición de los desarrolladores su API (http://code.google.com/apis/). Entre todos los servicios, decidimos utilizar la API de Google Calendar, ya que posee un servicio de alertas para teléfonos móviles. Desde Google Calendar es necesario configurar el número de teléfono móvil asociado a la cuenta Google. Esta cuenta es la que debe aparecer definida en el fichero de configuración XML para el técnico.

3.1.2.3 Configuración

Como ya se ha explicado, tanto en la aplicación cliente – servidor para controlar el robot remotamente como en la aplicación de alertas, es necesaria una configuración por XML para permitir el cambio de ciertos parámetros sin necesidad de recompilar el proyecto entero. A continuación vemos la configuración XML para cada aplicación:

3.1.2.3.1 Aplicación cliente – servidor

```
<configuracion>
  <mailTecnico mail="cuentadeltecnico@fdi.ucm.es" />
  <mailGmailSMS mail ="tecnico.fdi@gmail.com" pwd="pruebaparassii" />
  <puerto>16125</puerto>
</configuracion>
```

Figura 46 Configuración XML de la aplicación cliente - servidor

- En primer lugar, vemos que se debe configurar el mail del técnico. Este mail puede estar alojado en cualquier servidor. A él llegarán todos los avisos emitidos a través de la aplicación de control remoto del robot.
- En segundo lugar, es necesario tener creada una cuenta en GMail si se desean recibir los avisos a través de SMS. Esto es debido a que se utiliza el servicio GCalendar con alertas via SMS. Se dan más detalles acerca de esto en el punto 3.1.2.3.3. Como se puede observar, es necesario indicar en la configuración tanto el nombre de la cuenta GMail como la contraseña.
- Finalmente, para esta aplicación es necesario especificar el puerto usado por la aplicación para habilitar la conectividad. Es un parámetro de desarrollo, principalmente.

3.1.2.3.2 Aplicación de alertas

```
<configuracion>
  <mailTecnico mail="cuentadeltecnico@fdi.ucm.es" />
  <mailGmailSMS mail ="tecnico.fdi@gmail.com" pwd="pruebaparassii" />
  <limiteBateria>37</limiteBateria>
</configuracion>
```

Figura 47 Configuración XML de la aplicación de alertas

- En primer lugar, vemos que se debe configurar el mail del técnico. Este mail puede estar alojado en cualquier servidor. A él llegarán todos los avisos emitidos a través de la aplicación de control remoto del robot.
- En segundo lugar, es necesario tener creada una cuenta en GMail si se desean recibir los avisos a través de SMS. Esto es debido a que se utiliza el servicio GCalendar con alertas via SMS. Se dan más detalles acerca de

esto en el punto 3.1.2.3.3. Como se puede observar, es necesario indicar en la configuración tanto el nombre de la cuenta GMail como la contraseña.

- Finalmente, para esta aplicación es necesario especificar el límite para el nivel de la batería en tanto por ciento. De esta manera, podremos recibir las alertas cuando el robot alcance el límite deseado.

3.1.2.3.3 Configuración de alertas para móviles

En los puntos anteriores se ha comprobado que en la configuración es necesario declarar una cuenta de GMail y su contraseña asociada si se desea recibir alertas via SMS. Esto es debido al uso que se hace del servicio GCalendar de Google.

Este servicio permite añadir "notas de aviso" personales, y configurar avisos para recibir SMS cuando llegue la fecha anotada. Lo que nuestro componente "Comunicador" hace es utilizar esta funcionalidad para guardar un nuevo aviso en GCalendar un minuto después de la hora a la que se ha realizado la petición.

Así, unos instantes después, si se han configurado correctamente las alertas para móviles en GCalendar, se recibe el mensaje corto en el móvil con el asunto de la tarea anotada.

Mostramos a continuación una sencilla guía para configurar correctamente nuestro número de teléfono móvil en el servicio GCalendar:

- Accedemos a la página www.google.com/calendar y nos registramos con la cuenta que previamente habremos creado:

Figura 48 Registro en GCalendar I

- Al acceder a la página, obtendremos una pantalla como esta:

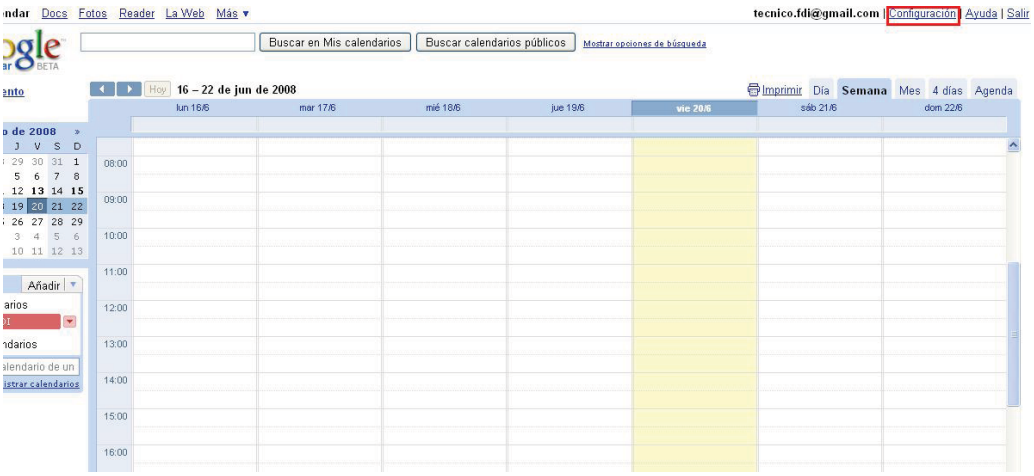


Figura 49 Registro en GCalendar II

Pincharemos entonces sobre la pestaña “Configuración” de la parte superior (marcada en rojo) y obtendremos la siguiente página en la secuencia:

Figura 50 Registro en GCalendar III

Pincharemos entonces sobre la pestaña “Configuración para móviles” (marcada en rojo) para configurar nuestros avisos de GCalendar, de forma que podamos recibir las notificaciones en nuestro teléfono via SMS.

- En esta página deberemos completar la información requerida, en el siguiente orden (secuencia marcada en rojo):

Figura 51 Registro en GCalendar IV

- Primero introducimos el número de teléfono donde queremos recibir las alertas (1).
- Después pulsamos el botón “Enviar código de verificación” (2). Tras unos instantes, recibiremos dicho código en nuestro móvil.
- Introducimos el código recibido en el campo habilitado para ello (3).
- Pulsamos el botón “Finalizar configuración” (4), que se iluminará cuando el código introducido sea auténtico.
- Finalmente pulsamos el botón “Guardar” (5), que mantendrá los cambios realizados.

Con estos sencillos pasos, por lo tanto, habremos configurado nuestro teléfono móvil para la recepción de alertas a través del servicio de GCalendar y, consecuentemente, podremos recibir los mensajes emitidos por las aplicaciones que utilicen el componente “Comunicador”.

3.1.3 Problemas encontrados

El principal problema fue la familiarización con tecnologías y librerías nuevas para la realización de pruebas. Una vez hechas algunas probatinas, se realizó un diseño de forma que se crearan los dos componentes

descritos. De esta manera, se permitiría su reutilización, así como una gran flexibilidad a la hora de aumentar su funcionalidad.

Es necesario indicar que el componente "Batería" funciona sobre cualquier ordenador portátil; sin embargo, la funcionalidad no se mantiene sobre el PCBox del robot debido, seguramente, a diferencias sustanciales en las distintas placas base.

Además, el componente "Comunicador" a veces no funcionó dentro de la Facultad de Informática, debido a un problema de infraestructura, ya que en la Facultad la conexión WiFi es muy inestable. Esto no permite conectar de manera robusta con los servicios remotos. Las pruebas realizadas con conexiones fiables han resultado satisfactorias en un 100% de los casos.

Se llevaron a cabo dos prototipos en paralelo, uno para el uso del componente "Batería" y otro para el componente "Comunicador". Una vez hecho esto, la integración tanto dentro del servidor como dentro de esta aplicación de alertas fue trivial.

En cuanto a la comunicación Socket, surgieron problemas a la hora de integrar una librería de Sockets de alto nivel (Solar Sockets) con la arquitectura del robot, que finalmente no se utilizó, realizando una implementación propia, ampliando la funcionalidad de la libraría de Sockets del proyecto anterior.

3.2 Localización mediante visión artificial

3.2.1 Introducción

La primera decisión que tuvimos que tomar llegados a la etapa de implementación fue si debíamos implementar nosotros mismos los algoritmos de tratamiento de imagen y captura de vídeo desde una cámara o si utilizábamos los ofrecidos por alguna librería.

Al final nos decantamos por utilizar las funciones ofrecidas por alguna librería de código abierto para así ahorrarnos tiempo en la implementación de los algoritmos. Otra de las razones por las que decidimos utilizar librerías fue la mayor fiabilidad y eficiencia que ofrecen sus algoritmos si la librería es robusta y contrastada.

La necesidad de disponer de un paquete de visión por computador y procesamiento de imágenes lo suficientemente flexible y potente como para proporcionar herramientas de alto nivel para el desarrollo de nuestro sistema de localización propició una búsqueda incesante de software que cumpliera dichas características y sobre todo que fuese gratuito, estuviese

soportado/revisado por personal cualificado y además se le intuyese una larga vida. Al principio tuvimos algunos problemas que comentamos más extensamente en el apartado 3.2.5, ya que no conseguíamos dar con la librería adecuada.

Después de investigar bastante y de encontrarnos con multitud de problemas como ya hemos comentado, al final escogimos la librería de código abierto OpenCV (Intel® Open Source Computer Vision Library).

Esta librería proporciona un marco de trabajo de alto nivel para el desarrollo de aplicaciones de visión por computador en tiempo real: estructuras de datos, procesamiento y análisis de imágenes, análisis estructural, etc. Este marco de trabajo facilita en gran manera la implementación de distintas técnicas de visión por computador, aislando al desarrollador de las peculiaridades de los distintos sistemas de visión.

Además, es la única librería gratuita de las que conseguimos encontrar que proporciona bibliotecas de tipos de datos estáticos y dinámicos (matrices, grafos, árboles, etc.), herramientas con la posibilidad de trabajar con la mayoría de las capturadoras/cámaras del mercado, entornos de desarrollo fáciles e intuitivos y todo ello, corriendo en dos de los sistemas operativos más utilizados del mundo Microsoft® Windows y Linux, un requisito imprescindible para nuestro proyecto. Se puede encontrar más información sobre esta librería en el apéndice A.

Una vez que teníamos clara la especificación y diseño de nuestro sistema y las herramientas necesarias para implementarlo, comenzamos la fase de implementación que explicamos con más detalle a continuación para cada una de las partes principales en que hemos dividido el sistema.

3.2.2 Captación de imágenes

El fin perseguido por esta parte de la aplicación es obtener imágenes a partir de una cámara conectada al sistema. Es una tarea fundamental, ya que sin ella, el programa no puede comenzar.

La implementación de esta parte es la que más problemas nos dio, por la dificultad de encontrar unas librerías que se adaptasen a nuestros requisitos. Esta parte se explica de forma más detallada en el apartado 3.2.5.1 Problemas encontrados. Se realizaron diversos prototipos con muchas librerías pero ninguno de ellos se ajustaba a nuestro objetivo de forma totalmente precisa. El mejor resultado lo obtuvimos con unas librerías realizadas por unos compañeros de años anteriores, pero su integración con nuestro sistema resultaba demasiado complicada, además de complicar el diseño e implementación del mismo por lo que decidimos abandonar esta alternativa.

Finalmente, conseguimos encontrar la solución con la misma librería que utilizábamos para el tratamiento de la imagen. Esto hecho hizo que la unión entre la parte de captación de imágenes y la parte de preprocesamiento fuese más sólida, sencilla y eficiente y que sus barreras fronterizas estuviesen más cercanas.

Una vez dispusimos de una librería que se ajustaba perfectamente a nuestros requisitos comenzamos la implementación definitiva de este apartado. Esta implementación fue realizada de forma incremental, de manera que comenzó con un prototipo básico que después se fue refinando y mejorando para conseguir llegar a la mejor solución para el problema de la forma más sencilla posible.

El hecho de disponer de una buena librería hizo que la implementación final sea bastante robusta, sencilla y fácil de comprender para cualquier posterior desarrollador que necesite comprenderla. A continuación, explicamos los pasos seguidos para llevar a cabo esta tarea:

1) Iniciar la captura desde la webcam. Para ello utilizamos la función cvCaptureFromCAM de la librería OpenCV. De esta forma, la webcam comienza la captura de vídeo de forma permanente hasta la finalización del programa. Si por algún motivo esta captura no se consiguiese iniciar, se mostraría por pantalla el correspondiente aviso.

2) El siguiente paso es comenzar la ejecución de un bucle infinito que se repite cada 200 ms (5 veces por segundo) y que finaliza cuando termina la ejecución de la aplicación. Cómo ya comentamos anteriormente, la frecuencia de repetición del bucle puede aumentarse o disminuirse para adaptarse a situaciones concretas.

3) Dentro del bucle, obtenemos un frame de la captura que será la imagen a la que le serán aplicadas las distintas técnicas de tratamiento.

4) Liberación en memoria de las imágenes una vez que han sido tratadas y sabemos perfectamente que nos las vamos a volver a utilizar.

5) Vuelta al paso 2.

6) Al finalizar el bucle detenemos la captura y la liberamos de memoria.

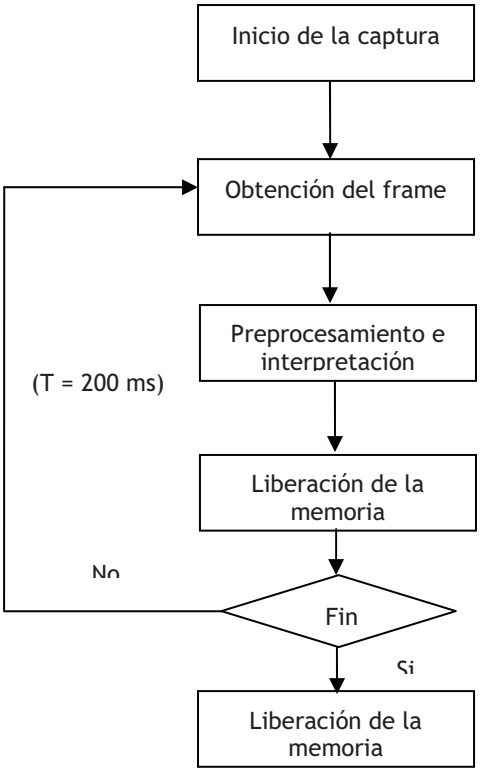


Figura 52 Pasos en el procesamiento de las fotografías

Como se puede observar, se obtienen 5 imágenes por segundo. Estas imágenes serán procesadas con las técnicas que se explican en el siguiente apartado y son las que servirán a nuestra aplicación para obtener la orientación y la posición en cada momento.

Es muy importante liberar la memoria cada vez que no se vaya a utilizar más una imagen. De lo contrario, esta imagen permanecerá en memoria de forma indefinida. Al cabo de un tiempo, todas las imágenes capturadas se encontraran en memoria, y llegará un momento en que no quede espacio libre. El sistema se puede volver inestable, incluso detener su ejecución por falta de memoria. En las primeras versiones que implementamos no tuvimos esto en cuenta de forma estricta. Esto ocasionaba que cuando la aplicación llevaba un tiempo en funcionamiento, el computador que la albergaba se quedaba sin memoria y se producía un error de memoria insuficiente. Por ello resaltamos la importancia de este factor.

3.2.3 Preprocesamiento e interpretación de imágenes

La imagen que nos llega del apartado anterior tiene que ser tratada para conseguir identificar las líneas rectas que se encuentran en ella, ya que estas líneas son las que nos ayudarán a deducir la posición y la orientación en la que nos encontramos en cada momento.

La implementación de este apartado ha sufrido diversos cambios y modificaciones desde su comienzo hasta su implementación final debido a que según comenzábamos a implementar los primeros diseños llegábamos a resultados poco satisfactorios y que tenían que ser mejorados.

La primera implementación realizada con la librería OpenCV obtenía buenos resultados. Sin embargo, no era lo suficientemente robusta cómo nos hubiese gustado y decidimos mejorarla, primero rehaciendo el diseño y después implementando este nuevo diseño mejorado. Los principales puntos débiles de esta primera implementación eran los siguientes:

- Se trataba de una implementación bastante complicada, ya que había que distinguir multitud de casos dependiendo del tipo de imagen.
- La luz afectaba mucho a la hora de detectar los bordes.
- Los dos factores anteriores unidos a otros hacían que en algunas imágenes no se detectasen todas las líneas rectas posibles, o que se detectasen como líneas rectas partes de la imagen que no lo eran.

El nuevo diseño, mejoraba prácticamente todos los puntos débiles comentados arriba y por ello fue el que finalmente se siguió para la implementación definitiva. A continuación, explicamos con más rigor los detalles de esta nueva implementación. Es una implementación más sencilla y que consigue una solución a nuestro problema bastante fiable. Consta de las siguientes partes:

1) Convertir la imagen a escala de grises. Este paso es necesario porque las imágenes en color utilizan 3 matrices para almacenarse (R, G, B). Cómo en nuestro caso, nos es indiferente el color, trataremos con una imagen en escala de grises que ocupa menos espacio y es más sencillo su tratamiento.

2) Obtener el histograma de la imagen y la varianza. Este paso es necesario para distinguir entre distintos tipos de imágenes. Por ejemplo una imagen tomada debajo de un fluorescente tendrá una varianza mayor que otra que no esté influenciada por ninguna fuente de luz.

3) A continuación aplicamos un suavizado Gaussiano para reducir el ruido. Es importante para que el algoritmo de detección de bordes sea más efectivo.

4) Después, empleamos el algoritmo de detección de bordes de Canny. En la primera implementación probamos con dos algoritmos diferentes, el operador de Sobel y la Laplaciana. Sin embargo, después de utilizar el algoritmo de Canny nos dimos cuenta que se comportaba mejor y además se adaptaba perfectamente al nuevo diseño. Este paso es muy importante ya que dependiendo de su resultado, se podrán detectar mejor o peor las rectas de la imagen con las que realizaremos el cálculo de la posición y la orientación.

5) Una vez detectados los bordes de la imagen, ya sólo queda averiguar cuales de ellos son líneas rectas. Para ello utilizamos la transformada de Hough. Esta función devuelve todas las líneas rectas encontradas en una estructura llamada lineas de tipo CvSeq*.

Para resolver cada uno de los apartados anteriores, utilizamos una serie de funciones de la librería OpenCV y otras definidas por nosotros que están especificadas en el apartado 2.4.3 y que hacen uso también de las funciones que ofrece la librería. Vamos a comentar de forma más detallada su implementación:

- funHistograma (imagen) » Se define el rango del histograma (256 niveles de gris) y una estructura de tipo CvHistogram* proporcionada por la librería. Posteriormente se llama a la función de la librería que calcula el histograma pasándole como parámetro la imagen y el histograma. Una vez obtenido el histograma calculamos la varianza asociada mediante la siguiente fórmula:

$$\sigma^2 = \sum_{g=0}^{L-1} (g - \overline{g})^2 P(g)$$

- funBordes (imagen, umbralCanny1, umbralCanny2) » Definimos una imagen auxiliar con un tamaño acorde con el soportado por la función CvCanny ofrecida por la librería donde clonamos la imagen original. Posteriormente definimos otra imagen a la que denominamos imagenBordes que será la imagen resultante de aplicar el algoritmo de Canny sobre la imagen original. Para finalizar llamamos a la función CvCanny con la imagen auxiliar, imagenBordes y los umbrales especificados.

- funHough (imagen, umbralHough) » Definimos una imagen auxiliar con un tamaño acorde con el soportado por la función CvHoughLines2 ofrecida por la librería donde clonamos imagenBordes. Después llamamos a la función con los parámetros necearios y almacenamos su salida en la variable lineas comentada anteriormente.
- funDibujarLineas (imagen, rectas) » Recorre las rectas devueltas por el algoritmo de Hough, obtiene dos puntos de cada una de ellas y las dibuja en la imagen con la función CvLine.
- funBrilloContrast (imagen, brillo, contraste): Esta función no es utilizada en la implementación final del algoritmo pero si que se utilizó en las primeras versiones. Sin embargo nos ha parecido interesante mantenerla, ya que puede resultar útil en un futuro. El algoritmo utilizado en esta función es muy parecido al utilizado en el conocido programa de retoque de fotográfico Photoshop.

Una vez completada esta parte, disponemos de todo lo necesario para realizar el cálculo de la posición y la orientación que se comenta en el siguiente apartado.

3.2.4 Cálculo de la posición y la orientación

El objetivo final de un sistema de localización es informar de la posición y orientación en la que se encuentra en cada momento. En los dos subapartados siguientes explicamos cómo nuestro sistema realiza esta tarea realizando una distinción entre el cálculo de la orientación y el de la posición.

3.2.4.1 Cálculo de la orientación

Como ya se ha comentado anteriormente en esta memoria, la orientación devuelta por nuestra aplicación, se expresa en referencia a una orientación inicial. Si se produce un giro hacia la izquierda, la orientación irá disminuyendo hasta alcanzar el mínimo de -180º y si gira hacia la derecha la orientación aumentará hasta alcanzar un máximo de 180º. Toda la especificación y diseño de esta parte se encuentran en el apartado 2.4.4.1. En la figura 53 se muestra la implementación del algoritmo en un diagrama de flujo:

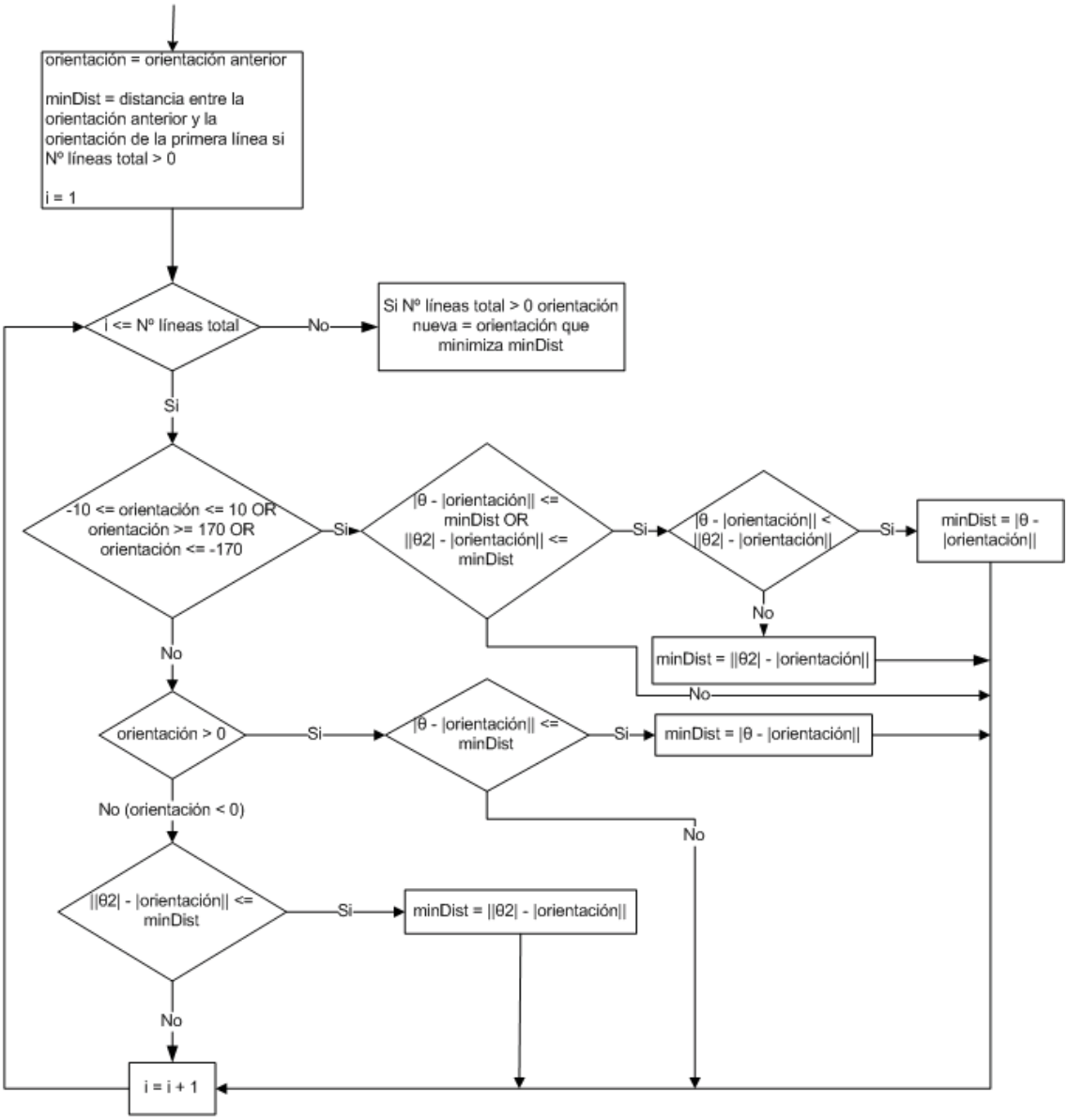


Figura 53 Diagrama de flujo que explica el cálculo de la orientación

Una vez implementado el algoritmo es importante darse cuenta de que cuando la cámara gira hacia la izquierda, la foto que captura gira hacia la derecha, por lo que la orientación obtenida no es la real, sino que está

cambiada de signo. La solución a este problema es trivial y consiste en multiplicar por -1 la orientación obtenida.

Para el cálculo de la posición es imprescindible conocer la orientación en la que nos encontramos.

3.2.4.2 Cálculo de la posición

El algoritmo de cálculo de posición tiene cuatro partes claramente diferenciadas:

- Calculo del punto de referencia. Supone hallar el punto de corte de rectas perpendiculares más cercano al robot.
- Seguimiento de un punto de referencia anterior.
- Calculo de la fiabilidad en cada paso.
- Seguimiento de balizas absolutas.

