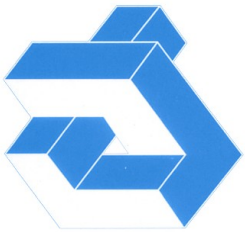
UNIVERSIDAD POLITÉCNICA DE MADRID



DEPARTAMENTO DE
AUTOMÁTICA
INGENIERÍA ELECTRÓNICA
E INFORMÁTICA INDUSTRIAL

División de Ingeniería de Sistemas
y Automática (DISAM)

Interfaz de teleoperación y control para plataforma robótica móvil

Autor: Julio César Larriba García

Tutor: Ricardo Sanz Bravo

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS INDUSTRIALES
José Gutiérrez Abascal, 2
28006 Madrid

Madrid septiembre 2009

Agradecimientos

A Ricardo, por la libertad, confianza y oportunidad ofrecida para realizar este proyecto. Por todas las atenciones, por el tiempo que ha invertido en mí, y sobre todo por su apoyo.

A Adolfo, que gracias a su infinita paciencia, a sus enormes conocimientos de programación y a la ayuda aportada han hecho posible que pueda estar escribiendo este documento.

A Carlos, por su disposición y tiempo, siempre que le he necesitado ha estado para echarme una mano. Gracias por los consejos y comentarios que han mejorado este proyecto.

A mis amigos del DISAM, José Alberto, Guillermo, Carlos, Roberto, Lorena, Patricia, Pablo, gracias por hacer mi estancia en el departamento un placer.

Al grupo de ASLab, Guadalupe, Marcos e Iñaki por las ayudas recibidas en el desarrollo del proyecto.

A los “habitantes” de la sala de cálculo, Vicente, Joao, Paloma, José, Antonio, y demás gentes, gracias por los buenos momentos.

A mis padres, a mi hermano y a toda mi familia, gracias por apoyarme en todo momento, si no fuera por ellos no estaría a punto de cumplir un sueño.

A mis amigos y compañeros de piso, gracias por compartir vuestras vidas conmigo, habéis hecho posible que esté donde estoy.

En memoria de mi tío Eusebio y de Lucas.

Índice general

I Preliminares

II DESARROLLO

III INFORMACIÓN COMPLEMENTARIA

Índice de figuras

Capítulo 1

1. INTRODUCCIÓN

1. 1 PREÁMBULO

La robótica móvil es un campo de desarrollo reciente y de fuerte investigación en prestigiosas universidades de todo el mundo. Para poder desarrollar de forma completa este documento, es necesario introducirnos dentro de los inicios de la robótica. Así púes, las primeras cuestiones a resolver son.

¿**Qué es un robot?**

Un robot es una máquina controlada por ordenador y programada para moverse, manipular objetos y realizar trabajos a la vez que interacciona con su entorno. Su objetivo principal es el de sustituir al ser humano en tareas repetitivas, difíciles, desagradables e incluso peligrosas de una forma más segura, rápida y precisa.

Algunas definiciones aceptadas son las siguientes:

- "*Dispositivo multifuncional reprogramable diseñado para manipular y/o transportar material a través de movimientos programados para la realización de tareas variadas*." (Robot Institute of America, 1979).

- "*Dispositivo automático que realiza funciones normalmente adscritas a humanos o máquina con forma humana*." (Webster Dictionary).

Esta última definición, sin embargo, no es la más acertada, ya que un robot no tiene porqué tener forma humana. Un lavavajillas es un robot, así como los satélites artificiales, el "tractor" lunar soviético Lunakhod o la sonda exploradora de la NASA Mars Pathfinder. Toda una refinería petrolífera controlada por computador también puede ser considerada un robot.

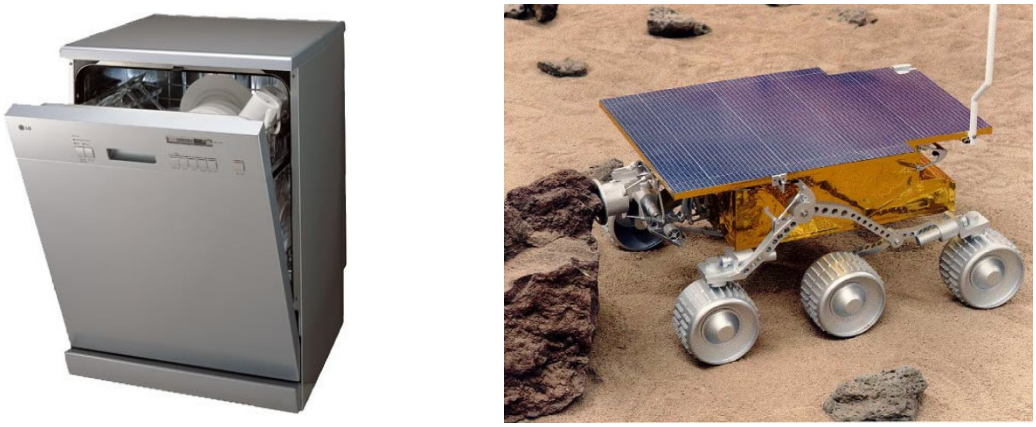


Figura 1.1: En el más amplio sentido de la palabra, hasta un lavavajillas (izquierda) puede ser un robot y al contrario que la creencia popular, los robots normalmente no tienen apariencia humana.

¿Qué significa la palabra robot?

El término procede de la palabra checa *robota*, que significa 'trabajo obligatorio'; fue empleado por primera vez en la obra teatral R.U.R. (Robots Universales de Rossum), estrenada en Enero de 1921 en Praga por el novelista y dramaturgo checo **Karel Capek**. La obra fue un éxito inmediato y pronto se estrenó en multitud de teatros por toda Europa y Estados Unidos. En ella, el gerente de una fábrica construía unos seres al absoluto servicio del hombre, que realizaban todas las tareas mientras los humanos se dedicaban al ocio permanente.

Cuando el gerente de la fábrica decide construir robots más perfectos que experimentaran felicidad y dolor, todo cambia. Los robots se sublevan contra los hombres y destruyen al género humano.

La nueva capacidad de las máquinas para comunicar y controlar procesos, dirigir operaciones y cumplir las órdenes, e incluso aprender, llevó al desarrollo de una nueva ciencia: la **Cibernética**, palabra que deriva del vocablo griego *Kybernetes*, que significa "timonel". Fundada en la década de 1940 por el matemático norteamericano **Norbert Wiener**, es la ciencia que estudia la comunicación entre el hombre y la máquina, y entre las propias máquinas. De la mano de la Cibernética se desarrolló la **Biónica**, ciencia que estudia todos los aspectos relativos a la simulación de actividades humanas y animales por medio de máquinas.

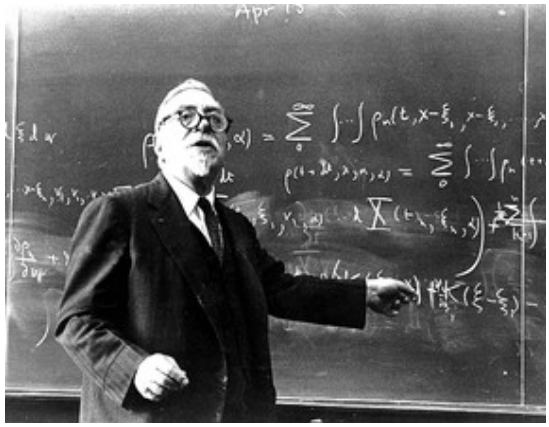


Figura 1.2: Issac Assimov Norbert Wiener

Una vez definido qué es un robot y su significado; ¿Qué caracteriza un robot?

Los robots exhiben tres elementos claves según la definición adoptada:

- **Programabilidad**, lo que significa disponer de capacidades computacionales y de manipulación de símbolos (el robot es un computador).
- **Capacidad mecánica**, que lo capacita para realizar acciones en su entorno y no ser un mero procesador de datos (el robot es una máquina).
- **Flexibilidad**, puesto que el robot puede operar según un amplio rango de programas y manipular material de formas distintas.

Con todo, se puede considerar un robot como una máquina complementada con un computador o como un computador con dispositivos de entrada y salida sofisticados. La idea más ampliamente aceptada de robot está asociada a la existencia de un dispositivo de control digital que, mediante la ejecución de un programa almacenado en memoria, va dirigiendo los movimientos de un brazo o sistema mecánico. El cambio de tarea a realizar se verifica ordenando el cambio de programa.

¿Qué es la robótica?

La **robótica** es la ciencia y la tecnología de los robots. Se ocupa del diseño, manufactura y aplicaciones de los robots. La robótica combina diversas disciplinas como son: la mecánica, la electrónica, la informática, la inteligencia artificial y la ingeniería de control. Otras áreas importantes en robótica son el álgebra, los autómatas programables y las máquinas de estados.

El área de conocimiento en la que se enmarca la Robótica es la **Automática**, definida por la Real Academia de las Ciencias como la *disciplina que se ocupa de los métodos y*

procedimientos cuya finalidad es la sustitución del operador humano por un operador artificial en la ejecución de una tarea física y mental, previamente programada. A partir de esta definición, en la Automática se pueden diferenciar dos componentes claros:

- Una **Unidad de Control** que gobierna las acciones a realizar. Este gobierno debe cumplir ciertos criterios u objetivos del control como la estabilización ante perturbaciones, o la evolución temporal y el comportamiento dinámico óptimo respecto a determinados parámetros de calidad. Los avances en el campo de la inteligencia artificial permiten dotar a estas unidades de aspectos más avanzados como la toma de decisiones o el aprendizaje.
- Un **Actuador** que realiza las acciones programadas bajo la supervisión de la unidad de control. Estos dispositivos pueden ir desde los casos más elementales, como accionadores hidráulicos, neumáticos o electromecánicos hasta máquinas más complejas como manipuladores, máquinas−herramientas y, quizás los autómatas por excelencia, los robots.

La coordinación entre ambos componentes mediante el intercambio de información es lo que permite conseguir la realización correcta de las tareas a realizar. Puesto que es posible definir la **Informática** como la *ciencia que estudia el tratamiento de la información*, es evidente que existe una relación clara entre Automática e Informática.

Será, en rasgos generales, la rama de la ciencia donde se desarrollará los principales temas tratados en este documento.

Y para definir conceptos esenciales ¿Qué tipos de robots existen?

Existen diferentes tipos y clases de robots, entre ellos con forma humana, de animales, de plantas o incluso de elementos arquitectónicos pero todos se diferencian por sus capacidades y se clasifican en 4 formas:

1. Androides: robots con forma humana. Imitan el comportamiento de las personas, su utilidad en la actualidad es de solo experimentación. La principal limitante de este modelo es la implementación del equilibrio a la hora del desplazamiento, pues es bípedo.
2. Móviles: se desplazan mediante una plataforma rodante (ruedas); estos robots aseguran el transporte de piezas de un punto a otro.

3. Zoomórficos: es un sistema de locomoción imitando a los animales. La aplicación de estos robots sirve, sobre todo, para el estudio de volcanes y exploración espacial.
4. Poliarticulados: mueven sus extremidades con pocos grados de libertad. Su utilidad es principalmente industrial, para desplazar elementos que requieren cuidados.

Nuestra plataforma robótica, está englobado dentro del grupo 2, con lo cual nos centraremos en ese tipo de robot.

Según la IFR los robot se pueden clasificar según su uso y aplicación:

- Servicio a humanos (personal, protección, entretenimiento, asistentes, ...)
- Servicio a equipos (mantenimiento, inspección, reparación, limpieza, ...)
- Otras funciones autónomas (vigilancia, transporte, adquisición de datos, ...)

De una manera más concreta se puede indicar que los robots de servicio operan en sectores y realizan actividades como: medicina, domestico y oficina, ocio y entretenimiento, servicios de comunidad y la industria de la construcción [ref 1].

1.2 ANTECEDENTES: ROBOTS MÓVILES

1.2.1 Ventajas de la robotización

La Robótica y la Automatización siempre han ofrecido al sector industrial un excelente compromiso entre productividad y flexibilidad, una calidad uniforme de los productos, una sistematización de los procesos y la posibilidad de supervisar y/o controlar las plantas según diferentes parámetros y criterios. Se pueden destacar cuatro ventajas principales de los sistemas robotizados: aumento de la productividad, alta flexibilidad, excelente calidad y mejora de la seguridad. Como resultado, la robotización permite mejorar la calidad y las condiciones de trabajo, sustituyendo tareas penosas por otras que se efectúan en condiciones mucho más ventajosas. Pero, además, la irrupción de la automatización en los servicios y el ocio permite mejorar la calidad de vida de los ciudadanos [ref 2].

1.2.2 Historia

Tradicionalmente las aplicaciones de la Robótica y la Automatización estaban centradas en los sectores manufactureros más desarrollados para la producción masiva: industria del automóvil, transformaciones metálicas, industria química, etc. aunque en la última década el peso de la industria manufacturera ha ido bajando. Los robots comienzan a fabricarse para tareas muy específicas después de la Segunda Guerra Mundial. Como por ejemplo los que se necesitaban en las investigaciones espaciales (como el Surveyor que aterriza en la Luna, en 1966, o el Viking, que aterriza en Marte diez años después) para aquellas funciones en las que se exigían ciertas destrezas para resolver situaciones no completamente definidas, o las que se requerían para trabajar en ambientes altamente nocivos para la vida humana como las centrales nucleares, en condiciones térmicas no soportable o ciertos ambientes químicos o biológicos de alta toxicidad.

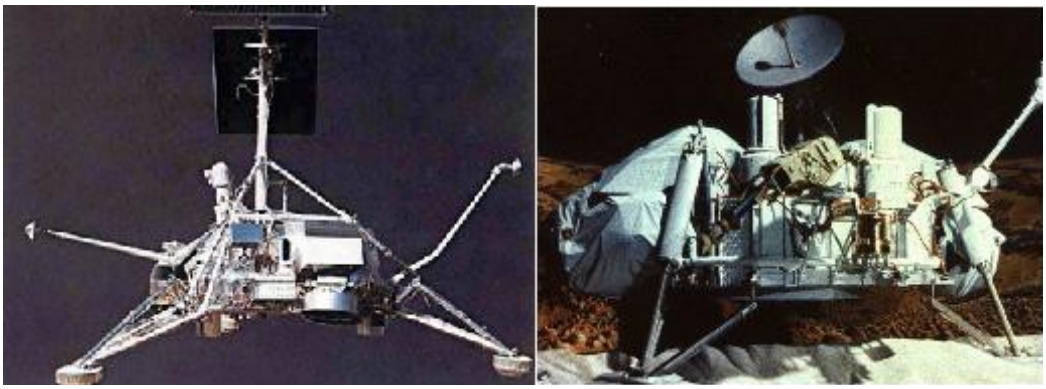


Figura 1.3: Surveyor (izquierda) Viking (derecha)

En general, estaban asociados a proyectos de investigación de presupuestos muy elevados, en los que incluir partidas para la construcción de costosos robots resultaba factible. De estas investigaciones también saldrían los diseños de robots aplicados a actuaciones militares. También por estos años surgieron algunos robots orientados a atender ciertas tareas en los procesos de fabricación. Aunque la primera patente de un robot industrial está registrada en Inglaterra en el año 1954, suelen considerarse como primeros robots industriales los Unimates instalados en las fábricas de la General Motors y construidos por George Devol y Joe Engelberger (a quienes se conoce como padres de la robótica) a finales de los años cincuenta y primeros de los sesenta.

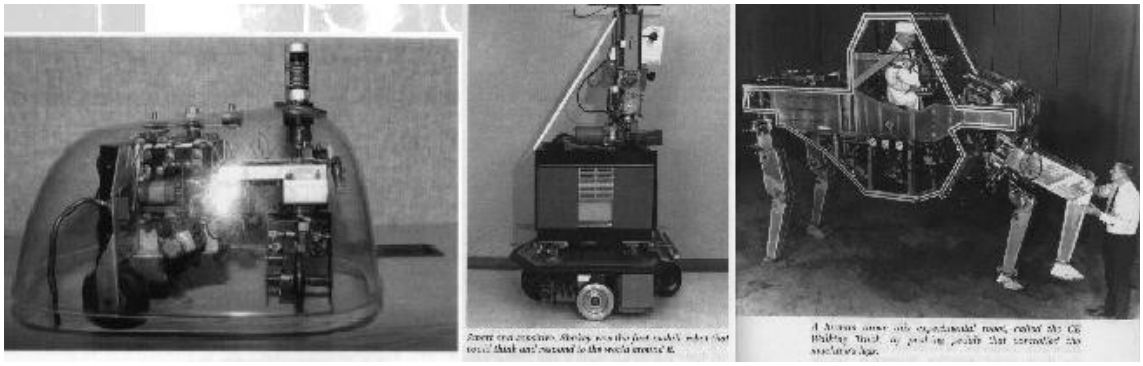


Figura 1.4: Tortuga de Grey Walter (izquierda). Shakey, primer robot capaz de pensar e interactuar (centro). Vehículo eléctrico caminante (derecha)

Tras un lento crecimiento inicial, el uso de los robots industriales experimentó durante los años 70 y 80 un vertiginoso crecimiento desde la aparición de los primeros robots servocontrolados, instalándose con autoridad en muchas de las industrias manufactureras, en particular la del automóvil. En el presente, hay unas 800.000 unidades operativas en todo el mundo. Este número de robots cubre en gran medida todas aquellas instalaciones en las que el uso del robot es económicamente rentable. Por este motivo parece probable que el número de robots industriales instalados se quede finalmente estancado en torno a esa cifra.

No obstante, desde mediados de los años 80 han surgido otras aplicaciones para los robots, no ligadas a la manufactura, y en la que se busca en el empleo del robot ventajas

Introducción al diseño de microrrobots móviles 9

Aplicaciones de Robots Móviles distintas a la del aumento de la productividad. Estas aplicaciones se caracterizan en muchos casos por desarrollarse fuera del ambiente estructurado propio de una fábrica, por lo que los robots que las desarrollan deben reunir una serie de características particulares.

Dado que sus funciones suelen estar ligadas al desarrollo de servicios útiles para los hombres o las máquinas, estos robots se han venido a denominar robots de servicio.

En resumen, podemos hacer una distinción de tres generaciones, dado que los cambios en Robótica se suceden tan deprisa que ya se ha pasado de unos robots relativamente primitivos a principios de los 70, a una segunda generación. La primera generación de

robots era reprogramable, de tipo brazo, dispositivos manipuladores que sólo podían memorizar movimientos repetitivos, asistidos por sensores internos que les ayudan a realizar sus movimientos con precisión. La segunda generación de robots entra en escena a finales de los 70, tienen sensores externos (tacto y visión por lo general) que dan al robot información realimentación) del mundo exterior. Estos robots pueden hacer elecciones limitadas o tomar decisiones y reaccionar ante el entorno de trabajo, se les conoce como robots adaptativos.

Figura 1.5: Telemanipuladores de Goertz. Argonne National Laboratory (1948)

1.2.3 Divisiones de las aplicaciones de robots móviles

Según el desarrollo histórico comentado, los ayudantes mecánicos del hombre se pueden dividir en dos grandes grupos de aplicaciones: aplicaciones tradicionales , centradas básicamente en la industria manufacturera, utilizados en la fabricación y cuya función está regida por un programa preestablecido; y aplicaciones innovadoras, cuya mayoría está centrada en el sector servicios, entendiendo éste de forma amplia y agrupando sectores no tradicionales, capacitados para 'razonar' ante un conjunto de situaciones o contexto.

El sector industrial, es el área que como consecuencia de su crecimiento impulso el desarrollo de la robótica. Como se ha comentado anteriormente son innumerables las ventajas que aporta la robótica a la industria en especial con respecto a la productividad.

La robótica empleada en el sector industrial en su mayor parte es asistida por ordenador, para así poder reprogramarse cuando se desee cambiar la función de dicho robot, como por ejemplo, si deseamos fabricar un modelo de un coche y tiempo después otro totalmente distinto. Estas ventajas son las que han impulsado la robótica, puesto que con una misma máquina podemos fabricar distintas cosas simplemente cambiando un programa.

Figura 1.6 Ejemplo de robots industriales

El sector servicios, es una de las áreas de aplicación más novedosa. Los robots incluidos en él son aquellos que realizan servicios en beneficio de los humanos o para el mantenimiento de infraestructuras y equipos, excluidas las operaciones de fabricación. De una manera más concreta, se puede indicar que los robots de servicio operan en sectores y realizan actividades como: espacio, construcción, médico, submarino, nuclear, limpieza, agricultura, doméstico y de oficina, militar y seguridad, ocio y entretenimiento. Se estima que en los próximos diez años el sector pueda requerir necesidades en robótica con un volumen de negocio comparable con el del sector industrial.

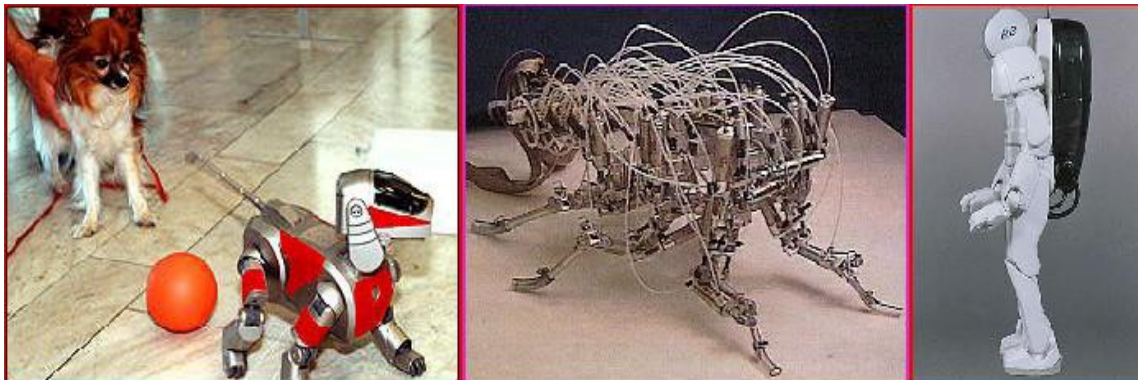


Figura 1.7: Robots de servicios. Robot perro Aibo de Sony (izquierda). 'Robot III' robot insecto hexapodo (centro). 'P3' humanoide de Honda (derecha).

El sector militar es, como en muchos campos científicos, uno de los impulsores de la robótica. Aunque su primer impulsor fue el área industrial actualmente uno de sus impulsores es el área militar. Esto es debido a que cualquier ejército busca tener el armamento tecnológicamente más avanzado y la robótica proporciona un aliado importante puesto que puede añadir funciones de vigilancia o combate sin representar un coste de vidas humano. Actualmente podemos encontrar robots que pueden ayudar en combate puesto que pueden vigilar o desactivar minas aunque la tendencia más creciente es a crear un robot que sustituya a la infantería, es decir, que se capaz de combatir detectando a los enemigos y no requiera la intervención de un ser humano.

Figura 1.8: Robots Militares: 'MERV' Desactivador de bombas remoto (izquierda). 'Cypher' helicóptero militar de control remoto (derecha)

Independientemente del sector militar podemos encontrar del sector de vigilancia que aunque su finalidad es igual a la del sector militar se estable un vinculo con el sector de servicios puesto que son robots creados para detectar intrusos en ciertos lugares y no enemigos en los campos de batalla.

Actualmente el presupuesto dedicado al sector de I+D en robótica es cada vez mayor debido a todas las ventajas comentadas con anterioridad. La investigación en robótica ha dejado de ser algo exclusivo de empresas y ha pasado a desarrollarse de manera pública en universidades donde existen numerosos grupos de trabajo en distintas áreas. Al contrario que ocurría con anterioridad cuando exclusivamente se trabajaba en robots industriales o militares ahora se pueden encontrar un gran elenco de desarrollos con utilidades distintas que pueden ser empleados en distintos tipos de robots. Proyectos para crear inteligencia artificial o para simular movimientos cada vez más humanos contribuyen a desarrollar el área de la robótica para poder crear robots cada vez más parecidos al ser humano y que puedan ayudarle con sus tareas cotidianas.

1.3. Sectores de la Robótica Móvil

1.3.1. Aplicaciones Industriales

La utilización de robots industriales está ampliamente extendida en todo tipo de fábricas y empresas industriales, obteniendo con ellos reducción de costes, aumento de la productividad, mejora de la calidad en la producción y eliminación de condiciones peligrosas de trabajo o mejora de las mismas. De este modo, la empresa industrial, a través de inversiones tecnológicas en el campo de la automatización industrial, podrá aumentar su competitividad en el mercado, corriendo el riesgo de quedarse rezagada en el mercado si descartase la utilización de la robótica en sus procesos de fabricación. El principal papel de los robots es articular diferentes máquinas y funciones productivas; transporte, manejo de materiales, maquinado, carga y descarga, etc. mediante su capacidad para desempeñar diversas tareas u operaciones. El robot industrial ha sido descrito como el elemento más visible de la fabricación asistida por computador y como la base técnica para la mayor automatización de la producción [ref 3].

La inmensa mayoría de los robots industriales se componen de un brazo articulado, a través del cual desempeñan su tarea en una cadena de producción. Este tipo de robots se sale del marco de nuestra asignatura, por no ser ni 'microrrorobots', ni 'móviles', de modo que dejaremos de lado esta amplia familia para indagar en robots más próximos a los tratados en clase, en busca de cierta inspiración para el posterior diseño de nuestros 'microrrobots móviles'. Dentro de las aplicaciones industriales, al más típicas son:

- **Limpieza Industrial**

    - Inspección y Limpieza de conductos
    - Inspección y Limpieza de barcos
    - Limpieza de fachadas (cristales)
    - Limpieza de aviones

- **Transporte de mercancías**

    - Vehículos de Guiado Automático (AGV)
    - Vehículos Laserguiados (LGV)
    - Vehículos Filoguiados (RGV)
    - Automated Material Transport System (AMTS)

- **Agricultura**

    - Tractor Autónomo
    - Cosechadora Autónoma
    - Sector Automovilístico

Figura 1.9: Ejemplos de robots industriales, Robot remolcador y robot de limpieza de suelos

1.3.2 Aplicaciones innovadoras y de servicio

Los ayudantes mecánicos del hombre están divididos en dos tipos principales: los automatas industriales, utilizados en la fabricación y cuya función está regida por un programa preestablecido, y los de servicio, capacitados para razonar, ante un conjunto de situaciones o contexto. El sector servicios, tanto personal como colectivo, es una de las áreas de aplicación más novedosa. Se estima que en los próximos diez años el sector pueda requerir necesidades en robótica con un volumen de negocio comparable con el del sector industrial (tradicional o manufacturero) [4].

DEF: IEEE (Institute of Electrical and Electronics Engineers), los robots de servicios son aquellos que de forma semiautomática o totalmente automática realizan servicios en beneficio de los humanos o para el mantenimiento de infraestructuras y equipos, excluidas las operaciones de fabricación. Los tipos de robots de innovación y servicio son:

- Medicina
- Domestico y oficina
- Ocio y entretenimiento
- Servicios de comunidad
- Industria de la construcción

Figura 1.10: El famoso robot de entretenimiento Aibo de Sony (izquierda) y el robot Rudy del centro médico de la universidad de california (derecha), su función principal consiste en permitir que los médicos interactúen con los pacientes recién operados sin necesidad de estar presentes.

1.3.3 Aplicaciones de vigilancia

Vigilancia: Cuidado y atención exacta en las cosas que están a cargo de cada uno. Servicio ordenado y dispuesto para vigilar.

A la vista de esta definición y aplicándolo al tema que nos ocupa que es la aplicación a la robótica móvil, tenemos una serie de robots que se crean para cumplir sobre todo con la segunda parte de esta definición. La primera parte de la definición también la podemos tener en cuenta en caso de que tengamos un robot vigilante telecontrolado.

Cada uno tendrá unas funciones específicas, como encargarse de patrullar por un entorno detectando anomalías, otros se dedicarán a observar una zona concreta para evaluar la situación, otros se usarán como apoyo a los militares en las misiones para controlar una zona ya tomada o buscar objetivos u otras misiones posibles que se les puede asignar.

Además de la propia función de vigilancia también se les podría poner armamento no letal para disuadir en su caso del intruso o para la autoprotección del robot.

Lo primero que haremos será ver la estructura sobre la cual se están montando este tipo de robots, según el medio que se vaya a tener controlado (tierra, aire o mar). Dentro de la tierra, lo que se verán las distintas plataformas sobre los que puede estar implementado, pues el robot se puede mover sobre ruedas, oruga, esférico, etc.

A continuación agruparemos las características comunes a este tipo de robots, tales como sensores que más o menos todos llevan. Una vez descritos estos sensores, su

funcionamiento, características específicas de algunos y detalles, lo que haremos será pasar a controlarlos, fusionar los resultados. Esto lo haremos mediante algoritmos específicos que se sustentarán en la información proporcionada del entorno para tratarla de distintas formas y después extraer lo que nos interese, como por ejemplo, posición o localización en un mapa dado, detección de movimiento, reconocimiento de formas y lugares y navegación.

A continuación, daremos una breve pasada por los distintos sistemas de control que tenemos aplicados en algunos robots o en proyectos que se están realizando por parte del DoD americano, ya sea telecontrol aplicado a robots (telerrobótica), autónomos, mixtos y con control jerárquico.

Posteriormente describiremos algunos ejemplos ya realizados de robots vigilantes y veremos sus características, comportamiento en distintos entornos, kits de montaje para hacerse uno mismo uno, comparativa entre robots que se pueden usar en un mismo entorno con las ventajas e inconvenientes de cada uno de ellos.

Por último, se describirán algunos de los proyectos (realizados o en curso) de robots vigilantes por parte del centro de investigación de San Diego, el cual forma parte del DARPA (DoD). Veremos las distintas partes de proyectos realizados, zonas de aplicación, especificaciones generales de lo que se deben cumplir y se verá cómo algunos de esos robots han pasado de ser experimentales a derivarse en otros robots que ya son reales y han actuado en diversos conflictos.

1.3.4 Aplicaciones militares

Desde las primeras civilizaciones la guerra ha sido una aspecto muy presente en la sociedad, desde la conquista de gran parte de Europa por parte de los Romanos, pasando por las cruzadas de Occidente por lograr el control de las tierras santas hasta las guerras napoleónicas y muchas otras, la historia del hombre siempre ha ido de la mano con la guerra.

Pero a principios del siglo XX, donde curiosamente se iban a producir las peores guerras de toda la historia, se cambio el concepto en cuanto a lo que era la base de las máquinas de guerra hasta entonces, la infantería.

La infantería era y sigue siendo en parte, un elemento un gran elemento a tener en cuenta para ganar una guerra. La infantería siempre está en primera línea, luchando en muchos casos cuerpo a cuerpo, es la fuerza de choque directo contra el enemigo, puede

luchar en todo tipo de terrenos y en grupos muy pequeños haciendo escaramuzas (como el ejército español durante la ocupación napoleónica) o en grandísimos ejércitos en campo abierto, como la infantería inglesa durante los siglos XVII, XVIII y XIX.

Al estar siempre en la primera línea de fuego, la infantería siempre ha sufrido todo tipo de desgracias y la mayor parte de las bajas (ya sean muertes, heridos o desaparecidos) de los ejércitos.

Pero a principios del siglo XX, da idea acerca del soldado de infantería cambio radicalmente y se le dejo de tratar como si fuese una arma más, y empezaron a darse cuenta que la gran cantidad de bajas en la infantería se podían reducir con medidas como es el uso de cascos de metal en vez de tela, la inclusión de médicos en la línea del frente y otras medidas, hicieron que se salvaran muchas vidas en todas las ramas de los ejércitos, pero en especial en la infantería, se cambio radicalmente la idea de que el soldado era una herramienta más y se empezó a valorar su vida como importante, no como en el pasado que eran considerados auténtica 'carne de cañón'.

Esta idea de la importancia de no perder de vidas durante el combate y poder recuperar soldados para tras curarlos poder volver a la batalla, ha ido imponiéndose durante todas las guerras del siglo XX, además de por ética y moral, por el decremento en el gasto militar si se un soldado se puede reincorporar. Alcanzo el nivel actual donde la guerra de Vietnam, donde el salvar a un soldado herido se convirtió en una máxima para el ejército, y se hacían todo tipo de esfuerzos por lograrlo.

Actualmente, la infantería ya no la forman miles de persona y la idea de esta es otra, no es una fuerza bruta de choque, que acaba con el enemigo por la fuerza de su ataque; actualmente se trata de grupos relativamente pequeños, cuyo objetivo es acabar con objetivos muy específicos y sin sufrir bajas de ningún tipo, aún así al estar en las primeras posiciones de la batalla o en misiones en territorio enemigo, es inevitable que existan muchas bajas en estos cuerpos.

Por ello, los grandes ejércitos, desde la Segunda Guerra Mundial hasta ahora, se han interesado en conseguir máquinas que fuesen capaces de hacer las mismas tareas que un soldado de la primera fila de fuego; ya que si se consiguiese, se ahorraría mucho dinero además de evitar la pérdida de muchas vidas.

Los militares no solo se han interesado en máquinas o robots soldado que hagan el trabajo de la infantería o de otras tropas, si no en todo tipo de tareas en las que los soldados que las realizan estuviesen en gran peligro; como es por ejemplo el espionaje

desde el aire, durante la Primera y Segunda Guerra Mundial, miles de pilotos que se dedicaban a la observación de las líneas enemigas perdieron sus vidas mientras realizaban estas misiones.

Al igual que ocurre con otras tareas como son la desactivación de minas o de bombas, en las que el más mínimo fallo puede hacer perder la vida al encargado de esa tarea.

Al sector militar está muy interesado en la robótica, porque conseguir grandes avances en ello, repercutiría en un gran ahorro; tanto económicamente ya que el mantenimiento de un ejército de robots sería mucho mejor que el de un ejército de personas, como un gran ahorro en vidas.

Los primeros robots llegan al campo de batalla durante la Segunda Guerra Mundial, cuando el ejército alemán saca a la luz a 'Goliat', una de sus extrañas armas secretas, se trata de un pequeño vehículo de un metro y medio de largo y unos 350 Kg. de peso, movido por orugas (su diseño es muy similar a un tanque), que era capaz de cargar de unos 60 Kg. de TNT.

El objetivo del Goliat era atacar nidos de ametralladora o pequeñas fortificaciones haciendo estallar su contenido una vez estuviese pegado a ellos, es decir, se inmolaba. Podríamos definir el Goliat como una mina móvil.

El Goliat se controlaba mediante control remoto, que transmitía las instrucciones mediante un cable, por lo tanto la distancia máxima de uso era de entre 600 y 1500 m. (la longitud del cable). El Goliat en sus primeras versiones solamente contaba con un motor eléctrico alimentado mediante unas potentes baterías, pero el rendimiento de estas no fue el esperado, así que se le introdujo un motor de gasolina, que le permitía recorrer la distancia hasta la línea del frente por sí mismo, sin tener que ser transportado, y cuando iba a producirse su ataque hacía uso del motor eléctrico.

Figura 1.11: Detalle de un Goliat (modelo SdKfz 302) que se ha podido conservar

Sin embargo, se empezaron a hacer mejoras en el diseño del Goliat, sobretodo en cuanto a su estructura, para que ocupando lo mismo, cupiesen unas baterías más potentes y un sistema de tracción mejorado. Con esto y una capa de blindaje adicional, empezaron a construirlo en serie durante una temporada. En 1944 se habían fabricado casi 3000 unidades.

Más tarde se introdujeron nuevas mejoras. La más importante fue la sustitución del motor eléctrico por uno de combustión, teniendo que llevar así un depósito de gasolina que disponía de 6 litros. Con estos cambios se incrementó un poco el tamaño del GOLIAT, así que elevaron la carga de explosivos a 70 Kg. y en otro modelo hasta los 100 Kg. La velocidad también aumento un poco.

Su mejorado sistema de tracción le permitía sortear zanjas de hasta 1 metro, y subir pendientes de 70º.

De este último modelo se llegaron a construir 5000 unidades, ya que el precio de su motor de combustión era mucho menor que el antiguo motor eléctrico. El Goliat no fue un gran éxito, debido a su gran coste y a la distancia de uso tan limitada que poseía.

A partir de estos robots, se fueron haciendo nuevas ideas de robots militares, que se fueron desarrollando paralelamente al sistema de guerra que había en cada caso.

Así llegaron las primeras aeronaves no tripuladas en la década de los 80, que fueron usadas para exploración y detección de blancos. Estas naves eran muy lentas al principio, ya que su misión de adquisición de blancos la desarrollaban una vez habían aterrizado, mientras que mientras volaban no hacían más que transportarse y explorar. Pero a partir de los años 90 se fue mejorando y se les dotó de un sistema de reconocimiento de blancos desde el aire, siendo así mucho más efectivo que el anterior.

Estos robots han sido usados recientemente, por ejemplo en la guerra de Irak, (se usaron unos 50) donde tuvieron un gran éxito. Con lo cual, este tipo de robots tiene un gran futuro, ya que controlando bien la posición de la nave, y los blancos que se quieren, se pueden preparar proyectos de aeronaves no tripuladas y armadas, para combatir sin daños humanos.

El Global Hawk, uno de los últimos aviones diseñados de esta clase, que se usó por primera vez en el 2001, tiene una autonomía de 24 horas, ya que casi todo su peso se debe al combustible que lleva. Vuela a una altura superior a los 45000 pies, y tiene una velocidad de 200 km/h.

Ingenieros de toda Europa, están unidos para diseñar conjuntamente proyectos de UCAV (Unmanned Combat Air Vehicle), que ya han visto la luz por ejemplo en aviones como el Dassault nEUROn, que hizo una demostración en Febrero del 2006.

Los robots militares más usados, son:

- Robots soldado
- Robots de reconocimiento aéreo
- Robots explorador o reconocimiento terrestre
- Robots de desminado o desactivación de explosivos
- Robots de ayuda al ingeniero
- Robots suicidas o minas móviles

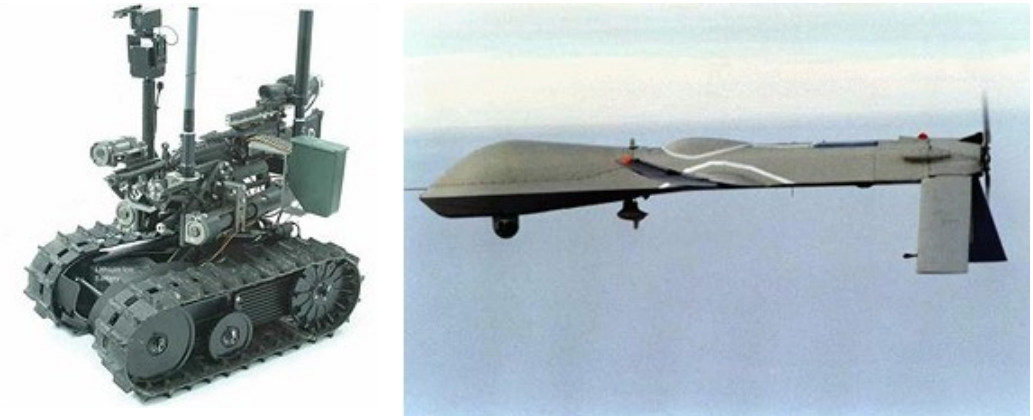


Figura 1.12: Robot Soldado TALON, equipado con un cañón de M16 (izquierda) y avión de espionaje autotripulado Predator (derecha), en pleno vuelo.

1.4. Motivación y objetivos

1.4.1 Objetivos del proyecto

El propósito del presente proyecto consiste en el desarrollo de una interfaz basada en una aplicación informática gráfica para la teleoperación y control remotos de una plataforma móvil Pioneer 2AT-8, basada en tecnologías modernas y escalables en lenguaje de alto nivel. Los servicios serán prestados al usuario a través de una interfaz gráfica que resulte a la vez cómoda para un usuario sin conocimientos específicos, que podrá acceder al estado del sistema y las lecturas de sus sistemas sensoriales, y que permita un acceso versátil y potente a la plataforma móvil para un operador experto.

Los principales objetivos que se deben cumplir son:

1. Diseñar y desarrollar una arquitectura software para el control de movimiento y posición de la plataforma robótica móvil y obtención gráfica y clara de la odometría del mismo, y que cumpla los siguientes requisitos:

    - El sistema será diseñado para utilizar tecnología **CORBA** de componentes, lo que proporcionará al sistema de la estabilidad e independencia de la capa física y de sistema operativo propias de esta tecnología.
    - Sistema desarrollado bajo los estándares de lenguaje **Java,** que nos proporcionará operar con independencia de la plataforma y sistema operativo.
    - Utilización de software libre.
    - Programación de la de la aplicación en base al paradigma cliente-servidor, en la que aplicación se conectará como cliente al servidor ya existente en la plataforma robótica para la obtención de datos y envío de comandos de control que permitan su teleoperación a distintos niveles, desde el control remoto manual de los actuadores hasta el envío de comandos de alto nivel como seguimiento de una trayectoria o misiones de navegación entre distintas localizaciones.
    - El sistema incorporará una interfaz gráfica de cómodo uso para el usuario, tanto para la visualización de los datos procedentes de la plataforma móvil (lecturas de sensores, estado de las baterías, imagen estereoscópica…) como para su teleoperación.

2. Obtención y muestra en pantalla en tiempo real en el ordenador cliente de las imágenes obtenidas por la cámara conectada al ordenador portátil servidor de imágenes situado en la repisa de la plataforma robótica móvil.

3. Dibujado y muestra por pantalla del mapa generado por la plataforma robótica móvil según su recorrido, obtenido por técnicas SLAM.

4. Dibujado y muestra de un mapa de puntos, en tiempo real que nos da el LASER de barrido que la plataforma robótica tiene en la parte delantera.

5. Control del movimiento de la plataforma robótica móvil a través de las teclas de dirección del teclado del cliente de control del robot móvil. Así podemos decidir hacia donde se va a dirigir posteriormente el robot.
6. Todas las conexiones entre el cliente de control y los dos servidores de datos (la plataforma móvil y la cámara estereoscópica), se hagan a través de una conexión vía wifi.

En resumen, se pretende que, a través de cualquier ordenador (gracias a la programación en java) y con una fiabilidad extrema (gracias a la tecnología CORBA), teleoperar remotamente (vía wifi, sin cables) el robot móvil, de tal manera que sepamos por donde se ha movido y donde está (gracias a los mapas generados por SLAM), que hay en su entorno (gracias a los datos obtenidos por el láser y por la cámara estereoscópica) y que podemos moverlo por donde se nos antoje.

1.4.2 Alcance del proyecto

Para obtener los máximos resultados y que el tiempo gastado en desarrollar el proyecto ha sido óptimo, es necesario proponerse unas metas bien definidas y tenerlas en mente en todo momento; pasito a pasito se irá viendo como se alcanzarán dichas metas. Para saber a dónde vamos, es necesario saber donde se está. El alcance del proyecto global implica las siguientes tareas:

- Estudio de las tecnologías existentes para realizar los objetivos y selección de la más adecuada.
- Selección del dispositivo de control remoto.
- Integración física de la cámara estereoscópica en la plataforma robótica.
- Desarrollo de la transmisión del vídeo.
- Desarrollo de la aplicación de control.
- Realización de pruebas a ambos sistemas.

- Desarrollo de un manual de usuario para ambas aplicaciones.

1.4.3 Objetivos personales

El principal objetivo personal es el de superar el reto que supone el aprender, comprender y poner en práctica todos los conocimientos nuevos necesarios para realizar este proyecto, ya que nunca antes había trabajado con este tipo de sistemas. Me resulta muy interesante la experiencia de programar una GUI (Graphics User Interface) por primera vez, y hacer que se comunique usando tecnologías tan avanzadas como CORBA y JAVA SWING.

También el hecho de programar a alto y bajo nivel, por las dificultades tanto de implementación como de depuración que conlleva, además de tener que aprender las peculiaridades del compilador que voy a utilizar para ello. Se espera que durante el desarrollo todos los problemas que vayan surgiendo, se solventen con eficacia y obteniendo las soluciones óptimas.

Capítulo 2

2. Estado del arte

2.1 Tecnologías de desarrollo de aplicaciones gráficas (*Graphics User Interfaces*)

En esta sección se pretende analizar las diferentes herramientas y APIs existentes para la confección de software gráfico de control. Se analizarán varias de las herramientas disponibles para realizar un estudio comparativo de las capacidades que ofrecen, este análisis se centra principalmente en los distintos métodos de interacción con el usuario ofrecidos por estas herramientas, las posibilidades de agrupación, distribución y control y las posibilidades de integración de las diferentes APIs entre otras. Las 3 tecnologías más usadas para el desarrollo de GUIs son Java Swing, Java AWT y C++ QT, a continuación se describirán estas tecnologías:

2.1.1 Java Swing

Swing es una biblioteca gráfica para el lenguaje de programación Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas. Las Internet Foundation Classes (IFC) eran una biblioteca gráfica para el lenguaje de programación Java desarrollada originalmente por Netscape y que se publicó en 1996.

Desde sus inicios el entorno Java ya contaba con una biblioteca de componentes gráficos conocida como AWT. Esta biblioteca estaba concebida como una API estandarizada que permitía utilizar los componentes nativos de cada sistema operativo. Entonces una aplicación Java corriendo en Microsoft Windows usaría el botón estándar de Windows y una aplicación corriendo en UNIX usaría el botón estándar de Motif. En la práctica esta tecnología no funcionó:

- Al depender fuertemente de los componentes nativos del sistema operativo el programador AWT estaba confinado a un mínimo denominador común entre ellos. Es decir que sólo se disponen en AWT de las funcionalidades comunes en todos los sistemas operativos.

- El comportamiento de los controles varía mucho de sistema a sistema y se vuelve muy difícil construir aplicaciones portables. Fue por esto que el eslogan de Java "Escríbalo una vez, ejecútelo en todos lados" fue parodiado como "Escríbalo una vez, pruébelo en todos lados".

En cambio, los componentes de IFC eran mostrados y controlados directamente por código Java independiente de la plataforma. De dichos componentes se dice con frecuencia que son componentes ligeros, dado que no requieren reservar recursos nativos del sistema de ventanas del sistema operativo. Además al estar enteramente desarrollado en Java aumenta su portabilidad asegurando un comportamiento idéntico en diferentes plataformas.