El cálculo del punto de referencia más cercano al robot, tenemos que realizar dos pasos:

Primero se calcula la ecuación de la recta más cercana al robot y si se encuentra una recta, se busca la siguiente recta más cercana al robot que forme con la anterior alrededor de 90º. El cálculo de la recta más cercana al robot se hace siguiendo el esquema ilustrado en la figura 54:

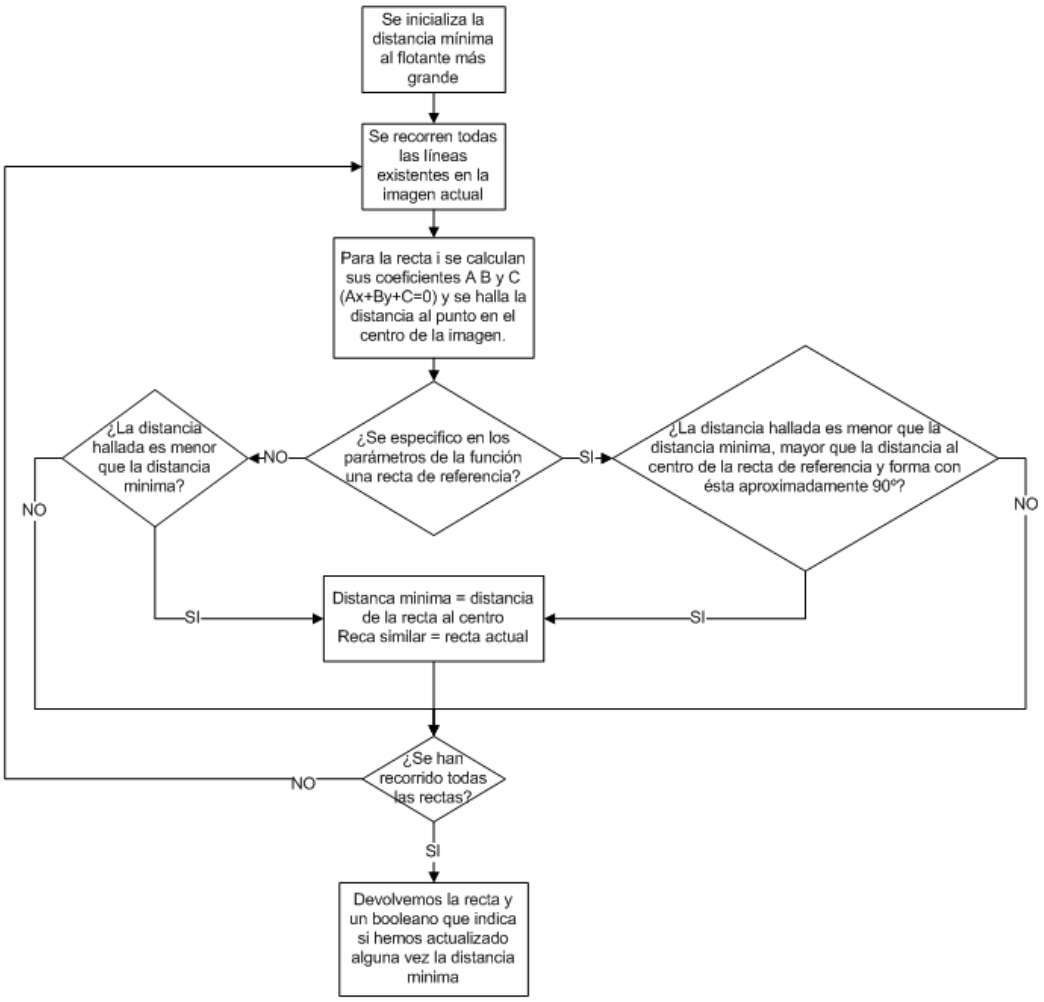


Figura 54 Diagrama de flujo para el cálculo de la recta más cercana al robot

A continuación, se halla el punto de corte de las dos rectas encontradas.

Para realizar el seguimiento de un punto de referencia, debemos realizar el seguimiento de las rectas con las que se calculo dicho punto. Para ello, suponiendo que en la iteración anterior se guardaron los parámetros de las rectas halladas, se procede buscando la primera recta, y si se encontró, entonces se busca la segunda.

Cada recta se busca como muestra la figura 55:

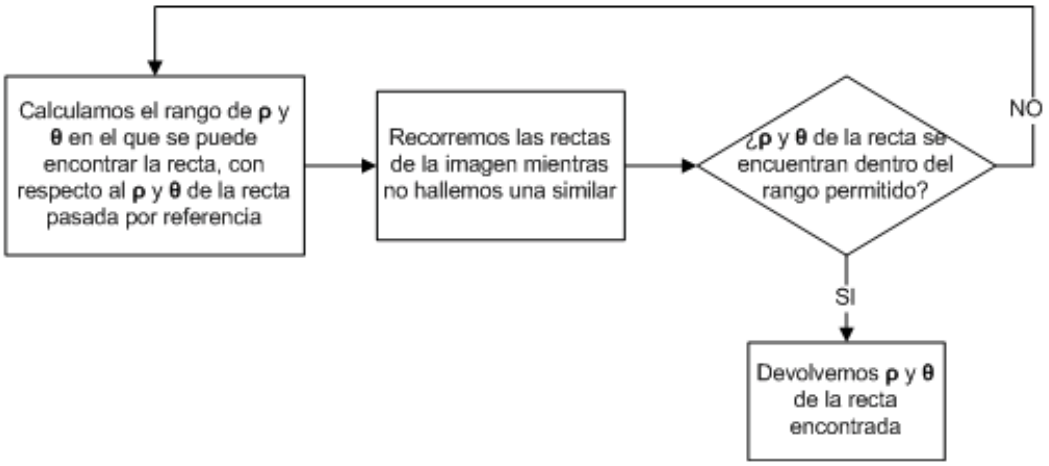


Figura 55 Diagrama de flujo para la búsqueda de rectas

Si se han encontrado las dos rectas, se busca el punto de corte de ambas.

La medida de similitud aplicada, tanto para rho como para theta, se puede configurar de forma dinámica en un fichero XML, ya que la distancia entre dos rectas depende tanto del retardo entre dos capturas consecutivas como de la velocidad del Robot.

El cálculo de la fiabilidad del algoritmo es una tarea sencilla, ya que se conocen las dos causas por las que el algoritmo podría obtener mediciones erróneas, estudiadas en la fase de análisis y diseño.

Si se pierde un punto de referencia debido a que en la nueva imagen no aparecen rectas similares a las halladas en la iteración anterior, y por lo tanto debiendo recalcular un punto de referencia nuevo perdiendo el posible incremento en la posición de esa captura, la fiabilidad debe disminuir.

Si en la imagen no hay rectas con las que calcular los puntos de corte, entonces evidentemente, no podremos realizar el seguimiento del punto de referencia anterior ni tampoco recalcular un punto de referencia nuevo, con lo cual la fiabilidad debe disminuir en mayor medida que en el caso anterior.

La disminución de fiabilidad es configurable mediante un XML, ya que al igual que ocurría con la medida de similitud, la magnitud del error de cálculo del algoritmo si se da una de las causas anteriores, es directamente proporcional a la velocidad que tenga el móvil que integre nuestro sistema

y al retardo entre la captura de dos imágenes consecutivas. Se detallará este punto en el apartado de integración del sistema de visión.

Como se ha comentado en el apartado de especificación y diseño, nuestro sistema de visión es completamente autónomo, pero necesita un cálculo de posición aproximado aportado por una fuente externa para suplir los errores de cálculo cometidos cuando se da alguna de las situaciones de error comentadas anteriormente. Para integrarlo en el robot del museo, se ha considerado que la fuente externa que aporta dicho cálculo aproximado sea la combinación de los encoders y la brújula. Se detallará este punto también en el apartado de integración del sistema de visión.

El seguimiento de balizas absolutas debe ser configurable de forma dinámica, con lo cual se ha generado un fichero XML, que es leído por la aplicación al comienzo de la ejecución.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE focos SYSTEM "focos.dtd">
<focos>
  <foco posX_1="200" posY_1="200" posX_2="240" posY_2="220">
      foco_1</foco>
  <foco posX_1="300" posY_1="200" posX_2="340" posY_2="220">
      foco_2</foco>
  <foco posX_1="400" posY_1="200" posX_2="440" posY_2="220">
      foco_3</foco>
  …
</focos>
```

Figura 56 Fichero de configuración XML para los focos

Cada foco tiene un nombre asociado, para una depuración más cómoda del algoritmo, y posee cuatro argumentos.

Los dos primeros definen una de las esquinas del foco, en concreto la más cercana al punto origen del mapa que representa el mundo en que el robot se mueve. Los dos siguientes definen el punto opuesto en la diagonal que une ambos puntos.

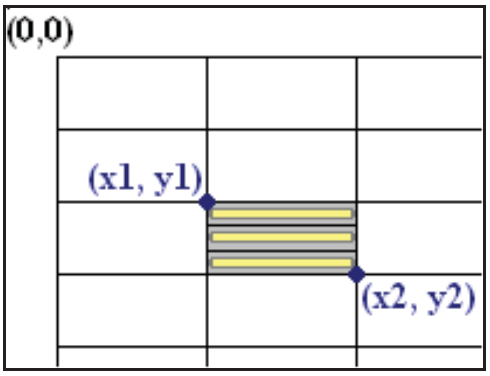


Figura 57 Situación de los focos sobre el mapa

Cuando encontramos una captura con varianza mayor que 6900, buscamos qué foco es el más cercano a la posición actual calculada, y de ese foco, a qué punto estamos más próximos, reconfigurando la posición calculada del robot a la posición del foco con fiabilidad 100%.

En la figura 58 se muestra un diagrama de flujo general que integra las cuatro partes del sistema funcionando en conjunto:

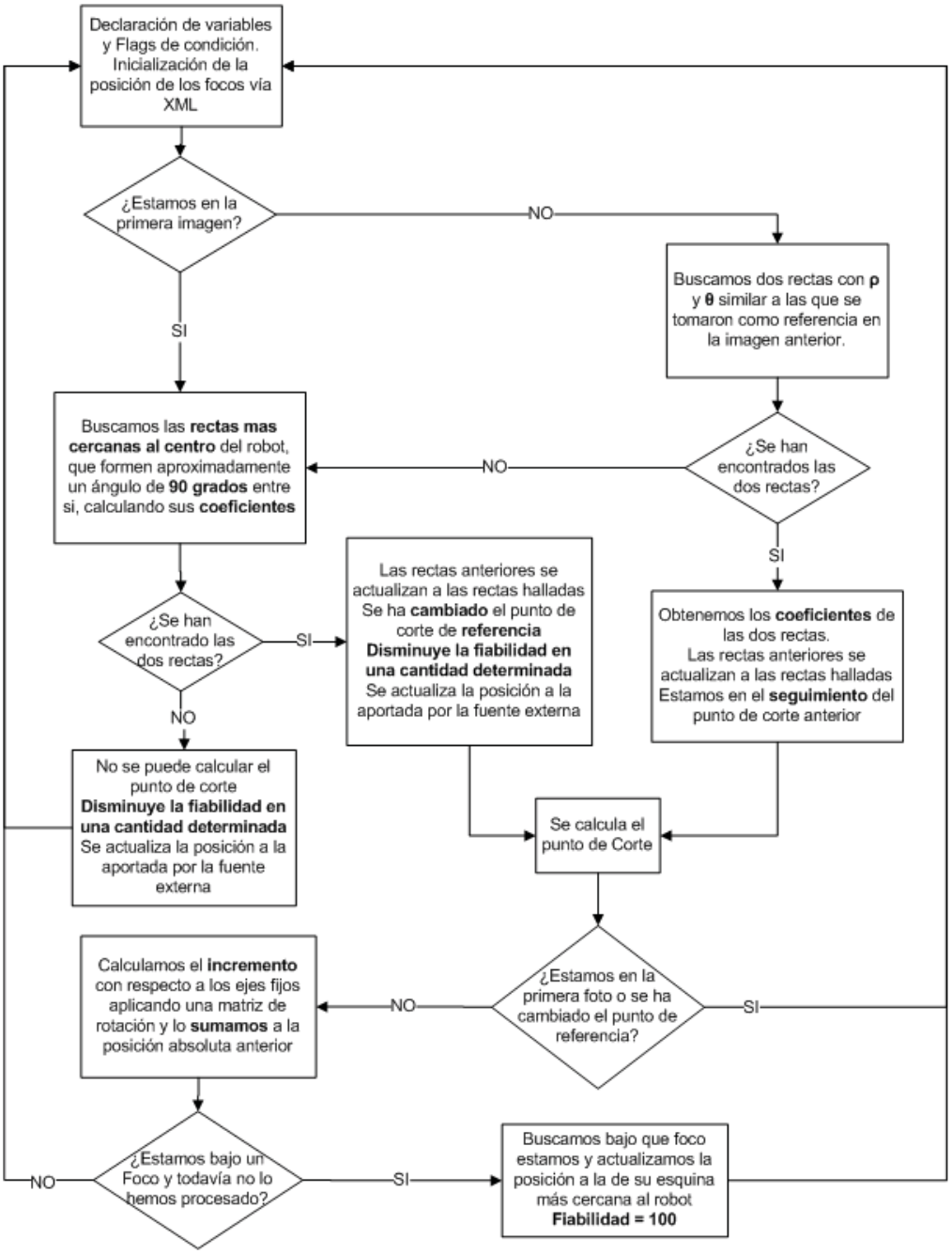


Figura 58 Funcionamiento completo del sistema de posicionamiento

3.2.5 Problemas encontrados

3.2.5.1 Problemas en la captación de imágenes

El primer problema al que nos enfrentamos dentro de este proyecto de visión artificial fue de nivel tecnológico: debíamos encontrar la forma de manipular en nuestro propio software una fotografía tomada por la webcam que habíamos comprado. Para ello los bytes que componen la fotografía deberían estar situados en una estructura tratable.

Evidentemente, se trataba de investigar en profundidad para descubrir alguna librería de libre distribución que ofreciera este servicio. La tarea no fue nada fácil, debido a que casi todo el código encontrado estaba destinado a Visual Basic. Además, la librería debería ser válida tanto para Windows como para Linux.

Inicialmente un profesor de la Facultad de Informática nos recomendó utilizar la librería "avicap32.dll" de Windows, pero la documentación al respecto era escasa. Se probaron algunos códigos fuente de aplicaciones libres de internet, pero no llegaron a ser operativos. Incluso se pidió ayuda a un departamento de Robótica de la Universidad de Sevilla, ya que buscando en la red descubrimos que hacían cosas similares a lo que estábamos buscando. Esto tampoco resultó.

Después de hacer multitud de pruebas con lo poco que habíamos encontrado, nuestro tutor de proyecto nos remitió a un antiguo alumno de Sistemas Informáticos, ya que éste había trabajado en un problema similar algunos años antes.

En efecto, este alumno había desarrollado en su Proyecto Fin de Carrera una librería bastante completa en C para captar imágenes a partir de una webcam. Esta librería era intuitiva y fácil de utilizar, y válida tanto para Windows como para Linux, lo cual encajaba perfectamente con nuestros requisitos.

Tras estudiar la librería, nos dispusimos a realizar pruebas con ella. Primeramente se desarrolló un prototipo en Borland C++, pero lo interesante era integrarlo en el proyecto del robot. Cuando nos dispusimos a hacer esto, obtuvimos una serie de errores de configuración que nos resultaron realmente difíciles de eliminar. De hecho, en este punto resultó imprescindible la ayuda del autor de la librería.

Sin embargo, al poco tiempo de haber conseguido integrarla en el proyecto, otro componente de nuestro grupo de trabajo, que a la vez estaba investigando sobre librerías de tratamiento de imágenes, descubrió que la librería OpenCV podía obtener de manera cómoda una estructura con la

información de una fotografía de la webcam. Por lo tanto, y teniendo en cuenta que esta última librería además resultaba de gran utilidad para otros aspectos del proyecto, decidimos finalmente hacer uso de la misma.

3.2.5.2 Problemas en el preprocesamiento e interpretación

Uno de los principales problemas que tuvimos para dar con una buena solución consistió en las dificultades para encontrar una librería potente, sencilla, gratuita y fiable que se adaptase a nuestras necesidades. Tampoco queríamos que la inclusión de la librería en el proyecto implicase la instalación de otras tecnologías adicionales que lo harían más complicado.

Antes de decantarnos por OpenCV realizamos algunas primeras implementaciones con librerías tales como VIPS o ImageMagik. Las complicaciones encontradas para su utilización así cómo el hecho de que no se correspondían exactamente con nuestras necesidades, hicieron que abandonásemos estos caminos y buscásemos soluciones mejores.

Las razones que nos hicieron decantarnos finalmente por OpenCV se explican en el apartado de introducción 3.2.1.

Otro de los problemas que encontramos en este apartado, es la entrada de luz a la cámara en el momento de realizar la captura, ya que las imágenes captadas en esas circunstancias son muy complicadas de tratar. Se ha intentado que el algoritmo sea lo más robusto posible, pero sería conveniente que la cámara estuviese cubierta por los laterales para así evitar en cierta medida la exposición a la luz que le puede entrar desde por ejemplo las ventanas y evadir casos en los que el algoritmo de tratamiento no sea capaz de reconocer las líneas de forma correcta.

3.3 Adaptación de comunicación con microcontroladores a USB

3.3.1 Implementación

Como ya hemos comentado en el diseño de esta adaptación (apartado 2.5.3.2), se llevó a cabo tanto un cambio en el formato de comandos como en la comunicación serie. Vemos los detalles de implementación por separado:

3.3.1.1 Cambio en la comunicación

El objetivo era cambiar la comunicación serie existente entre el módulo "Pic-comm" y el propio PIC a través de una nueva librería de comunicación USB que se nos facilitó. Para ello era necesario cambiar el proceso llevado a cabo en el módulo "Pic_comm".

A continuación explicaremos los procesos implementados mediante hilos, tanto para el control de los motores a través del PIC como para el de sensores.

3.3.1.1.1 Introducción al Pic_comm de motores

Las acciones principales de esta parte, donde se lleva a cabo la comunicación con el PIC mediante USB, están representadas mediante el siguiente diagrama:

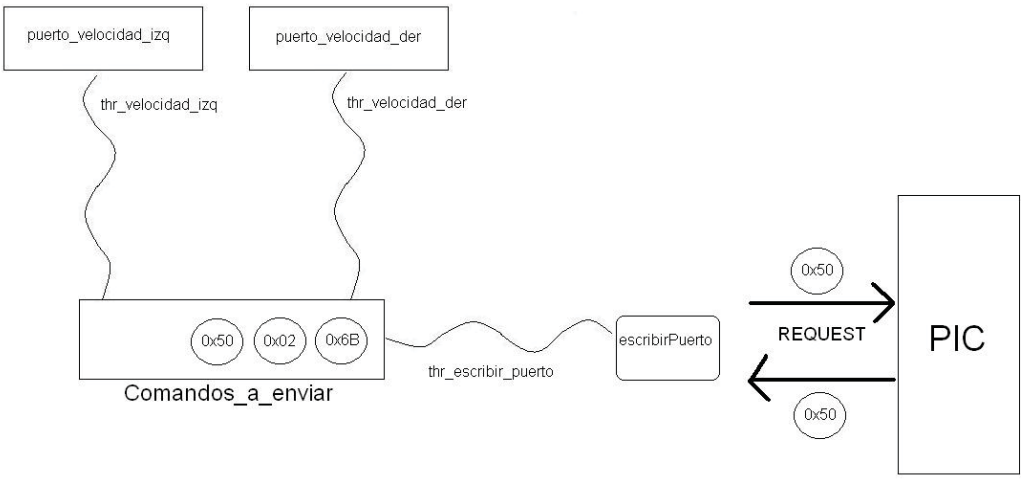


Figura 59 Proceso principal en "Pic_comm" de motores

Entre los puertos de entrada que posee el módulo "Pic_comm" se encuentran los puertos que reciben las velocidades de las ruedas derecha e izquierda. La información que contienen estos puertos llegan del módulo "Motores", y el objetivo es que el módulo "Pic_comm" haga de intermediario para enviar esas órdenes al PIC de Motores (más información

sobre las estructuras de módulos en las memorias de los proyectos anteriores).

Los hilos "thr_velocidad_der" y "thr_velocidad_izq" realizan el papel de consumidores, sacando periódicamente las peticiones de las colas. Esas peticiones se encolan en otra estructura FIFO conocida como "Comandos_a_enviar". Con respecto a esta cola, ambos hilos son además productores.

Otro hilo más (thr_escribir_puerto) se encarga de consumir los comandos a enviar periódicamente, enviándolos al PIC mediante la función "escribirPuerto". **Esta función fue modificada** para realizar llamadas al PIC mediante comunicación USB. La llamada REQUEST que aparece en el esquema, envía un comando al PIC y recibe otro comando de confirmación, que debe tener el mismo código al ser un ACK. De esta forma se controla que el comando ha sido correctamente procesado.

3.3.1.1.2 Implementación del Pic_comm de motores

En un principio las funciones que comunicaban directamente con el PIC fueron fácilmente modificables, adaptándolas a las llamadas propias de la nueva librería que permitía la comunicación USB. Sin embargo, el módulo no se comportaba como se esperaba, deteniendo además en numerosas ocasiones el flujo habitual de la aplicación. Era necesario, pues, analizar minuciosamente todo el proceso para detectar cuáles eran los problemas y poder solucionarlos.

Es necesario indicar que en la anterior versión con comunicación serie del proyecto de años pasados, se enviaban los mismos datos alrededor de 500 veces con la seguridad de que alguno de esos envíos resultaría exitoso. Esto, evidentemente, no ofrecía ninguna seguridad de que los datos se pudieran enviar a la primera tras cambiar la comunicación a USB.

Por lo tanto, y después de comprobar con distintas configuraciones y diversos cambios que el módulo efectivamente no funcionaba como debería, se decidió realizar un estudio profundo partiendo desde cero, ya que la traza con todos los módulos conectados era realmente compleja (teniendo en cuenta además la gran cantidad de hilos corriendo al mismo tiempo).

- El primer paso fue aislar el módulo "Pic_comm" para probar la comunicación con el PIC, que en todo momento estaría conectado al PC. Pero esto presentaba un problema: al no conectar el módulo "Motores", los puertos de las velocidades estarían siempre vacíos (ver diagrama anterior), por lo que los hilos "thr_velocidad_izq" y

"thr_velocidad_der" no podrían introducir ningún comando en la cola común. Se decidió, por tanto, crear otros dos nuevos hilos productores independientes que generasen comandos para introducirlos en la cola.

- Se probó inicialmente con sólo un hilo productor, observando un buen comportamiento por parte del PIC cuando se enviaba siempre el mismo comando. El siguiente paso era el envío de comandos distintos por parte de un solo hilo. En este punto observamos que el PIC frecuentemente respondía un comando distinto al que se le había enviado. El efecto era como si los comandos se "mezclasen":

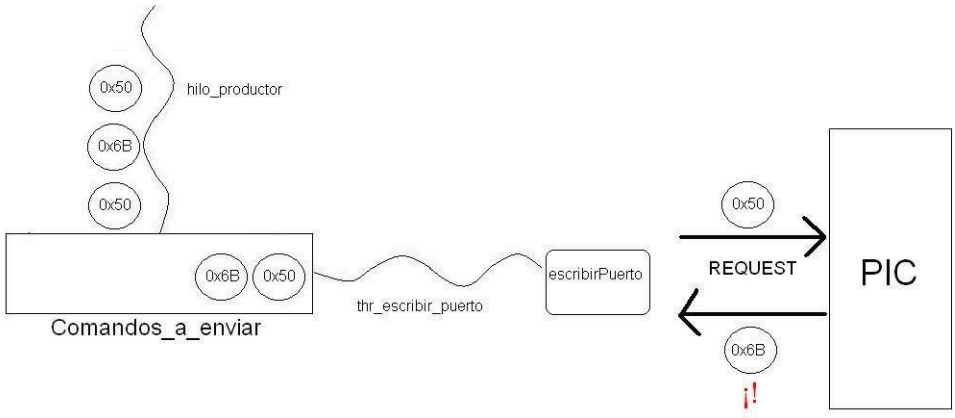


Figura 60 Problemas con un solo hilo productor

En el gráfico, por ejemplo, el hilo productor únicamente envía de forma alterna los comandos 0x50 y 0x6B. Un buen comportamiento por parte del PIC supone reenviar como ACK el mismo comando que hubiera recibido. Sin embargo, como hemos comentado, en ocasiones reenviaba un comando diferente al esperado. Este problema debía ser resuelto.

- Como solución a esto, se pensó en dormir el hilo "thr_escribir_puerto" 100 milisegundos para que le diese tiempo a procesar la petición evitando así interferencias, pero esto tampoco resultó. A continuación, se probó a reenviar varias veces el mismo comando al PIC hasta que éste respondiese adecuadamente. Al tratarse de una llamada síncrona, no debería suponer ningún problema de sincronización entre hilos. En principio esta solución parecía adecuada, pero en ocasiones la aplicación terminaba de forma anómala si no había elementos en la cola

de comandos a enviar. De todas maneras, era claro que el siguiente paso debía ser probar con dos hilos productores.

- Estos dos hilos productores en un principio enviarían un solo comando cada uno, pero distintos entre sí. Las pruebas parecían indicar que el PIC contestaba bien; sin embargo, la aplicación seguía terminando inesperadamente y sin razón.
- Después de hacer muchas más pruebas que no nos llevaron a ningún lado, se creyó conveniente ir hacia atrás y volver al problema del fallo en los reenvíos cuando usamos un solo hilo con dos comandos diferentes, ya que con total seguridad este extraño comportamiento repercutía en la interacción entre dos hilos productores, como era el caso.
- Se usó un único hilo que alternase dos comandos de nuevo, y las trazas mostraron que siempre se tomaba el segundo. Esto hizo pensar que el comando 2 "se escribía encima" del comando 1, por lo que se podía tratar de un problema de referencias en C.
- Los comandos, a pesar de ser datos hexadecimales, se representaban a través de una estructura **char ***, en la que se incluía el código del comando en la *primera* casilla del vector, utilizando el resto para información que *algunos* comandos necesitaban transmitir:

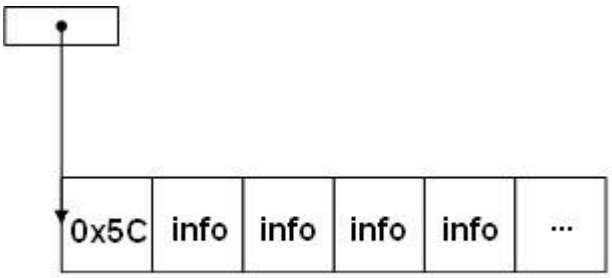


Figura 61 Estructura de los comandos

- Se observó que siempre que se enviaba un comando nuevo, se utilizaba una *misma* variable local de tipo char*. Este hecho, sumado a las sospechas anteriores, hizo pensar que siempre se estaba utilizando la misma referencia, sobrescribiendo el comando a enviar. Esto explicaría que siempre se enviaba el último comando que se había ordenado. Las trazas confirmaron esta hipótesis.
- La solución, por tanto, era tan simple como crear nuevas referencias cada vez que se enviase un comando. Esto se probó y se solventaron de

esa manera muchos de los problemas anteriores; sin embargo, algunas anomalías persistían cuando interactuaban dos hilos diferentes.

- Se observó que la cola "comandos_a_enviar" (ver gráfico en 3.3.1.1.1) suponía una sección crítica que *no había sido tenida en cuenta*. Por lo tanto, la instrucción para meter un nuevo comando en ella se rodeó de un mutex de foma que se controlase este acceso.
- Estos cambios, sumados a algunos otros que se llevaron a cabo para sincronizar la concurrencia, derivó en la perfecta adaptación a comunicación USB, integrando el módulo "Pic_comm" en la aplicación del robot, y consiguiendo que las ruedas respondieran al control manual.

3.3.1.1.3 Introducción al Pic_comm de sensores

El proceso que consulta información al PIC acerca de los sensores se ha simplificado y se ha hecho más robusto. Anteriormente había hilos innecesarios realizando operaciones que se podrían haber ejecutado más fácilmente. Vemos en el siguiente esquema el proceso actual:

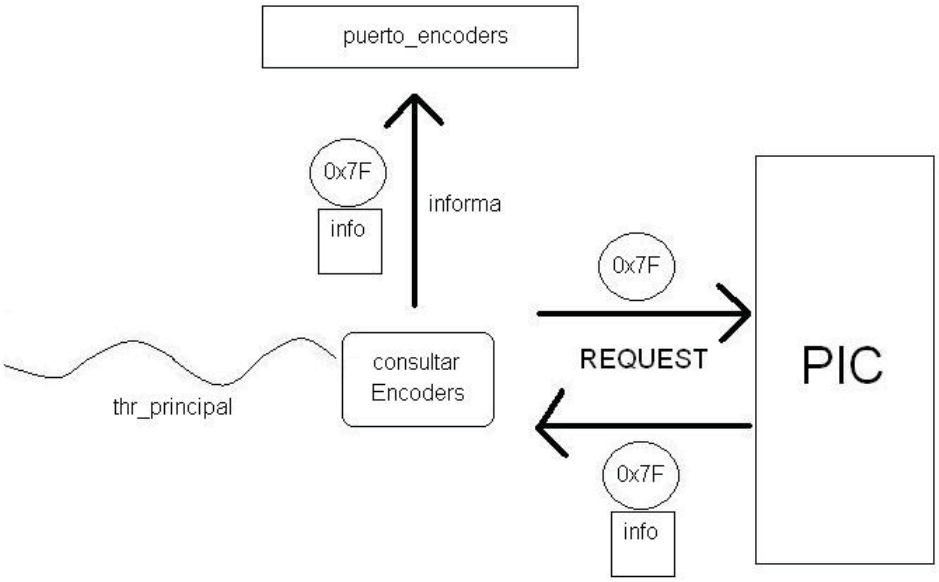


Figura 62 Proceso principal en "Pic_comm" de sensores

El hilo principal realiza consultas constantemente sobre el estado de los encoders. Lo hacemos a través de una función que solicita esta información al PIC. Le enviamos inicialmente el código correspondiente a los encoders (0x7F), y recibimos el mismo código (como ACK), además de la **información asociada.** Esta información se reenvía al puerto de los encoders, que estará conectado al módulo "Encoders" de manera que éste pueda conocer en todo momento el estado de los mismos.

La información devuelta por el PIC acerca del estado de los encoders posee el siguiente formato:

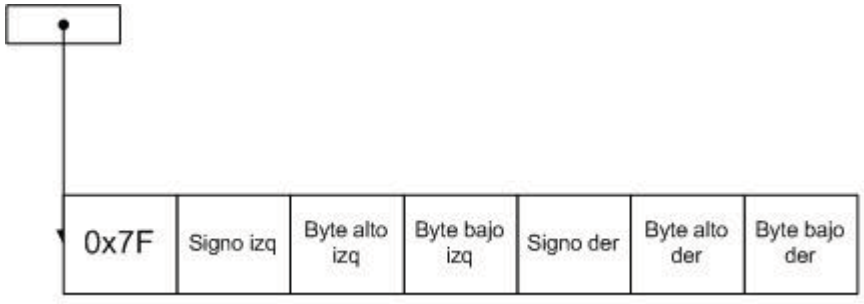


Figura 63 Estructura para el comando de Encoders

El primer byte devuelve el código ACK de la orden de encoders. Los 3 siguientes bytes indicarán el valor de incremento / decremento del encoder izquierdo, mientras que los 3 últimos hacen lo propio con el encoder derecho. Así, por ejemplo, para el encoder izquierdo conoceremos su valor de incremento de la siguiente manera:

*Valor_incremento = ((Byte alto izq * 256) + (Byte bajo izq)) * signo izq*,

siendo *signo izq* +1 o -1, dependiendo del sentido de la rueda.

3.3.1.1.4 Implementación del Pic_comm de sensores

La implementación de esta parte fue más sencilla, al haber aprendido los errores de concurrencia que ocurrían en el proceso de Pic_comm de motores. El proceso se simplificó al haber reducido el número de hilos activos para el Pic_comm de sensores a 1.

Aún así, había un problema que hubo que solucionar. Como ambos procesos en Pic_comm (sensores y motores) utilizaban una misma función para

comunicarse con el PIC, era necesario utilizar exclusión mutua al tratarse de una clara sección crítica.

Los pasos dados en esta implementación fueron similares a los dados en el Pic_comm para motores, trazando hilos de forma separada y descartando errores para llegar al resultado final.

3.3.1.2 Cambio en la representación de comandos

Como se ha comentado anteriormente en el punto 2.5.3.2.2, se cambió la representación de los comandos enviados. Al mandar comandos entendibles directamente por el firmware del PIC, el procesamiento es mucho más simple.

La implementación pasó simplemente por desechar la anterior estructura utilizada por los comandos (explicada en el diseño), y modificar todas las funciones donde ésta se utilizaba. A continuación presentamos los códigos de algunos comandos entendibles por el PIC:

```
#define LED_AM_ON 0x01;
#define LED_AM_OFF 0x02;
#define TESTING 0x30;
#define MOTOR_IZQ_0 0x50
#define MOTOR_IZQ_1 0x51;
#define MOTOR_IZQ_2 0x52;
#define MOTOR_IZQ_3 0x53;
#define MOTOR_IZQ_4 0x54;
#define MOTOR_IZQ_5 0x55;
#define MOTOR_IZQ_6 0x56;

#define MOTOR_DER_0 0x5A;
#define MOTOR_DER_1 0x5B;
#define MOTOR_DER_2 0x5C;
#define MOTOR_DER_3 0x5D;
#define MOTOR_DER_4 0x5E;
#define MOTOR_DER_5 0x5F;
#define MOTOR_DER_6 0x60;
```

Figura 64 Codificación de algunos comandos interpretados por el PIC

3.3.2 Problemas encontrados

Muchos han sido los problemas encontrados a la hora de realizar este cambio, especialmente en el paso a comunicación USB. Los detalles del proceso de implementación con las mayores dificultades encontradas se pueden consultar en el apartado 3.3.1.1.2 y el apartado 3.3.1.1.4.

La parte más complicada fue la integración del nuevo módulo con el resto del proyecto. Se realizaron infinidad de ejecuciones con distintos archivos XML de configuración de módulos para llevar a cabo diversas pruebas.

Eclipse (entorno de desarrollo utilizado a lo largo del proyecto), por otro lado, no posee un buen depurador para C. Esto, sumado a la dificultad de trazar procesos concurrentes, hizo que esta parte fuera especialmente complicada, al tener que utilizar instrucciones tipo *printf* para saber qué hacía exactamente la aplicación en cada momento.

El mayor problema de todos fue tener que modificar la implementación que había hecha en el proyecto anterior del módulo "Pic_comm". Ésta no contemplaba el envío de los datos de una sola vez, sino que se reenviaran 500 veces mediante comunicación serie, suponiendo de esta forma que de algún modo llegarían. Por esta causa, anteriormente no influía demasiado el usar las mismas referencias o el hecho de no cuidar algunas secciones críticas. Estos puntos, sin embargo, eran vitales para el correcto funcionamiento de nuestro módulo al adaptarlo a USB.

Hemos tenido que hacer frente, en definitiva, a una implementación de otro proyecto anterior producto de una mala ingeniería, lo que ha provocado un retraso considerable en una adaptación de comunicación que debería haber sido trivial.

4 Integración

4.1 Introducción

4.2 Integración del control remoto

En esta parte podemos distinguir 3 aplicaciones distintas: la cliente, la servidor y la de alertas, estando distribuidas. En un mismo terminal (el robot) podemos encontrar la aplicación que actúa como servidor y la de alertas al técnico, mientras que en otro/s terminal/es se halla/n la aplicación cliente. Vemos todo el proyecto de control remoto distribuido en el siguiente gráfico:

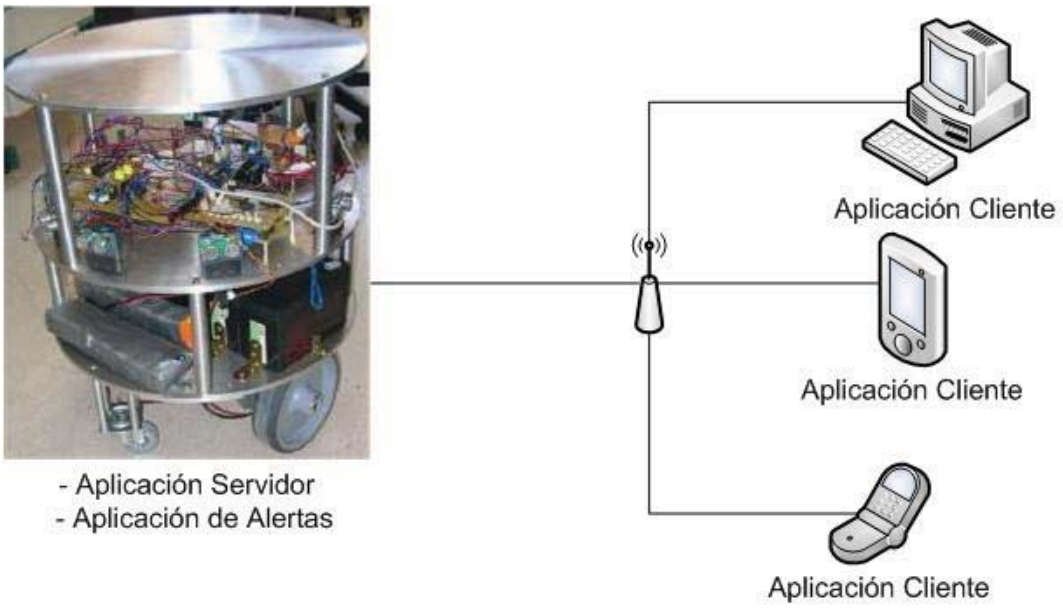


Figura 65 Distribución de las aplicaciones

4.2.1 Aplicación cliente – servidor

Como comentábamos anteriormente, el "*Control remoto*" en si, se divide en dos componentes principales, el Servicio Web Java, y el módulo servidor de Sockets en el Robot.

El servicio web apenas requería integración con la implementación antigua, aunque sí requería un estudio del formato de los mensajes que tenían que enviarse al Robot, para reutilizar el código existente del Robot lo máximo posible.

En cuanto al servidor de Sockets del Robot, sí requería integración con el resto de la aplicación.

El servidor de sockets antiguo seguía este esquema de conexiones:

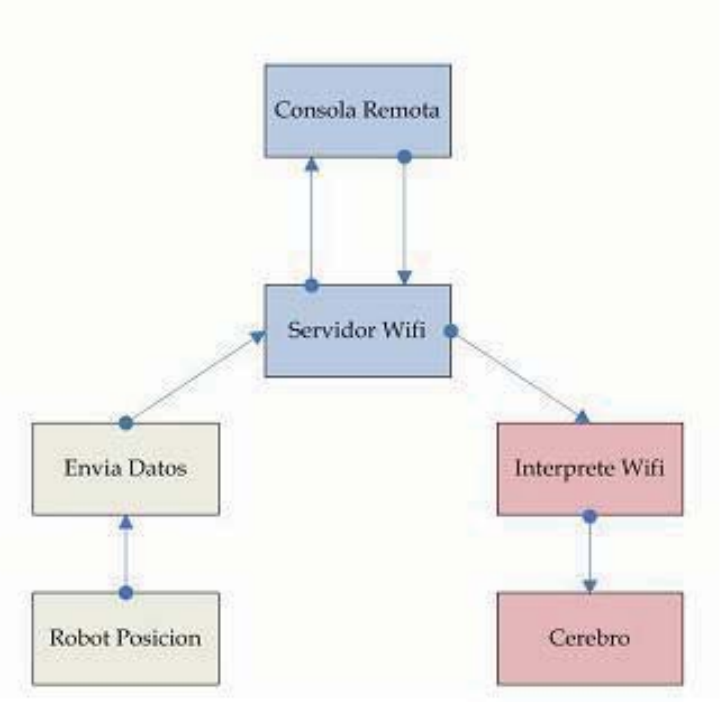


Figura 66 Conexiones del antiguo servidor de Sockets

Al haber heredado la estructura de los comandos enviados, se ha reutilizado por completo el módulo "Interprete WiFi", con lo cual la conexión es idéntica.

La nueva especificación del diseño de Sockets, tan solo requería comunicación entrante con respuestas atómicas, debido al carácter

síncrono de la comunicación y nunca el envio de información al servidor Web que este no hubiera solicitado, con lo cual se prescinde por completo del módulo “Envía Datos”.

No obstante, el servicio Web seguía requiriendo la posicion y orientación actuales del robot, con lo cual se debía crear una conexión con el módulo “Robot Posicion”. Por lo tanto se simplificó el diseño de las interconexiones, creando una conexión directa del módulo “Robot Posición” al módulo “Servidor”, con el fin de mantener esta información actualizada en variables de clase del Servidor de Sockets.

El nuevo esquema de interconexiones tiene este aspecto:

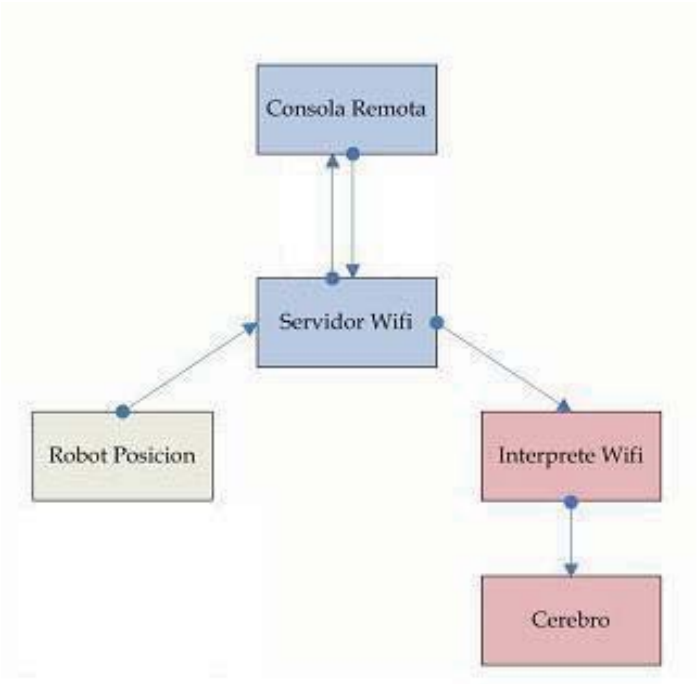


Figura 67 Conexiones del nuevo servidor de Sockets

La interconexión entre los módulos se especifica en el fichero XML de configuración, que detallaremos en el apartado de interconexión de módulos.

4.2.2 Aplicación para alertar al técnico en caso de nivel bajo de batería.

La integración de esta aplicación no supuso ninguna complicación, debido a que funciona de manera totalmente *independiente* en la máquina que actúa como servidor (esto es, el robot). Basta con instalar la aplicación en el terminal seleccionado y ejecutarla.

4.3 Integración del sistema de localización mediante visión artificial

Hemos hablado a lo largo de toda la memoria, que el sistema de visión diseñado e implementado es autónomo y se ofrece en forma de API, para poder ser integrado en cualquier sistema.

Así, se ha hecho utilización de la arquitectura del robot aprovechando el sistema de configuración con ficheros XML, y la capacidad de interconexión de módulos con colas para obtener los datos de entrada necesarios y servir los datos calculados a otros posibles módulos.

El trabajo de integración en sí con el Robot, no solo suponía acoplar nuestro nuevo módulo, sino también hacerlo cohabitar con otros módulos que ya se utilizaban para calcular la situación del móvil de forma aproximada, tales como el módulo de encoders o el módulo brújula. Estamos hablando por lo tanto de **integración multisensorial**.

El módulo "Webcam" está compuesto por dos hilos principales:

- Hilo de recepción de la situación aproximada, calculada por un medio externo.
- Hilo de cálculo de la posición y orientación con cierta fiabilidad, y escritura de la misma en el puerto de salida del módulo.

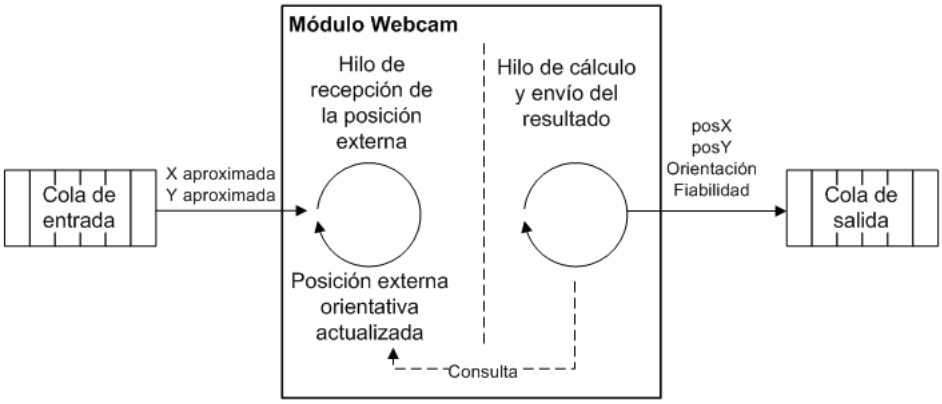


Figura 68 Aspecto del módulo "Webcam"

La integración del módulo supone, tanto la creación de XML que lo define, configura e interconecta con otros módulos, y la modificación de módulos existentes para implementar la integración sensorial.

4.3.1 XML de configuración

La parte del fichero XML de configuración que define el módulo es la siguiente:

```
<modulo nombre="Webcam" ruta="Webcam" estado="activo">
  <argumento nombre="ruta_focos_xml">xml/focos.xml</argumento>
  <argumento nombre="debug">0</argumento>
  <argumento nombre="tam_cola_info">5</argumento>
  <argumento nombre="mostrar_ventanas">1</argumento>
  <argumento nombre="umbral_hough">100</argumento>
  <argumento nombre="umbral_canny_1">20</argumento>
  <argumento nombre="umbral_canny_2">40</argumento>
  <argumento nombre="orientacion">0</argumento>
  <argumento nombre="tiempo_captura">200</argumento>
  <argumento nombre="recibe_pos">1</argumento>
  <argumento nombre="envia_info">1</argumento>
  <argumento nombre="escala">1</argumento>
  <argumento nombre="fiabilidad_cambio_punto">5</argumento>
  <argumento nombre="fiabilidad_perdida_lineas">10</argumento>
```

```
  <argumento nombre="error_rho">10</argumento>
  <argumento nombre="error_theta">20</argumento>
  <conexion
       origen="puerto_webcam_pos_ent"
       destino="puerto_robot_posicion_sal"
       cola="5">Robot_Posicion</conexion>
  <salida puerto="puerto_webcam_sal">5</salida>
</modulo>
```

Figura 69 Configuración XML del módulo "Webcam"

Como podemos observar, se definen los siguientes argumentos:

- **ruta_focos_xml:** Ruta relativa del fichero XML de configuración de la localización en el mapa de los focos que iluminan la habitación en la que se mueve el Robot.
- **debug:** Atributo binario que indica si se desea o no que se muestre información de depuración en la consola del Robot.
- **tam_cola_info:** Atributo entero que indica el tamaño de la cola circular en la que se coloca el resultado para servirlo al puerto de salida.
- **mostrar_ventanas:** Atributo binario que indica si se desea o no que se muestren las ventanas en las que se realiza el tratamiento de las imágenes capturadas.
- **umbral_hough:** Umbral de Hough para el algoritmo de extracción de rectas de la imagen.
- **umbral_canny_1 y 2:** Umbrales de canny, para el algoritmo de extracción de bordes de la imagen.
- **orientacion:** Orientación inicial en grados desde la que parte el robot.
- **tiempo_captura:** Retardo entre capturas para el tratamiento de estas.
- **recibe_pos:** Atributo binario que indica si se desea la recepción de la posición calculada por un medio externo.
- **envia_info:** Atributo binario que indica si desean colocar en el puerto de salida los datos calculados por el algoritmo.
- **escala:** El algoritmo trabaja con distancias medidas en pixeles. Para hacer independiente el algoritmo de la distancia de la

cámara al techo, el envío o recepción de información se hace en centímetros, con lo cual la escala se configura en [pixeles/cm].

- **fiabilidad_cambio_punto:** Pérdida de fiabilidad cuando se recalcula el punto de referencia.
- **fiabilidad_perdida_lineas:** Pérdida de fiabilidad cuando no se pueden extraer rectas de la imagen. Supuestamente debería ser mayor que la perdida de fiabilidad con el cambio de punto. Se debe tener en cuenta que la fiabilidad oscila en el conjunto de valores [0,100].

También se define una conexión entrante de la fuente de calculo de posición externa, en nuestro caso el módulo "*Robot_Posición*", que explicaremos en el apartado de integración multisensorial, y un puerto de salida para la escritura de la solución del algoritmo en cada paso.

4.3.2 Integración multisensorial

Como ya se comentó en el apartado de especificación y diseño, el proyecto del que se partió, ya calculaba una posición aproximada, con la utlización de la cuenta de los encoders, y la orientación con los mismos encoders o con el módulo "*Brujula*", que controla una brujula electrónica.

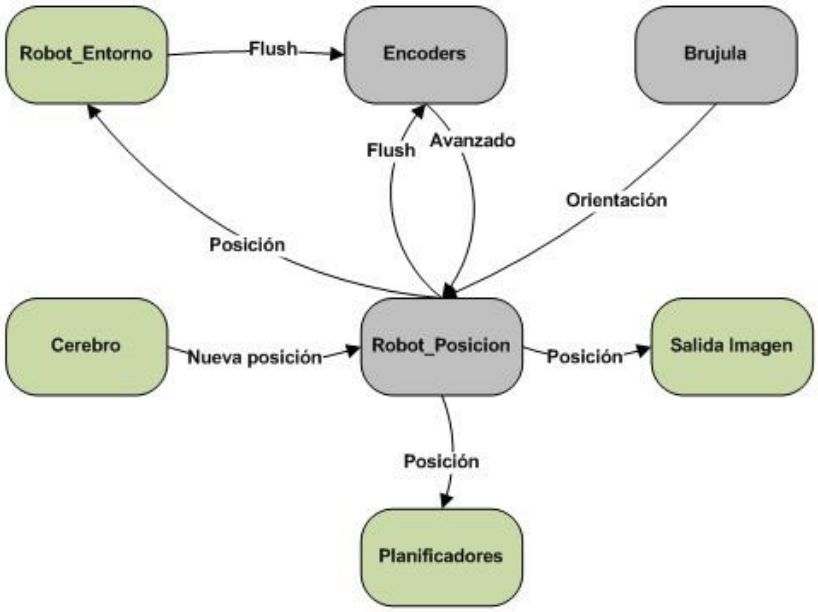


Figura 70 Interconexión de módulos de bajo y alto nivel

Integrar un módulo nuevo que realizara un cálculo que ya aportaban otros módulos, como ocurre con la Webcam, supone tanto la creación de las conexiones pertinentes, como la adición de soporte a los módulos existentes que manejan las entradas y las salidas de la Webcam.

Como podemos ver en la siguiente ilustración, el modulo que realiza la integración multisensorial es "*Robot_Posicion*", y debe ser este por lo tanto, quien decida finalmente cual es la posición del robot en cada momento, realizando los cálculos pertinentes con los datos aportados por la brújula, los encoders y la webcam.

Se han creado por lo tanto dos nuevas conexiones entre el módulo "*Robot_Posicion*" y el módulo "Webcam", una de ellas sirve la posición orientativa a la "*Webcam*" y la otra devuelve a "*Robot_Poscion*" los datos calculados por la "*Webcam*".

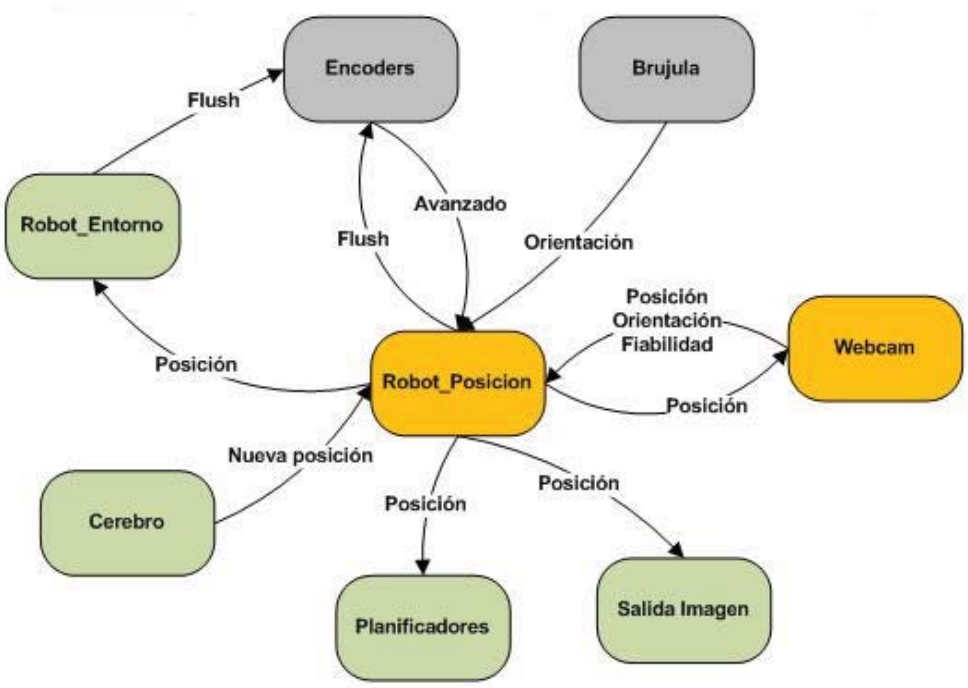


Figura 71 Integración del módulo "Webcam"

Como comentábamos en párrafos anteriores, se ha tenido que incluir una nueva funcionalidad al módulo "*Robot_Posicion*" para recoger los datos de la Webcam, y combinarlos con los de los Encoders y la Brujula.

La frecuencia de datos recibidos de la "*Webcam*" es mucho menor que la frecuencia de datos aportados por los módulos "*Encoders*" y "*Brujula*", por

lo tanto se ha creado un hilo en el módulo Robot_Posicion que recoge los datos de la Webcam y activa un flag de datos nuevos para que el método encargado de procesarlos los combine una sola vez con la otra información.

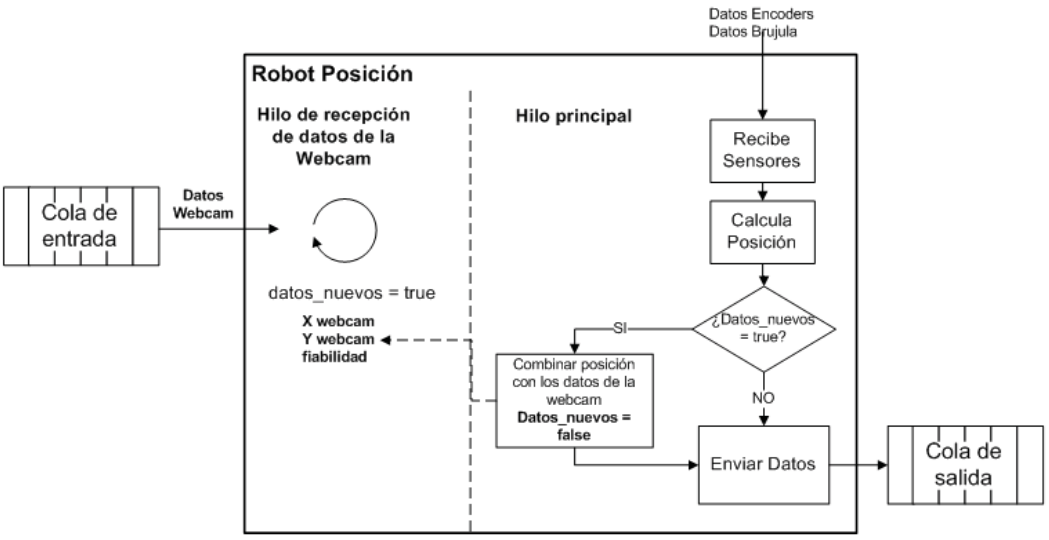


Figura 72 Fusión sensorial al calcular la posición del robot

La funcionalidad de integración sensorial se ha realizado en el método "combinar posición". Este método realiza una media ponderada entre los datos de los encoders y la brújula, y los datos de la webcam, haciendo uso de la fiabilidad enviada por esta última.

La fórmula de cálculo de la posición y orientación por lo tanto es la siguiente:

```
xCombinado = ((posXWebcam * fiabilidad) +
            (posXSensores*(100-fiabilidad))) / 100;

yCombinado = ((posYWebcam * fiabilidad) +
            (posYSensores * ( 100-fiabilidad ))) / 100;

orientacion = ((orientacionWebcam * fiabilidad) +
             (orientacionSensores * (100-fiabilidad)))
            / 100;
```

4.4 Integración del pic

El módulo "Pic_comm" es utilizado tanto para manejar el PIC de motores como el de sensores. Las conexiones existentes con este módulo para los dos casos son las siguientes:

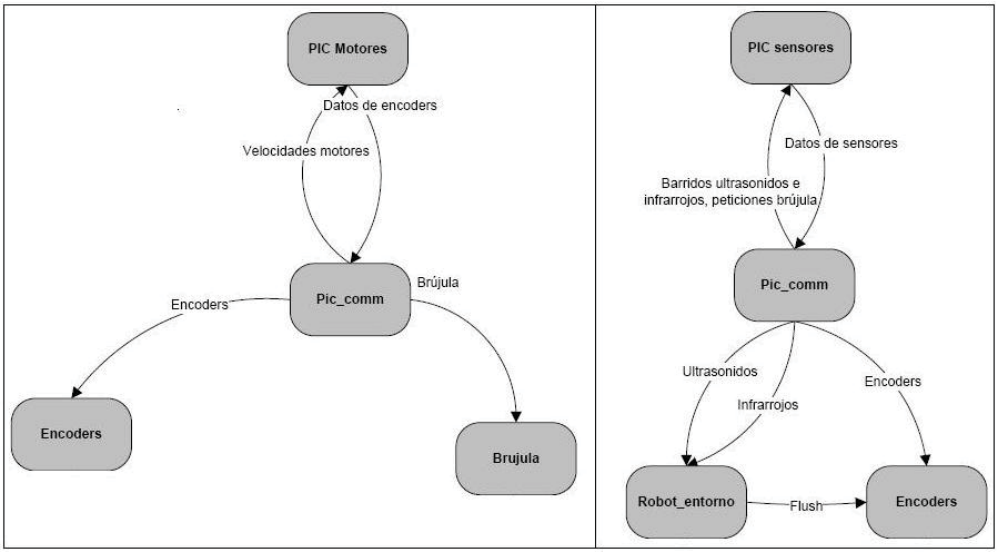


Figura 73 Integración del módulo "Pic_comm"

Al integrar el módulo, se han tenido que respetar todas las conexiones previas para mantener el comportamiento correcto.

Este módulo ya existía previamente en el proyecto heredado, y estaba integrado con el resto de la arquitectura. Los cambios realizados en él han afectado puramente a su implementación interna, no al diseño. Las mayores modificaciones se han dado entre el módulo "Pic_comm" y el microcontrolador PIC, por lo que estos cambios han sido transparentes para el resto.

En definitiva, lo más destacado de esta parte del proyecto no es tanto su integración (ya que se mantiene las anteriores conexiones y configuraciones) como su diseño e implementación, explicado todo ello en los apartados correspondientes.

5 Resultados del Proyecto

5.1 Pruebas

5.1.1 Pruebas de control remoto

La realización de pruebas sobre el control remoto, incluía la realización de pruebas de comunicación atómicas sobre las distintas partes que lo componen, para después realizar pruebas del sistema completo y pruebas sobre otras arquitecturas, como por ejemplo, un teléfono móvil o un Pocket PC.

A raíz de los dos flujos de conexiones existentes (posible fuente de errores de conexión), se implementó un comando especial para realizar un test de conexiones, y saber en todo momento, que ambos flujos funcionaban correctamente.

5.1.1.1 Prueba de comunicación Socket

Primera prueba de la comunicación de sockets con el robot con un cliente sencillo, no integrado con el servicio Web:

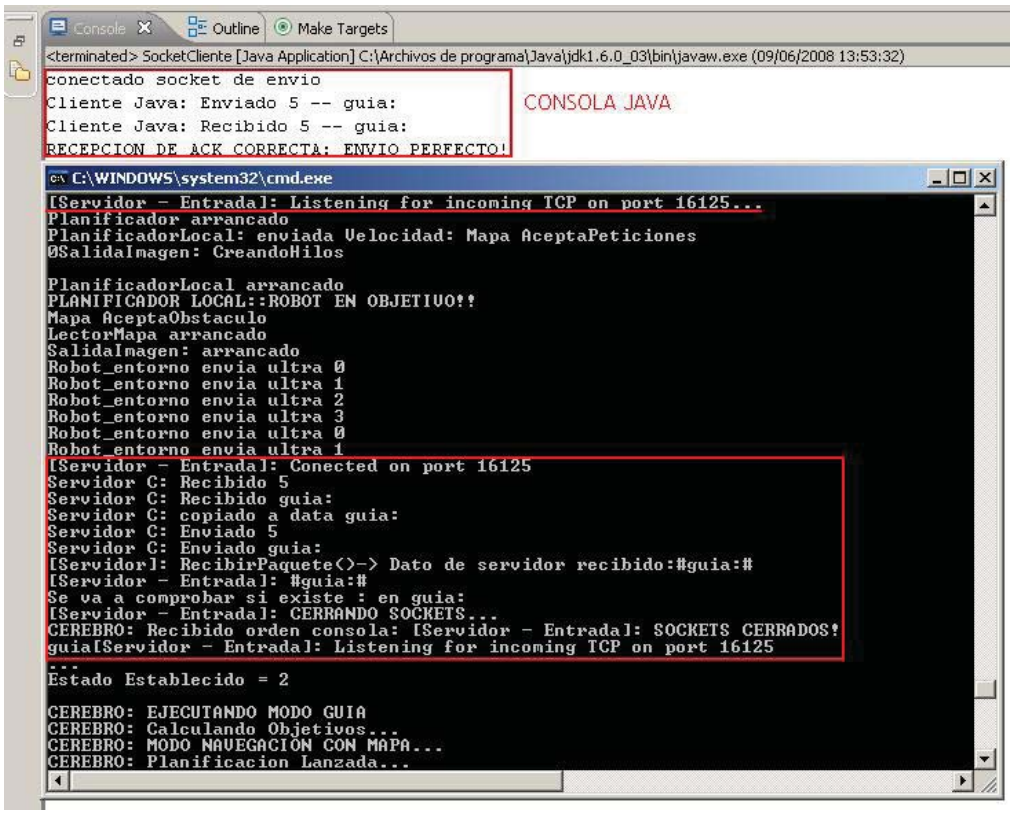


Figura 74 Prueba de comunicación Socket

Se realizó el envío de un comando, en este caso “guia:” para comprobar que el robot lo recibía, lo gestionaba, y enviaba la respuesta de ACK correctamente.

En la ilustración se observa, como el cliente de Socket Java envía el comando, el robot lo recibe, lo envía al interprete wifi y por lo tanto al cerebro (notificando este la recepción), y responde con el mismo comando, para que el cliente lo compare con lo que envió, y escriba por consola que el envió fue correcto.

5.1.1.2 Prueba de comunicación HTTP asíncrona

Pruebas de conexión asíncrona de un cliente (Navegador Web) con la interfaz de llamadas asíncronas del servicio Web:

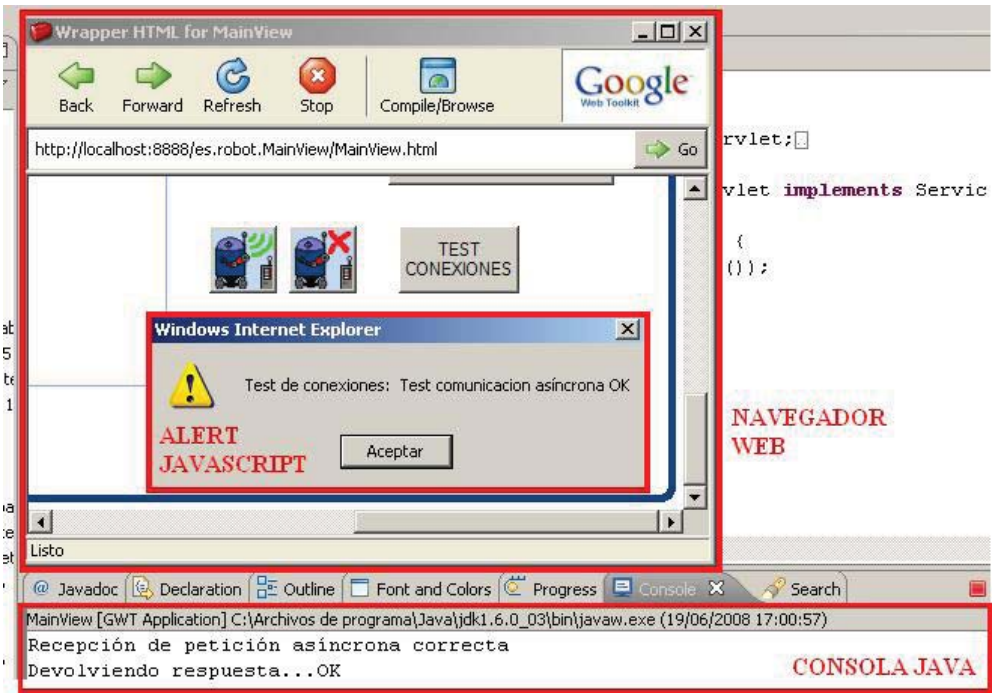


Figura 75 Prueba de comunicación http asíncrona

En la figura 75 se puede observar, como haciendo uso del botón "Test Conexiones", se envía una petición asíncrona al servidor, quien muestra por su consola un mensaje con la recepción del mensaje, y devuelve la cadena "OK", para que el cliente la muestre por pantalla en forma de "Alert" JavaScript.

Cabe destacar que esta fue la primera versión de la interfaz gráfica del cliente, y el botón de "Test" tan solo realizaba la prueba de comunicación asíncrona.

5.1.1.3 Prueba de comunicación completa

Prueba completa de comunicación desde un cliente (Navegador Web) conectado al servidor, y envío de la orden recibida por Sockets al Robot:

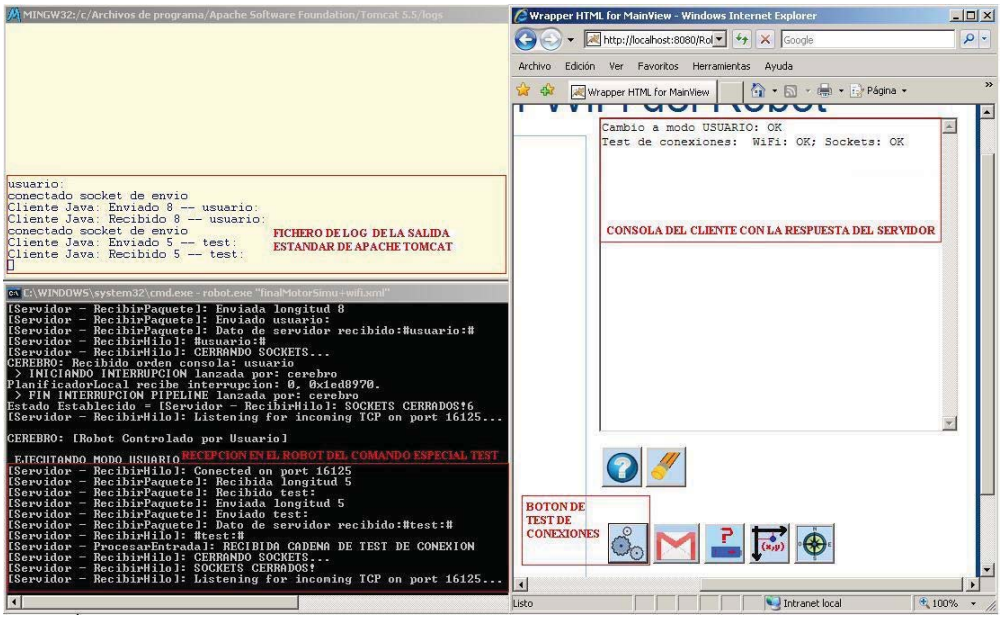


Figura 76 Prueba completa de comunicación

Prueba sobre la versión final del Servicio Web, ya instalado en una máquina, sobre un contenedor de JSP's y servlet's Apache Tomcat.

En la mitad derecha se encuentra en ejecución un cliente sobre el navegador web "Internet Explorer", y en la mitad izquierda, en la parte superior el fichero de log de Tomcat de la salida estándar, y en la parte inferior la aplicación del Robot.

Se observa, como trás pasar a modo usuario (se muestra tanto en la consola del cliente, como en el fichero de log de Tomcat en el servidor, y en el robot), se ha pulsado el boton del test de conexiones. El servidor recibe la petición asíncrona, conecta via Socket con el Robot, le envía el comando especial de test "test:" y este le responde con la misma cadena. El servidor compara las cadenas, comprobando que los sockets funcionan correctamente, y devuelve al cliente la respuesta del test de conexiones, mostrando este la respuesta en la consola.

5.1.1.4 Pruebas del servicio en diferentes arquitecturas

Prueba de la visualización y funcionamiento del servicio Web final desde un terminal movil (Nokia E65) con una red wireless Ad-hoc:

Figura 77 Pruebas del servicio desde un teléfono móvil

En la figura 77, se muestra como desde el terminal móvil, se realiza un cambio en la posición del robot, utilizando un comando de modo autónomo, y en la consola aparece la respuesta del servidor tras ser la orden enviada al servidor, y posteriormente al Robot por Sockets.

La figura 78 muestra los controles de modo usuario en el terminal. En este caso, se envió el comando que hace que el Robot gire hacia la derecha mientras avanza.

Figura 78 Control remoto del robot desde un teléfono móvil

5.1.1.5 Prueba del flujo de datos en el servidor de Sockets del Robot

Prueba de escritura de la orientación y la posición en el Servidor de Sockets del robot, con varias peticiones del cliente de la posición actual:

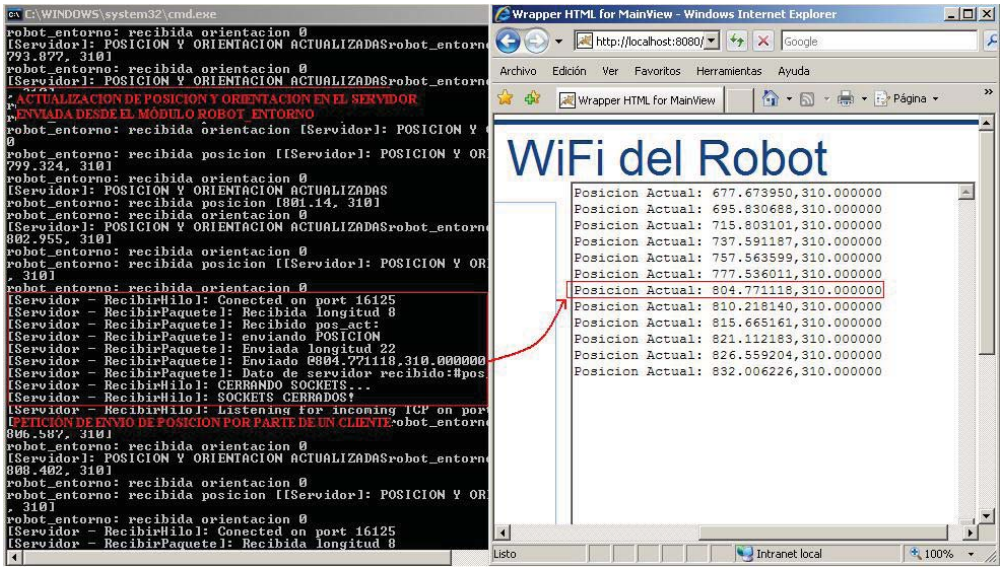


Figura 79 Prueba del flujo de datos en el servidor de Sockets del robot

Se pueden observar varias actualizaciones en la aplicación del Robot, de la posición y la orientación (se muestra en la cadena "[Servidor]: POSICION Y ORIENTACION ACTUALIZADAS").

Desde el cliente en el navegador web se hacen varias consultas de la posición. Se muestra una de ellas, en la que la posición mostrada por la consola remota es [804.77, 310.00].

5.1.1.6 Prueba de envío de alertas

Una vez fue testeado y prototipado el componente "Comunicador", se procedió a integrarlo con el resto de la aplicación. Su funcionalidad la proporciona a través del botón

Figura 80 Botón de envío de mensajes

Al pulsar dicho botón, aparece un pop-up que nos permite seleccionar el tipo de alerta (sms y/o mail), y el tipo de mensaje que queremos enviar:

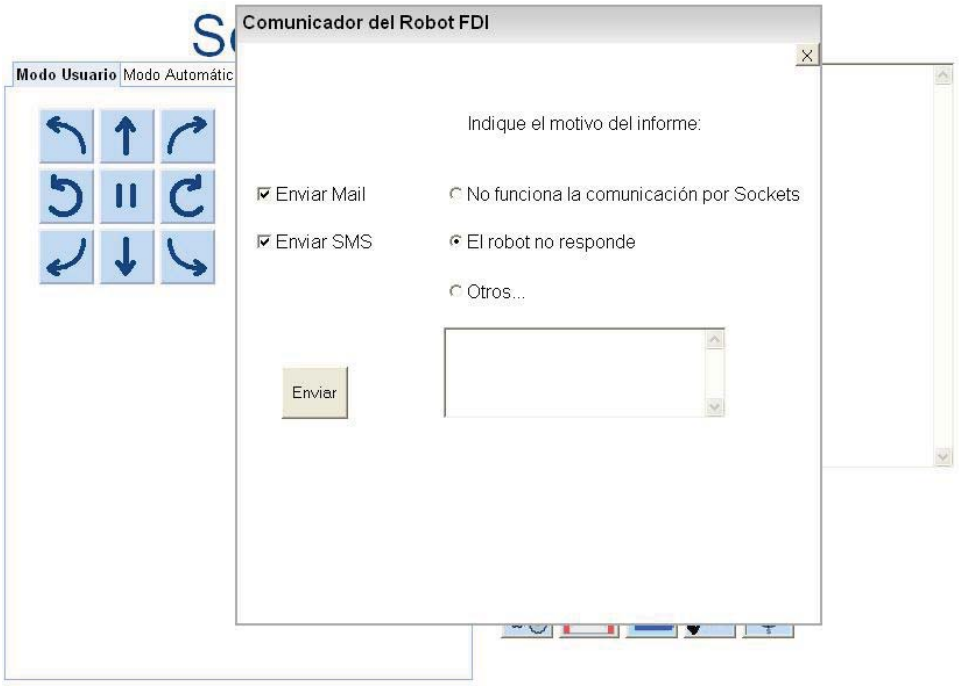


Figura 81 Pop - up para envío de mensajes

Al pulsar el botón “Enviar” se procede al envío, en este caso, del mail y el SMS al técnico responsable. En la consola de la aplicación se imprimirá unos mensajes que nos indicará si ha sido posible realizar la conexión adecuadamente (en el dibujo se han enmarcado en rojo):

Figura 82 Confirmación del envío de mensajes

Como podemos comprobar en la figura 83, el técnico recibió el mail correctamente por parte del robot:

Figura 83 Recepción del e-mail

Y también recibió correctamente el sms en su teléfono móvil:

Figura 84 Recepción del sms

5.1.1.7 Prueba de carga de batería

Tras la realización de diversos prototipos con el componente "Batería", éste se integró completamente en la arquitectura de la aplicación cliente – servidor, ofreciendo su funcionalidad a través del botón

Figura 85 Botón de consulta de batería

Al pulsar dicho botón, se ofrece un pop-up indicando el estado de la batería. Si ésta se está descargando, aparecerá algo de este estilo:

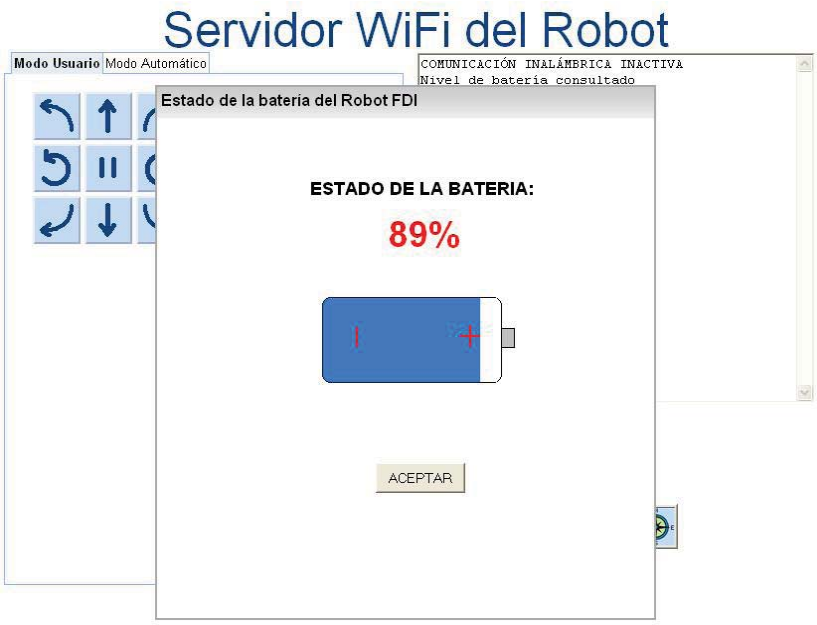


Figura 86 Pop – up que muestra el estado de la batería descargándose

Si por el contrario, la batería está cargada y conectada a la red, la aplicación nos informará a través del siguiente pop-up:

Figura 87 Pop – up que muestra el estado de la batería cargándose

5.1.2 Pruebas de visión artificial

5.1.2.1 Pruebas de Visión de bajo y medio nivel.

A continuación se pueden observar los pasos seguidos por el apartado de tratamiento de imágenes para reconocer las líneas rectas partir de la imagen original obtenida por la cámara. Se muestran pruebas con distintos tipos de imágenes capturadas para que se pueda observar como actúa el algoritmo en los diferentes casos.

En primer lugar, se muestra una captura de una imagen idónea para el funcionamiento de nuestro sistema, ya que no presenta ninguna dificultad en su tratamiento:

1) Captación de la imagen:

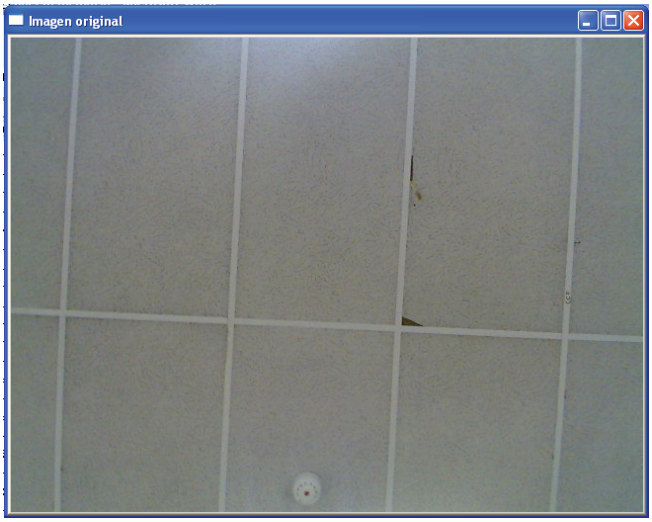


Figura 88 Prueba de captación de la imagen

2) Transformación de la imagen a escala de grises para trabajar únicamente sobre una imagen y no sobre tres (R, G, B) ya que no nos importa el color:

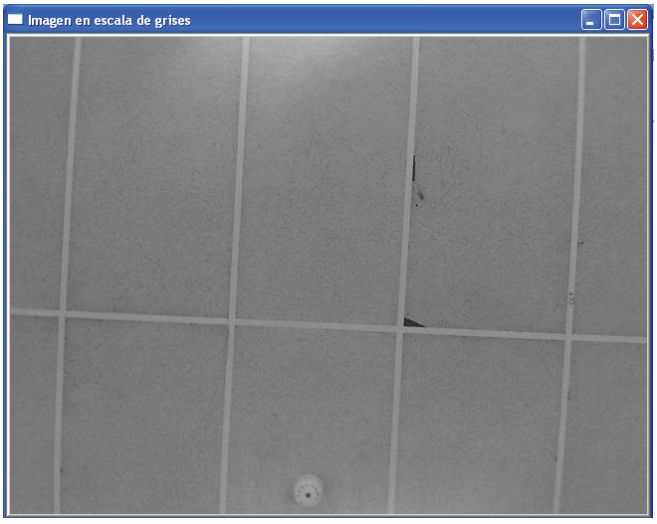


Figura 89 Prueba de transformación a escala de grises

3) Suavizado Gaussiano para eliminar ruido:

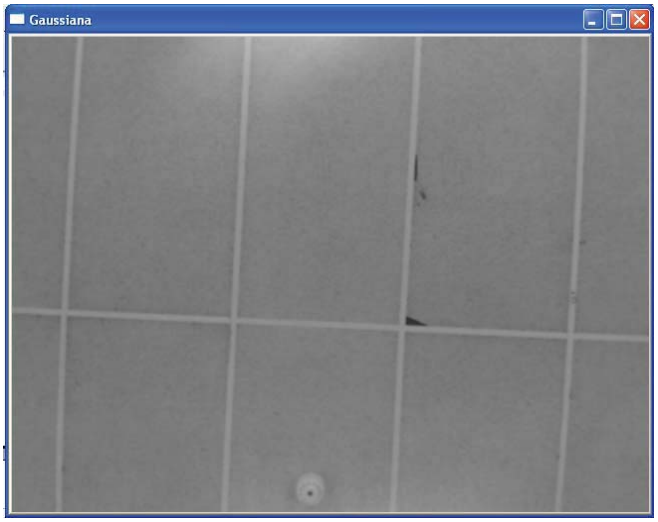


Figura 90 Prueba de suavizado Gaussiano

4) Detector de bordes de Canny:

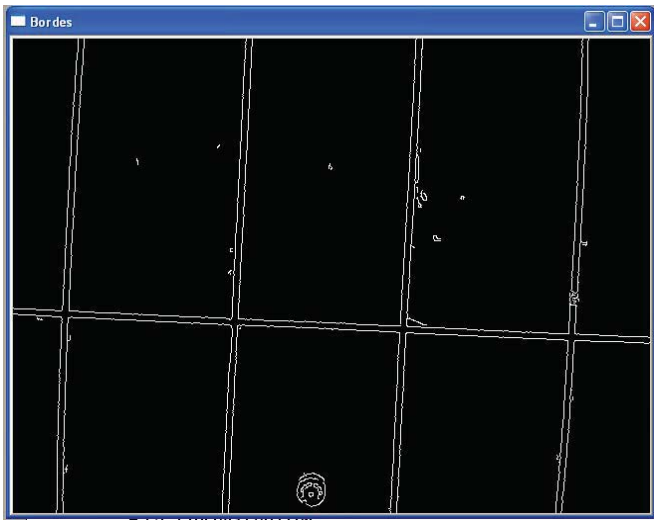


Figura 91 Prueba de detección de bordes

5) Transformada de Hough. En rojo aparecen las líneas rectas detectadas:

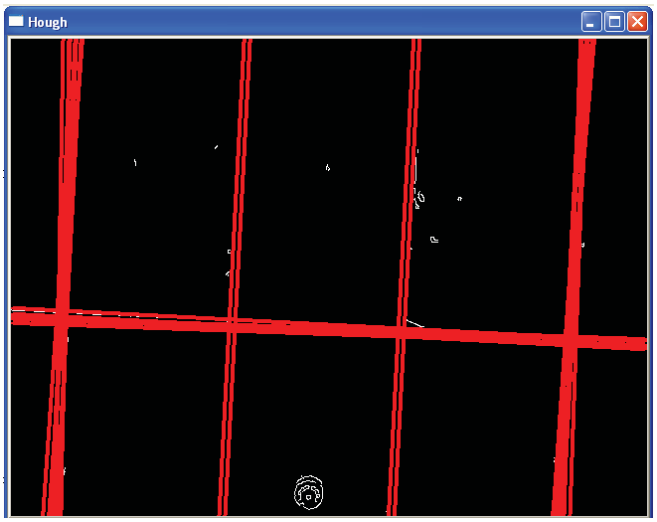


Figura 92 Prueba de transformada de Hough

Seguidamente mostramos cómo actúa la parte de preprocesamiento e interpretación de imágenes cuando la imagen está tomada debajo de una fuente de luz:

1) Captación de la imagen:

Figura 93 Prueba de captación de la imagen bajo un foco

2) Transformación de la imagen a escala de grises:

Figura 94 Prueba de transformación a escala de grises bajo un foco

3) Suavizado Gaussiano para eliminar ruido:

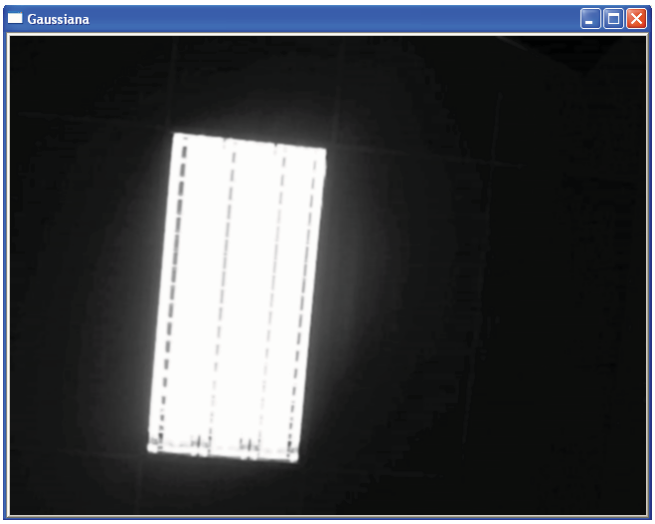


Figura 95 Prueba de suavizado Gaussiano bajo un foco

4) Detector de bordes de Canny. En este caso, la influencia de la luz hace muy complicado que el detector de bordes detecte todas las líneas de la imagen. Sin embargo es suficiente con que encuentre los bordes del

fluorescente, pues sólo con ellos podemos seguir calculando la posición y la orientación. Además, los fluorescentes pueden actuar como balizas absolutas, corrigiendo si existieran pequeños errores en el cálculo de la posición y la orientación:

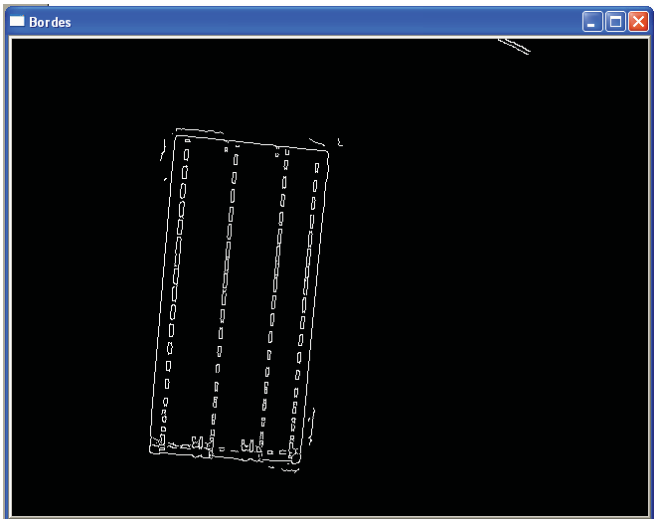


Figura 96 Prueba de detección de bordes bajo un foco

5) Transformada de Hough. En rojo aparecen las líneas rectas detectadas:

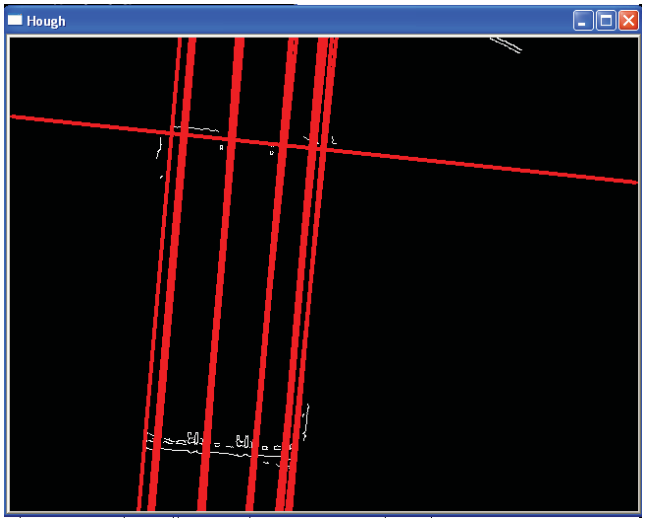


Figura 97 Prueba de transformada de Hough bajo un foco

Por último mostramos una prueba realizada sobre una imagen captada al lado de una ventana y de una fuente de luz para comprobar como responde nuestro sistema ante este tipo de situaciones:

1) Captación de la imagen:

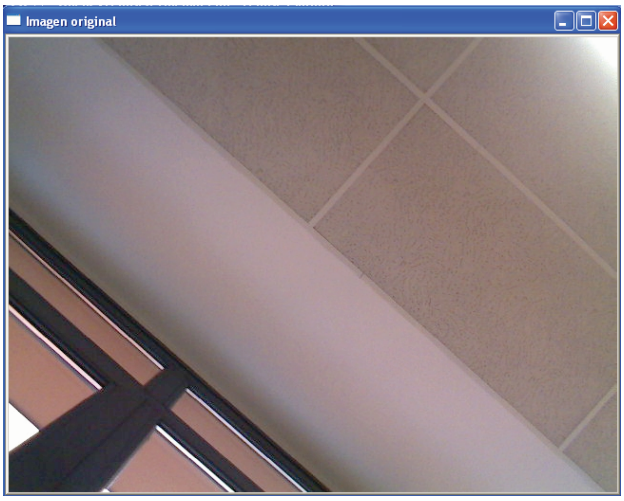


Figura 98 Prueba de captación de imagen al lado de una ventana

2) Transformación de la imagen a escala de grises:

Figura 99 Prueba de transformación a escala de grises al lado de una ventana

3) Suavizado Gaussiano para eliminar ruido:

Figura 100 Prueba de suavizado Gaussiano al lado de una ventana

4) Detector de bordes de Canny:

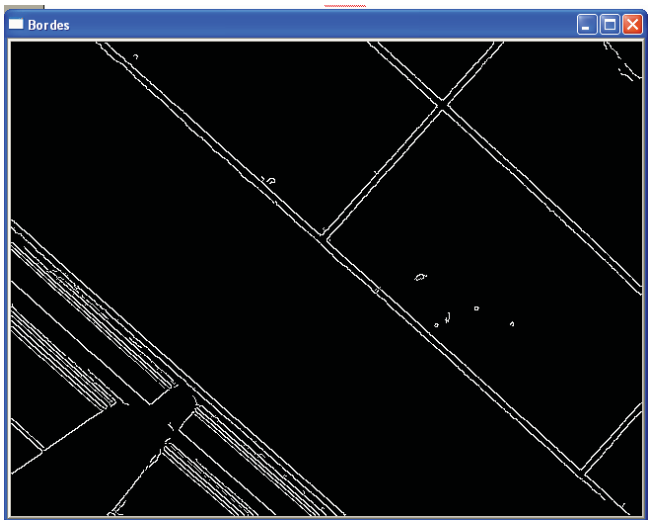


Figura 101 Prueba de detección de bordes al lado de una ventana

5) Transformada de Hough. En rojo aparecen las líneas rectas detectadas. Nos damos cuenta que ha detectado una recta que va de esquina a esquina de la imagen y que no nos interesa. De todas formas, esto no supone ningún problema. Los problemas se darían si no detectase alguna de las rectas con las que vamos a realizar el cálculo de la posición y la orientación:

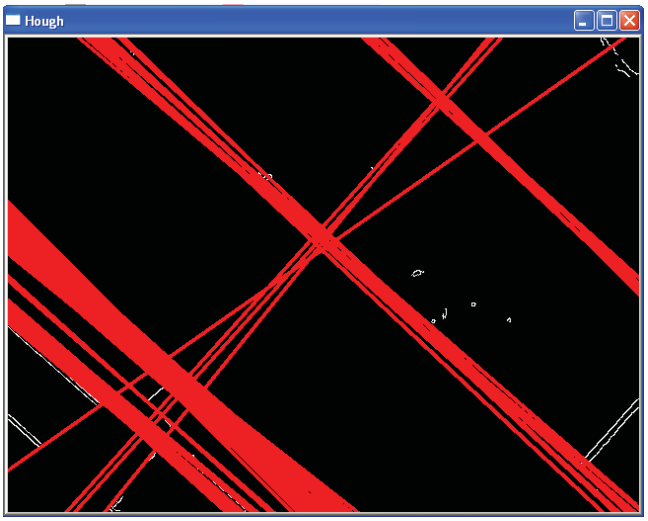


Figura 102 Prueba de transformada de Hough al lado de una ventana

Como hemos podido observar, nuestro sistema se comporta de forma correcta en los casos más habituales.

5.1.2.2 Pruebas de visión de alto nivel

En este apartado se realizarán pruebas para la comprobación de la correcta interpretación de las imágenes capturadas y tratadas previamente.

Para ello se realizarán pruebas de la obtención de la orientación a partir de las rectas halladas mediante la transformada de Hough, y seguidamente pruebas del cálculo incremental, realizando el seguimiento de los puntos del techo, y absoluto de la posición.

5.1.2.2.1 Pruebas de cálculo de la orientación.

Primeramente, se ha realizado una prueba con orientación de 0 grados, para ver si se capta bien la orientación de la línea inicial que servirá como referencia para obtener la orientación en sucesivas capturas.

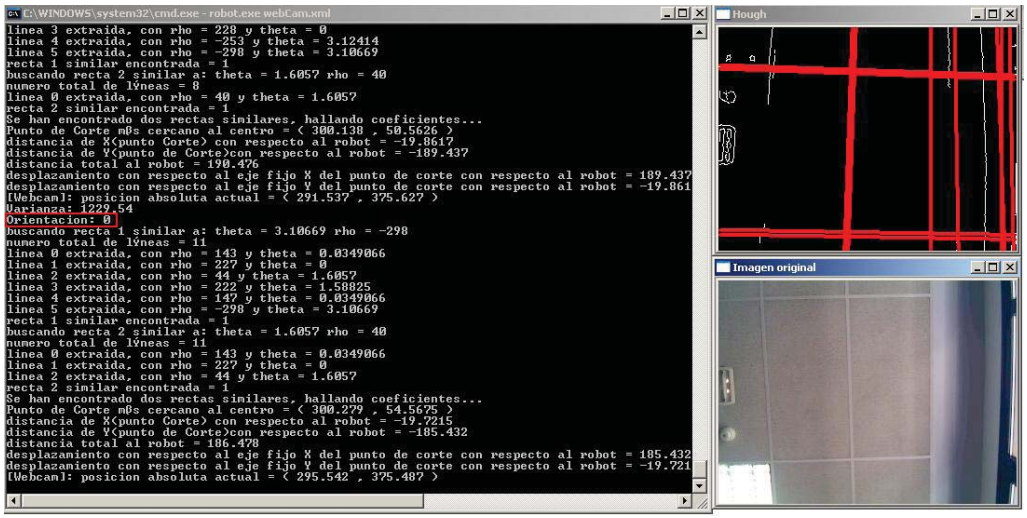


Figura 103 Prueba de visión con 0 grados de orientación

Podemos observar cómo con las líneas formadas a partir del los lados más largos de los rectángulos del techo, paralelas al eje Y de la fotografía, se obtienen 0 grados de orientación. Se puede observar en el recuadro rojo en la consola.