En 1997, Sun Microsystems y Netscape Communications Corporation anunciaron su intención de combinar IFC con otras tecnologías de las Java Foundation Classes. Además de los componentes ligeros suministrados originalmente por la IFC, Swing introdujo un mecanismo que permitía que el aspecto de cada componente de una aplicación pudiese cambiar sin introducir cambios sustanciales en el código de la aplicación. La introducción de soporte ensamblable para el aspecto permitió a Swing emular la apariencia de los componentes nativos manteniendo las ventajas de la independencia de la plataforma. También contiene un conjunto de herramientas que nos permiten crear un interfaz atractivo para los usuarios [ref 5].

Es una plataforma independiente, Model-View-Controller Gui framework para Java. Sigue un simple modelo de programación por hilos, y posee las siguientes características principales:

- Independencia de plataforma.
- Extensibilidad: es una arquitectura altamente particionada: los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. Se puede extender clases existentes proveyendo alternativas de implementación para elementos esenciales.
- Customizable: dado el modelo de representación programático del framework de swing, el control permite representar diferentes 'look and feel' (desde MacOS look and feel hasta Windows XP look and feel). Además, los usuarios pueden proveer su propia implementación look and feel, que permitirá cambios uniformes en el look and feel existente en las aplicaciones Swing sin efectuar ningún cambio al código de aplicación.

Sus ventajas e inconvenientes principales son:

- El diseño en Java puro posee menos limitaciones de plataforma.
- El desarrollo de componentes Swing es muy activo.
- Los componentes de Swing soportan gran cantidad de características.
- Capacidad de Swing para sincronizar eventos.
- Control sobre el desarrollo menor que con C++ QT y java AWT.
- Más sencillo y ameno que QT y AWT.

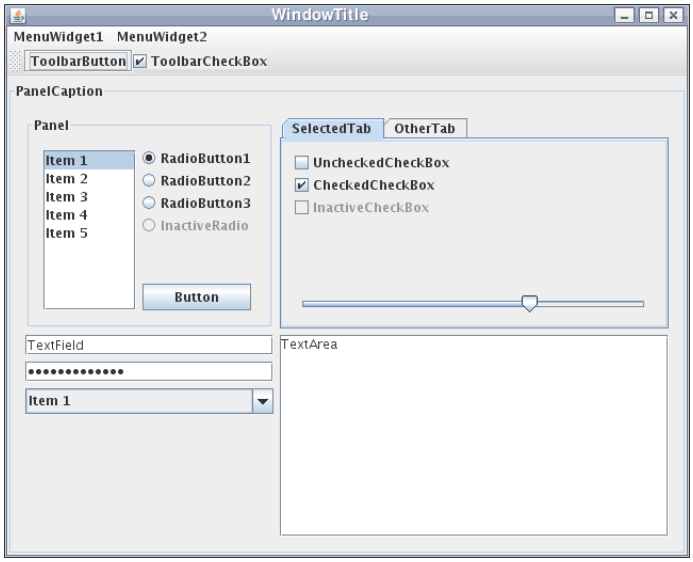


Figura 2.1: Ejemplo de widget Swing en Java 5 funcionando en Linux X Window System.

2.1.2 Java AWT

Abstract Window Toolkit (AWT, en español Kit de Herramientas de Ventana Abstracta) es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java. AWT es ahora parte de las Java Foundation Classes (JFC) - la API estándar para suministrar una interfaz gráfica de usuario (GUI) para un programa Java.

Cuando Sun Microsystems liberó Java en 1995, AWT suministró solo un nivel de abstracción muy fino sobre la interfaz de usuario nativa subyacente. Por ejemplo, crear una caja de verificación AWT causaría que AWT directamente llame a la subrutina nativa subyacente que cree una caja de verificación. Sin embargo, una caja de verificación en Microsoft Windows no es exactamente lo mismo que una caja de verificación en Mac OS o en los distintos tipos de UNIX. Algunos desarrolladores de aplicaciones prefieren este modelo porque suministra un alto grado de fidelidad al kit de herramientas nativo subyacente y mejor integración con las aplicaciones nativas. En

otras palabras, un programa GUI escrito usando AWT parece como una aplicación nativa Microsoft Windows cuando se ejecuta en Windows, pero el mismo programa parece una aplicación nativa Apple Macintosh cuando se ejecuta en un Mac, etc. Sin embargo, algunos desarrolladores de aplicaciones desprecian este modelo porque prefieren que sus aplicaciones se vean exactamente igual en todas las plataformas.

En J2SE 1.2, los widgets de AWT fueron ampliamente superados por aquellos del kit de herramientas Swing. Además de proveer un conjunto más rico de widgets UI, Swing dibuja sus propios widgets (usando Java 2D para llamar a las subrutinas de bajo nivel en el subsistema de gráficos local) en lugar de confiar en el módulo de interfaz de usuario de alto nivel del sistema operativo. Swing suministra la opción de usar un aspecto nativo o de plataforma cruzada para la aplicación.

AWT continúa suministrando el núcleo del subsistema de eventos GUI y la interfaz entre el sistema de ventanas nativo y la aplicación Java, suministrando la estructura que necesita Swing. También suministra gestores de disposición básicos, un paquete de transferencia de datos para uso con el Bloc de notas y Arrastrar y Soltar, y la interface para los dispositivos de entrada tales como el ratón y el teclado.

Sus ventajas principales son:

- El diseño en Java puro posee menos limitaciones de plataforma.
- El desarrollo de componentes AWT es muy controlable por el programador.
- Los componentes de AWT son ideales para la muestra de mapas.
- Es más difícil de programar que swing.

Figura 2.2: Elemento ejemplo de ventana usando librerías AWT.

2.1.3 C++ QT

Qt es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica como herramientas de la consola y servidores. Qt es utilizada principalmente en KDE, Google Earth, Skype, Qt Extended, Adobe Photoshop Album, VirtualBox y Opie. Es producido por la división de software Qt de Nokia, que entró en vigor después de la adquisición por parte de Nokia de la empresa noruega Trolltech, el productor original de Qt, el 17 de junio de 2008.

Qt es utilizada en KDE, un entorno de escritorio para sistemas como GNU/Linux o FreeBSD, entre otros. Qt utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de *bindings*.

Funciona en todas las principales plataformas, y tiene un amplio apoyo. El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de otros para el manejo de ficheros, además de estructuras de datos tradicionales.

Distribuida bajo los términos de GNU Lesser General Public License (y otras), Qt es software libre y de código abierto.

Qt se encuentra disponible para sistemas tipo unix con el servidor gráfico X Window System (Linux, BSDs, Unix), para Apple Mac OS X, para sistemas Microsoft Windows, para Linux empotrado (en inglés *Embedded Linux*, para sistemas integrados como PDA, Smartphone, etc.) y para dispositivos que utilizan Windows CE.

Qt Software anunció el 20 de octubre de 2008 una versión de Qt para la plataforma S60.

Adicionalmente también está disponible QSA (*Qt Scripts for Applications*), que, basándose en ECMAScript/JavaScript, permite introducir y crear scripts en las aplicaciones creadas con Qt.

Hay tres ediciones de Qt disponibles en cada una de estas plataformas, llamadas:

- **GUI Framework** – edición con nivel reducido de GUI, orientado a redes y bases de datos.
- **Full Framework** – edición completa comercial
- **Open Source** – edición completa Open Source.

Sus mayores virtudes y defectos son:

- Programación más compleja que usando Swing y Awt.
- Desarrollo de la GUI dependiente de la plataforma debido al uso de C++.
- Al ser C++, programas en un nivel más bajo que con Java.
- Su compilación y puesta en marcha más complicada que con las otras tecnologías.
- Genera interfaces que corren muy rápidas.

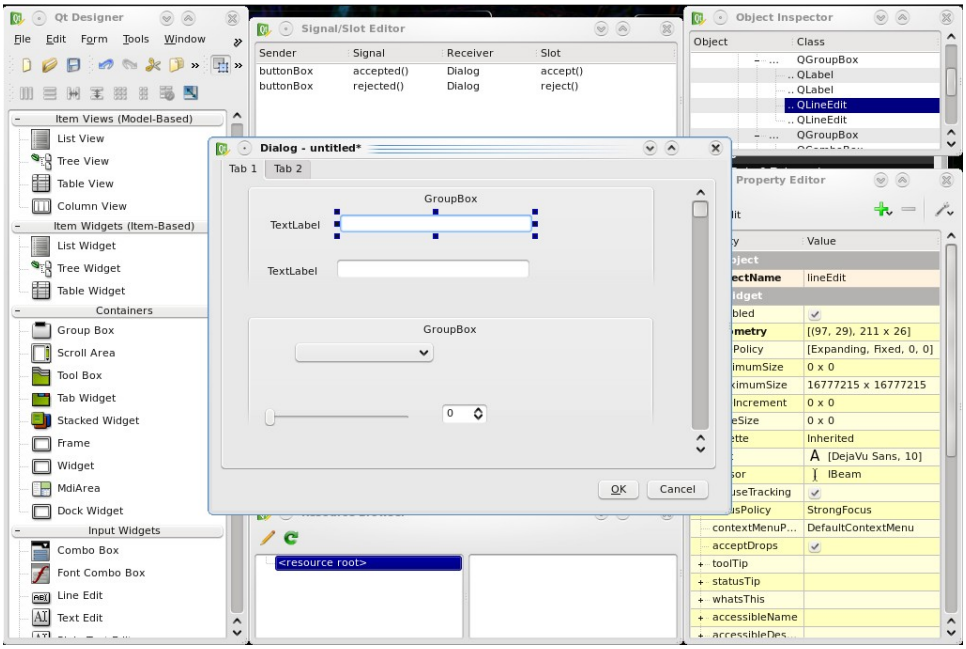


Figura 2.3: Ejemplo de interfaz gráfica desarrollada en QT, es una de las librerías gráficas más potentes

2.2 Métodos de control de robots

Existen dos tipos básicos de enfoques para controlar la navegación de un robot móvil: técnicas globales y locales. En las técnicas globales, como son los métodos geométricos y la programación dinámica, se asume totalmente conocida la descripción geométrica del entorno en el que se va a mover el robot. Se trata de métodos potentes y eficaces para generar tanto trayectorias a seguir como secuencias de comandos a ejecutar. Son métodos usados para el control de robots que trabajan en entornos sin ninguna variabilidad y que realizan tareas repetitivas en las que se conocen en todo momento el valor de todas sus variables de estado.

Los métodos locales o reactivos, por el contrario, consideran que el robot va a moverse en un entorno no conocido a priori y proporcionan unas conductas estándar para

reaccionar ante lecturas de los sensores del robot (evitar obstáculo, seguir pared, entrar en puerta, alinearse con objeto, etc.). Estas conductas son aplicables en un gran número de entornos distintos y se suelen usar junto con un control de alto nivel que se encarga de secuenciarlas.

En la propuesta que se presenta en este trabajo se formula el problema del control local de un robot móvil como un problema de reconocimiento de situaciones.

Asociado a cada uno de los esquemas de actuación se define un conjunto de acciones aplicables y se aprenden las situaciones perceptuales en las que estas deben aplicarse.

Para obtener muestras de aprendizaje de un funcionamiento correcto del esquema se optimiza una función que considera la trayectoria seguida por el robot, tanto en el espacio cartesiano como en el espacio de velocidades lineales y angulares, y que premia trayectorias consistentes con el esquema. Por ejemplo, para un esquema seguir-pared se considera que la distancia cartesiana a la pared debe ser pequeña y uniforme, premiando aquellas trayectorias con mayores velocidades lineales. Dado que el espacio de búsqueda es enorme y que la función a optimizar no es susceptible de ser diferenciada, se utiliza para su resolución la técnica de algoritmos genéticos, diseñando una codificación de las trayectorias del robot y un método de cruzamiento que han demostrado tener una alta efectividad.

A partir de las trayectorias obtenidas se generan todas las parejas de percepción y acción que el robot ha ido encontrando en la misma y se agrupan en situaciones percibidas para un mismo tipo de acción.

En el problema de la generación de trayectorias óptimas, la elección del uso de algoritmos genéticos se fundamenta en dos razones principales: En primer lugar, es una técnica adecuada para realizar búsquedas en espacios de dimensión elevada, como en este caso. Por otro lado, el método impone pocas restricciones de tipo matemático en la forma de la función a optimizar, de tal manera que es aplicable a la generación de trayectorias para cualquier tipo de comportamiento (evitar obstáculos, seguir paredes, etc.). Los algoritmos genéticos parten de una población inicial que se suele generar de manera aleatoria y que representa un conjunto de posibles soluciones al problema planteado. En este caso cada individuo de la población representara una posible trayectoria del robot en el espacio de velocidades.

En las sucesivas generaciones, los individuos de la población son medicados mediante la aplicación de una serie de operadores genéticos. El operador de cruce combina dos

individuos con el fin de obtener una solución mejor, mientras que la mutación modifica aleatoriamente algunas de las características de un individuo determinado.

Los individuos resultantes de los cruces y las mutaciones, junto con los ya existentes, pasan a competir por formar parte de la nueva generación, de manera que los individuos que representan soluciones mejores tienen más probabilidades de sobrevivir.

Es necesario disponer de una función de evaluación que permita medir la idoneidad de cada individuo (de cada trayectoria). La forma de la función de evaluación variaría según cuál sea el esquema de comportamiento deseado.

Una vez se han generado un conjunto de trayectorias correctas para el esquema que se está aprendiendo, se trata de caracterizar las situaciones en las que el robot se va a encontrar cuando evolucione siguiendo ese esquema. A partir de las trayectorias generadas por algoritmo genéticos se generan los estados que ha encontrado el robot en su evolución siguiendo dichas trayectorias. Estos estados se agrupan en conjuntos de situaciones prototipo que son aprendidos y reconocidos. Tecnologías de este estilo serían muy interesantes en Higgs. Por ejemplo, en su posible uso en vigilancia de exteriores podría ir reconociendo el terreno, sus modificaciones, objetos extraños, caminos más cortos. . .

2.3 Construcción de mapas con un robot móvil

Uno de los mayores problemas que conciernen a la navegación de robots móviles consiste en la determinación de su localización con un elevado grado de precisión. Para cumplir con el objetivo de autonomía no basta con situar el robot en un sistema de referencia global sino que es imprescindible conocer su posición relativa tanto respecto a posibles obstáculos móviles como respecto al modelo estático de su entorno. Para ello, existen diferentes alternativas utilizando mapas que, o bien se introducen previamente al robot, o bien se elaboran a medida que el mismo se mueve por un determinado lugar. Esta segunda opción empieza a desarrollarse a finales de los años 80. En un principio se desacoplaron los problemas de construcción del mapa y de la localización del robot en el mismo; pronto se descubriría que la solución rigurosa requiere tratar ambos aspectos al mismo tiempo en lo que se conoce como SLAM (Simultaneous Localization and Mapping). La solución al problema SLAM está considerada como pieza clave en la búsqueda de la completa autonomía de un robot móvil y concentra los principales esfuerzos de investigación de grupos de robótica móvil de todo el mundo que han conseguido importantes avances y continúan intentando resolver dicho problema.

El origen de las dificultades en la localización y la construcción de mapas se derivan de la existencia de ruido en las medidas de los sensores y en las limitaciones en el rango de las mismas. Un poco más en profundidad, los principales factores que impiden que el proceso sea más sencillo son los siguientes:

- Las observaciones se obtienen con respecto al sistema de referencia propio del robot, cuya posición viene afectada de un cierto grado de incertidumbre inherente a la odometría. Así, la imprecisión en las observaciones se añade a la ya existente en la posición del robot y se complica extremadamente la minimización de los errores.
- En multitud de casos es necesaria la representación de mapas de tamaño grande, lo cual supone un mayor coste computacional y una mayor imprecisión en la odometría según aumentan los desplazamientos del robot.
- Los entornos suelen ser dinámicos, sobre todo en el caso de robots guía o domésticos. Si se considera que las observaciones corresponden a puntos fijos representados con carácter permanente en el mapa se simplifica el problema, pero se tienen discrepancias con la realidad. Una aproximación parcialmente eficiente consiste en ir borrando objetos transitorios del mapa a lo largo del tiempo, tratándolos a modo de ruido. El rápido avance de la visión artificial y el desarrollo de dispositivos sensores que diferencien entre obstáculos móviles y estructuras estáticas dentro de un marco de referencia móvil permitirán notables mejoras en este aspecto.
- La asociación de las observaciones con los objetos del mapa puede resultar compleja si éstos son parecidos entre sí. En muchos casos no puede efectuarse la correspondencia de forma determinista y ha de emplearse una formulación probabilista.
- Los entornos son tridimensionales pero contemplar este aspecto introduce un mayor grado de complejidad y ha de tenerse en cuenta que generalmente los sensores están configurados para una percepción plana horizontal. Aunque existen algunos trabajos en la representación de mapas tridimensionales, la mayor parte de los algoritmos más empleados hasta el momento aceptan la hipótesis de bidimensionalidad del entorno. El paso a modelos en tres dimensiones es uno de los próximos objetivos a seguir.

2.4 Transmisión de imágenes por red TCP/IP

La transmisión de vídeo sobre redes de telecomunicaciones está llegando al punto de convertirse en un sistema habitual de comunicación debido al crecimiento masivo que ha supuesto Internet en estos últimos años. Lo estamos utilizando para ver películas o comunicarnos con conocidos, pero también se usa para dar clases remotas, para hacer diagnósticos en medicina, videoconferencia, distribución de TV, vídeo bajo demanda, para distribuir multimedia en Internet...

Debido a la necesidad de su uso que se plantea en el presente y futuro, se han proporcionado distintas soluciones y sucesivos formatos para mejorar su transmisión. Pero hoy, ya hemos oído hablar negativamente de los sistemas actuales de distribución de vídeo debido a su dudosa calidad en redes como Internet.

Estas aplicaciones normalmente demandan un elevado ancho de banda y a menudo crean cuellos de botella en las redes. Este es el gran problema al que está sometida la transmisión de vídeo. ¿Por qué es el vídeo tan problemático?

¿Qué es el vídeo?, El vídeo no es nada más que la reproducción en forma secuencial de imágenes, que al verse con una determinada velocidad y continuidad dan la sensación al ojo humano de apreciar el movimiento natural. Junto con la imagen, el otro componente es el sonido. La transmisión digital y la distribución de información audiovisual permite la comunicación multimedia sobre las redes que soportan la comunicación de datos, brindando la posibilidad de enviar imágenes en movimiento a lugares remotos. Pero no es todo tan bonito a la hora de transmitirlo por red, debido a que nos encontramos con sucesos como lentitud entre la reproducción de imágenes, errores de transmisión, o pérdidas de datos...

Existen dos formas de transmisión de datos, analógico y digital. Una de las características del vídeo es que está compuesto por señales analógicas, con lo que se pueden dar las dos formas de transmisión. En los últimos años la transmisión de datos se ha volcado hacia el mundo digital ya que supone una serie de ventajas frente a la transmisión analógica. Al verse la información reducida a un flujo de bits, se consigue una mayor protección contra posibles fallos ya que se pueden introducir mecanismos de detección de errores, se elimina el problema de las interferencias, podemos disminuir el efecto del ruido en los canales de comunicación, conseguir codificaciones más óptimas y encriptado, mezclar con otros tipos de información a través de un mismo canal, y poder manipular los datos con ordenadores para comprimirlos, por ejemplo.

Además si queremos difundir el vídeo por vías digitales tendremos que digitalizarlo, con lo que debe ser capturado en su formato analógico y almacenado digitalmente logrando así que sea menos propenso a degradarse durante la transmisión.

Existen dos tipos de redes de comunicación, de conmutación de circuitos y de conmutación de paquetes. En la conmutación de circuitos, donde la comunicación está permanentemente establecida durante toda la sesión, un determinado ancho de banda es asignado para la conexión, y el tiempo de descarga del vídeo puede predecirse, pero tienen la desventaja de que las sesiones son punto a punto y limitan la capacidad de usuarios.

En la conmutación de paquetes pueden acomodarse más fácilmente las conferencias multipunto. Aquí el ancho de banda esta compartido pero es variable, lo que supone una importante mejora puesto que, si el bit-rate (o número de bits por segundo) es fijo la calidad de la imagen variará dependiendo del contenido de los fotogramas. Debe de cumplirse que el ancho de banda, la resolución, y la compresión de audio sean idénticos para cada cliente que recibe el vídeo, lo que dificulta la configuración del sistema.

El vídeo es muy sensible al retardo de la red, ya que puede provocar cortes en las secuencias. La pérdida de alguna información en el vídeo sin comprimir no es muy relevante, ya que al perderse un fotograma, el siguiente fotograma proporciona la suficiente información para poder interpretar la secuencia. En cambio el vídeo comprimido es mucho más sensible a errores de transmisión, ya que las técnicas de compresión que se valen de la redundancia espacial y temporal pueden perder la información de esta redundancia y los efectos de la falta de datos pueden propagarse en los próximos fotogramas. Es por eso que actualmente la comunicación con vídeo vía Internet no prometen una elevada fiabilidad de transmisión.

Algunas técnicas de compresión compensan esta sensibilidad a la pérdida de datos enviando la información completa sobre un fotograma cada cierto tiempo, incluso si los datos del fotograma no han cambiado. Esta técnica también es útil para los sistemas de múltiples clientes, para que los usuarios que acaban de conectarse, reciban las imágenes completas.
Nos podemos preguntar cuál es la tecnología de red adecuada para las aplicaciones de vídeo, pero siempre dependeremos del entorno en el que trabajemos. Por ejemplo si disponemos de una alto ancho de banda el tipo de red adecuada sería ATM; para un entorno de red de área local podríamos usar Fast Ethernet, y actualmente para que el usuario de Internet, ADSL.

Figura 2.4: Gracias al aumento de las conexiones de banda ancha, la transmisión de imágenes a través de internet es cada día más común.

2.5 Middleware

En este punto, se explicará que es un middleware, el porqué queremos usar uno, cuales son los principales middlewares que existen y cuál es la razón por la que se ha decantado para usar el middleware CORBA.

Este punto se hacía realmente importante a raíz de que el uso y entendimiento del middleware, posiblemente, fue el punto más complejo y difícil de todas los conocimientos y métodos explicados en este documento. Además de que la implementación en los sistemas puede llegar a ser laboriosa y compleja, sin embargo, el uso de esta tecnología supone un gran potencia, versatilidad y portabilidad de todos los sistemas que hagan uso de ella.

2.5.1 ¿Qué es un middleware?

El **middleware** es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El middleware nos abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipo de servicios de middleware.

Por lo general el middleware del lado cliente está implementado por el Sistema Operativo subyacente, el cual posee las librerías que implementan todas las funcionalidades para la comunicación a través de la red.

El **middleware** es un módulo intermedio que actúa como conductor entre sistemas permitiendo a cualquier usuario de sistemas de información comunicarse con varias fuentes de información que se encuentran conectadas por una red.

Desde un punto de vista amplio una solución basada en productos middleware debe permitir conectar entre sí a una variedad de productos procedentes de diferentes proveedores. De esta forma se puede separar la estrategia de sistemas de información de soluciones propietarias de un sólo proveedor.