Las figuras 104 y 105 muestran giros a derecha e izquierda respectivamente, y cómo el algoritmo actualiza la orientación correctamente.

En las consolas de las ilustraciones, se puede observar la obtención del resultado recuadrado en un rectángulo rojo.

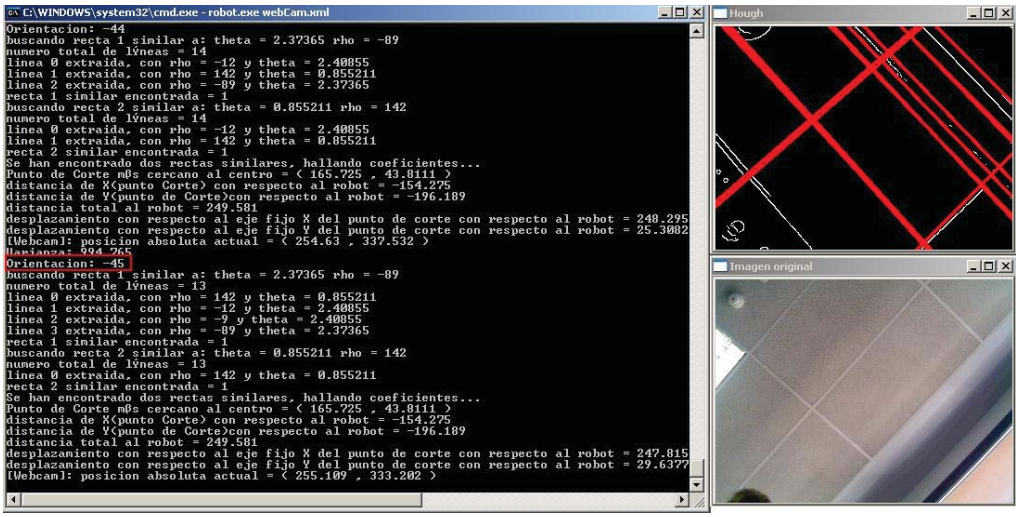


Figura 104 Prueba de cálculo de orientación realizando un giro hacia la derecha.

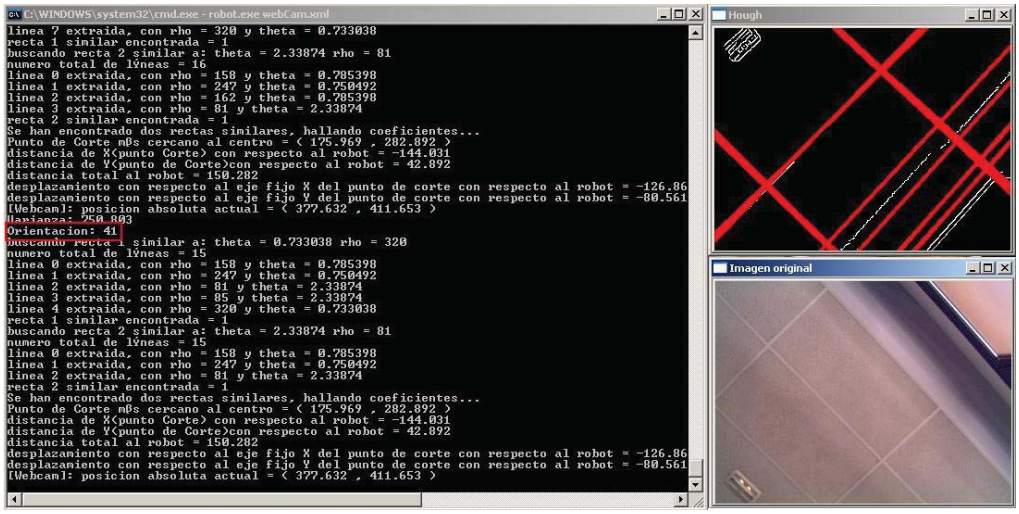


Figura 105 Prueba de cálculo de orientación realizando un giro hacia la izquierda.

La última prueba de la orientación, se trata de un test de la robustez del algoritmo forzando la pérdida de la línea de referencia entre imagen e imagen.

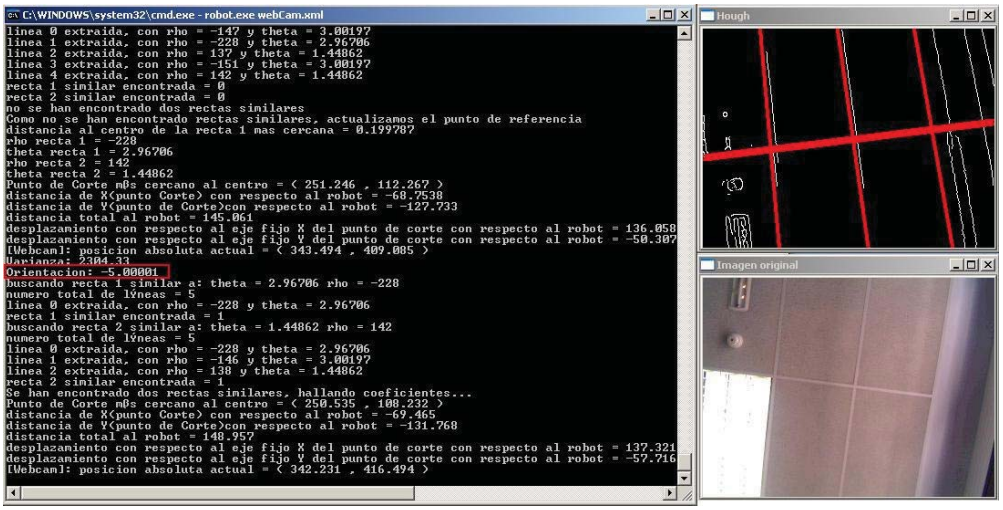


Figura 106 Imagen con la última orientación previa a la pérdida.

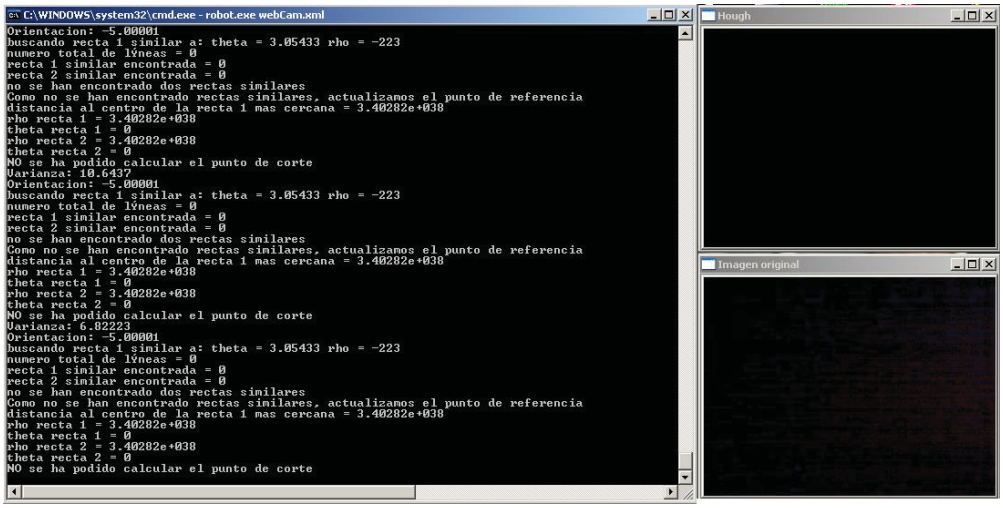


Figura 107 Imagen con pérdida de líneas forzada, tapando intencionadamente el objetivo de la Webcam.

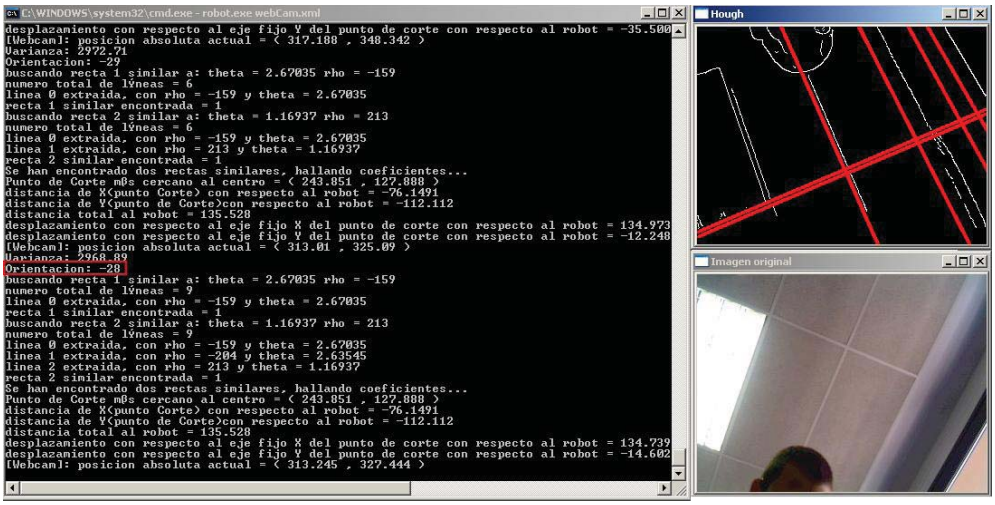


Figura 108 Nuevo cálculo de la orientación tras la recuperación de la línea de referencia.

Se puede observar cómo tras destapar el objetivo de la Webcam, se obtiene una orientación coherente, con respecto a la calculada anteriormente. Además, se muestra también la validez de la prueba para comprobar que aun interponiendo obstáculos en el campo de visión de la Webcam, los cálculos se realizan correctamente.

5.1.2.2.2 Pruebas de cálculo de la posición.

La realización de pruebas sobre el cálculo de la posición, se ha dividido en dos grupos.

Primeramente se han realizado pruebas forzando la fiabilidad del algoritmo del módulo Webcam al 100%; de este modo, se comprueba que la implementación del algoritmo es robusta, pudiéndose utilizar como sistema autónomo.

Para la primera prueba de posicionamiento, se ha realizando un recorrido sencillo en “L” por el pasillo, partiendo de un supuesto punto inicial.

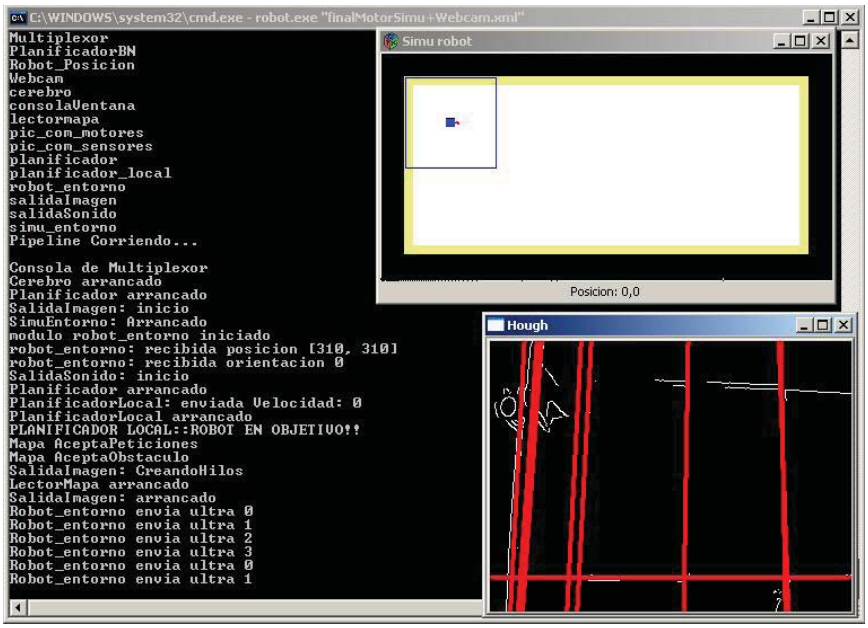


Figura 109 Inicio del recorrido en "L" con la Webcam como único sistema de navegación.

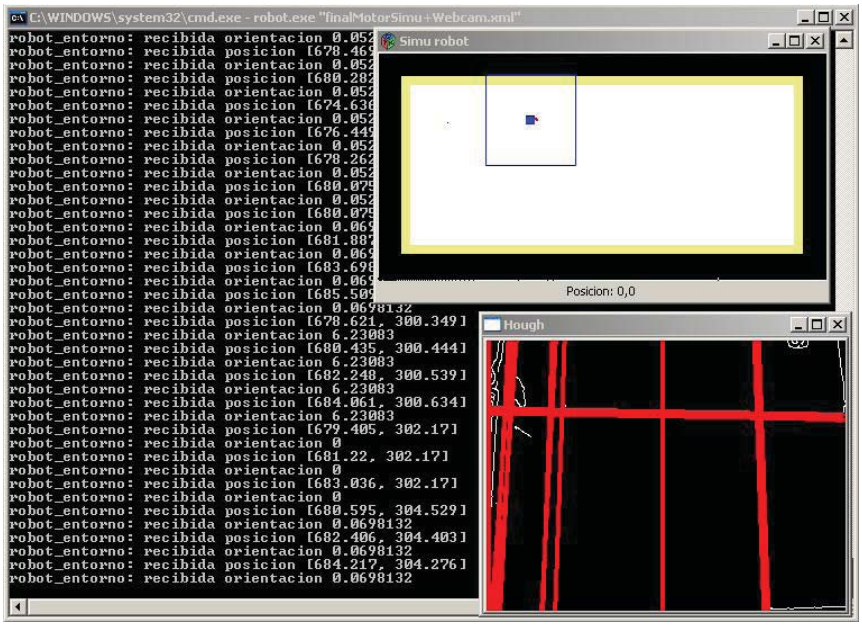


Figura 110 Punto medio del recorrido en forma de "L".

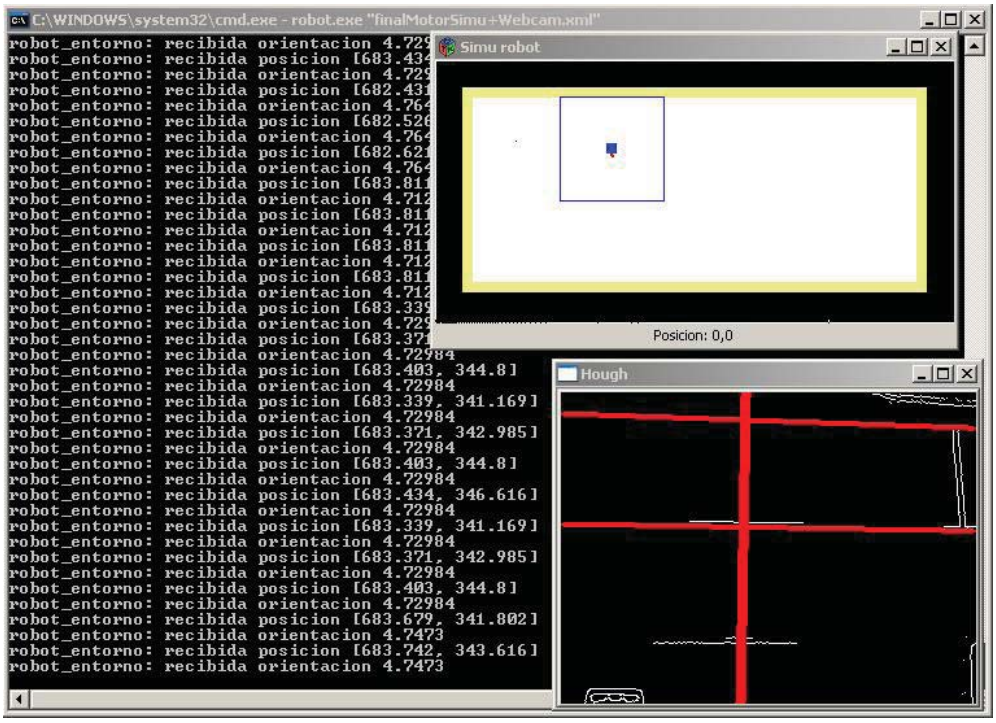


Figura 111 Fin del recorrido en forma de "L".

Se puede observar como la parte final del recorrido de la L fue muy corta a pesar del avance realizado. Esto se debe a la pérdida de líneas por las condiciones de iluminación de la habitación. Con la integración multisensorial, que veremos más adelante, este problema se minimiza.

Para la siguiente y última prueba del primer grupo, se ha realizado un recorrido en diagonal, utilizado para mostrar el correcto funcionamiento del algoritmo que aplica la matriz de transformación de los ejes móviles (los de la captura) para pasarlos al sistema de coordenadas fijas (las de la habitación).

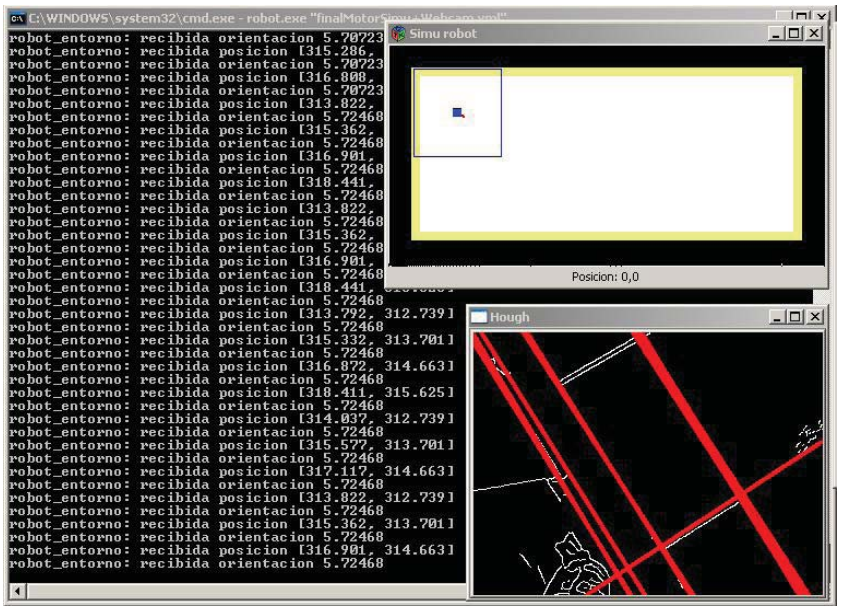


Figura 112 Inicio del recorrido en diagonal con la Webcam como único sistema de navegación.

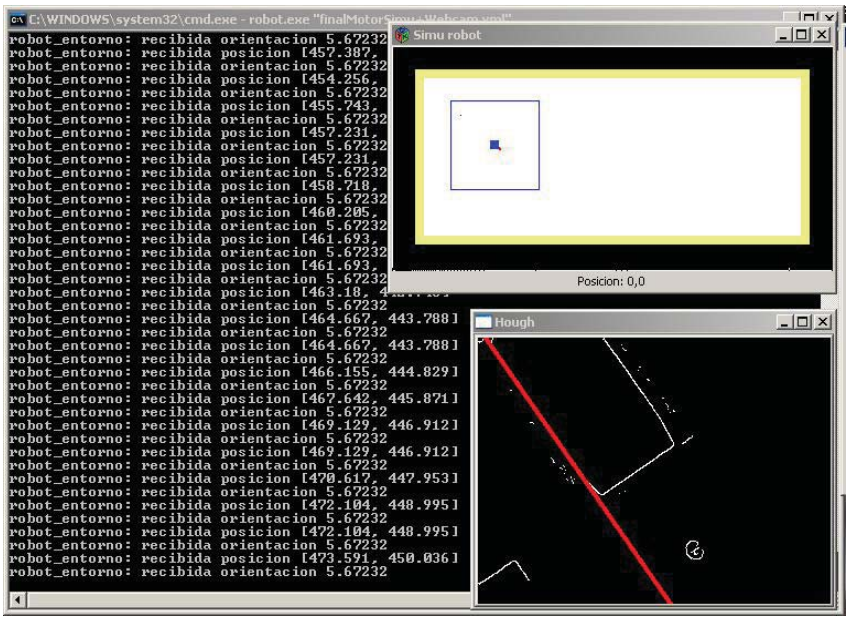


Figura 113 Final del recorrido en diagonal con la Webcam como único sistema de navegación.

El segundo grupo de pruebas de posicionamiento, incluye la integración multisensorial en la simulación del robot; es decir, la integración de la simulación de los encoders junto con el módulo de la webcam.

La primera prueba realizada hace referencia al sistema de posicionamiento absoluto, utilizando balizas conocidas por la aplicación (focos de la habitación).

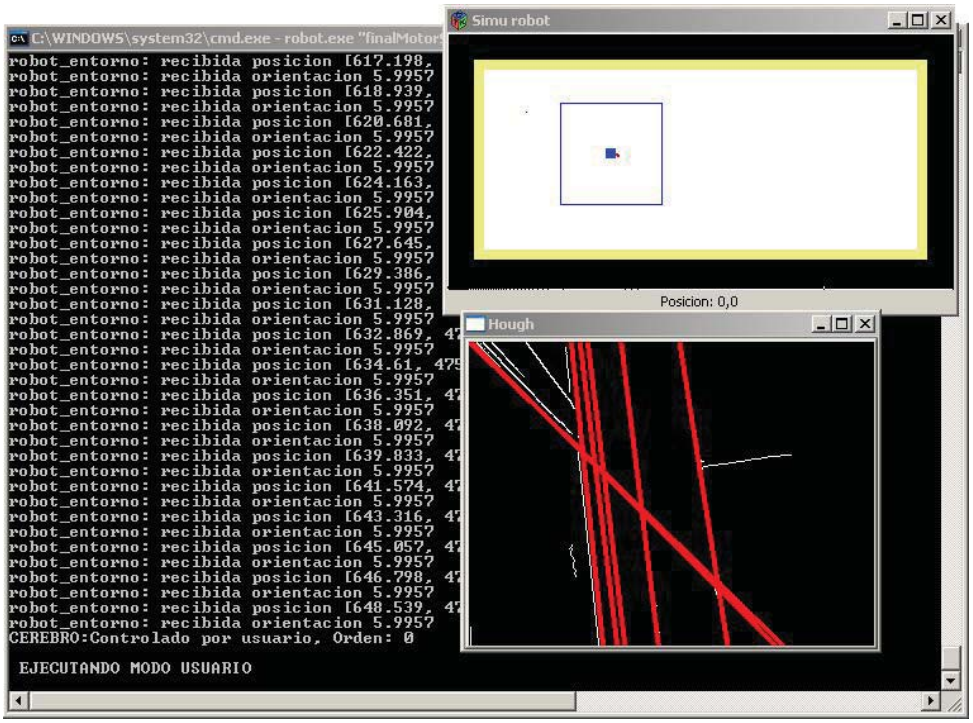


Figura 114 Captura que muestra al robot perdido.

En la figura 114 se muestra cómo el robot se encuentra perdido, debido a la acumulación de errores en el cálculo con los encoders y a la disminución a cero de la fiabilidad del cálculo realizado por la webcam.

En la figura 115, se observa cómo tapando el objetivo de la Webcam y posicionándonos bajo un foco, se corrige la posición del Robot con fiabilidad 100%.

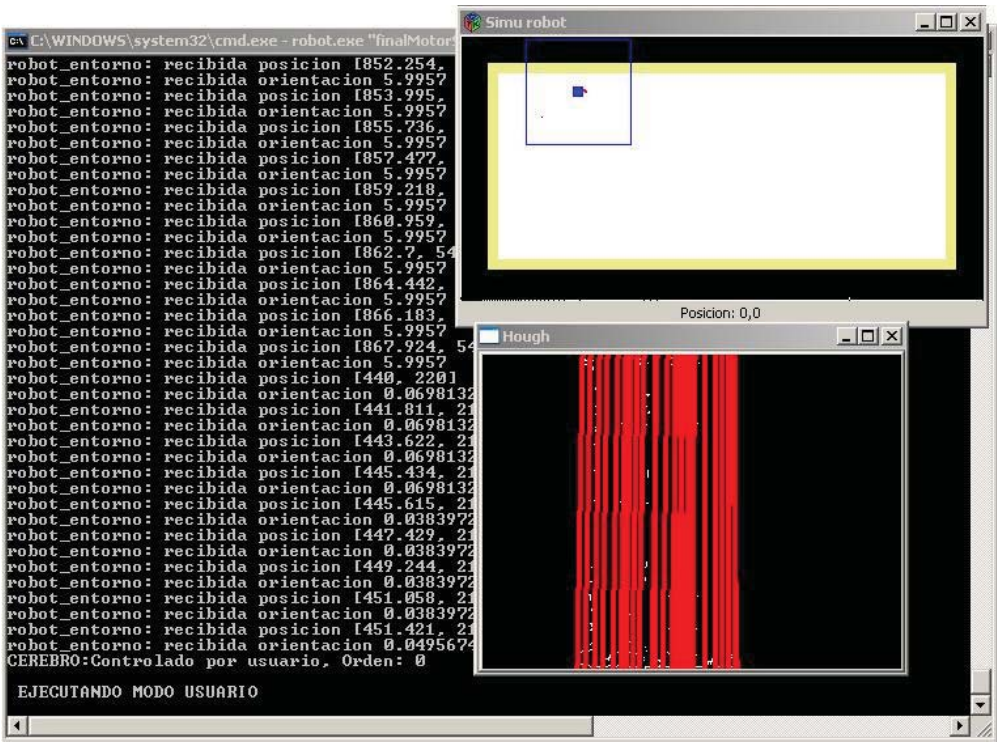


Figura 115 Recuperación de la posición al encontrar una baliza absoluta.

La última prueba correspondiente al grupo de integración multisensorial, muestra un recorrido en forma de "L", similar al realizado para la primera prueba de posición de este apartado en la que se utilizaba como sistema de navegación únicamente la Webcam.

Se ha obtenido un recorrido perfecto, ya que los errores acumulados en el cálculo de la posición efectuado por el módulo Webcam, debidos a la pérdida de líneas y/o puntos de referencia, se han corregido por el cálculo aproximado realizado por los encoders.

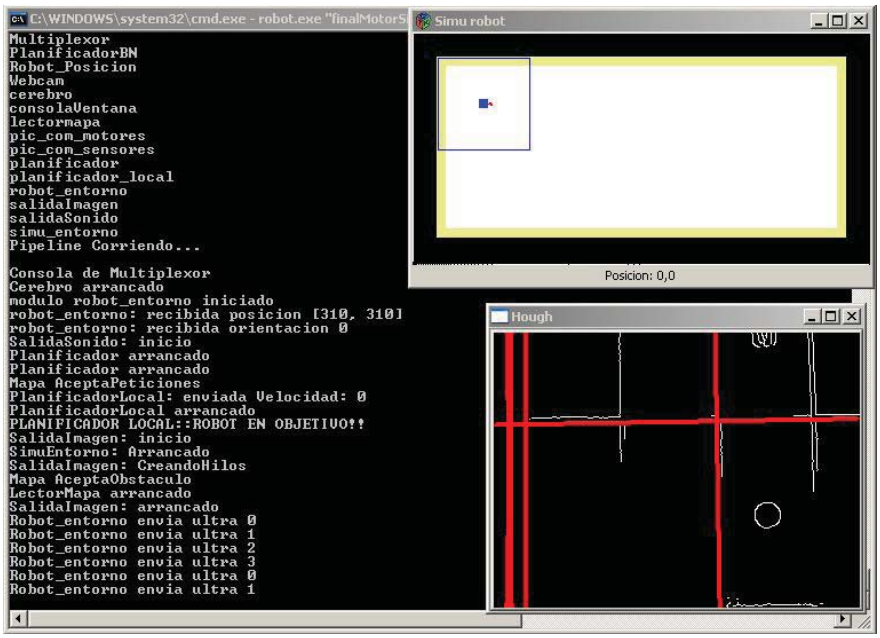


Figura 116 Inicio del recorrido en "L" con integración multisensorial.

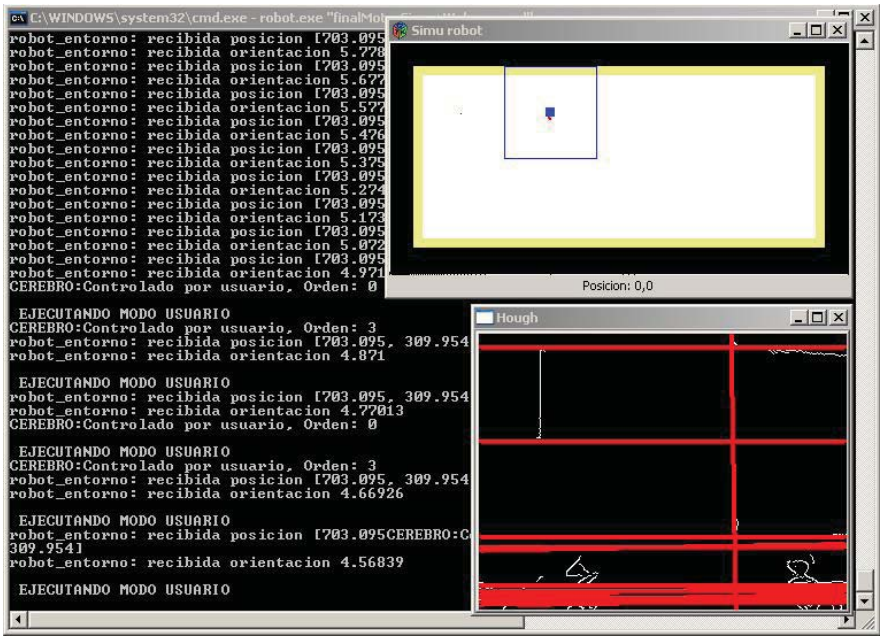


Figura 117 Punto medio del recorrido en "L" con integración multisensorial.

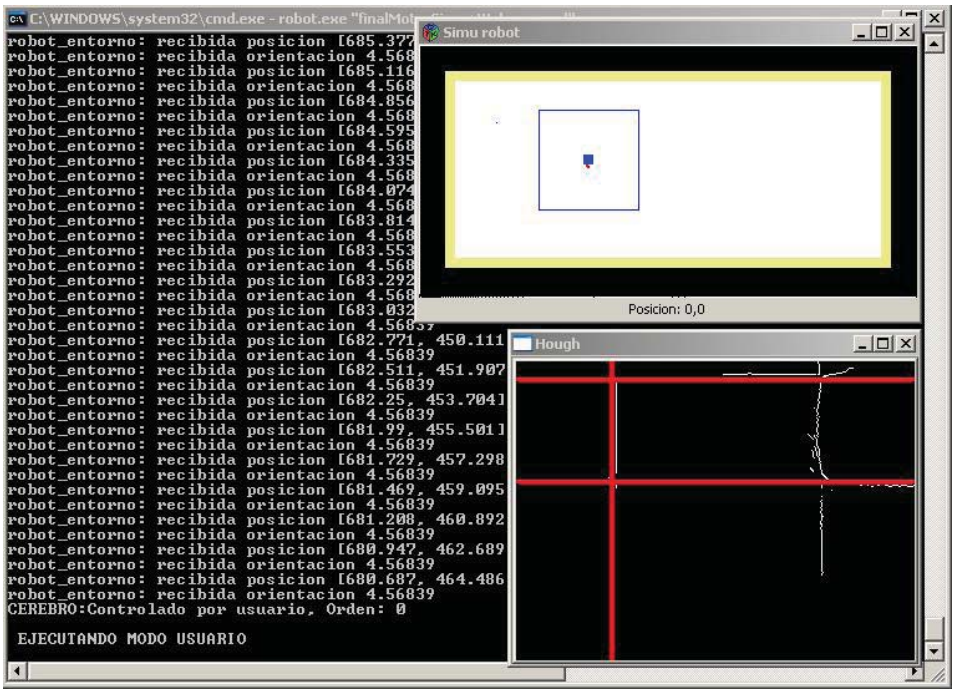


Figura 118 Final del recorrido en "L" con integración multisensorial.

5.1.3 Pruebas con el pic

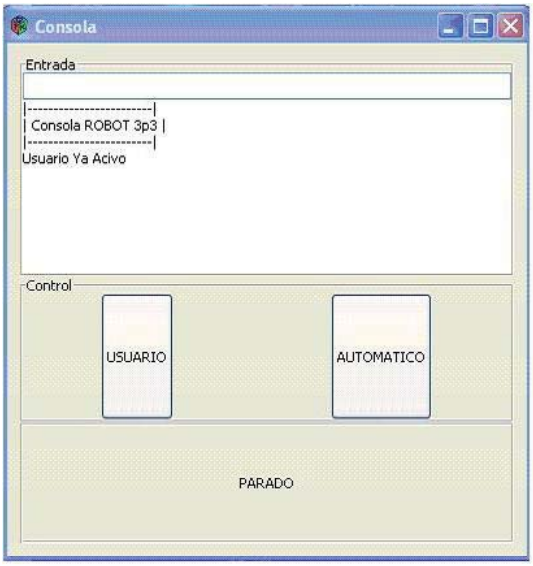
Una vez realizadas las modificaciones pertinentes en el módulo "Pic_comm" para adaptarlo a comunicación USB y para cambiar la representación de los comandos transferidos, llegaba el momento de realizar pruebas para comprobar el funcionamiento.

Como la traza del proyecto se hacía muy complicada, debido tanto al uso de múltiples hilos como a la comunicación intermedia USB con el microcontrolador, las pruebas se realizaron probando distintas configuraciones especificadas en diferentes archivos xml.

Además, fue crucial el uso de un PIC programado con un firmware especial que iluminaba un LED de distintas formas según el comando que recibiera. De esta manera se pudo comprobar si el PIC contestaba de manera adecuada a las peticiones que se le solicitaban.

Tras múltiples pruebas, aquí explicamos las que se realizaron sobre las configuraciones que fueron más significativas:

- "bajo_nivel.xml": sobre este fichero de configuración de módulos, inicialmente se hicieron pruebas para enviar al PIC **siempre** la misma orden, con el fin de testear si el microcontrolador conectaba adecuadamente y respondía con el ACK apropiado. Este comando llamado "LED_AM_OFF" posee el valor hexadecimal 0x02, y el efecto sobre el firmware programado en el PIC es que apaga un LED que inicialmente aparece parpadeando. Una vez esto funcionó, se reprogramó el firmware del PIC para que el LED actuase de distintas formas en función del comando enviado. De esta manera se comprobaría que el módulo funcionaba para el resto de las órdenes referentes a las velocidades de los motores. En este fichero estaban conectados los módulos de bajo nivel. Como tras la adaptación a la nueva librería USB, la aplicación no funcionaba correctamente, se decidió trazar el comportamiento de "Pic_comm" aislándolo en el siguiente fichero.
- "bajo_nivel_pic.xml": en este fichero se incluyó únicamente el módulo "Pic_comm" para asegurar su funcionamiento por separado, sin que estuviese influido por ningún otro módulo. Sobre este fichero se llevó a cabo el proceso de implementación descrito en el apartado 3.3.1.1.2. Ahí se pueden encontrar los pasos seguidos hasta hacerlo funcionar con la nueva comunicación, descritos con detalle.
- "finalMotorReal.xml": una vez que se logró hacer funcionar al módulo "Pic_comm" de manera aislada, era evidente que se debía integrar con el resto de la aplicación. Por ello, en este fichero se unieron todos los módulos, tanto de alto como de bajo nivel. Se pudo comprobar cómo el control manual del robot funcionaba a la perfección. De hecho, tras la simulación, se conectó el PIC a las ruedas y respondieron adecuadamente. Vemos el funcionamiento en la figura 119:



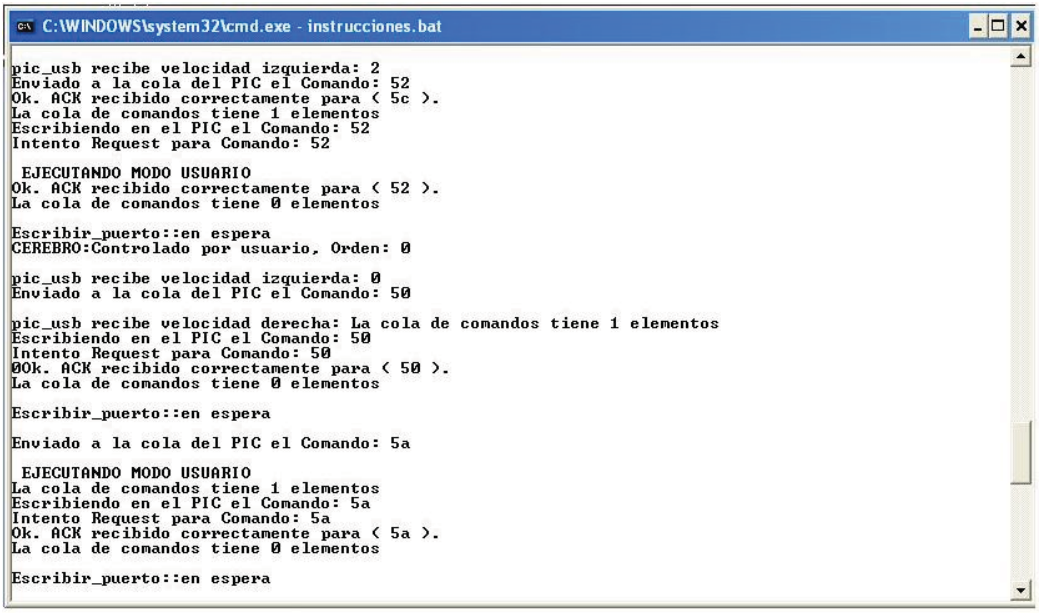


Figura 119 Prueba de devolución de ACK por parte del PIC de motores

En la consola se seleccionó el modo usuario para utilizar el control manual del robot. Como se puede apreciar en la segunda ventana, se envió una orden para que el robot caminara hacia adelante, y luego otra para que se parara:

- W (hacia delante): comprobamos cómo el PIC envía el comando 0x5C para la rueda derecha y el comando 0x52 para la izquierda. Los ACK confirman que el PIC ha procesado correctamente los comandos. Las trazas también permiten ver la cantidad de elementos que hay en la cola "comandos_a_enviar" en cada instante.
- S (parado): se envía al PIC el comando 0x50 para la rueda izquierda, y el 0x5A para la derecha. De igual forma se puede comprobar en las trazas cómo se reciben dichos comandos de los puertos de entrada y cómo se mandan al PIC.

5.1.4 Pruebas en conjunto después de la integración

5.2 Trabajo futuro

Aunque durante todo el curso se ha trabajado en los 3 objetivos planteados inicialmente, tuvimos en consideración otras posibles mejoras que se podrían realizar al robot, pero que se escapaban de nuestro marco de trabajo. No obstante, en este apartado se sugieren algunas ideas que podrían servir como trabajo futuro.

5.2.1 Trabajo futuro en el sistema de visión

En un futuro se podría ampliar el apartado de visión artificial de modo que fuese cada vez más completo y fuese capaz de tratar nuevos casos.

Sería interesante que distinguiese casos aislados, para que la posición y la orientación se calculasen de forma más precisa cuando estos casos se presenten, incluso que el móvil llegase a pararse si no tuviese información suficiente para continuar. Para ello, habría que analizar qué tipo de casos extraños podrían aparecer en distintos lugares y situaciones. Por ejemplo, alguien podría poner un objeto sobre la webcam, con lo que no sería capaz de realizar su tarea. También podría ocurrir que de pronto se apagase la luz y las imágenes tomadas por la cámara fuesen demasiado oscuras cómo para obtener algún tipo de información sobre ellas. En estos casos, el móvil debería detenerse, ya que de lo contrario se perdería y podría llegar a una situación de peligro, como por ejemplo golpearse con algún obstáculo.

También sería interesante que el sistema se diese cuenta de que está al lado de una pared, algo que se podría hacer distinguiendo texturas y que sería complementario con el trabajo de los sensores de ultrasonidos e infrarrojos.

Otra mejora posible, sería ampliar el sistema de balizas. Se podrían poner marcas en diferentes lugares del techo, de forma que cuando el móvil se encontrase debajo de alguna de ellas conociera con exactitud su localización y de esta forma corregir pequeños errores acumulados. Un ejemplo de baliza podrían ser puntos de colores, de manera que cada color estuviese asociado a una posición determinada.

5.2.2 Trabajo futuro en el servicio web

En el apartado de especificación y diseño, comentábamos que el servicio Web permite varias conexiones remotas simultáneas, y que podría suponer un problema con el diseño actual. Por otra parte, se realizo la implementación con un servicio Web multiusuario, pensando en las futuras ampliaciones que comentamos a continuación.

Una posibilidad que ofrece un servicio Web alojado en el Robot es colocar otra webcam al robot que funcionara como su "visión humana", captando imágenes como si se tratara de sus ojos. Estas imágenes se podrían emitir por streaming a través del servicio web, de forma que la aplicación cliente podría ver todo lo que "el robot viera".

También se ha comentado en el diseño de la comunicación por Sockets del servicio Web con el Robot, que se ha realizado una comunicación Socket síncrona, es decir, con espera de respuesta del cliente con cada envío de comando.

De cara a un sistema multiusuario, sería interesante sustituir la comunicación actual por una comunicación Socket orientada a conexión para permitir el envío asíncrono de datos del Robot al Servicio Web. De este modo, y junto con una cámara IP de visión real, sería posible realizar un control remoto sin necesidad de estar en presencia del robot, ya que este podría comunicar a todos y cada uno de los usuarios conectados su posición y orientación cada vez que esta cambiara, mostrándola en una interfaz gráfica en forma de mapa.

Tan sólo sería necesario cambiar la forma en la que se gestiona la comunicación, tanto en el cliente como en el servidor de sockets, de modo que no se cerrara después de cada envío o recepción respectivamente. Para lograr comunicación asíncrona, deberían crearse dos flujos distintos de datos. El mayor inconveniente de esta opción, sería el mantenimiento de la conexión activa.

Evidentemente, este nuevo concepto de control remoto, conllevaría la creación de un sistema de permisos de usuario en el servicio web, de modo que los usuarios más privilegiados pudieran realizar acciones importantes tales como desconectar el robot remotamente, reiniciarlo o provocar

cambios de configuración. Los usuarios más restringidos podrían realizar operaciones únicamente de consulta, y un solo usuario, preferiblemente el técnico, debería ser el encargado de utilizar las opciones que ofrece el diseño actual.

Otra idea muy interesante, aprovechando que el robot puede detectar su nivel de batería de forma automática, es que él mismo busque algún punto de toma de corriente y se recargue.

5.2.3 Trabajo futuro en la comunicación con el Hardware

La modificación realizada en el módulo "Pic_comm" se llevó a cabo con el fin de adaptar la comunicación entre éste, los motores y los encoders mediante USB. Sin embargo, queda pendiente cambiar la comunicación del resto de sensores (infrarrojos, ultrasonidos...), ya que esto no resultó fundamental para la realización de nuestro sistema, y no debería ocasionar grandes problemas de integración.

Además, una buena idea para un trabajo futuro es el **chequeo de errores**. Esto es algo que en la actualidad se realiza mediante mensajes de traza en modo *debug*. Sin embargo, el sistema está totalmente preparado para obtener información detallada acerca de los posibles errores en la comunicación USB. Estos errores pueden ser emitidos desde el firmware del PIC mediante distintos códigos de error, y el módulo "Pic_comm" podría interpretarlos, de manera que sería mucho más sencillo el proceso de localización de fallos en la comunicación.

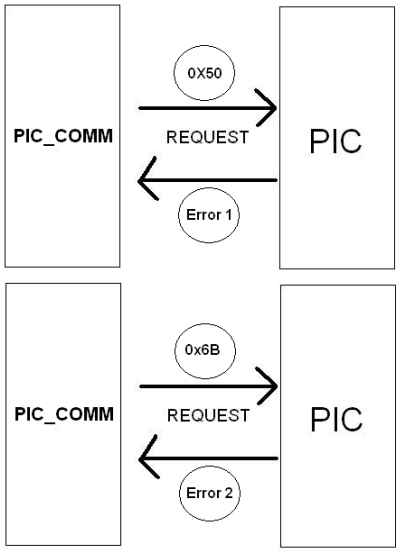


Figura 120 Chequeo de errores, distinguiéndolos según el código

6 Conclusiones

Cuando comenzamos el proyecto, tuvimos que analizar un trabajo anterior en el que se había construido un robot que tenía un sistema software para navegar de forma tanto autónoma como manual. El sistema de posicionamiento y guiado que utilizaban se basaba en el procesamiento de la información recibida por varios sensores, tales como los encoders, la brújula, infrarrojos y ultrasonidos.

El funcionamiento autónomo del robot no permitía realizar una visita larga del museo, puesto que la acumulación de errores procedentes de los encoders provocaba una imprecisión en el cálculo de la posición, e impedía que se pudiera navegar autónomamente durante largo plazo.

Hay que tener en cuenta que el error en los encoders es algo intrínseco a cualquier sistema de navegación, y que sin medidas de corrección externas, ningún sistema robótico podría navegar de forma autónoma. En nuestro caso, esta medida de corrección externa se obtiene a través de la webcam.

Se partía de los siguientes objetivos:

- Dotar al robot guía de la Facultad de Informática de un sistema de visión artificial que le permitiera localizarse utilizando una webcam como sensor de posicionamiento incremental ayudándose del reconocimiento de balizas absolutas.
- Consecución del control del robot mediante red WiFi.
- Cambiar la comunicación serie con los PIC por comunicación USB.

Siguiendo un **proceso de ingeniería** basado en análisis, diseño y síntesis, estos objetivos se completaron, cuidando especialmente el diseño realizado y permitiendo así la reutilización del código.

Se llevó a cabo un análisis descendente y un diseño modular ascendente, lo que permitió realizar una implementación del código mucho más flexible y escalable.

Entre otras cosas, se consiguió un **sistema de localización basado en visión artificial**, que cumple los requisitos propuestos inicialmente. Por otro lado, se ha realizado un sistema de **control inalámbrico** fácilmente ampliable, y más robusto que con el que contaba el robot en un principio. También se ha logrado una mejora sustancial en la comunicación con la parte

hardware, al haber realizado un **cambio de serie a USB**, logrando una transmisión más rápida y fiable.

Además de estos tres objetivos principales, fueron surgiendo **nuevas ideas** durante el proceso de desarrollo, siendo éstas aplicadas por su interés de cara al aumento de funcionalidad del sistema y comodidad para las personas encargadas de su mantenimiento. Entre estos objetivos secundarios se encuentran el sistema de aviso de batería limitada, y el servicio de alertas por e-mail y SMS.

Una de las mayores dificultades que hemos afrontado ha sido la de **integrar** nuestro trabajo en un sistema ya realizado, lo que hizo que tuviésemos que llevar a cabo un gran trabajo previo de análisis.

Ha sido fundamental en el desarrollo de nuestro proyecto la buena **organización y planificación** entre los tres componentes del grupo. Esto ha permitido llevar a cabo la realización de cada uno de los objetivos de forma paralela.

Para finalizar, cabe señalar que la visión por computador es un campo muy reciente en el que, aunque el trabajo realizado es cada vez mayor, aún queda mucho por hacer. De todas formas prevemos que este área tendrá una importancia cada vez mayor en los próximos años.

Apéndice A. Tratamiento de imágenes

Introducción

El tratamiento digital de imágenes es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información.

A continuación, se explican más detalladamente las principales técnicas utilizadas en el proyecto, así como la librería utilizada.

Histograma de una imagen

En estadística, un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados. En el eje vertical se representan las frecuencias, y en el eje horizontal los valores de las variables. En visión por computador, también se utilizan histogramas para extraer características de imágenes.

El histograma de una imagen contiene la información de la probabilidad de aparición de las distintas tonalidades de color que se pueden dar en cada caso, ya que podemos trabajar en distintos tipos de colores o en escala de grises. En el caso de una imagen en color, no podemos hablar de un único histograma que caracterice a la imagen sino de tres histogramas, uno para cada color (RGB, por ejemplo). Si bien el uso del histograma y sus posteriores modificaciones son más aplicables a imágenes en escala de grises.

La creación de un histograma, en el caso de una imagen en escala de 256 tonalidades del blanco al negro (escala de grises), se realiza fácilmente mediante medios informáticos, de manera que, en primer lugar crearemos un vector (array) que contenga 256 posiciones, una por cada nivel de gris, el algoritmo recorrerá cada uno de los pixeles de la imagen, aumentando en una unidad el valor guardado en la posición del array correspondiente al tono del pixel en cuestión.

El histograma proporciona una descripción de la apariencia global de una imagen. De forma que si los niveles de gris están concentrados hacia el extremo oscuro del rango de la escala de gris, la apariencia global de la imagen será oscura; mientras que si sucede justo lo contrario, la imagen

correspondiente será brillante. Por su parte, un histograma que presente un perfil estrecho corresponderá a una imagen de bajo contraste y un histograma con una dispersión considerable a una imagen de alto contraste.

En el caso de la imagen en color, aparecen tres histogramas, de forma que el tratamiento de imágenes en color se complica por la aparición de nuevos componentes. Ahora el histograma no contiene los tonos del negro al blanco, sino del negro al color correspondiente (rojo, verde ó azul -caso RGB-). Al elaborar el histograma, el algoritmo debe separar el color correspondiente a cada pixel en sus componentes RGB (rojo, verde y azul).

Veamos a continuación un ejemplo del histograma de una imagen del techo de la Facultad de Informática, idóneo para el funcionamiento de nuestro sistema de localización:

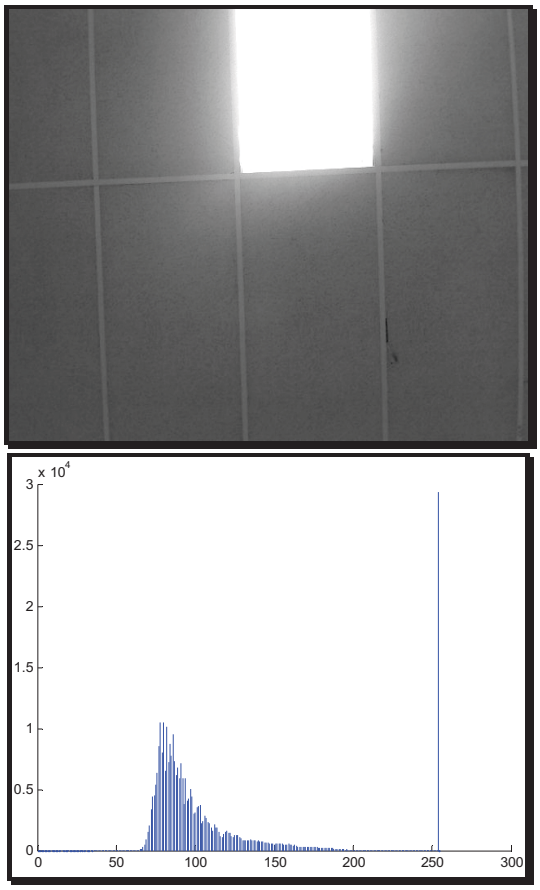


Figura 121 Histograma de una fotografía del techo

Algunas propiedades estadísticas que se pueden obtener a partir del histograma de una imagen son:

Media: $$\bar{g} = \sum_{g=0}^{L-1} gP(g) = \sum_i \sum_j \frac{I(i,j)}{M}$$

Varianza: $$\sigma^2 = \sum_{g=0}^{L-1} (g - \bar{g})^2 P(g)$$

Asimetría: $$a = \sum_{g=0}^{L-1} (g - \bar{g})^3 P(g)$$

Entropía: $$e = -\sum_{g=0}^{L-1} P(g) \log_2 \left[P(g) \right]$$

Suavizado Gaussiano

Todas las imágenes tienen una cierta cantidad de ruido, valores distorsionados, bien debidos al sensor CCD de la cámara o al medio de transmisión de la señal. El ruido se manifestará generalmente en píxeles aislados que toman un valor de gris diferente al de sus vecinos. Los algoritmos de filtrado se basan en esta característica y tienen como objetivo eliminar el ruido. También se suelen utilizar para desenfocar imágenes.

El filtrado gaussiano es una operación local sobre las imágenes, que puede interpretarse como el uso de una máscara de convolución que recorre la imagen píxel por píxel. En este tipo de operaciones locales la máscara se va haciendo coincidir con cada píxel mediante un barrido por filas o columnas. Una vez superpuesta la máscara, el número contenido en cada celda de esta es multiplicada por el valor del píxel coincidente en la imagen original y luego todos estos se suman para obtener el valor del píxel de la imagen de salida. Puesto que el valor del píxel de salida debe estar entre 0 y 255, es común que se requiera una operación de redondeo.

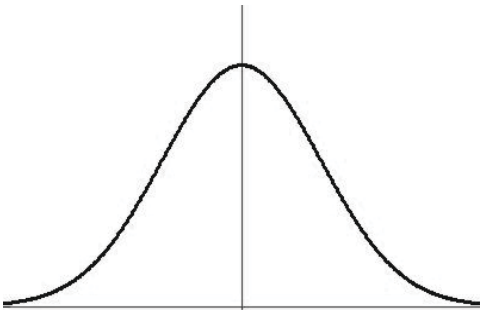


Figura 122 Suavizado Gaussiano

Para el filtrado gaussiano, suele usarse una máscara de 3x3 o una de 5x5 que intenta imitar la forma de una gausiana y los valores de las celdas dependen del valor de la desviación estándar, σ, seleccionada. A mayor σ mayor suavización de la imagen de salida. Si σ = 0,391 píxeles:

1	4	1
4	12	4
1	4	1

Figura 123 Máscara gaussiana

El filtro Gaussiano se calcula mediante la siguiente ecuación:

$$G(i,j) = e^{-\frac{\left(i^2+j^2\right)}{2\sigma^2}}$$

i, *j* se refieren a la fila y columna en la máscara tomando como fila y columna 0 el centro de la misma.

Operador de bordes de Canny

Una de las informaciones más útiles que se encuentran en una imagen la constituyen los bordes, ya que al delimitar los objetos definen los límites entre ellos y el fondo y entre los objetos entre sí. Las técnicas usadas en la detección de bordes tienen por objeto la localización de los puntos en los que se produce una variación de intensidad.

El detector de bordes de Canny lo propuso J. Canny en 1993. Se obtiene a partir de la optimización de una serie de condiciones:

- Error: Se deben detectar todos y sólo los bordes.
- Localización: La distancia entre el píxel señalado como borde y el borde real debe ser tan pequeña como se pueda.
- Respuesta: No debe identificar varios píxeles como bordes cuando sólo exista uno.

Estas tres condiciones pueden ser expresadas de forma matemática como:

$$SNR = \frac{A\left|\int_{0}^{W} f(x)dx\right|}{n_o\sqrt{\int_{-W}^{W} f^2(x)dx}}$$

$$L = \frac{A\left|f(0)\right|}{n_o\sqrt{\int_{-W}^{W} f^2(x)dx}}$$

$$X_r = \prod\left(\frac{\int_{-W}^{W} f^2(x)dx}{\int_{-W}^{W}\left(f'(x)\right)^2 dx}\right)^{1/2}$$

Canny busca la optimización del producto de la relación señal ruido por la localización y el tercero como una condición. Con ello se llega a que el operador óptimo es la derivada de una gausiana. Para obtener los bordes se siguen los siguientes pasos:

- Se obtiene la imagen I.
- Se tiene una gausiana unidimensional G.
- Se obtienen las derivadas (unidimensionales) de la gausiana G_x y G_y.
- Se convoluciona las derivadas de G con la imagen obteniendo I_xI_y.
- Se obtiene la magnitud M como la raíz cuadrada de la suma de los cuadrados de I_xI_y.

- El siguiente paso es la supresión de aquellos píxeles que no sean máximos. Para ello se compara el valor de la magnitud para cada píxel con sus vecinos. Sólo aquellos que son máximos en su entorno de dejan como están.
- Para detectar los bordes se utilizarán dos umbrales T1 y T2, siendo este último el mayor. Valores típicos para estos umbrales son 0.1 y 0.5, respectivamente, aunque se recomienda que T2 y T1 tengan una relación entre 2:1 y 3:1, dependiendo de la relación señal ruido, en el caso de que este valor sea conocido. Si se usase el menor se detectarían bordes poco importantes, si se usase sólo el mayor se dejarían de detectar puntos pertenecientes a los bordes. Por ello la condición para pertenecer a un borde es ser mayor de T2 o mayor que T1 siempre que uno de sus vecinos sea mayor que T2. Esta etapa realiza los cálculos siguientes:
  - Un pixel I(i,j), se considera borde definitivo si I(i,j) >= T2.
  - Un pixel I(i,j), se considera fondo definitivo si I(i,j) < T1.