El concepto de middleware no es un concepto nuevo. Los primeros monitores de teleproceso de los grandes sistemas basados en tecnología cliente servidor ya se basaban en él, pero es con el nacimiento de la tecnología basada en sistemas abiertos que el concepto de middleware toma su máxima importancia.

Las categorías del middleware podríamos definirlas de la siguiente forma:

- Monitores de proceso de transacciones distribuidos (DTPM's Distributed Transaction Processing Monitors) . Herederos de la tecnología mainframe, son ampliamente demandados para intercomunicar distintos sistemas en distintos entornos.
- Llamadas a procedimientos remotos (RPC's Remote procedure Call) Diseñado como servicios síncronos para permitir gestión remota de redes.
- Middleware orientado a mensajes (MOM Messaging Oriented Middleware) Diseñado para servicios de mensajes con tecnología asíncrona.
- (ORB Objects Request Broker) Middleware para tecnologías orientadas a objetos. Objetos piden servicios de objetos que se encuentran en la red. El estándar más conocido de esta tecnología es CORBA Common Object Request Broker Arquitecture.
- Middleware de acceso a Bases de Datos (Data Base Access Middleware). Para acceso estándar a bases de datos. Permite desarrollar sistemas independizándolo de la base de datos que lo soporte. En la actualidad representa el 50% del mercado del middleware.

2.5.2 Los middlewares más usados

RMI

RMI (*Java Remote Method Invocation*) es un mecanismo ofrecido por Java para invocar un método de manera remota. Forma parte del entorno estándar de ejecución de Java y provee de un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java [ref 6]. Si se requiere comunicación entre otras tecnologías debe utilizarse CORBA o SOAP en lugar de RMI.

RMI se caracteriza por la facilidad de su uso en la programación por estar específicamente diseñado para Java; proporciona paso de objetos por referencia (no permitido por SOAP), recolección de basura distribuida (Garbage Collector distribuido) y paso de tipos arbitrarios (funcionalidad no provista por CORBA).

Por medio de RMI, un programa Java puede exportar un objeto, lo que significa que éste queda accesible a través de la red y el programa permanece a la espera de peticiones en un puerto TCP. A partir de este momento, un cliente puede conectarse e invocar los métodos proporcionados por el objeto.

La invocación se compone de los siguientes pasos:

- Encapsulado (marshalling) de los parámetros (utilizando la funcionalidad de serialización de Java).
- Invocación del método (del cliente sobre el servidor). El invocador se queda esperando una respuesta.
- Al terminar la ejecución, el servidor serializa el valor de retorno (si lo hay) y lo envía al cliente.
- El código cliente recibe la respuesta y continúa como si la invocación hubiera sido local.

Desde la versión 1.1 de JDK, Java tiene su propio ORB: RMI (Remote Method Invocation). A pesar de que RMI es un ORB en el sentido general, no es un modelo compatible con CORBA. RMI es nativo de Java, es decir, es una extensión al núcleo del lenguaje. RMI depende totalmente del núcleo de la Serialización de Objetos de Java, así como de la implementación tanto de la portabilidad como de los mecanismos de carga y descarga de objetos en otros sistemas...

El uso de RMI resulta muy natural para todo aquel programador de Java ya que éste no tiene que aprender una nueva tecnología completamente distinta de aquella con la cual desarrollará. Sin embargo, RMI tiene algunas limitaciones debido a su estrecha integración con Java, la principal de ellas es que esta tecnología no permite la interacción con aplicaciones escritas en otro lenguaje.

RMI como extensión de Java, es una tecnología de programación, fue diseñada para resolver problemas escribiendo y organizando código ejecutable. Así RMI constituye un punto específico en el espacio de las tecnologías de programación junto con C, C++, Smalltalk…

La arquitectura RMI puede verse como un modelo de cuatro capas:

Primera capa: La primera capa es la de aplicación y se corresponde con la implementación real de las aplicaciones cliente y servidor. Aquí tienen lugar las llamadas a alto nivel para acceder y exportar objetos remotos. Cualquier aplicación que quiera que sus métodos estén disponibles para su acceso por clientes remotos debe declarar dichos métodos en una **interfaz** que extienda java.rmi.Remote. Dicha interfaz se usa básicamente para "marcar" un objeto como remotamente accesible. Una vez que los métodos han sido implementados, el objeto debe ser exportado. Esto puede hacerse de forma implícita si el objeto extiende la clase UnicastRemoteObject (paquete java.rmi.server), o puede hacerse de forma explícita con una llamada al método exportObject() del mismo paquete.

Segunda capa: La capa 2 es la capa proxy, o capa stub-skeleton. Esta capa es la que interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones junto con sus parámetros y retorno de objetos tienen lugar en esta capa.

Tercera capa: La capa 3 es la de referencia remota, y es responsable del manejo de la parte semántica de las invocaciones remotas. También es responsable de la gestión de la replicación de objetos y realización de tareas específicas de la implementación con los objetos remotos, como el establecimiento de las persistencias semánticas y estrategias adecuadas para la recuperación de conexiones perdidas. En esta capa se espera una conexión de tipo stream (stream-oriented connection) desde la capa de transporte.

Cuarta Capa: La capa 4 es la de transporte. Es la responsable de realizar las conexiones necesarias y manejo del transporte de los datos de una máquina a otra. El protocolo de transporte subyacente para RMI es JRMP (Java Remote Method Protocol), que solamente es "comprendido" por programas Java.

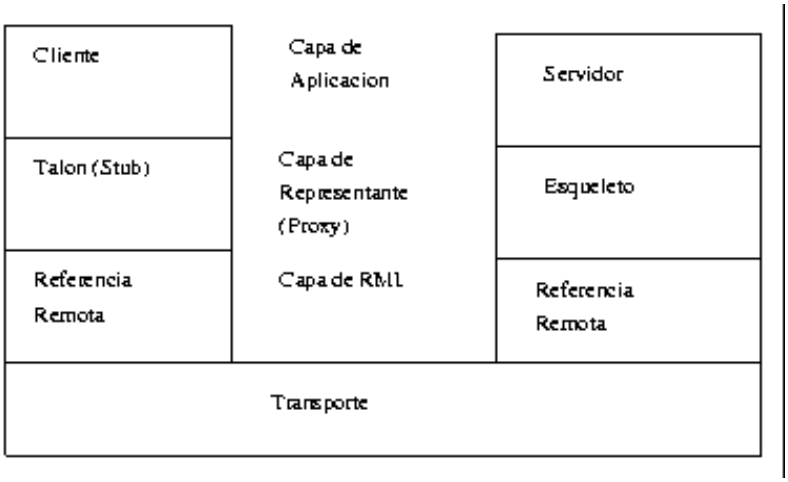


Figura 2.5 Capas RMI

DCOM

Distributed Component Object Model (DCOM), en español Modelo de Objetos de Componentes Distribuidos, es una tecnología propietaria de Microsoft para desarrollar componentes software distribuidos sobre varios ordenadores y que se comunican entre sí. Extiende el modelo COM de Microsoft y proporciona el sustrato de comunicación entre la infraestructura del servidor de aplicaciones COM+ de Microsoft. Ha sido abandonada en favor del framework .NET

La adición de la "D" a COM fue debido al uso extensivo de DCE/RPC, o más específicamente la versión mejorada de Microsoft, conocida como MSRPC.

En términos de las extensiones que añade a COM, DCOM tenía que resolver los problemas de:

- Aplanamiento - Serializar y deserializar los argumentos y valores de retorno de las llamadas a los métodos "sobre el cable".
- Recolección de basura distribuida, asegurándose que las referencias mantenidas por clientes de las interfaces sean liberadas cuando, por ejemplo, el proceso cliente ha caído o la conexión de red se pierde.

Uno de los factores clave para resolver estos problemas es el uso de DCE/RPC como el mecanismo RPC subyacente bajo DCOM. DCE/RPC define reglas estrictas en cuanto al aplanamiento y a quién es responsable de liberar la memoria.

DCOM fue uno de los mayores competidores de CORBA. Los defensores de ambas tecnologías sostenían que algún día serían el modelo de código y servicios sobre

Internet. Sin embargo, las dificultades que suponía conseguir que estas tecnologías funcionasen a través de cortafuegos y sobre máquinas inseguras o desconocidas, significó que las peticiones HTTP normales, combinadas con los navegadores web les ganasen la partida. Microsoft, en su momento intentó y fracasó anticiparse a esto añadiendo un transporte extra HTTP a DCE/RPC denominado "ncacn_http" (Connection-based, over HTTP).

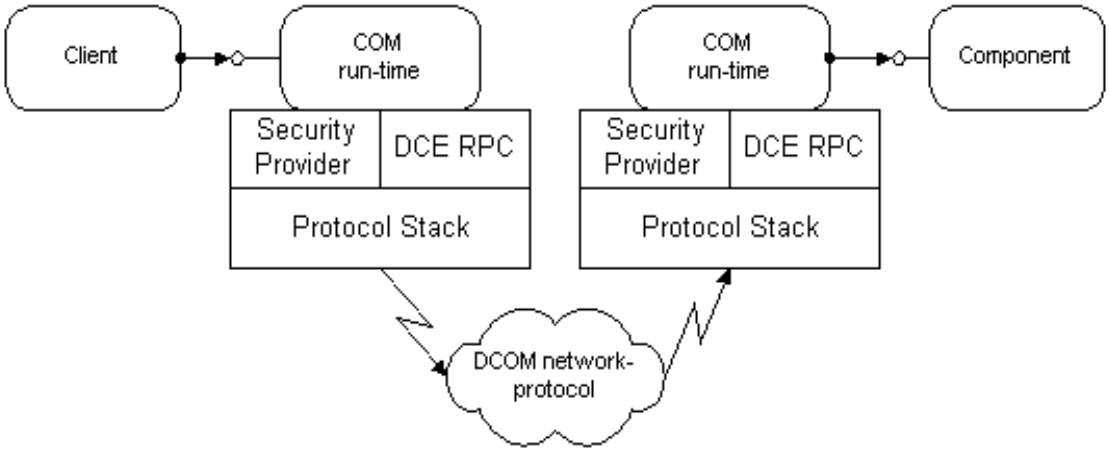


Figura 2.6: Arquitectura DCOM

CORBA

El middleware CORBA fue el elegido para el desarrollo de este proyecto, con lo cual no se hablará mucho sobre él, ya que hay un apartado mucho más amplio sobre él. Aquí puntualizaremos son características más importantes para poder comparar con los middlewares antes mencionados.

En computación, **CORBA** (*Common Object Request Broker Architecture* — arquitectura común de intermediarios en peticiones a objetos), es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

CORBA fue definido y está controlado por el *Object Management Group* (OMG) que define las APIs, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida.

En un sentido general, CORBA "envuelve" el código escrito en otro lenguaje, en un paquete que contiene información adicional sobre las capacidades del código que

contiene y sobre cómo llamar a sus métodos. Los objetos que resultan, pueden entonces ser invocados desde otro programa (u objeto CORBA) desde la red. En este sentido CORBA se puede considerar como un formato de documentación legible por la máquina, similar a un archivo de cabeceras, pero con más información.

CORBA utiliza un lenguaje de definición de interfaces (IDL) para especificar las interfaces con los servicios que los objetos ofrecerán. CORBA puede especificar a partir de este IDL, la interfaz a un lenguaje determinado, describiendo cómo los tipos de dato CORBA deben ser utilizados en las implementaciones del cliente y del servidor. Implementaciones estándar existen para Ada, C, C++, Smalltalk, Java, Python, Perl y Tcl.

Al compilar una interfaz en IDL se genera código para el cliente y el servidor (el implementador del objeto). El código del cliente sirve para poder realizar las llamadas a métodos remotos. Es el conocido como *stub*, el cual incluye un *proxy* (representante) del objeto remoto en el lado del cliente. El código generado para el servidor consiste en unos *skeletons* (esqueletos) que el desarrollador tiene que rellenar para implementar los métodos del objeto [Ref 7].

2.5.3 Razones para la elección de CORBA como middleware

Existen 2 razones de mucho peso por las cuales no se optó por DCOM como middleware para este proyecto:

- DCOM, es software privativo, y va en contra de las especificaciones del proyecto.
- Es todavía una tecnología poco desarrollada y que tiene mucho problemas en su implementación.

En el caso de RMI existía una limitación que la hacía a todas luces inviable, que es, que sólo se puede implementar en lenguaje JAVA, lo cual no se podía usar dentro de la plataforma robótica móvil, donde todo está programado en C++.

Las razones las por las cuales se optó por CORBA para ser el middleware de este proyecto fueron las siguientes:

*a) Disponibilidad y Versatilidad*Muchas arquitecturas y sistemas operativos cuentan con una implementación de CORBA, lo que hace suponer que se

puede usar CORBA en virtualmente cualquier proyecto de sistemas distribuidos.

b) Eficiencia La libertad de desarrollo ha favorecido la existencia de una pléyade de implementaciones del estándar que se adaptan a multitud de posibles necesidades de los usuarios, generando una competencia que favorece aquellas implementaciones de mayor calidad y con más características.

c) Adaptación a Lenguajes de programación Además, es posible emplear los servicios de CORBA desde cualquier lenguaje de programación, desde C++, C ó Java, hasta COBOL ó Ada.

d) Es software libre lo cual entra dentro de la filosofía de los requisitos del proyecto.

De todas formas, CORBA posee ciertas dificultades:

1) *Complejidad* Permitir la interoperabilidad de distintos lenguajes, arquitecturas y sistemas operativos hace que sea un estándar bastante complejo, y su uso no sea tan transparente al programador como sería deseable:
   a) Hay que usar un compilador que traduce una serie de tipos de datos estándares a los tipos del lenguaje en el que se vaya a programar (IDL)
   b) Hay que ser conscientes a la hora de diseñar qué objetos van a ser remotos y cuáles no (los remotos sufren restricciones en cuanto a sus capacidades con respecto a un objeto normal)
   c) Es necesario emplear tipos de datos que no son los que proporciona de manera habitual el lenguaje de programación (muchas veces hayque emplear tipos de datos adaptados de IDL).

2) *Incompatibilidad entre implementaciones* Muchas empresas ofrecen implementaciones CORBA, si bien el grado de cumplimiento es diverso. Las divergencias entre ORBs radican en detalles que, aunque no hacen imposible aplicar en uno el mismo diseño de un programa pensado para otro, hacen que

la adaptación sea fastidiosa. Cuestiones como la colocación de librerías o las diferentes formas de implementar la gestión de la concurrencia, hacen difícil la portabilidad del código y obligan al programador a reciclarse cuando quiere cambiar de ORB. Además, donde el estándar no concreta, las implementaciones pueden variar entre sí, lo que da lugar a molestas incompatibilidades que complican la vida al usuario.

2.6 CORBA

En informática, CORBA (Common Object Request Broker Architecture | arquitectura común de intermediarios en peticiones a objetos) es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un lenguaje orientado a objetos. CORBA fue definido y está controlado por el Object Management Group (OMG) que define las APIs, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida.

En un sentido general, CORBA “envuelve" el código escrito en otro lenguaje en un paquete que contiene información adicional sobre las capacidades del código que contiene, y sobre cómo llamar a sus métodos. Los objetos que resultan pueden entonces ser invocados desde otro programa (u objeto CORBA) de la red. En este sentido CORBA se puede considerar como un formato de documentación legible por la máquina, similar a un archivo de cabeceras pero con más información.

CORBA utiliza un lenguaje de definición de interfaces (IDL) para especificar las interfaces con los servicios que los objetos ofrecerán. Puede especificar a partir de este IDL la interfaz a un lenguaje determinado, describiendo cómo deben ser utilizados los tipos de dato CORBA en las implementaciones del cliente y del servidor. Implementaciones estándar existen para Ada, C, C++, Smalltalk, Java y Python. Hay también implementaciones para Perl y TCL.

IDL (*Interface Definition Language*) es un lenguaje de programación pensado exclusivamente para especificar los interfaces de las clases cuyas instancias queremos hacer públicas a objetos remotos que las usaran como clientes.

La necesidad de un IDL viene dada por la *independencia de CORBA respecto a la arquitectura y al lenguaje de programación*. Distintos lenguajes soportan diferentes tipos de datos y tienen distintas formas de especificar clases. Incluso limitándonos a un

lenguaje, la ordenación5 y el tamaño de un tipo de datos determinado no tiene porqué ser el mismo entre arquitecturas diferentes (por ejemplo, no es lo mismo un entero en un 386 con MS-DOS que en un UltraSparc con Solaris 7) [ref 9].

IDL pone de acuerdo a distintos lenguajes en el formato y tamaño de sus especificaciones. El compilador de IDL transforma una especificación neutral para la plataforma y el lenguaje en otra que puedan entender dicho lenguaje y plataforma.

Al compilar una interfaz en IDL se genera código para el cliente y el servidor (el implementador del objeto). El código del cliente sirve para poder realizar las llamadas a métodos remotos. Es el conocido como stub, el cual incluye un proxy (representante) del objeto remoto en el lado del cliente. El código generado para el servidor consiste en unos skeletons (esqueletos) que el desarrollador tiene que rellenar para implementar los métodos del objeto.

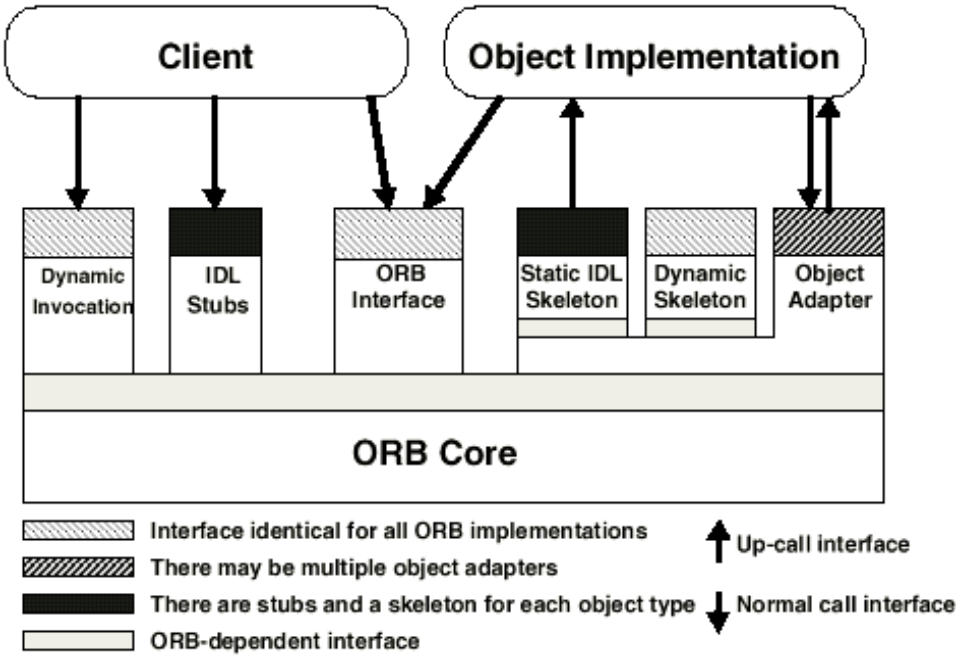


Figura 2.7 Esquema implementación CORBA

CORBA es más que una especificación multiplataforma, también define servicios habitualmente necesarios como seguridad y transacciones, es un middleware.

Resumiendo, se puede decir que CORBA es un middleware que permite la comunicación entre varias aplicaciones de forma transparente. Esto significa que no incluye el lenguaje en el que estén escritas dichas aplicaciones, ya que los servicios entre ellas serán comunes.

Los ORBs, núcleo decualquier implementación CORBA, transmiten losmensajes que se intercambian cliente y servidor, para lo que se ocupan de:

1. Canalizar las comunicaciones entre los objetos locales y los remotos.
2. Empaquetar los parámetros que el cliente pasa al método remoto y el resultado que el método devuelve al cliente.
3. Localizar al objeto remoto a partir de una referencia.

Dentro de CORBA, existen varias especificaciones de ORB que se podrían implementar dentro del proyecto presente. Estas tecnologías se describen a continuación

2.6.1 RMI-IIOP

leído RMI sobre IIOP, denota la interfaz RMI de Java sobre el sistema CORBA.Este estándar fue creado intentando simplificar el desarrollo de las aplicaciones CORBA, mientras preservaba todos los beneficios principales. RMI-IIOP está ampliamente basado en el concepto Objeto por Valor (descrito en la página CORBA) que sirve como un contenedor o un reemplazo directo para estructuras CORBA, uniones, secuencias, arrays y strings. La IDL no se usa. En cambio, las definiciones de la estructura de datos "supuestas" automáticamente, recolectan los datos necesarios a través de mecanismos de reflexión. Cuando CORBA normalmente necesita una clase generada suplementaria para cada estructura de datos no trivial que está siendo transferida, RMI-IIOP solo usa el código generado para los objetos remotos. Menos código generado resulta en menos huella.

Ambas CORBA y RMI-IIOP usan el mismo estándar de comunicación GIOP. Si se necesitara, es posible generar las definiciones de IDL para las estructuras de datos RMI-IIOP involucradas y usar esas definiciones para alcanzar la interoperabilidad entre las aplicaciones RMI-IIOP y CORBA planas.

Las versiones recientes de RMI-IIOP derivan sus *servants* desde la clase estándar Servant (CORBA). Por lo tanto es posible conectarlos al CORBA ORB manualmente, involucrando, si es necesario, el Adaptador de Objeto Portable, Interceptores Portables, servicio de nombrado CORBA y todas las demás características estándar CORBA.

Este ORB solo puede usarse cuando todo software, tanto el del/los servidor/es, como el del/los cliente/s, esté programado en Java, con lo cual no nos sirve para el propósito del proyecto.

2.6.2 MICO

MICO (Mico Is COrba, mico es corba en español) es un implementación estándar de CORBA en licencia GNU y por lo tanto de código abierto. Gracias a su sencillez y versatilidad, es uno de los ORB más usados.

En principio GNOME, el entorno de escritorio de Linux más usado, usó una implementación de CORBA denominada MICO, esta implementación también era usada por el proyecto KDE, no obstante, pronto se vieron sus problemas, MICO era muy lento y pesado, así pues, no era una solución viable para un escritorio, se precisaba de una implementación rápida y ligera. La solución fue escribir ORBit, que hoy por hoy es la implementación más rápida de CORBA existente. Lamentablemente el equipo del KDE tomó la decisión de abandonar CORBA por completo e implementar un nuevo sistema de comunicación mucho más limitado que CORBA.

2.6.3 TAO

TAO es una implementación de libre distribución de RT-CORBA que lleva varios años de evolución y desarrollo. Las aplicaciones se programan en C++ y la versión que hemos utilizado es la de Linux con TCP/IP. Se ofrece como una implementación de la especificación completa.

TAO parte de ACE, un framework orientado a objetos que implementa muchos patrones del núcleo para comunicación concurrente entre software.

Se aplicaron estos patrones y componentes del framework de ACE para desarrollar ACE ORB (TAO), que es el middleware que permite a clientes invocar operaciones en objetos distribuidos sin preocuparse de la localización del objeto, el lenguaje de programación, el sistema operativo, los protocolos de comunicación y el hardware. TAO se ha desarrollado para dar altas prestaciones y alta calidad de servicio en aplicaciones distribuidas de tiempo real.

2.6.4 GLADE

GLADE es la implementación original de GNAT del DSA de Ada para soportar el desarrollo de aplicaciones distribuidas con requisitos de tiempo real. Se sigue manteniendo hoy en la actualidad aunque evolucionará hacia PolyORB cuando esté completamente operativo. La planificación se realiza por prioridades fijas e implementa dos políticas para la distribución de prioridades al estilo de RT-CORBA (*Client Propagated* y *Server Declared*). De nuevo la plataforma de ejecución es Linux y TCP/IP.

2.6.5 Java IDL

Java IDL es una tecnología para objetos distribuidos -- es decir, objetos interactuando sobre diferentes plataformas a través de una red. Java IDL es similar a RMI (Remote Method Invocation), que soporta objetos distribuidos escritos enteramente en Java. Sin embargo, Java IDL permite intractuar a los objetos sin importar si están escritos en Java o en cualquier otro lenguaje como C, C++, COBOL, etc.

Esto es posible porque Java IDL está basado en "Common Object Request Brokerage Architecture" (CORBA), un modelo industrial estandard para objetos distribuidos. La característica principal de CORBA es IDL, (Interface Definition Language). Caa lenguaje que soporte CORBA tiene su propio mapeo IDL -- y como su nombre implica, Java IDL soporta el mapeo para Java. CORBA y los mapeos IDL son el trabajo de un consorcio industrial conocido como OMG (Object Management Group). Sun es un miembro fundacional del OMG, y el equipo Jva IDL ha jugao un papel activo en la definición del mapeo IDL-a-Java.

Para soportar la interacción entre objetos de programas separados, Java IDL proporciona un Object Request Broker, o ORB. El ORB es una librería de clases que permite una comunicación de bajo nivel entre aplicaciones Java IDL y aplicaciones compatibles con CORBA. Esto está empezando a parecer una sopa de letras, no te preocipes, los detalles sobre CORBA, IDL y ORB vienen en la próxima página.

Esta sección camina a través del diseño y desarrollo de una sencilla pareja de aplicaciones Java IDL que interactuan. Empiza mostrando la arquitectura general de CORBA, luego continúa con una introducción a los pasos para construir aplicaciones CORBA en Java IDL. Finalmente, realiza cada paso para producir un cliente y un servidor que interactuan usando CORBA.

2.6.6 Eleccion de los ORBs

CORBA es una tecnología con muchas implementaciones válidas para este proyecto, pero las 2 implementaciones que mejor se adaptaban a nuestro trabajo, fueron TAO (para C++) y Java ORB (para Java):

- La elección de Java ORB, para la parte desarrollada en Java, fue sencilla debido a que otras tecnologías de estándar de sistemas distribuidos, como RMI, no nos servían, ya que la parte del servidor iba a estar programada en C++, y necesitábamos una tecnología compatible con Java y C++.
- La elección de TAO fue más de diseño, ya que TAO permite la programación en tiempo real, mientras que otras como MICO no lo tienen. Además ere el ORB más rápido que se han probado en el laboratorio de ASLab.

Las motivaciones surgidas en el desarrollo de ACE+TAO y Java ORB son las siguientes:

1. Determinar empíricamente las capacidades necesarias para habilitar el ORB CORBA y Java ORB de tiempo real, para soportar las misiones críticas de sistemas SER (distribuidos, de tiempo real y embebidos) con requisitos de calidad software y hardware.
2. Combinar las estrategias para arquitecturas de subsistemas de entrada/salida de tiempo real y optimizarlas con ORBs para proveer sistemas finales integrados verticalmente que soporten paso de información de extremo a extremo, latencias y requisitos de calidad.
3. Capturar y documentar las bases del diseño y los principios de optimización necesarios para desarrollar ORBs basados en estándares, portables y con ampliaciones en calidad habilitadas.
4. Proveer un middleware CORBA de código abierto, de alta calidad y gratuito que pueda ser utilizado de manera efectiva por investigadores y desarrolladores.
5. Guiar varias estandarizaciones relacionadas con CORBA uniendo esfuerzos con el OMG.

En Higgs está instalado y compilado TAO. Como objetivo de un proyecto anterior se desarrolló un servidor en tiempo real que provee de las funcionalidades básicas. Es con esta aplicación con la que interacciona el control remoto de Higgs, aprovechando las funciones que ofrece.

II DESARROLLO

Capítulo 3

3. ARQUITECTURA DEL SISTEMA

En este capítulo describiremos cada una de las partes físicas que representan el sistema, y de las herramientas del laboratorio ASLab que usamos para cumplir los requisitos y objetivos del sistema de la plataforma robótica móvil.

3.1 Visión global del sistema.

El sistema, que iremos viendo poco a poco, consiste en la plataforma robótica móvil (de nombre (HIGGS), que es el elemento principal. Conectado en el interior de Higg, existe una tarjeta de conexión inalámbrica que nos permitirá conectarnos a una red wifi. Sobre ella van montados 3 partes del sistema: La cámara firewire, el ordenador ultraportatil Sony Vaio y Las baterías de alimentación. Las baterías de alimentación dan energía de funcionamiento a la cámara firewire y al propio robot Higgs, el ordenador Sony Vaio lleva su propia batería y no hace falta conectarlo a dichas baterías. La cámara Firewire está conectada al ordenador portátil Sony Vaio, que será la que reciba las imágenes digitales y las reenvie a través de la red hacia el PC que tenga el software de control, visión de imágenes y dibujado del mapa. La cámara Firewire, está apoyada en una muñeca móvil que sirve para orientar ésta en la dirección deseada.

Existe un PC, conectado a la red inalámbrica, que puede ejecutar un software que sirve para el control, muestra el mapa de posicionamiento del robot y muestra de imágenes que llegan de la cámara (con el PC Sony Vaio como intermediario). A este PC, que gracias a las especificaciones del proyecto, puede ser cualquier ordenador que tenga instalada una máquina virtual de Java, que actualmente está soportada por casi todos los sistemas operativos que existen. Así haremos el sistema todo los portable que podamos, y gracias al middelware CORBA podemos interactuar entre un programa escrito en C++, que mucha veces es dependiente de la plataforma, y Java, que es independiente de la plataforma elegida.

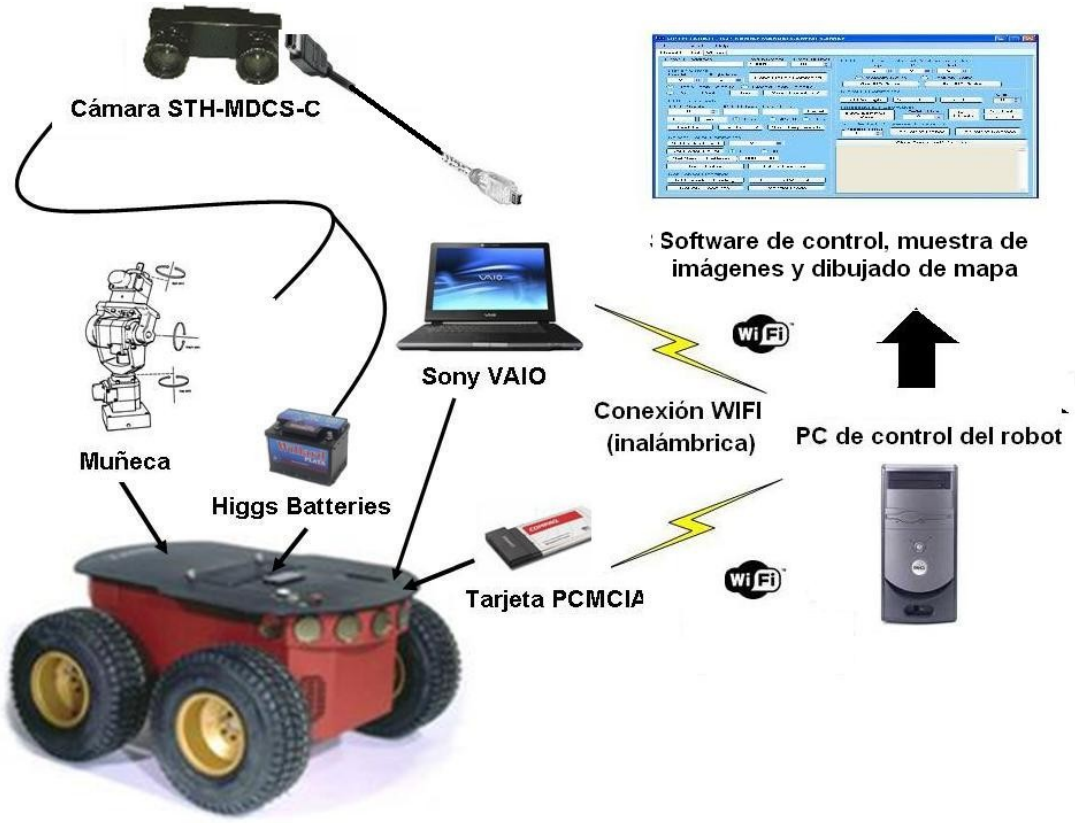


Figura 3.1: Esquema global del sistema completo.

3.2 Plataforma robótica móvil (Higgs)

Este es el nombre con el que se conoce en el laboratorio a la plataforma móvil de pruebas, un robot Pioneer 2AT-8. Como curiosidad se recomienda buscar la definición del Bosón de Higgs, que es de donde proviene el nombre del robot.

En el laboratorio de trabajo se han realizado múltiples actividades sobre esta plataforma. La que más interesa de todas ellas, debido a que se necesitará interaccionar con ella, es Versión RT-CORBA del servidor Pioneer 2AT-8 [Ref 11]. También se recomienda leer el manual de Higgs (Ref 12).

Higgs es un robot móvil Pioneer modelo 2AT-8 diseñado por ActivMedia Robotics y adquirido por nuestro grupo de trabajo (ASLab) para utilizarse como plataforma de pruebas. El objetivo de esta adquisición es incorporar al laboratorio una plataforma móvil e implementar sistemas de control inteligentes en la misma. Con la incorporación del robot al laboratorio ASLab se dispone de una nueva plataforma con la que comenzar a hacer pruebas de control inteligente y por tanto avanzar hacia la consecución de su

objetivo a largo plazo: la creación de sistemas de control capaces de aprender a controlar cualquier sistema y de tener conciencia de sí mismos.

Pioneer 2-AT8 es una plataforma robusta que incorpora todos los elementos necesarios para la implementación de un sistema de navegación y control en el robot para un gran número de entornos del mundo real. A continuación se verán algunas de sus características, las más relevantes para este proyecto. Para conocer las características completas, se recomienda acudir al manual de Higgs.

Características físicas

El tamaño del robot es pequeño en comparación con sus prestaciones. Pesa 14 kg con una batería incluida. Su estructura es de aluminio. Estas características le permiten transportar hasta 30 kg sobre el mismo. El microcontrolador que gobierna los diversos dispositivos electrónicos conectados es un Hitachi H8S 2.

Cuerpo del robot: Es de aluminio y aloja las baterías, los motores, los circuitos electrónicos y el resto de componentes. Además existe espacio para alojar diversos accesorios, como un PC de a bordo, un módem radio o ethernet radio o para incorporar sensores. La parte delantera del robot se puede desmontar fácilmente (sólo es necesario quitar tres tornillos y separar la pieza delantera del cuerpo del robot). Es donde se aloja el PC de a bordo. También es el sitio adecuado para situar otros accesorios propios por la facilidad de acceso a esa zona del robot. Tiene una mini-puerta en la parte trasera para acceder de forma sencilla a las tres baterías de las que está dotado. En la parte interior de la misma se encuentra el número de serie del robot. Existe un accesorio (una ventosa) para la cómoda extracción de las baterías del cuerpo del robot.

Hardware para la conexión inalámbrica

Una vez instalado el computador de a bordo, es necesario añadir los elementos de hardware necesarios para habilitar la comunicación inalámbrica con la red local. El laboratorio ASLab cuenta con una red inalámbrica compuesta por varios equipos portátiles que está basada en el estándar 802.11b. Los elementos que se utilizan para la comunicación entre el robot y la red de ASLab son una tarjeta PCMCIA y un punto de acceso que permite comunicar la red inalámbrica con la red cableada del laboratorio . Debido a que la tarjeta va insertada en la placa que va dentro del cuerpo del robot, y al ser _este de aluminio y estar cerrado en condiciones de trabajo, es necesario utilizar una antena amplificadora externa para permitir la comunicación inalámbrica. Esta antena fue adquirida junto con la tarjeta wireless y el Punto de Acceso. La tarjeta viene

preparada para conectarla a la antena. Una vez realizada la conexión, se colocó la antena en la plataforma exterior del robot.

Figura 3.2: Sistema Wireless de Higgs: La tarjeta inalámbrica con conexión PCMCIA (izquierda), Router inalámbrico (centro) y antena amplificadora de señal wireless (derecha).

3.3 Baterías

El robot está alimentado con tres baterías de 12 voltios y 7 amperios-hora cada una. Son intercambiables entre sí y accesibles a través de una mini-puerta en la parte trasera del robot. Por lo tanto disponemos en total de 252 watios-hora, lo que asegura varias horas de autonomía para la plataforma.

Para ver el estado de las baterías del robot nos fijamos en el color del LED BATTERY del panel de control (verde si >12.5 voltios y rojo si <11.5 voltios). Puede configurarse que el robot emita un pitido cuando las baterías estén bajas y necesiten recargarse. En cualquier caso cuando el voltaje cae por debajo de 11 voltios el watchdog del microcontrolador desconecta a cualquier cliente conectado y le envía una señal de apagado para evitar la pérdida de datos o que el sistema se corrompa debido a que las baterías están bajas.

Ha sido necesario añadir nuevas baterías de tal manera que se pudiese proporcionar la potencia necesaria a la tensión adecuada para todos los equipos añadidos (cámara, muñeca, PC, láser).

3.4 Cámara Estéreo: STH-MDCS-C

La cámara estéreo con la que se trabaja es fabricada por Videre Design. Es una cámara compacta, de bajo consumo (< 1W) con una interfaz digital basada en el IEEE 1394. Consiste en dos captadores de imagen CMOS progresivos, de 1.3 megapíxeles cada uno. Están montados en un cuerpo rígido, el cual tiene integrado el módulo 1394 (firewire).

Figura 3.3: Cámara estereo

El software Small Vision System de SRI tiene interfaz con esta cámara, y viene incluído con ella. Se ha de calibrar previamente, realizar la correlación estéreo y ver los resultados en 3D.

Las dos lentes están separadas entre ellas 9 cm. La montura de las lentes forman parte integral de la estructura y dos monturas a color son atornilladas a ella. Existe también un filtro IR. Un LED de estado indica la actividad de la cámara. Se encenderá cuando el dispositivo esté encendido y conectado a una tarjeta IEEE 1394. El LED comenzará a parpadear cuando las imágenes estén siendo adquiridas por el computador servidor. El parpadeo del LED será la mitad de la tasa de adquisición.

En el módulo de captación estéreo, los dos CMOS, cada uno de 1280 x 1024 píxeles, digitalizan la luz incidente en un stream digital. Estos captadores de imagen operan en modo progresivo solamente [4].

Esta cámara se alimenta a través del cable firewire de 6 pines. En el caso del proyecto, el cable firewire irá conectado a un ordenador portátil Sony VAIO, que tan sólo dispone de un puerto de cuatro pines. Los dos pines de diferencia son los pines de la alimentación.

Para solucionar este problema se adquirió un cable firewire que tiene 6 pines en un extremo y 4 en el otro, sacando la alimentación con dos conectores auxiliares.

Estos últimos se conectarán a las baterías de Higgs de tal manera que la tensión sea la adecuada.

La máxima transferencia de vídeo es a 12 megapíxeles por segundo desde cada captador, los cuales están sincronizados con un reloj común. De esta forma los píxeles correspondientes de las dos imágenes son dados exactamente en el mismo instante de tiempo. La electrónica convierte los dos streams individuales en un sólo stream entrelazado. Este último contiene un bit de la imagen izquierda, uno de la derecha, el siguiente de la izquierda.... [ref 13].

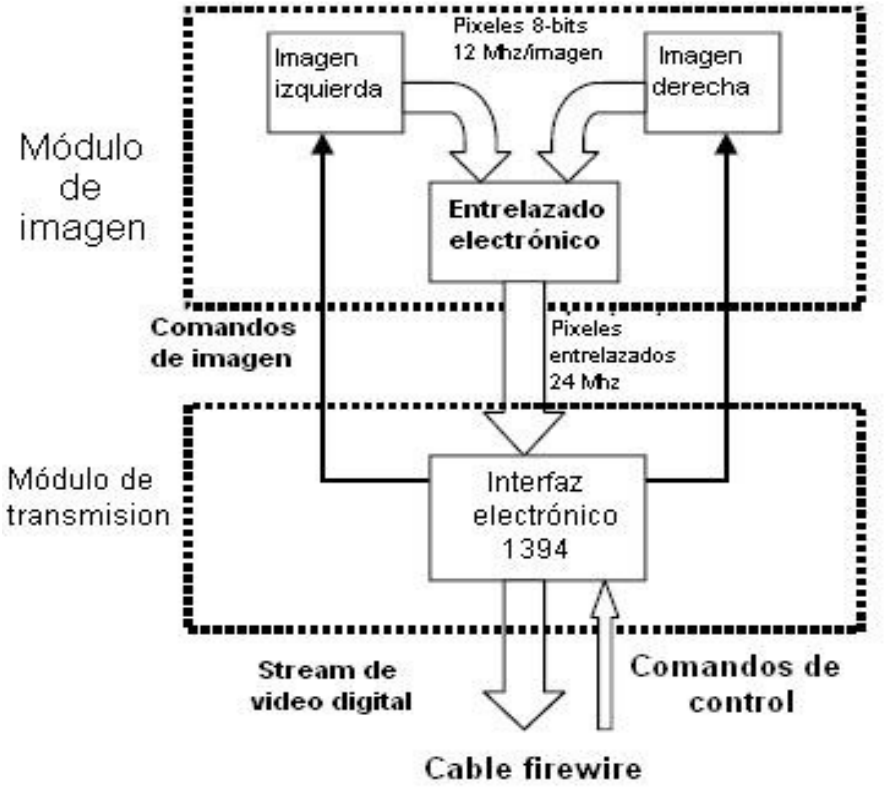


Figura 3.4: Esquema de funcionamiento de la cámara esterostrópica firewire.

La especificación de las cámaras digitales se hizo teniendo en mente cámaras monoculares. Para cumplir con esta especificación, la STH-MDCS-C utiliza el tipo de dato YUV, enviando la imagen izquierda en y la derecha en UV. Cada imagen de la cámara estéreo tiene píxeles de 8 bits, donde la información del color es codificada con un patrón de Bayer.

En el PC servidor, el software de SVS convierte el stream YUV en las imágenes izquierda y derecha. También procesa la información del color, convirtiendo el patrón de Bayer en imágenes RBG.

Las especificaciones técnicas podemos encontrarlas en la tabla siguiente:

Elemento	**Especificaciones**
Captadores de imagen	Formato CMOS 1/2 pulgada, área 1280x960
Versión	1.3
Formatos	1980x960,640x480,Monocromático de 8 bits o Patrón de Bayer
Consumo	Menor de 1 Watio
Sensibilidad	2.5 V = lux-sec
Error estadístico (1 Sigma)	5 mm
Clase láser	1 (eye safe)
Máxima distancia	80 metros
Tiempo de respuesta	13ms/26ms/53ms
Error sistem_atico (modo mm)	15mm
Peso	4.5 kgr
Ancho	155 mm
Alto	210 mm
Profundo	256 mm

La cámara irá montada sobre una muñeca mecánica que servirá en un futuro para la inclinación de la cámara para una visión móvil del entorno.

Figura 3.5: Sistema robótico montado.

Capítulo 4

4 Visión de video en tiempo real a través de red wireless.

En este capítulo, veremos con que técnicas, métodos y tecnologías, hemos logrado la visión desde un ordenador de las imágenes en tiempo real de la cámara que está montada sobre Higgs, así un operario sentado en el terminal de control, podrá saber lo que está "viendo" Higgs en cada momento, y poder controlarlo de forma mucho más óptima.

4.1 requisitos y decisiones de diseño

Para llegar a las especificaciones del proyecto, se habrán tenido que satisfacer una serie de requisitos:

1. La emisión de datos debe de ser continua y sin pausa, para que el control y manejo de Higgs sea óptima, es decir que sea en tiempo real y haya el menor "*delay*" posible de imágenes.
2. La transmisión de datos debe de ser a través de una red TCP/IP wireless, ya que es la manera de comunicación de redes más común y barata que existe. El robot al ser móvil no debe de estar atado o anclado por medio de un cable.
3. El software de recepción de imágenes debe de poder estar programado tanto en Java, como en C++.
4. La emisión de imágenes debe de ser esterostrópica y se debe de poder decidir si queremos el visionado de las dos imágenes o solo la derecha o la izquierda.

Para atajar estos problemas, se tomaron decisiones de diseño para poder satisfacer todos y cada uno de los puntos anteriores:

1. La emisión de datos se realizará a traves de una red wireless tipo G de 48 megabits por segundo, lo que nos dará un ancho de banda suficiente como para poder emitir imágenes sin retraso.
2. La cámara de video es firewire, e irá conectado a un ordenador ultraportatil Sony Vaio que se encargará de recoger las imágenes por el dicho puerto firewire, y las

enviará a través de la red wireless hacia el cliente de imágenes. Tanto el servidor como el cliente estarán conectados a través de tarjetas inalámbricas a la red wireless comentada en el punto anterior.

3. Usaremos la tecnología CORBA como sistema de envío y recepción de datos. Esta tecnología es compatible con el estándar TCP/IP. Además se puede implementar tanto en lenguaje C++ (en este caso con TAO) y Java (aquí se usará JavaORB), con lo que el cliente puede ser desarrollado con los dos lenguajes mencionados antes.
4. La imagen se enviará entrelazada al cliente, estando así codificada dos imágenes en una (ocupando el doble de espacio). Cuando llega al servidor la desentrelaza y la convierte otra vez en dos imágenes diferentes, así obtenemos imágenes esteroscópicas.