  - Todos los píxeles en un vecindario 3 x 3 de los píxeles considerados como borde definitivo, I(k,l), se consideran también borde definitivo si I(k,l) >= T1.

La información normalizada no binaria de salida del método de Canny indica la fuerza del borde a partir de un análisis del gradiente global entre pixeles vecinos. Este método es uno de los más robustos contra el ruido (filtro óptimo), en comparación con los métodos de Roberts, Sobel o Prewitt, por ejemplo.

Transformada de Hough

La transformada de Hough es una técnica utilizada para aislar características de forma particular dentro de una imagen. Esta transformada busca formas geométricas en toda la imagen encontrando los parámetros de aquella que contenga a más puntos en ella. La idea básica es encontrar curvas que puedan ser parametrizadas como líneas rectas, polinomios y círculos. En este apéndice nos centraremos en la transformada de Hough para rectas.

Para evitar singularidades que pueden dar las rectas de pendiente infinita, la ecuación de una línea recta puede escribirse como:

$$\rho = x \cos \theta + y \sin \theta$$

quedando por tanto cada recta definida por su par de parámetros θ y ρ. Todos los puntos que pertenezcan a la misma recta coincidirán en las

componentes θ y ρ donde ρ es la longitud de una normal desde el origen hasta la línea y θ es el ángulo de ρ con respecto al eje x.

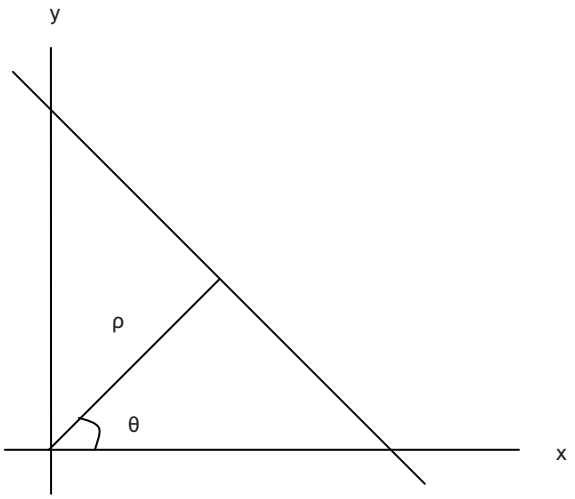


Figura 124 Teoría sobre la transformada de Hough

Por tanto se pasaría primero un detector de bordes y se consideraría como puntos pertenecientes a él aquellos cuyo valor sea más alto de un valor prefijado. Para cada punto (x_i, y_i) detectado se calcula la curva:

$$\rho = x_i \cos \theta + y_i \sin \theta$$

donde ahora θ y ρ son variables y que representa todas las rectas que pasan por el punto (x_i, y_i). Para dos puntos cualesquiera la intersección de sus dos curvas senoidales estará en aquellos valores (θ_i, ρ_i) que definen la recta que los contiene. Al final aquellos valores más repetidos coinciden con la recta que contiene el mayor número de píxeles.

Existe un umbral que descarta las líneas que no sobrepasan el número de puntos especificado en este umbral.

Un inconveniente de la transformada de Hough es su coste computacional ya que el número de operaciones es alto. Aunque este coste cada vez es menor debido al desarrollo de los ordenadores existen varias simplificaciones. Una de ellas consiste en aprovechar la información del ángulo del gradiente para simplificar la búsqueda ya que entonces basta con sustituir los tres valores (x_i, y_i, θ_i) en la ecuación para obtener ρ.

$$\rho = x_i \cos \theta_i + y_i \sin \theta_i$$

Otra variación es darle un peso a cada píxel dependiendo de su respuesta al gradiente.

La gran ventaja de la transformada de Hough es que utiliza toda la información de la imagen por lo que es más inmune a pérdidas parciales de los bordes que los métodos locales. Sus inconvenientes son el coste computacional ya mencionado, que es grande, y sobre todo que las rectas detectadas son de longitud infinita. Por ello no se sabe realmente dónde empieza y acaba el segmento de recta presente en la imagen y se pueden tomar varios segmentos como pertenecientes al mismo objeto cuando en realidad corresponden a objetos distintos.

Librería OpenCV

El 13 de Junio del 2000, Intel® Corporation anunció que estaba trabajando con un grupo de reconocidos investigadores en visión por computador para realizar una nueva librería de estructuras/funciones en lenguaje C. Esta librería proporcionaría un marco de trabajo de nivel medio-alto que ayudaría al personal docente e investigador a desarrollar nuevas formas de interactuar con los ordenadores. Este anuncio tuvo lugar en la apertura del IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). Había nacido The Open Computer Vision Library y lo hacía bajo licencia BSD (Software Libre).

La librería OpenCV es una API de aproximadamente 300 funciones escritas en lenguaje C que se caracterizan por lo siguiente:

- Su uso es libre tanto para su uso comercial como no.
- No utiliza librerías numéricas externas, aunque puede hacer uso de alguna de ellas, si están disponibles, en tiempo de ejecución.
- Es compatible con The Intel® Processing Library (IPL) y utiliza The Intel® Integrated Performance Primitives (IPP) para mejorar su rendimiento, si están disponibles en el sistema.
- Dispone de interfaces para algunos otros lenguajes y entornos: EiC -intérprete ANSI C escrito por Ed Breen. Hawk y CvEnv son entornos interactivos (escritos en MFC y TCL, respectivamente) que utilizan el intérprete EiC; Ch - intérprete ANSI C/C++ creado y soportado por la compañía SoftIntegration; Matlab® - gran entorno para el cálculo numérico y simbólico creado por Mathworks; y muchos más.

Estructura y características

La librería OpenCV esta dirigida fundamentalmente a la visión por computador en tiempo real. Entre sus muchas áreas de aplicación destacarían: interacción hombre-máquina; segmentación y reconocimiento de objetos; reconocimiento de gestos; seguimiento del movimiento; estructura del movimiento; y robots móviles.

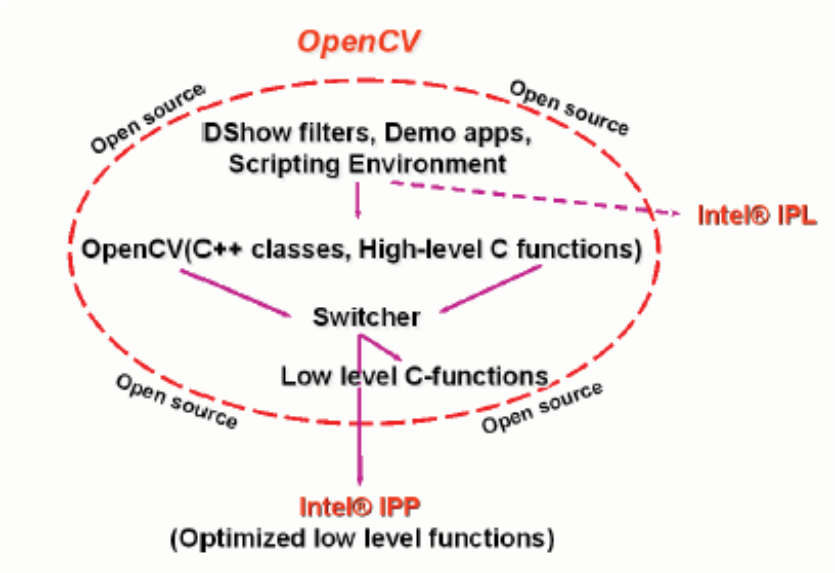


Figura 125 Librería OpenCV

La librería OpenCV proporciona numerosos elementos de alto nivel, que facilitan sobre manera el trabajo al usuario. Por ejemplo, proporciona filtros Microsoft® DirectShow para realizar tareas tales como: calibración de la cámara, seguidores de objetos (Kalman tracker y ConDensation tracker), etc. Todos ellos se pueden utilizar en Microsoft® GraphEdit para ilustrar de forma bastante sencilla numerosas aplicaciones de visión.

De igual forma, la librería OpenCV proporciona numerosas aplicaciones de ejemplo que ilustran como emplear las distintas funciones de la librería.

También los entornos de scripting hacen uso de estas funciones para implementar su funcionalidad. La librería OpenCV proporciona una gran diversidad de entornos.

Todas estas herramientas de alto nivel hacen uso de un paquete de clases C++ y funciones C de alto nivel que utilizan a su vez funciones

muy eficientes escritas en C. Concretamente, el conjunto de funciones suministradas por la librería OpenCV se agrupan en los siguientes bloques:

- Estructuras y operaciones básicas: matrices, grafos, árboles, etc.
- Procesamiento y análisis de imágenes: filtros, momentos, histogramas, etc.
- Análisis estructural: geometría, procesamiento del contorno, etc.
- Análisis del movimiento y seguimiento de objetos: plantillas de movimiento, seguidores, flujo óptico, etc.
- Reconocimiento de objetos: objetos propios, modelos HMM, etc.
- Calibración de la cámara: morphing, geometría epipolar, estimación de la pose, etc.
- Reconstrucción tridimensional (funcionalidad experimental): detección de objetos, seguimiento de objetos tridimensionales, etc.
- Interfaces gráficos de usuarios y adquisición de video.

La librería OpenCV puede hacer uso de librerías propietarias como son en este caso The Intel® Image Processing Library (IPL) y The Intel® Integrated Performance Primitives (IPP) en caso de disponer de ellas. Estas aplicaciones tratan de mejorar el rendimiento de la librería con primitivas optimizadas para procesadores Intel®. Sin embargo, la licencia por las que se rigen estas aplicaciones no es software libre y por tanto carece de interés para nuestro proyecto.

Conclusiones

Según lo expuesto a lo largo de este apéndice, The Open Computer Vision Library constituye una de las apuestas más interesantes e importantes en visión por computador a lo largo de los últimos tiempos y además Software Libre. Su estabilidad, escalabilidad, su rápido crecimiento y desarrollo garantizan su continuidad y la hacen especialmente útil en el ámbito universitario, para la realización de tareas de docencia e investigación a corto, medio y largo plazo. Es por ello que nos decantamos por utilizarla en nuestro proyecto.

Podemos encontrar más información en la siguiente dirección: http://www.intel.com/technology/computing/opencv/index.htm

Apéndice B. Instalación del entorno de desarrollo del Robot.

A continuación se explica cómo instalar el entorno de desarrollo de la aplicación de alto nivel del robot. La instalación del entorno de desarrollo para el servicio Web, se puede encontrar en la página de GWT, incluida en la Bibliografía.

Software Necesario:

- JRE y JDK de java (www.java.sun.com).
- Entorno de desarrollo Eclipse Europa (www.eclipse.org).
- Plugin CDT para eclipse (www.eclipse.org/cdt) entorno de desarrollo de C/C++.
- Plugin subclipse para eclipse (http://subclipse.tigris.org/) gestión de configuraciones.
- Herramienta para gestión de configuración "Tortoise SVN".
- Compilador de C/C++ MinGW versión 5.0.2 con gcc 3.4.x.
- Phyton 2.4.2 con scons 0.96.1 para los scripts de compilación.
- Msys 1.0.10 Entorno de consola mínima de Linux para Windows + Parche.
- Depurador gdb 5.2.x.
- GTK Entorno de desarrollo de Linux compuesto por:
  - GTKdev de glade versión 2.10.11.
  - GTKmm devel 2.10.11.
  - GTKmm runtime versión 2.10.11 → no necesaria su instalación.
- Librerías de búsqueda en grafos boost versión 1.33.1.
- Librerías de tratamiento de imágenes OpenCV.

Pasos para la instalación:

Lo primero que se debe instalar es el entorno de desarrollo JDK de java, para poder ejecutar eclipse.

El siguiente paso será la instalación de eclipse y sus plugins. Para ello seguir las instrucciones que aparecen en los sitios web mencionados en el apartado de "software necesario".

Instalar en orden los siguientes programas: *MinGW*, *Phyton*, *scons*, *Msys*, *GTKdev glade*, *GTKmm devel* y por último las librerías de *boost* en los directorios de instalación por defecto de cada programa. Como caso especial mencionar la librería boost, que al no tener instalador, debe descomprimirse en "*C:*\".

Es posible que MinGW se instale con la última versión de gcc, con lo cual, se debe reinstalar la versión 3.4.2.

Para la depuración de código, es necesario instalar gdb.

Se debe descomprimir el parche "MSYS-1.0.11-20071204.tar.bz2" en la carpeta de instalación de MSYS, con el fin de solucionar en la ejecución de comandos en dicha aplicación.

Para la ampliación del proyecto de 2008, se deben instalar también las librerías de tratamiento de imágenes OpenCV ejecutando el instalador.

Configuración del entorno:

Configuración de las variables de entorno:

Tras efectuar las instalaciones, setear las variables de entorno del **Sistema** de Windows accediendo desde *"Panel de Control → Sistema → Opciones Avanzadas → Variables de Entorno"* como sigue:

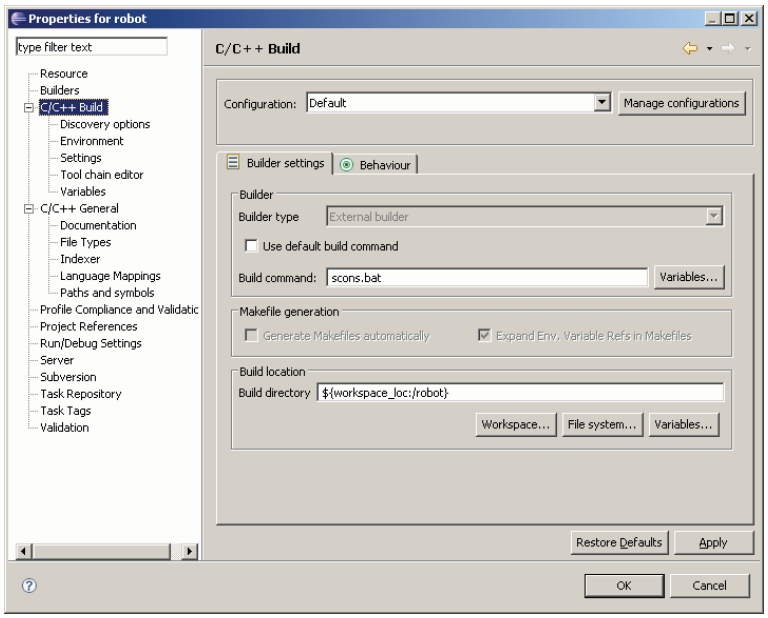
Nombre de la Variable	Valor de la Variable
GTK_BASEPATH	C:\GTK
GTKMM_BASEPATH	C:\GTK
INCLUDE	C:\GTK\INCLUDE;C:\GTK\INCLUDE\GTK-2.0;C:\GTK\INCLUDE\GLIB-2.0;C:\GTK\INCLUDE\PANGO-1.0;C:\GTK\INCLUDE\CAIRO;C:\GTK\INCLUDE\ATK-1.0;C:\GTK\INCLUDE\GTKGLEXT-1.0;C:\GTK\LIB\GTK-2.0\INCLUDE;C:\GTK\LIB\GLIB-2.0\INCLUDE;C:\GTK\LIB\GTKGLEXT-1.0\INCLUDE;C:\GTK\INCLUDE\LIBGLADE-2.0;C:\GTK\INCLUDE\LIBXML2;
LIB	C:\GTK\LIB;
Path	… ;%GTK_BASEPATH%\bin;%SystemRoot%\system32; %SystemRoot%;%SystemRoot%\System32\Wbem; … ;C:\msys\1.0\bin;C:\Python24;C:\MinGW\bin;
PKG_CONFIG_PATH	c:\gtk\lib\pkgconfig

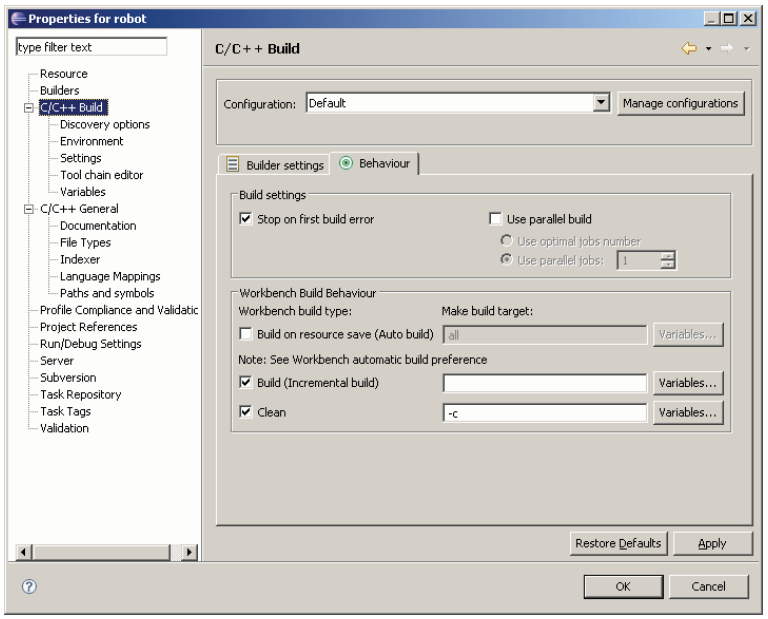
Configuración del eclipse:

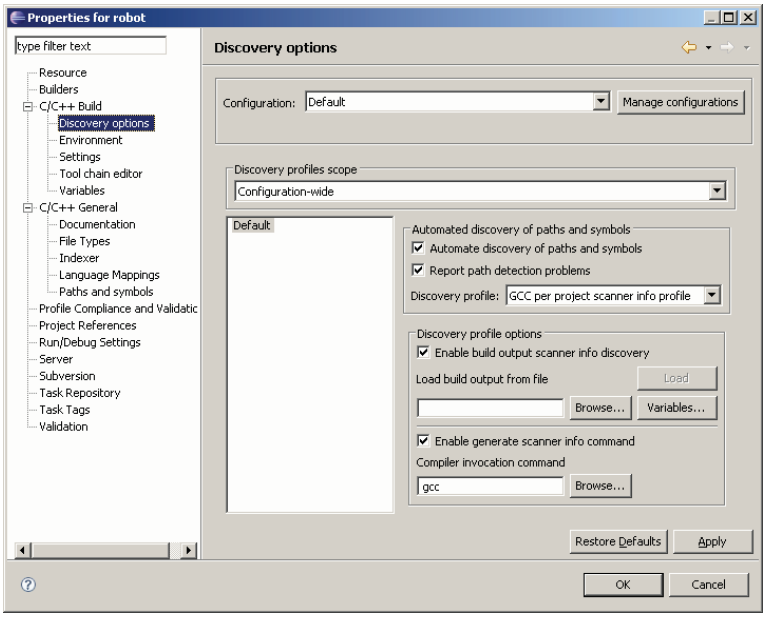
Para crear un proyecto nuevo hay que seleccionar la siguiente opción del menú *"File → New → C++ Project"* y seleccionar en "*Project types*" "*Makefile Project*" (OJO, sin entrar dentro de la carpeta) y en la sección colindante "*ToolChain*", "*MinGW GCC*". Desmarcar la casilla "*Use default location*" y cambiar la localización por la carpeta descomprimida del código del robot.

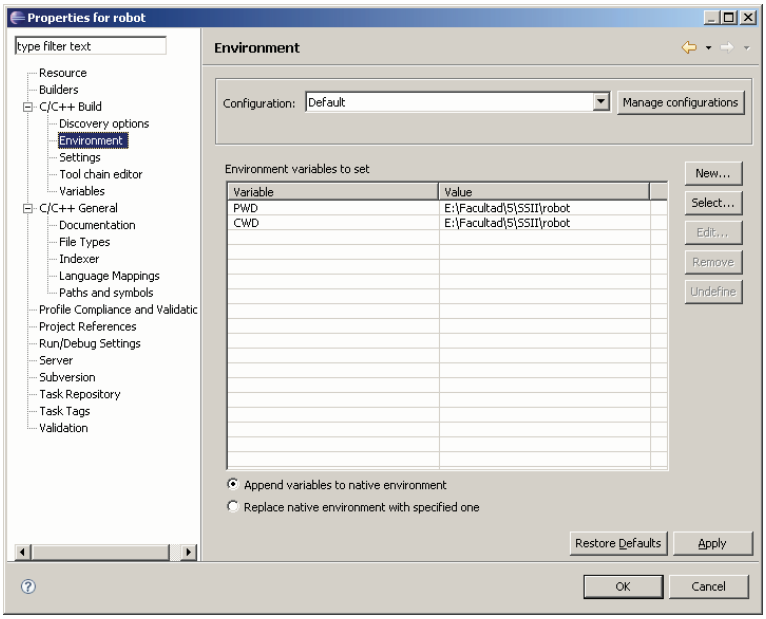
Cuando ya este creado el proyecto, hay que hacer clic derecho en el nombre del proyecto en el "*Project Explorer*" y seleccionar "Properties".

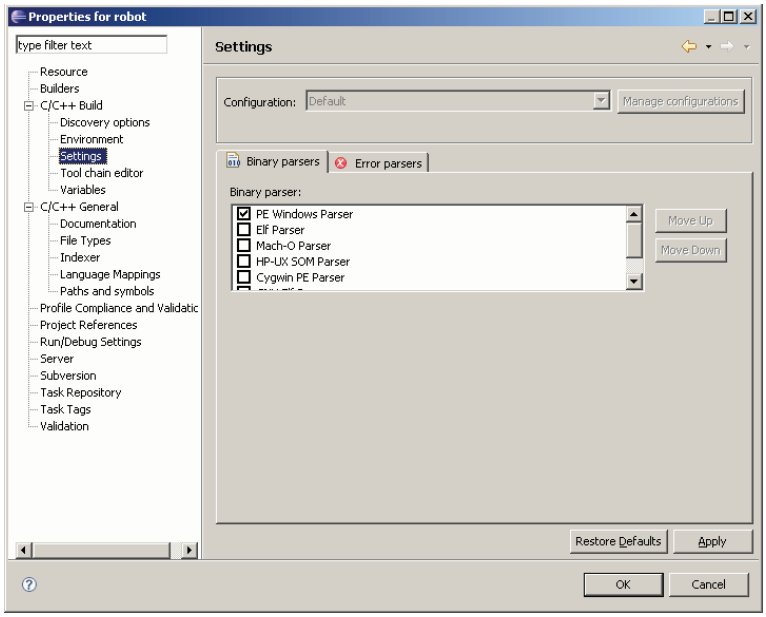
La configuración debe quedar como se muestra en las siguientes capturas:

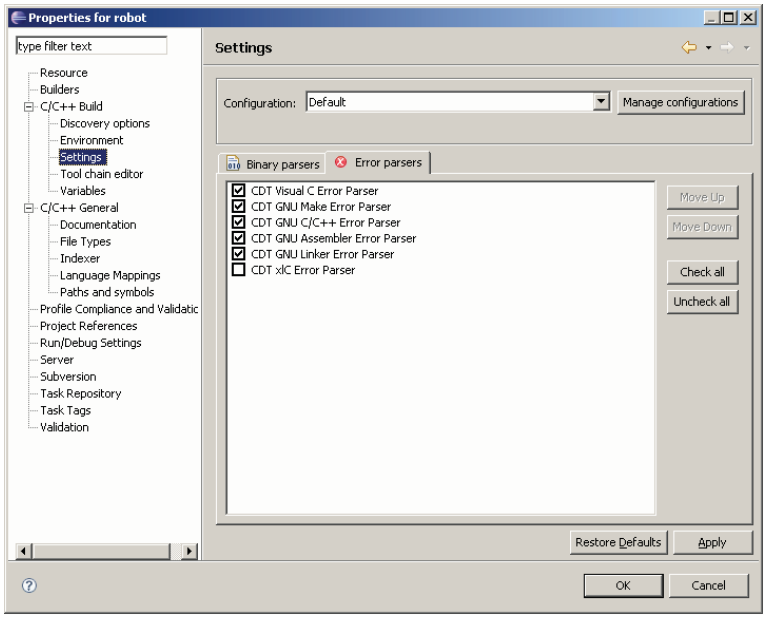

Properties for robot
type filter text
Resource
Builders
C/C++ Build
Discovery options
Environment
Settings
Tool chain editor
Variables
C/C++ General
Documentation
File Types
Indexer
Language Mappings
Paths and symbols
Profile Compliance and Validatic
Project References
Run/Debug Settings
Server
Subversion
Task Repository
Task Tags
Validation
C/C++ Build
Configuration: Default
Manage configurations
Builder settings
Behaviour
Builder
Builder type
External builder
Use default build command
Build command: scons.bat