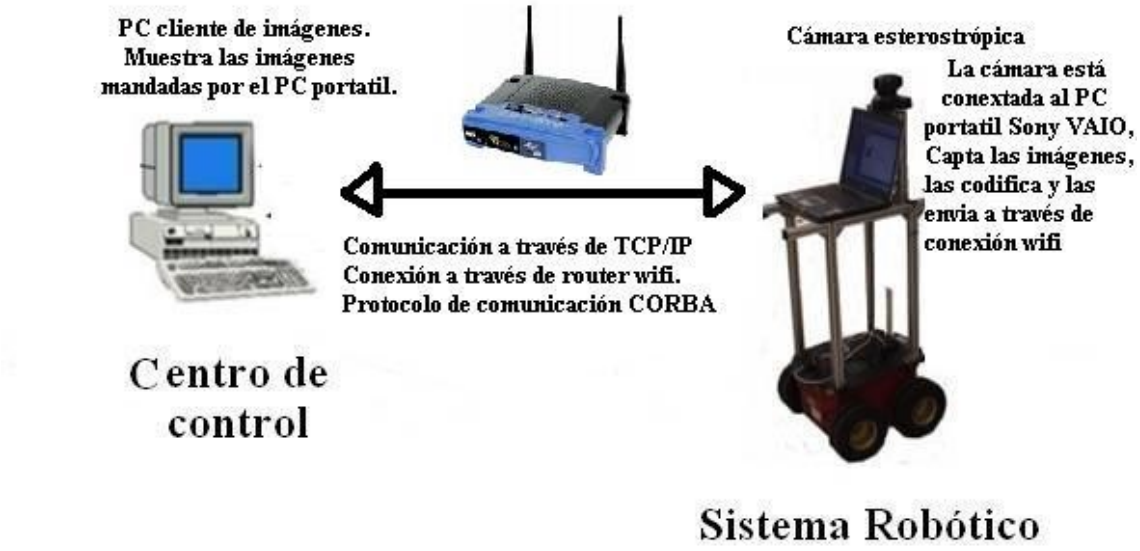


Figura 4.1: Esquema general del sistema de visionado de imágenes remotas.

4.2 ¿Qué es FireWire?

El IEEE 1394, FireWire o i.Link es un estándar multiplataforma para entrada/salida de datos en serie a gran velocidad. Suele utilizarse para la interconexión de dispositivos digitales, como cámaras digitales y videocámaras, a ordenadores. Este es el médio de conexión entre la cámara y el ultraportatil Sony Vaio.

El FireWire fue inventado por Apple Computer a mediados de los 90, para luego convertirse en el estándar multiplataforma IEEE 1394. A principios de este siglo fue adoptado por los fabricantes de periféricos digitales hasta convertirse en un estándar

establecido. Sony utiliza el estándar IEEE 1394 bajo la denominación i.Link, que sigue los mismos estándares pero sólo utiliza 4 conexiones de las 6 disponibles en la norma IEEE 1394, suprimiendo las dos conexiones encargadas de proporcionar energía al dispositivo, que tendrá que proveerse de ella mediante una toma separada [ref 14].

Las características del estándar IEEE 1394 son las siguientes:

1. Elevada velocidad de transferencia de información.
2. Flexibilidad de la conexión.
3. Capacidad de conectar un máximo de 63 dispositivos.

Su velocidad hace que sea la interfaz más utilizada para audio y vídeo digital. Así, se usa mucho en cámaras de vídeo, discos duros, impresoras, reproductores de vídeo digital, sistemas domésticos para el ocio, sintetizadores de música y escáneres. Existen dos versiones:

1. FireWire 400: tiene un ancho de banda 30 veces mayor que el USB 1.1.
2. IEEE 1394b, FireWire 800 o FireWire 2: duplica la velocidad del FireWire 400.

Así, para usos que requieran la transferencia de grandes volúmenes de información, resulta muy superior al USB.

El FireWire es una tecnología muy empleada en la actualidad. Presenta una serie de ventajas que lo convierten en una de las mejores opciones del mercado:

- Alcanza una velocidad de 400 megabits por segundo.
- Es hasta cuatro veces más rápido que una red Ethernet 100Base-T, y 40 veces más rápido que una red Ethernet 10Base-T.
- Soporta la conexión de hasta 63 dispositivos con cables de una longitud máxima de 425 cm.
- No es necesario apagar un escáner o una unidad de CD antes de conectarlo o desconectarlo, y tampoco requiere reiniciar el ordenador.
- Los cables FireWire se conectan muy fácilmente: no requieren números de identicación de dispositivos, conmutadores DIP, tornillos, cierres de seguridad ni terminadores. FireWire funciona tanto con Macintosh como con PC.
- FireWire 400 envía los datos por cables de hasta 4,5 metros de longitud, mediante fibra óptica profesional, FireWire 800 puede distribuir información por cables de hasta 100 metros.

4.3 Visión estereoscópica

La **estereoscopía**, **imagen estereográfica**, o **imagen 3D (tridimensional)** es cualquier técnica capaz de recoger información visual tridimensional o de crear la ilusión de profundidad en una imagen. La ilusión de la profundidad en una fotografía, película, u otra imagen bidimensional es creada presentando una imagen ligeramente diferente para cada ojo, como ocurre en nuestra forma habitual de recoger la realidad. Muchas pantallas 3D usan este método para transmitir imágenes. Fue inventado primero por Sir Charles Wheatstone en 1838.

La estereoscopía es usada en fotogrametría y también para entretenimiento con la producción de estereogramas. La estereoscopía es útil para ver imágenes renderizadas de un conjunto de datos multi-dimentionales como los producidos por datos experimentales. La fotografía tridimensional de la industria moderna puede usar escáners 3D para detectar y guardar la información tridimensional. La información tridimensional de profundidad puede ser reconstruida partir de dos imágenes usando una computadora para hacer relacionar los pixels correspondientes en las imágenes izquierda y derecha. Solucionar el problema de correspondencia en el campo de la visión por computadora apunta crear información significativa de profundidad a partir de dos imágenes.

La fotografía estereoscópica tradicional consiste en el crear una ilusión 3-D a partir de un par de imágenes 2D. La forma más sencilla de crear en el cerebro la percepción de profundidad es proporcionando a los ojos del espectador dos imágenes diferentes, que representan dos perspectivas del mismo objeto, con una pequeña desviación similar a las perspectivas que de forma natural reciben los ojos en la visión binocular.

Si se quiere evitar la fatiga visual y la distorsión, cada una de las imágenes 2D se debe presentar preferiblemente al ojo correspondiente del espectador de tal manera que cualquier objeto a distancia infinita percibido por el espectador debe ser percibido por ese ojo mientras está orientado justo derecho hacia adelante, los ojos del espectador no son cruzados ni divergen. Cuando la imagen no contiene ningún objeto de distancia infinita, como un horizonte o una nube, la imágenes deben ser espaciadas correspondientemente más cerca.

Literalmente, estereoscopía es: ver con dos ojos. La estereoscópica también llamada visión en tres dimensiones, o visión en relieve, resulta de la capacidad del sistema visual de dar aspecto tridimensional a los objetos a partir de las imágenes en dos dimensiones

obtenidas en cada una de las retinas de los ojos. Estas imágenes son procesadas y comparadas por el cerebro, el cual acaba creando una sensación espacial.

Por lo que si tomamos o creamos dos imágenes con un ángulo ligeramente distinto y se las mostramos a cada ojo por separado, el cerebro podrá reconstruir la distancia y por lo tanto la sensación de profundidad. De aquí se extrae la conclusión de que las variaciones horizontales que hacen que las imágenes tengan un ángulo ligeramente diferente pueden ser interpretadas por nuestro cerebro como una realidad con volumen.

Percepción en relieve de las imágenes:

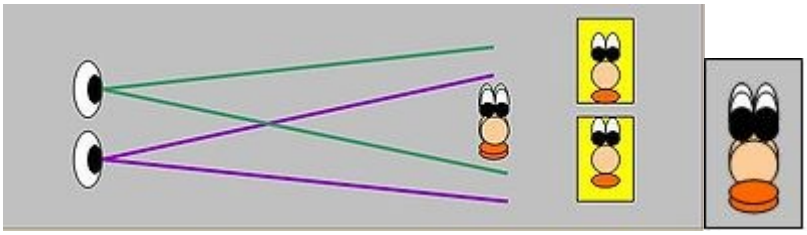


Figura 4.2: El ojo percibe los objetos en diferentes ángulos, creando la ilusión de profundidad de los objetos.

Las variaciones verticales son indiferentes en lo que respecta a creación de sensación de volumen (a no ser que esta diferencia sea demasiado grande).

Esta tercera dimensión es capaz de reconstruirse en nuestro cerebro gracias a una serie de complejos procesos fisiológicos y psicológicos relacionados con la visión tanto monocular como la binocular:

Dos imágenes estereoscópicas, como las que veremos en pantalla de la cámara

4.4 Entrelazado de imágenes

Entrelazado es una técnica consistente en organizar la información digital de forma no contigua para mejorar las prestaciones de un sistema.

El entrelazado (*interleaving* en inglés) es empleado principalmente en telecomunicaciones (por ejemplo vía satélite o ADSL), tecnologías multimedia, acceso a memoria y formatos de archivo. El término "multiplexado" se utiliza a veces para referirse al entrelazado de una señal digital.

El entrelazado se emplea también para organizar estructuras de datos multidimensionales. El entrelazado es utilizado en transmisión digital de datos como técnica para proteger la información frente a los errores de ráfaga (*burst errors*). Este tipo de errores ocasionales afectan a varios bits seguidos, e invalidan las propiedades correctoras de error de los códigos redundantes que se emplean en la transmisión de datos. Al emplear técnicas de entrelazado, los errores de ráfaga se ven distribuidos entre varias palabras, facilitando la labor correctora del código empleado.

La cámara firewire empleada en el proyecto, interlaza las imágenes para evitar fallos en la recepción de datos tal y como se muestra arriba, a la hora de obtener la imagen ya digitalizada, nosotros tenemos que trabajar con datos y píxeles, con lo cual lo que hacemos es separar los bytes paras con los impares.

1º Accedemos a la memoria donde se aloja la imagen.

2º Separamos los bytes pares de los impares.

3º Unimos los bytes colocándolos de forma contigua.

4º Guardamos las imágenes otra vez en memoria pero ahora una imagen contigua a la otra y de forma separada.

4.5 Las librerías dc1394

Libdc1394 es una librería que proporciona una interfaz de programación de alto nivel (API) para desarrolladores que desean tener un gran control en todas las cámaras que usen el estándar IEEE 1394 de cámaras firewire. Esta librería está desarrollada para Linux y para Mac OSX. Esta interfaz incluye detección de cámaras, comandos de control de ancho de banda, actuadores de control de disparo, compatibilidad con todos

los modos de video, funciones propias para la transformación de video y otras características interesantes [ref 9].

Estas librerías fueron desarrolla originalmente en el año 2000. Antes, la programación de cámaras firewire era realmente complicada y necesitaba una serie de conocimientos por parte de los usuarios de lenguajes de programación de muy bajo nivel. De todas formas, no fue hasta la salida de la versión 1.0 del año 2004, cuando estas librerías empezaron a soportar gran cantidad de cámaras firewire, y empezó a ser realmente popular. Ese mismo año se empezó a trabajar en la versión 2.0, y fue dado a luz el año 2008 con compatibilidad en Mac OSX, también se añadieron soluciones a múltiples bugs de la anterior versión esta versión es la que se usará en el proyecto.

4.5.1 Velocidad de transmisión ISO

Define la velocidad utilizada para la transmisión en el bus FireWire. Actualmente la velocidad máxima que se obtiene en un bus FireWire es de 800 Mbps. Para la aplicación se utiliza la velocidad de 400 Mbps:

DC1394_ISO_SPEED_400

Así la velocidad de transferencia puede se puede cambiar modificando el parámetro 400 por otros valores estandarizados:

DC1394_ISO_SPEED_100, DC1394_ISO_SPEED_200,

DC1394_ISO_SPEED_400, DC1394_ISO_SPEED_800,

DC1394_ISO_SPEED_1600, DC1394_ISO_SPEED_3200

Es importante hacer notar, que las velocidades de 1600 y 3200 todavía no están soportadas por las limitaciones de la conexión firewire.

4.5.2 Velocidad de captura

Define la velocidad de captura de la cámara. Para realizar las medidas en este proyecto se empleará una velocidad de captura de 15 imágenes por segundo, aunque la cámara tiene una capacidad de captar 30 imágenes por segundo. Esta decisión la tomamos partiendo de la base que nuestro principal problema a la hora de mandar imágenes a través de la red, era el ancho de banda de la misma. El ancho de banda hace de cuello de botella a la hora de hacer llegar un “chorro” fluido de imágenes y como actualmente las

redes wireless tienen una velocidad de transmisión limitada, nunca llegaríamos a tener más de 15 imágenes por segundo. La función que configura la cámara a 15 frames por segundo, sería:

DC1394_FRAMERATE_15

De todas formas se podría gestionar el número de frames con otros comandos:

DC1394_FRAMERATE_1_875 DC1394_FRAMERATE_3_75,

DC1394_FRAMERATE_7_5 DC1394_FRAMERATE_30,

Las siguientes configuraciones no serían posibles debido a limitaciones del hardware a un máximo de 30 imágenes por segundo:

DC1394_FRAMERATE_60 DC1394_FRAMERATE_120,

DC1394_FRAMERATE_240

También se podría definir de forma dinámica las imágenes por segundo y poner un valor mínimo y máximo que se desea:

#define DC1394_FRAMERATE_MIN

#define DC1394_FRAMERATE_MAX

#define DC1394_FRAMERATE_NUM

4.5.3 Resolución de video

Definen la resolución de la imagen capturada. La **resolución de imagen** indica cuánto detalle puede observarse en una imagen. El término es comúnmente utilizado en relación a imágenes de fotografía digital, pero también se utiliza para describir cuán nítida (como antónimo de granular) es una imagen de fotografía convencional (o fotografía química). Tener mayor resolución se traduce en obtener una imagen con más detalle o calidad visual. Para las imágenes digitales almacenadas como mapa de bits, la convención es describir la resolución de la imagen con dos números enteros, donde el primero es la cantidad de filas de píxeles (cuántos píxeles tiene la imagen a lo alto) y el segundo es la cantidad de columnas de píxeles (cuántos píxeles tiene la imagen a lo ancho). En el código que se incluye a continuación se muestran sólo los modos para los que la aplicación desarrollada funciona o que son aceptados por las cámaras empleadas.

Resaltar que a mayor resolución, la velocidad que se puede conseguir al capturar es menor, ya que se emplea un mayor ancho de banda para la transmisión por el cable. La cámara firewire que usamos soporta dos modos de video: 640 x 480 y 1980 x 960. Ya que la segunda resolución es muy alta, decidimos usar la opción de 640 x 480. Con la segunda resolución sería imposible un envío constante y fluido de imágenes por la red [ref 17].

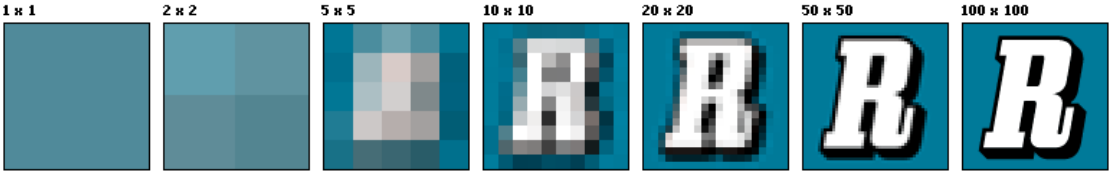


Figura 4.4: Es muy importante tener resoluciones altas para tener imágenes nítidas

4.5.4 Codificación de color

Para entender las diferentes transformaciones y otras operaciones para llegar al formato que necesitamos, es necesario definir varios estándares de codificación de color, siendo las 3 más importantes las que vamos a tratar.

Modo raw

Cuando una cámara digital hace un disparo, el sensor de imagen (ya sea de tipo CCD o CMOS), recoge la cantidad de luz que llega a cada uno de sus componentes. Esto se lee como un voltaje de un nivel determinado. Los circuitos de conversión de analógico a digital de la cámara transforman esta señal de voltaje analógica en un valor digital.

RAW (en inglés significa *crudo*) es un formato de archivo digital de imágenes que contiene la totalidad de los datos de la imagen tal y como ha sido captada por el sensor digital de la cámara de video.

Este tipo de imágenes ocupan una gran cantidad de información y no son aptas para el propósito de este proyecto. En resumen, hay que transformala, en este caso codificarla.

Modo rgb

La descripción RGB (del inglés *Red*, *Green*, *Blue*; "rojo, verde, azul") de un color hace referencia a la composición del color en términos de la intensidad de los colores primarios con que se forma: el rojo, el verde y el azul. Es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores luz primarios. El modelo de color RGB no define por sí mismo lo que significa exactamente rojo, verde o azul, por lo que los mismos valores RGB pueden mostrar colores notablemente diferentes en diferentes dispositivos que usen este modelo de color. Aunque utilicen un mismo modelo de color, sus espacios de color pueden variar considerablemente.

Para indicar con qué proporción mezclamos cada color, se asigna un valor a cada uno de los colores primarios, de manera, por ejemplo, que el valor 0 significa que no interviene en la mezcla y, a medida que ese valor aumenta, se entiende que aporta más intensidad a la mezcla. Aunque el intervalo de valores podría ser cualquiera (valores reales entre 0 y 1, valores enteros entre 0 y 37, etc.), es frecuente que cada color primario se codifique con un byte (8 bits). Así, de manera usual, la intensidad de cada una de las componentes se mide según una escala que va del 0 al 255.

En las pantallas de computadoras, la sensación de color se produce por la mezcla aditiva de rojo, verde y azul. Hay una serie de puntos minúsculos llamados píxeles. Cada punto de la pantalla es un píxel y cada píxel es, en realidad, un conjunto de tres subpíxeles; uno rojo, uno verde y uno azul, cada uno de los cuales brilla con una determinada intensidad.

Al principio, la limitación en la profundidad de color de la mayoría de los monitores condujo a una gama limitada a 216 colores, definidos por el cubo de color. No obstante, el predominio de los monitores de 24-bit, posibilitó el uso de 16.7 millones de colores del espacio de color HTML RGB.

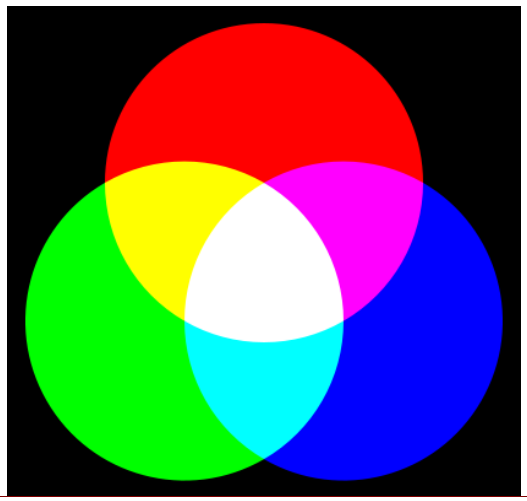


Figura 4.5:Mezcla aditiva de colores

Modo YUV

Y'UV Es un espacio de color tipicamente usado como parte de un conducto de imagen en color. Codifica una imagen o vídeo en color teniendo en cuenta la percepción humana, permitía el ancho de banda reducido para los componentes de crominancia, de ese modo típicamente hace que los errores de transmisión o las imperfecciones de compresión se oculten más eficientemente a la percepción humana que usando una representación RGB "directa". Otros espacios de color tienen propiedades similares, y la principal razón para implementar o investigar propiedades de Y'UV sería para interactuar con televisión analógica o digital o equipamiento fotográfico que se ajusta a ciertos estándares Y'UV.

El modelo **YUV** define un espacio de color en términos de una componente de luminancia y dos componentes de crominancia.

Las siguientes ecuaciones se usan para calcular *Y*, *U* y *V* a partir de *R*, *G* y *B*:

$$Y = 0{,}299R + 0{,}587G + 0{,}114B$$
$$U = 0{,}492(B - Y) = -0{,}147R - 0{,}289G + 0{,}436B$$
$$V = 0{,}877(R - Y) = 0{,}615R - 0{,}515G - 0{,}100B$$

O usando las matrices:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0,299 & 0,587 & 0,114 \\ -0,147 & -0,289 & 0,436 \\ 0,615 & -0,515 & -0,100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

El "Subsampling" (submuestreo) consiste en reducir la información de color preservando intacta la luminosidad. Eso se expresa de la siguiente manera:

- 4:4:4 mantiene intacta tanto la información de la luminosidad (primer "4") como la del color (los otros dos "4"'s)
- 4:2:2 reduce el muestreo del color a la mitad
- 4:1:1 reduce el muestreo de color a la cuarta parte
- 4:2:0 elimina uno de los valores de color dejando el otro valor en la mitad.

(Los valores empleados en la compresión JPG y MPEG son 4:1:1 y 4:2:0)

Para saber el porqué se le llama "sub-muestreo" hay que recordar que en un televisor la imágen se forma de forma entrelazada entre 625 líneas horizontales (525 para NTSC). Por tanto, el "submuestreo" hace referencia a que para comprimir la imagen tan sólo se utiliza el muestreo vertical, puesto que las líneas son horizontales y se cuentan de "arriba a abajo".

Debido al poco espacio que ocupa este formato, será este formato YUV 422 el que usaremos para enviar las imágenes por la red perdiendo la menor cantidad de información posible.

Una vez explicado cada una de las cosas, la forma de tratar las imágenes, sería:

1º Cogemos la imagen en crudo (raw) entrelazada de la cámara firewire. Esta imagen es tratada desde el cliente.

2º Desentrelazamos la imagen de la cámara firewire, a partir de entonces tenemos dos imágenes, la derecha y la izquierda.

3º Transformamos las dos imágenes de crudo a RGB por medio de la codificación de bayer.

4º Comprimimos las dos imágenes de RGB a YUV 422 con un algoritmo matemático de transformación.

5º Enviamos la imagen con CORBA a través de la red.

6º El cliente recibe las imágenes y las muestra en pantalla.

Este método es explicado en el siguiente diagrama de flujos:

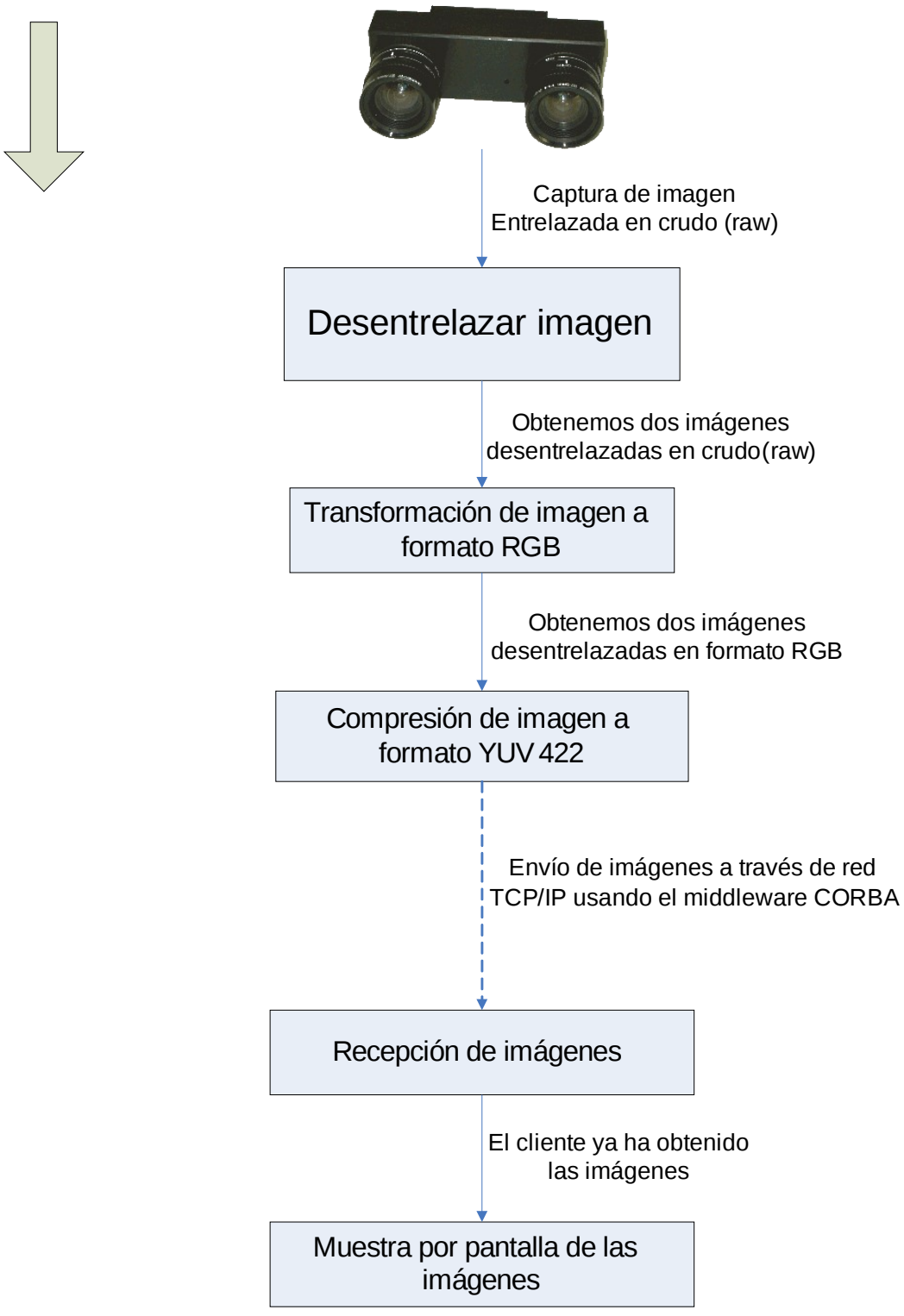


Figura 4.6: Diagrama de flujo del tratamiento de imágenes

4.5.5 Modos de disparo

Define los tipos de trigger externo que se pueden emplear. Cada cámara puede, o no, aceptar un número variable de ellos. Además de los modos de trigger externo que la cámara acepte, también puede funcionar haciendo uso del trigger interno.

En el modo 0 la cámara empieza a capturar cuando se recibe un flanco activo, y el tiempo que permanece capturando está determinado por el valor del parámetro shutter.

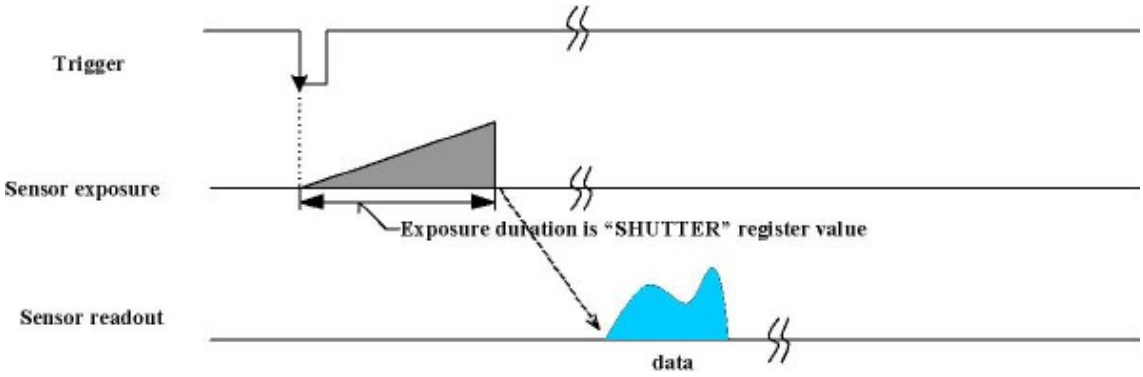
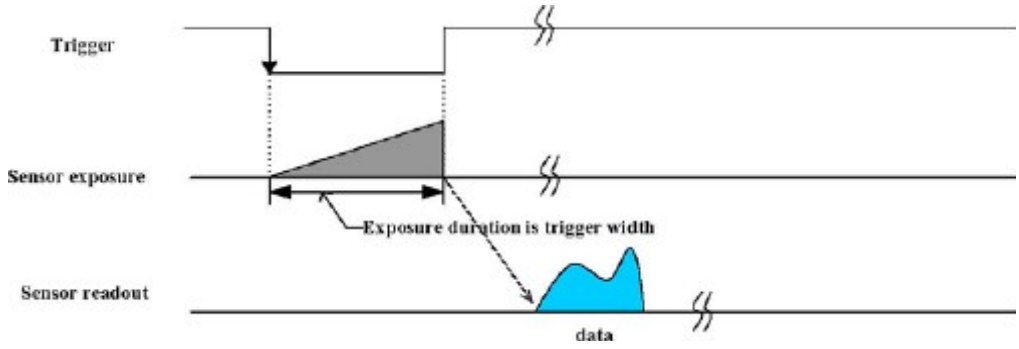


Figura 4.7: Cronograma del trigger modo 0

Para realizar las pruebas se ha empleado el modo 1, en el cual la captura empieza cuando se recibe un flanco de reloj y termina cuando se recibe otro flanco de reloj. Se puede configurar la cámara para que el disparo sea activo en el flanco de bajada o de subida. Para la aplicación desarrollada se ha configurado el trigger como activo en el flanco de bajada. De esta forma el tiempo de exposición puede controlarse mediante el tiempo que la línea se encuentra a nivel bajo.



Cronograma del trigger modo 1

4.5.6 Política de espera para capturar las imágenes

Define la política de espera cuando se realiza la captura de imágenes. Es posible que, al llamar a la función de captura, ésta se quede bloqueada hasta que en la cámara se haya capturado un frame, o simplemente sondear la cámara para comprobar si hay algún frame disponible.

La configuración:

DC1394_VIDEO1394_WAIT

hace que el programa sondee la cámara para ver si hay alguna imagen disponible en el buffer y, en caso contrario, permanece a la espera bloqueando el programa hasta que una imagen sea capturada.

Con la otra configuración:

DC1394_VIDEO1394_POLL

se realiza el sondeo de la cámara pero no se bloquea el programa hasta que una imagen haya sido capturada, sino que se continúa con el _ujo normal de éste.

Básicamente, con este parámetro se puede definir la función

dc1394_capture_dequeue()

como bloqueante, eligiendo

DC1394_VIDEO1394_WAIT

o no bloqueante, eligiendo

DC1394_VIDEO1394_POLL

A continuación se incluye un ejemplo de llamada a la función de captura:

dc1394_capture_dequeue(camera, DC1394_CAPTURE_POLICY_WAIT, &frame);

En este ejemplo se realiza un sondeo de la cámara apuntada por camera para ver si hay imágenes disponibles. En caso de que no haya ninguna, el _ujo del programa se bloquea

hasta que una imagen sea capturada. La imagen se almacena en uno de los campos de la estructura frame.

4.6 Funcionamiento del buffer de captura

Una vez que la cámara está configurada y se ha ejecutado *dc1394_capture_setup()*, un bloque de memoria contigua es reservado como un bufier DMA circular esperando a ser rellenado. Cuando se inicia la captura con *dc1394_video_set_transmission()*, cada ranura del buffer empieza a rellenarse, capturando la cámara a la velocidad configurada.

La primera vez que se llama a la función *dc1394_capture_dequeue()*, ésta retorna un puntero a la primera estructura del buffer (dc1394frame_t). Se puede configurar la función para que espere a que haya un frame disponible o que retorne inmediatamente, aunque no haya ningún frame disponible. Cada vez que se llama a la función de captura a partir de ahora, se obtiene un puntero al siguiente frame en el buffer circular. Si se llama a la función *dc1394_capture_enqueue()* se libera la ranura correspondiente a la imagen más antigua capturada. No se liberará el espacio correspondiente a una trama en el buffer hasta que no se llame a la función *dc1394_capture_enqueue()*.

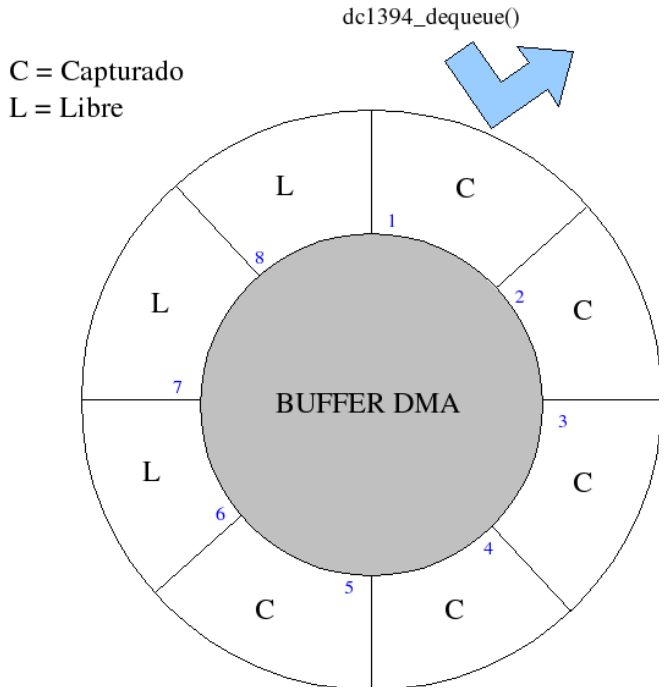
Puede ocurrir que la cámara intente guardar una imagen en el buffer cuando esté lleno.

En este caso la cámara no puede guardar las nuevas tramas, por lo que se produce una pérdida de datos. Esto es un problema bastante importante cuando se captura con cámaras de este tipo, ya que se debe garantizar que nunca se llenará el buffer DMA, puesto que si esto ocurriera la cámara no podría capturar a la velocidad marcada (fps), y se observarían saltos en la grabación. Si no se garantiza esto, los frames que se guarden estarán siempre colocados en el buffer en orden cronológico, pero pueden estar irregularmente capturados en el tiempo. El tamaño de este buffer se puede establecer por medio de la librería libdc1394, gracias al parámetro NUM_BUFFERS, utilizado con la función:

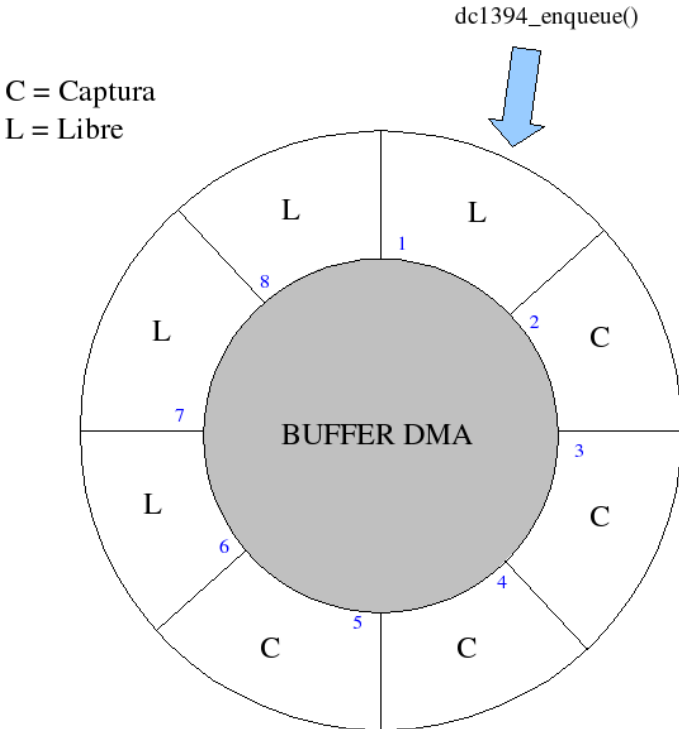
dc1394_capture_setup(camera, NUM_BUFFERS)

Para solucionar este problema y garantizar que se está accediendo a la trama más reciente al comenzar a grabar, se debe vaciar el bu_er DMA antes de realizar la petición de la primera trama, y luego llamar a la función de captura. Esta función debería ser llamada antes de empezar a grabar en el disco. Básicamente, lo que hace es ir sacando todas las tramas capturadas y almacenadas en el buffer, liberando el espacio que ocupan en éste.

En el momento en que no queden frames en el buffer, la función concluye. Esta función, llamada antes de *dc194_dequeue()*, garantiza que el frame que devuelve dicha función es el último capturado.



☎ *La función dc1394_dequeue() requiere un frame del buffer DMA.*



(b) La función dc1394_enqueue() libera una ranura del buffer DMA.

4.7 Gráficos con X11

En Unix hay un programa que es el encargado de dibujar sobre la pantalla, es el servidor de X. Este programa lo suele arrancar Unix al encender el ordenador. A partir de ese momento estará siempre pendiente de cualquier otro programa que quiera dibujar en pantalla. Cualquier programa unix que quiera dibujar sobre la pantalla, debe hacerlo a través del servidor. También disponemos de unas librerías para conectarnos y dibujar gráficos con dicho servidor (las librerías de X11).

Las librerías de X11 son las de más "bajo nivel" con las que podemos pintar gráficos sobre Unix. Únicamente nos van a permitir abrir ventanas gráficas y dibujar en ellas con las primitivas de pintado habituales en cualquier lenguaje de programación (pintar puntos, lineas, arcos, etc).

Si queremos dibujar algo "más complejo", como una ventanas con botones, listas y menús desplegables, debemos irnos a librerías de más "alto nivel" (Gtk, motif, etc), que normalmente están basadas a su vez en las librerías de X11.

Las librerías que se usan para el uso de X11 son las Xlib. **Xlib** son un conjunto de funciones y macros realizadas en C y utilizadas por un 'cliente' como el interfaz con la versión 11 de X Window System. En resumen, es un interfaz de programación de bajo nivel para X. Xlib está basada en la filosofía de eventos (o mensajes). La biblioteca apareció alrededor del 1985.

Algunas aplicaciones utilizan directamente Xlib, sin embargo es muy común que se utilicen bibliotecas que a la vez la utilizan, entre ellas se encuentran: Intrinsics (Xt), Xaw, XView, Motif, GTK+, Qt (versión para X11), Tk.

Xlib proporciona un método relativamente sencillo para realizar peticiones de protocolo X al servidor. Incluye funciones para realizar más livianamente las tareas más frecuentes, permitiendo al programador no usar el protocolo X para dichas tareas. Proporciona rutinas para facilitar la labor de crear y manipular recursos básicos (ventanas, contextos gráficos,...), el dibujo de elementos gráficos (botones, líneas,...) o generar la entrada y salida de texto, entre otros.

4.7.1 Crear una ventana para dibujar

El cliente tiene que ser capaz de crear una ventana para mostrar las imágenes por pantalla. Para ello tenemos varias funciones, pero la más sencilla es *Window XCreateSimpleWindow ()*. Los parámetros de esta función son:

- El display obtenido anteriormente con *XOpenDisplay()*. De esta forma indicamos en qué servidor de X queremos crear la ventana.

- Una ventana padre para la que estamos creando. Nuestra ventana heredará algunos de los atributos de la ventana padre, como por ejemplo, la paleta de colores. Como es habitual en los entornos de ventanas, hay una ventana "raíz" que es la pantalla completa. A partir de ahí y en forma de "árbol" se van creando subventanas. Luego se podrían hacer cosas como que al ocultar una ventana "padre", que se oculten todas sus "hijas", que una ventana "hija" no se pueda salir fuera de la ventana "padre, etc. Para evitar complicaciones, en este parámetro pondremos la ventana "raíz". La ventana "raíz" de una pantalla nos la da la función Window *XDefaultRootWindow (display)*, a la que como es de suponer, hay que pasar el display obtenido con *XOpenDisplay()***.**

- Posición x de la ventana en la pantalla (en pixels).

- Posición y de la ventana en la pantalla (en pixels). La posición que debemos dar es la de la esquina superior izquierda de la ventana. La posicion (0,0) es la esquina superior izquierda de la ventana padre (en nuestro caso, de la pantalla). De todas formas, en ningún sistema que he probado funcionan estos dos parámetros. Al final siempre hay que mover la ventana con la función *XMoveWindow()* después de que esté visible.

- Ancho de la ventana (en pixels).

- Alto de la ventana (en pixels).

- Ancho en pixels del borde de la ventana. Este no tiene nada que ver con la barra de título y los bordes de los que podemos "estirar". Este borde es interno a la ventana gráfica.

- Indice de color para el borde anterior. El índice de color es un número de 0 en adelante. Según la configuración de nuestro servidor, el máximo puede ser 255 o un número mucho más grande.

- El índice del color de fondo de la ventana. Toda la ventana aparecerá inicialmente rellena de este color.

4.7.2 Redibujado de la ventana

El redibujado del contenido está dentro de un lazo para eventos. Antes que entre a este lazo, se establece la clase de eventos de la aplicación, en este caso se realiza con la llamada a *XSelectInput*. El lazo de evento espera por la llegada de un evento: si el evento es una tecla presionada, la aplicación termina; si el evento es *Expose*, se redibuja el contenido de la ventana.

La función *XNextEvent* bloquea y vuelca el buffer requerido si no hay ningún evento en la cola. La función que hace que se pinte en pantalla es *XMapWindow***.** Los parámetros son, como no, la conexión con el servidor de X y el identificador de la ventana que queremos que visualice. La llamada *XCreateGC* crea un contexto gráfico donde se guardan las propiedades de la ventana creada en el apartado anterior.

4.8 Transmisión de datos del servidor al cliente

En este apartado se explicará como conseguimos el envío de datos a través de la red usando tecnología CORBA. Dentro del servidor se define una función *Imagen_impl* que tendrá como misión obtener la primera imagen del buffer DMA de imágenes que hemos explicado anteriormente, esta función es invocada de forma remota por el cliente que devolverá un string de octetos (o de chars, que ocuapan cada uno 1 byte)que contienen la imagen capturada. Debido a que la diferencia entre lenguajes de programación y por limitaciones de la tecnología CORBA, no se pueden transladar localizaciones de memoria entre funciones remotas, por ello, se debe coger cada uno de los bytes de la imagen alojados en memoria y transladarlos a un string de octetos o chars que será enviado por red. Sabiendo que esa imagen está en formato UYV422, podemos reconstruir la imagen dentro del cliente, y por lo tanto mostrarla en pantalla. Para ello el programa servidor y cliente hacen las siguientes operaciones:

1º El cliente pide a través de CORBA, una imagen al servidor.

2º El servidor coge una imagen del buffer anterior y la aloja en memoria.

3º El servidor transforma la imagen a formato uy422 como hemos visto en apartados anteriores.

4º El servidor transforma los bytes que forman la imagen y la translada a un string con una longuitud de 640 x 480 x 2 x 2 (el priemer dos por el formato uyv422 usado, el segundo 2 por ser dos imágenes) de octetos o chars, sabemos que esa imagen está en formato uyv422.

5º El servidor envía el string de octetos que contiene la imagen

6º El servidor elimina la imagen enviada

7º El servidor manda a la cámara obtener otra imagen e introducirla en el buffer DMA.

8º El cliente recibe la imagen en forma de string y lo guarda en memoria.

9º El cliente destransforma la imagen sabiendo que está en formato uyv422, y la aloja en memoria como en su estado antes de que se convirtiera en un string de datos.

10º Usando las librerías X11, el servidor muestra la imagen por pantalla (que en realidad son 2).

Este proceso es explicado en el siguiente diagrama de flujo:

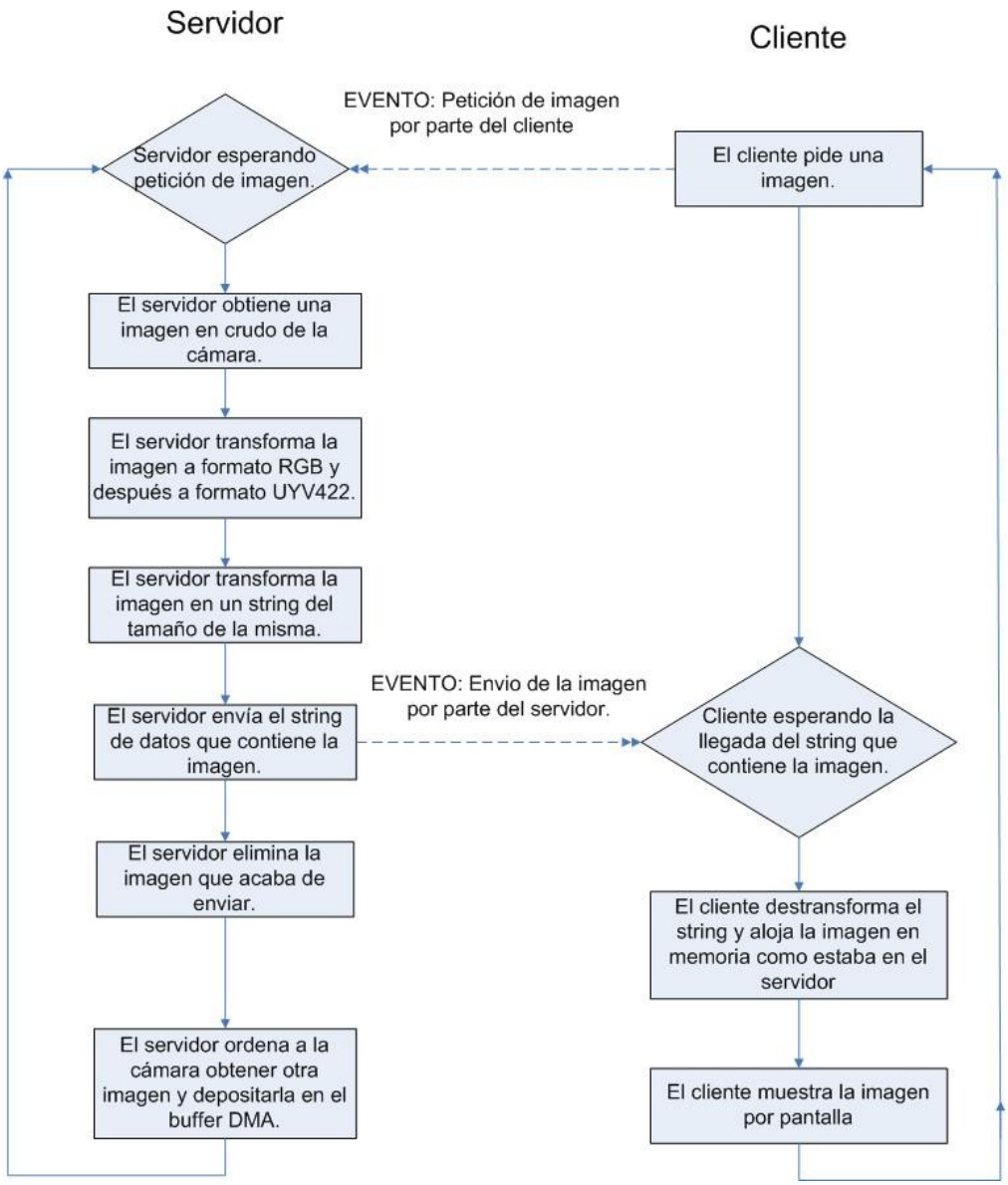


Figura 4.10 Diagrama de flujo de transmisión de datos

4.9 Proceso de captura, transformación, envío y visión en pantalla de imágenes

Para terminar este capítulo, describiremos cada uno de los pasos dados para que finalmente podamos mostrar en el servidor las imágenes obtenidas desde la cámara situada en Higgs. Al final de este apartado, veremos un diagrama de flujo del proceso entero, para un visionado completo de todo lo que hemos explicado anteriormente.