Variables...
Makefile generation
Generate Makefiles automatically
Expand Env. Variable Refs in Makefiles
Build location
Build directory ${workspace_loc:/robot}
Workspace...
File system...
Variables...
Restore Defaults
Apply
OK
Cancel

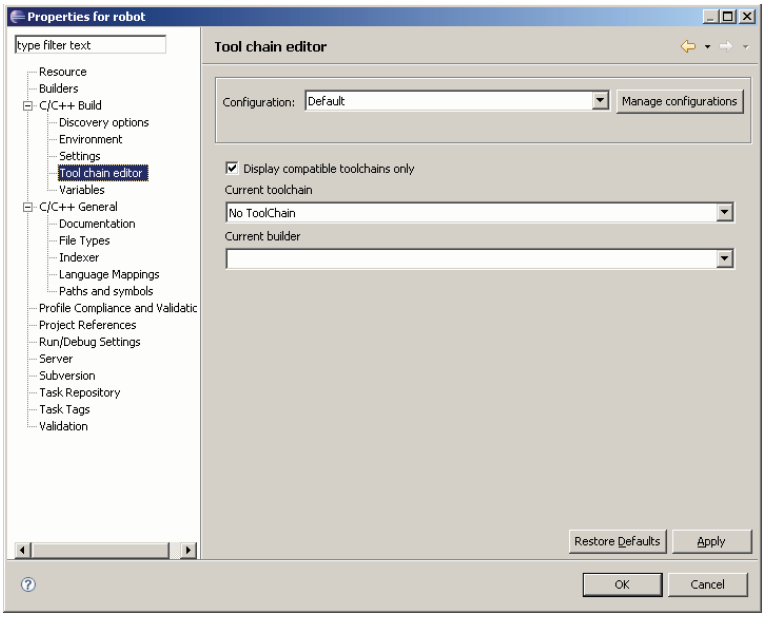

Properties for robot
type filter text
Resource
Builders
C/C++ Build
Discovery options
Environment
Settings
Tool chain editor
Variables
C/C++ General
Documentation
File Types
Indexer
Language Mappings
Paths and symbols
Profile Compliance and Validatic
Project References
Run/Debug Settings
Server
Subversion
Task Repository
Task Tags
Validation
C/C++ Build
Configuration: Default
Manage configurations
Builder settings
Behaviour
Build settings
Stop on first build error
Use parallel build
Use optimal jobs number
Use parallel jobs: 1
Workbench Build Behaviour
Workbench build type:
Make build target:
Build on resource save (Auto build)
all
Variables...
Note: See Workbench automatic build preference
Build (Incremental build)
Variables...
Clean
-c
Variables...
Restore Defaults
Apply
OK
Cancel

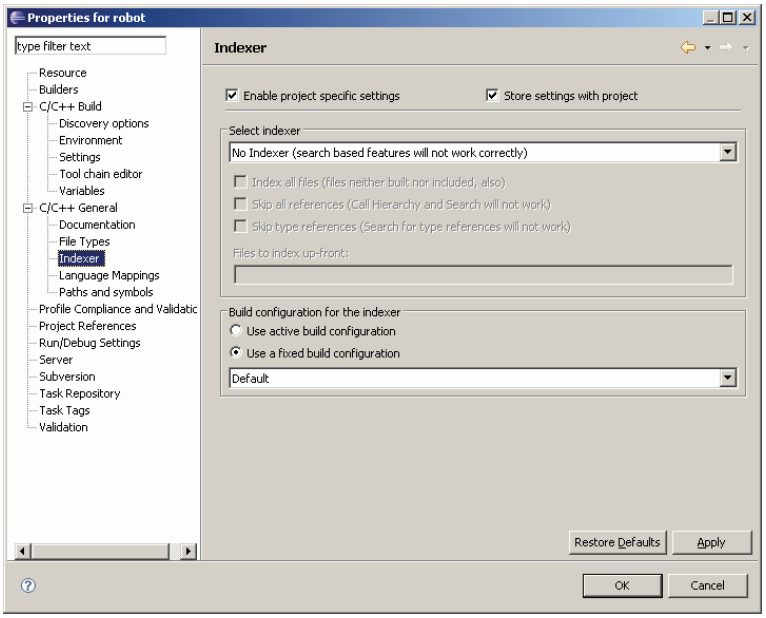

Properties for robot
type filter text
Resource
Builders
C/C++ Build
Discovery options
Environment
Settings
Tool chain editor
Variables
C/C++ General
Documentation
File Types
Indexer
Language Mappings
Paths and symbols
Profile Compliance and Validatic
Project References
Run/Debug Settings
Server
Subversion
Task Repository
Task Tags
Validation
Discovery options
Configuration: Default
Manage configurations
Discovery profiles scope
Configuration-wide
Default
Automated discovery of paths and symbols
Automate discovery of paths and symbols
Report path detection problems
Discovery profile: GCC per project scanner info profile
Discovery profile options
Enable build output scanner info discovery
Load build output from file
Load
Browse...
Variables...
Enable generate scanner info command
Compiler invocation command
gcc
Browse...
Restore Defaults
Apply
OK
Cancel


Properties for robot
type filter text
Resource
Builders
C/C++ Build
Discovery options
Environment
Settings
Tool chain editor
Variables
C/C++ General
Documentation
File Types
Indexer
Language Mappings
Paths and symbols
Profile Compliance and Validatic
Project References
Run/Debug Settings
Server
Subversion
Task Repository
Task Tags
Validation
Environment
Configuration: Default
Manage configurations
Environment variables to set
Variable
Value
PWD
E:\Facultad\5\SSII\robot
CWD
E:\Facultad\5\SSII\robot
New...
Select...
Edit...
Remove
Undefine
Append variables to native environment
Replace native environment with specified one
Restore Defaults
Apply
OK
Cancel


Properties for robot
type filter text
Resource
Builders
C/C++ Build
Discovery options
Environment
Settings
Tool chain editor
Variables
C/C++ General
Documentation
File Types
Indexer
Language Mappings
Paths and symbols
Profile Compliance and Validatic
Project References
Run/Debug Settings
Server
Subversion
Task Repository
Task Tags
Validation
Settings
Configuration: Default
Manage configurations
Binary parsers
Error parsers
Binary parser:
PE Windows Parser
Elf Parser
Mach-O Parser
HP-UX SOM Parser
Cygwin PE Parser
Move Up
Move Down
Restore Defaults
Apply
OK
Cancel


Properties for robot
type filter text
Resource
Builders
C/C++ Build
Discovery options
Environment
Settings
Tool chain editor
Variables
C/C++ General
Documentation
File Types
Indexer
Language Mappings
Paths and symbols
Profile Compliance and Validatic
Project References
Run/Debug Settings
Server
Subversion
Task Repository
Task Tags
Validation
Settings
Configuration: Default
Manage configurations
Binary parsers
Error parsers
CDT Visual C Error Parser
CDT GNU Make Error Parser
CDT GNU C/C++ Error Parser
CDT GNU Assembler Error Parser
CDT GNU Linker Error Parser
CDT xlC Error Parser
Move Up
Move Down
Check all
Uncheck all
Restore Defaults
Apply
OK
Cancel


Properties for robot
type filter text
Resource
Builders
C/C++ Build
Discovery options
Environment
Settings
Tool chain editor
Variables
C/C++ General
Documentation
File Types
Indexer
Language Mappings
Paths and symbols
Profile Compliance and Validatic
Project References
Run/Debug Settings
Server
Subversion
Task Repository
Task Tags
Validation
Tool chain editor
Configuration: Default
Manage configurations
Display compatible toolchains only
Current toolchain
No ToolChain
Current builder
Restore Defaults
Apply
OK
Cancel


Properties for robot
type filter text
Resource
Builders
C/C++ Build
Discovery options
Environment
Settings
Tool chain editor
Variables
C/C++ General
Documentation
File Types
Indexer
Language Mappings
Paths and symbols
Profile Compliance and Validatic
Project References
Run/Debug Settings
Server
Subversion
Task Repository
Task Tags
Validation
Indexer
Enable project specific settings
Store settings with project
Select indexer
No Indexer (search based features will not work correctly)
Index all files (files neither built nor included, also)
Skip all references (Call Hierarchy and Search will not work)
Skip type references (Search for type references will not work)
Files to index up-front:
Build configuration for the indexer
Use active build configuration
Use a fixed build configuration
Default
Restore Defaults
Apply
OK
Cancel

Compilación del proyecto

Para compilar, tan solo hay que acceder a la opción de la barra de menú "*Project → Build All*" habiendo seleccionado el proyecto previamente en el "*Project Explorer*".

Limpieza del proyecto

Para limpiar, tan solo hay que acceder a la opción de la barra de menú "*Project → Clean*" habiendo seleccionado el proyecto previamente en el "*Project Explorer*".

Depuración o Ejecución del proyecto bajo eclipse

Para depurar o ejecutar el proyecto, se debe crear una nueva configuración de depuración, especificando el depurador gdb de MinGW (en caso de querer depurar) y poniendo el fichero XML de configuración en la pestaña de argumentos.

Apéndice C. Microcontrolador PIC

Los '**PIC**' son una familia de microcontroladores tipo RISC fabricados por Microchip Technology Inc. y derivados del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instruments.

El nombre actual no es un acrónimo. En realidad, el nombre completo es **PICmicro**, aunque generalmente se utiliza como ***P**eripheral **I**nterface **C**ontroller* (Controlador de Interfaz Periférico).

El PIC original se diseñó para ser usado con la nueva UCP de 16 bits CP16000. Siendo en general una buena UCP, ésta tenía malas prestaciones de E/S, y el PIC de 8 bits se desarrolló en 1975 para mejorar el rendimiento del sistema quitando peso de E/S a la UCP. El PIC utilizaba microcódigo simple almacenado en ROM para realizar estas tareas; y aunque el término no se usaba por aquel entonces, se trata de un diseño RISC que ejecuta una instrucción cada 4 ciclos del oscilador.

En 1985, dicha división de microelectrónica de General Instruments se convirtió en una filial y el nuevo propietario canceló casi todos los desarrollos, que para esas fechas la mayoría estaban obsoletos. El PIC, sin embargo, se mejoró con EPROM para conseguir un controlador de canal programable. Hoy en día multitud de PICs vienen con varios periféricos incluidos (módulos de comunicación serie, UARTs, núcleos de control de motores, etc.) y con memoria de programa desde 512 a 32.000 palabras (una *palabra* corresponde a una instrucción en ensamblador, y puede ser 12, 14 o 16 bits, dependiendo de la familia específica de PICmicro).

Más información en

http://www.microchip.com/

http://www.todopic.com.ar/

Apéndice D. Comunicación USB

El **Universal Serial Bus** (bus universal en serie) es un puerto que sirve para conectar periféricos a una computadora. Fue creado en 1996 por siete empresas: IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC.

El estándar incluye la transmisión de energía eléctrica al dispositivo conectado. Algunos dispositivos requieren una potencia mínima, así que se pueden conectar varios sin necesitar fuentes de alimentación extra. La gran mayoría de los concentradores incluyen fuentes de alimentación que brindan energía a los dispositivos conectados a ellos, pero algunos dispositivos consumen tanta energía que necesitan su propia fuente de alimentación. Los concentradores con fuente de alimentación pueden proporcionarle corriente eléctrica a otros dispositivos sin quitarle corriente al resto de la conexión (dentro de ciertos límites).

El diseño del USB tenía en mente eliminar la necesidad de adquirir tarjetas separadas para poner en los puertos bus ISA o PCI, y mejorar las capacidades plug-and-play permitiendo a esos dispositivos ser conectados o desconectados al sistema sin necesidad de reiniciar. Cuando se conecta un nuevo dispositivo, el servidor lo enumera y agrega el software necesario para que pueda funcionar.

El USB puede conectar los periféricos como mouse, teclados, escáneres, cámaras digitales, teléfonos celulares, reproductores multimedia, impresoras, discos duros externos, tarjetas de sonido, sistemas de adquisición de datos y componentes de red. Para dispositivos multimedia como escáneres y cámaras digitales, el USB se ha convertido en el método estándar de conexión. Para impresoras, el USB ha crecido tanto en popularidad que ha empezado a desplazar a los puertos paralelos porque el USB hace sencillo el poder agregar más de una impresora a una computadora personal.

En el caso de los discos duros, el USB es poco probable que reemplace completamente a los buses como el ATA (IDE) y el SCSI porque el USB tiene un rendimiento un poco más lento que esos otros estándares. El nuevo estándar Serial ATA permite tasas de transferencia de hasta aproximadamente 150/300 MB por segundo. Sin embargo, el USB tiene una importante ventaja en su habilidad de poder instalar y desinstalar dispositivos sin tener que abrir el sistema, lo cual es útil para dispositivos de almacenamiento externo. Hoy en día, una gran parte de los fabricantes

ofrece dispositivos USB portátiles que ofrecen un rendimiento casi indistinguible en comparación con los ATA (IDE).

El USB no ha remplazado completamente a los teclados AT y mouse PS/2, pero virtualmente todas las placas base de PC traen uno o más puertos USB.

Más información en:

http://www.usb.org/home

Apéndice E. AJAX

AJAX, acrónimo de *Asynchronous JavaScript And XML* (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o **RIA** (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante *XMLHttpRequest*, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

AJAX es una combinación de cuatro tecnologías ya existentes:

- **XHTML** (o **HTML**) y hojas de estilos en cascada (CSS) para el diseño que acompaña a la información.
- Document Object Model (DOM) accedido con un lenguaje de scripting por parte del usuario, especialmente implementaciones ECMAScript como JavaScript y JScript, para mostrar e interactuar dinámicamente con la información presentada.
- El objeto **XMLHttpRequest** para intercambiar datos asincrónicamente con el servidor web. En algunos frameworks y en algunas situaciones concretas, se usa un objeto `iframe` en lugar del XMLHttpRequest para realizar dichos intercambios.
- XML es el formato usado generalmente para la transferencia de datos solicitados al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSON y hasta EBML.

Como el DHTML, LAMP o SPA, AJAX no constituye una tecnología en sí, sino que es un término que engloba a un grupo de éstas que trabajan conjuntamente.

Más información en:

http://www.uberbin.net/archivos/internet/ajax-un-nuevo-acercamiento-a-aplicaciones-web.php

http://www.librosweb.es/ajax/index.html

Bibliografía

- Cómo conectar por Sockets Java y C++
  http://www.chuidiang.com/java/sockets/cpp_java/cpp_java.php
- Librería de Sockets
  http://www.adp-gmbh.ch/win/misc/sockets.html
- Página de GWT
  http://code.google.com/webtoolkit/
- GWT in action
  Robert Hanson
  Adam Tacy
  Ed. Manning
- Página de Scons
  http://www.scons.org/
- Visión por computador. Fundamentos y métodos.
  Arturo de la Escalera.
  Ed. Prentice Hall.
- Visión por computador: imágenes digitales y aplicaciones.
  Gonzalo Pajares Martinsanz
  Ed. Ra-Ma
- Wikipedia
  http://www.wikipedia.org
- Open Source Computer Vision Library
  http://www.intel.com/technology/computing/opencv/index.htm
  http://openvlibrary.sourceforge.net

- Memorias de proyectos anteriores:
  "Robot guía del Museo García Santesmases"
  "Sistema Software para el Robot Guía del Museo de Informática García Santesmases"
- Ingeniería del software. Un enfoque práctico.
  R. Pressman
  Ed. McGraw-Hill