4.1.1 Pasos del programa servidor

1º Definir las librerías a usar y reservar memoria para el almacenaje de las imágenes

2º Definir las funciones a usar, especialmente aquella que obtiene la imagen y la envía a través de la red wifi. Esta función puede ser invocada de forma remota.

3º Configurar el ORB y la tecnología CORBA.

4º Verificar la cámara y configurarla: el modo de disparo, el número de imágenes por segundo, la resolución de las imágenes a capturar (640x480 en este caso).

5º Iniciar el ORB y ponerse en espera para la petición de imágenes.

6º Cuando el cliente pide una imagen se pone en marcha el bucle de envío de imágenes: El servidor coge una imagen del buffer anterior y la aloja en memoria.

7º Cogemos la imagen alojada en memoria, en crudo (raw) entrelazada de la cámara firewire. Esta imagen es tratada desde el cliente.

8º Desentrelazamos la imagen de la cámara firewire, a partir de entonces tenemos dos imágenes, la derecha y la izquierda.

9º Transformamos las dos imágenes de crudo a RGB por medio de la codificación de bayer.

10º Comprimimos las dos imágenes de RGB a YUV 422 con un algoritmo matemático de transformación.

11º El servidor transforma los bytes que forman las imágenes y la traslada a un string con una longitud de 640 x 480 x 2 x 2 (el primer dos por el formato uyv422 usado, el segundo 2 por ser dos imágenes) de octetos o chars, sabemos que esa imagen está en formato uyv422.

12º El servidor envía el string de octetos que contiene las imágenes.

13º El servidor elimina las imágenes enviadas.

14º El servidor manda a la cámara obtener otra imagen e introducirla en el buffer DMA.

15º El servidor vuelve al punto de espera, explicado en el punto 5, y el bucle se repite hasta que se cierra el programa.

4.1.2 Pasos del programa cliente

1º El programa cliente define las librerías gráficas X11 y reserva espacio en memoria para el aojamiento de imágenes

2º El cliente contacta con el ORB.

3º El cliente entra en un bucle infinito. El cliente pide una imagen al servidor.

4º El cliente espera la llegada de la imagen.

8º El cliente recibe la imagen en forma de string y lo guarda en memoria.

9º El cliente destransforma la imagen sabiendo que está en formato uyv422, y la aloja en memoria como en su estado antes de que se convirtiera en un string de datos.

10º Usando las librerías X11, el servidor muestra la imagen por pantalla (que en realidad son 2).

11º El servidor sigue con su bucle infinito y vuelve al punto número 3, este bucle no termina hasta que no sales de él.

4.1.3 Pasos de la cámara firewire

1º Una vez conectada se pone a la espera de una verificación

2º Una vez se verifica, recibe los comandos de configuración deseada y llena el buffer de imágenes.

3º Una vez configurada, espera ordenes de enviar la primera imagen de la cola del buffer.

4º Cuando el cliente recoge una imagen del buffer de memoria, la cámara la elimina es imagen. Corre las imágenes del buffer un lugar.

5º La cámara se vuelve a poner en espera.

6º El servidor pide que la cámara capture una nueva imagen y que la ponga en al final del buffer.

7º La cámara se pone en espera, volviendo al punto 3

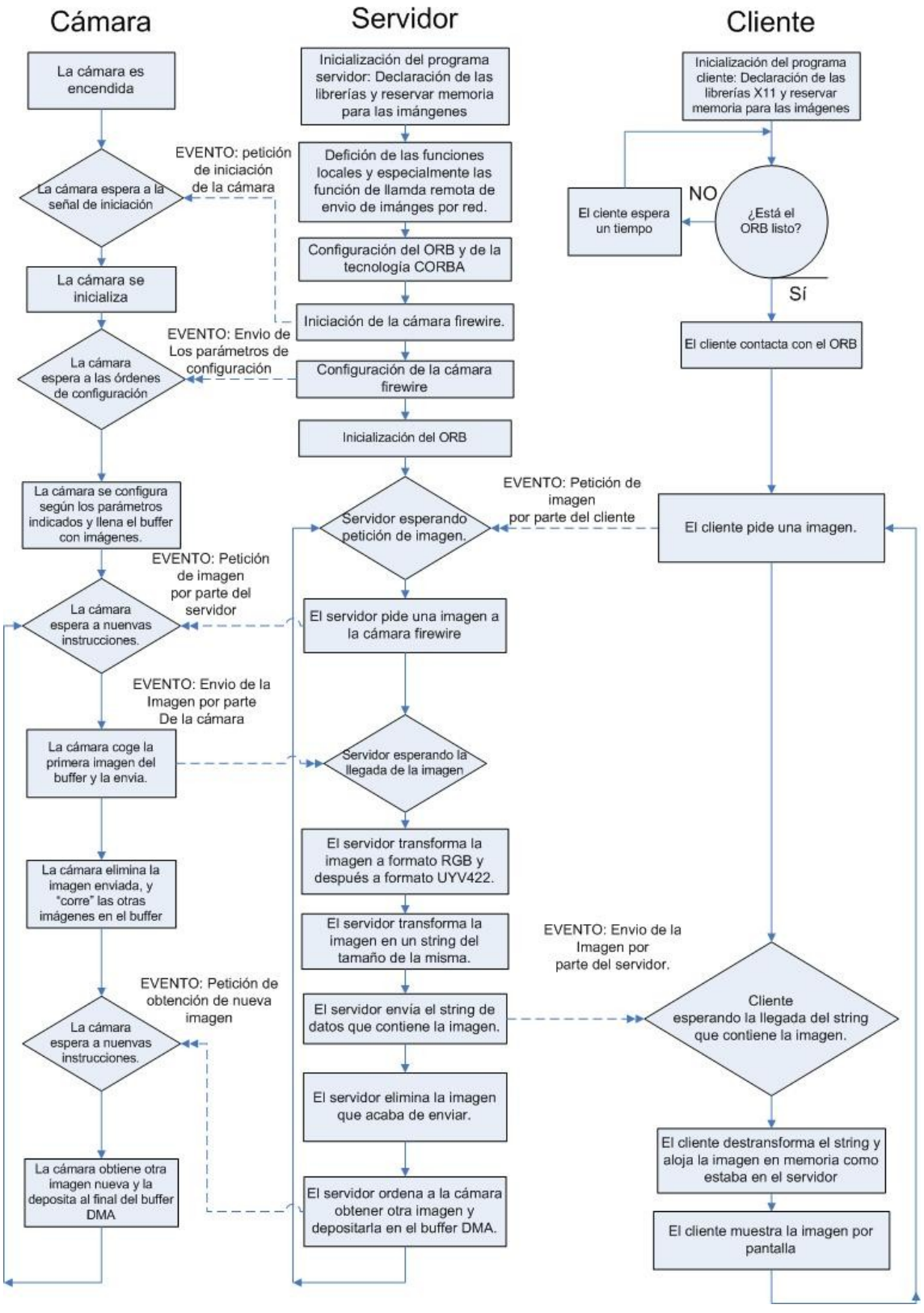


Figura 4.11: Diagrama de flujo de sistema de visión de imágenes completo.

Capítulo 5

Dibujado del mapa robótico

En este capítulo veremos cómo hemos construido el mapa que dibuja el robot mientras se está moviendo. El mapa se representará en pantalla usando las librerías gráficas AWT, definiremos el tipo de mapa que construiremos y como se va dibujando en pantalla.

5.1 Como se construye el mapa

El proceso básico de construcción del mapa, se basa en el dibujado dinámico de puntos que el Higgs nos va mandando a través de red mientras explora el entorno. El sistema se puede englobar en 3 elementos:

1º Los sensores del robot, que serán los “ojos” de lo que ve el mapa mientras explora. Gracias a la odometría podemos obtener los puntos de los obtáculos y paredes que se va encontrando la plataforma robótica móvil.

2º El ordenador de abordo, que hace de servidor y está dentro de Higgs. Gracias a la técnica de SLAM, podemos saber dónde está el robot en cada momento, y obtener datos coherentes de los sensores. Éste ordenador manda los datos obtenidos a través de red. En este proyecto no se habla nada de SLAM debido a que esa parte está siendo desarrollada por un miembro del mismo grupo.

3º El PC cliente que pide los datos al servidor y muestra a través de las librerías gráficas AWT el mapa en pantalla.

Este documento solo tratará con el parte del PC cliente, mostrando cual es la metodología seguida y los pasos dados para conseguir dibujar el mapa en pantalla. El programa cliente sigue estos pasos:

1º El programa define las librerías AWT y dibuja la ventana principal donde se va a dibujar el mapa.

2º El programa contacta con el ORB del servidor. Si no es capaz de contactar el programa se termina, imprimiendo una excepción por pantalla.

3º Si el cliente es capaz de contactar con el ORB del servidor, define un temporizador que se activa cada 0.3 segundos.

4º Cuando se activa el temporizador, el cliente pide al servidor la posición del robot y la su dirección, los puntos de los obstáculos y la visión del láser de barrido SICK.

5º El servidor cliente dibuja en pantalla una representación de la posición del robot mirando en la dirección real, junto a los puntos de los obstáculos y la visión del láser.

6º Cuando pasan 0.3 segundos, el proceso se repite empezando por el punto 4.

Nota: El sistema coge como referencia como punto (0,0) en cartesianas su punto de partida, y su dirección inicial como el ángulo de 0 grados.

En la siguiente página se muestra un diagrama de los pasos seguidos por el programa para mostrar el mapa.

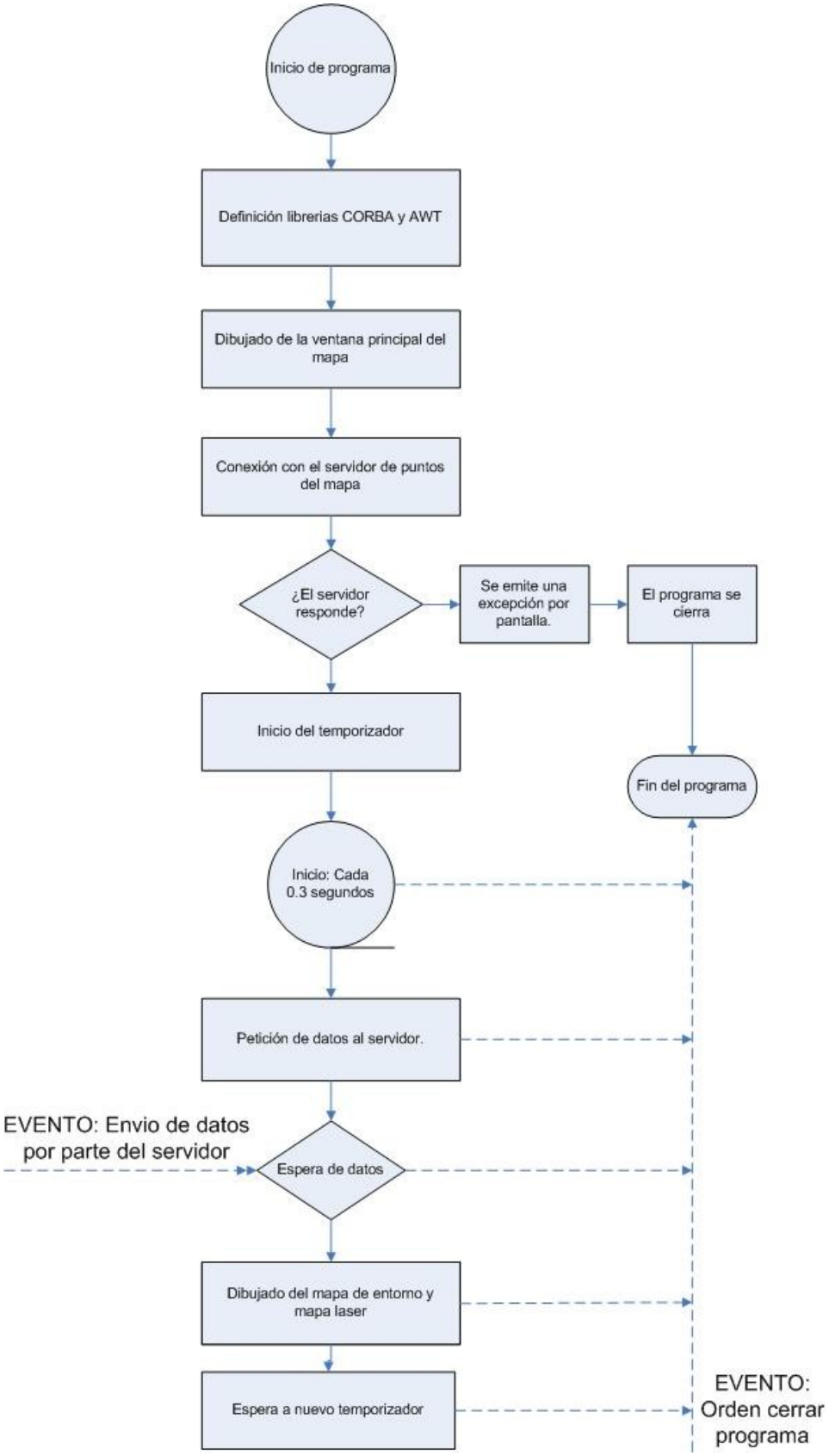


Figura 5.1: Digrama de flujo de dibujado de mapas

5.2 Mapas en robótica

Una de las habilidades principales en robótica es el movimiento. Es necesario tener una representación del entorno para moverse adecuadamente y para ello se construyen los mapas. Los mapas son una representación del entorno. Es necesario definir y obtener las propiedades del entorno: ocupación (para no chocar contra los obstáculos), color… Los humanos también utilizamos mapas.

¿Para qué se utiliza un mapa? navegación, por ejemplo comportamiento ir-a-punto Históricamente introducidos a mano, recientemente construcción autom ´atica desde los sensores.

5.2.1 Tipos de mapas

Cuanto mayor sea la complejidad del mapa a emplear, mayor será la complejidad computacional de su construcción, de la localización y de la navegación.

La precisión del mapa vendrá condicionada en cada caso por la precisión que se necesite en el movimiento del robot y por la precisión de los sensores de que se disponga.

Se establecen tres niveles posibles de representación en la definición de un mapa: geométrico, topológico y semántico. Una representación completa del entorno debería incorporarlos todos pero con frecuencia las soluciones halladas se centran en uno sólo de ellos y, a lo sumo, incluyen uno o los dos restantes como mera información extra que pueda mejorar la navegación [ref 18].

El nivel métrico consiste en representar las coordenadas y propiedades de los objetos del mapa. Este modelo puede a su vez ser geométrico, en el que se representan elementos discretos del entorno de modo que almacenan su parametrización geométrica, o discretizado, en el que la ocupación del entorno se analiza mediante una división del mismo. Así, los mapas métricos geométricos representan objetos con determinadas características geométricas y diferentes grados de complejidad dependiendo de las capacidades sensoriales y de extracción de información.

Este tipo de mapas es ampliamente utilizado en entornos estructurados debido principalmente a la facilidad de visualización que ofrecen y la compacidad que presentan. Otra ventaja fundamental es la filtración de los objetos dinámicos al hacerse la extracción previa de características del entorno. Los sensores necesarios para

construir estos mapas no pueden generar mucho ruido, puesto que han de permitir distinguir los diferentes elementos del entorno.

Otro inconveniente a resaltar es su incapacidad para proporcionar un modelo completo del espacio que rodea al robot. Los puntos que no se identifican como características geométricas del mundo real son eliminados, con lo que para ganar en robustez y compacidad se pierde información de los sensores.

Esta limitación afecta a tareas como la planificación de trayectorias y la exploración de entornos, reduciendo consiguientemente la utilidad de estos mapas en la navegación de robots móviles.

Hay muchos tipos de mapas.

- Locales vs globales.

  - **Global**: representa toda el área de movimiento del robot.
  - **Local**: sólo el entorno próximo a la ubicación actual.

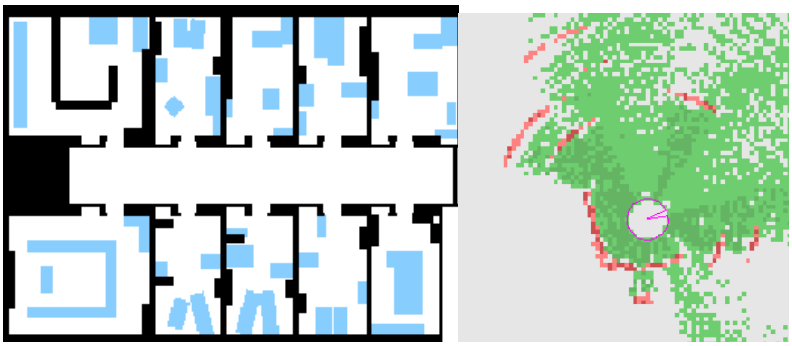


Figura 5.2: Mapa local (izquierda) y mapa global (derecha)

- Topológicos vs métricos.

  - **Mapas topológicos**: tiene Grafos cíclicos con nodos y arcos. Nodos son lugares relevantes y arcos los pasajes entre nodos. No se pueden inferir distancias precisas. En este tipo de mapas se puede planificar trayectoria.

  - **Mapas métricos**: Se pueden inferir distancias y ángulos. Pero Necesitan un sistema de coordenadas.

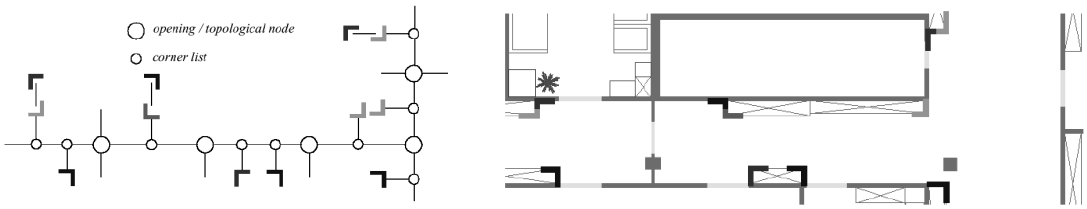


Figura 5.3: Mapa topológico (izquierda) y mapa métrico (derecha)

- Rejilla vs elementos geométricos.

  - **Rejilla**: No hay primitivas, sólo celdillas. Las observaciones sensoriales se relacionan con las celdillas. El estado de cada celdilla se estima continuamente.

  - **Mapas de elementos geométricos**: Usa como puntos básico primitivas de percepción: esquinas, segmentos, etc. Las observaciones sensoriales se relacionan con las primitivas. La posición de los elementos se estima continuamente.

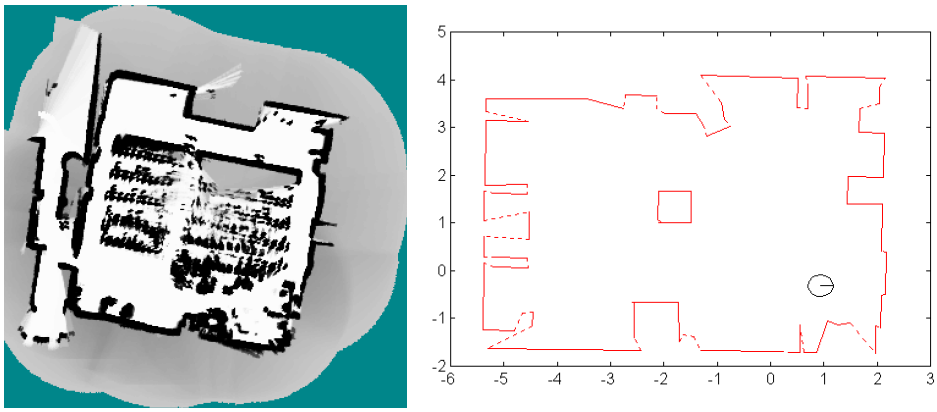


Figura 5.4: Mapa de rejilla (izquierda) y mapa de elementos geométricos (derecha)

- Estáticos vs dinámicos.
- Bidimensionales vs 3D.

Según estas indicaciones nuestro mapa será:

a) Local: La plataforma robótica no está pensada para explorar areas enormes
b) Métrico: Usamos nodos para guiarnos y tenemos el origen de coordenadas en el punto de inicio del robot.

c) Es un mapa de elementos geométricos: Usamos esquinas y las unimos para generar el mapa.
d) Dinámico: Gracias a las librerías AWT, el estado del mapa es variable con el tiempo y cambia con el entorno.
e) Bidimensional: Un mapa tridimensional no sería tan intuitivo para el usuario como el bidimensional.

5.2.2 Tipo de mapa a dibujar

El método utilizado para construir el mapa del entorno es el siguiente: El robot parte de una posición inicial, ubicado en una posición que se tomará como punto de inicio y con una orientación dada aleatoria.

Una vez que el robot inicia la tarea de exploración, se lleva un control odométrico de sus movimientos, lo que significa que por medio de sensores internos del robot se puede ontabilizar el número de revoluciones de las ruedas y el sentido de éstas. Con base en esta información se calcula que píxeles deben iluminarse en la matriz de píxeles que rerpresentará el mapa del entorno.

La tarea de exploración puede concluir en cualquier momento que desee el usuario, cuando tengo información suficiente del ambiente explorado a criterio del operador humano.

Al realizarse diversas pruebas en tareas de exploración del ambiente, se observó que el mapa resultante tenía pequeñas variaciones con respecto al amibiente real, esto es debido a pequeños fallos odométricos y deslizamiento de ruedas.

5.3 La clase Graphics de AWT.

La clase Graphics es el sistema básico de todas las operaciones gráficas de las librerías AWT por lo que parece fundamental que se entienda bien su concepto antes de ponernos a tratar gráficos o imágenes.

Esta clase abstracta tiene la misión de conformar el contexto gráfico y la de proporcionar los métodos necesarios para poder dibujar en la pantalla.

5.3.1 El contexto gráfico.

Consiste en encapsular la información necesaria para las operaciones de renderizado básicas, es decir, toda la información que afecte a las operaciones de dibujado conforma el **contexto gráfico** y la salida a la ventana en la que se va a dibujar tiene que pasar por este contexto.

Incluye las siguientes **propiedades** más importantes:

· El objeto Component sobre el que se va a dibujar. Puede ser la ventana principal de un applet, de un frame o de una ventana hija.

· Un origen de coordenadas. Se especifican en pixels y comienzan en la esquina superior izquierda (0,0). No obstante, recordar que las coordenadas son relativas a la ventana en la que estemos trabajando ya que la posición real que ocupa una ventana madre en la pantalla no tiene porqué ser la misma que la de una ventana hija que despleguemos en algún momento.

· El "clipping". Esto no es más que una ventana (un rectángulo invisible) sobre el que vamos a poder pintar o renderizar nuestras imágenes. Todo lo que quede fuera será cortado (clipped).

· Los colores del fondo y el primer plano (backgroun y foreground). Por defecto son negro y blanco respectivamente.

· La fuente de texto. Por defecto es la fuente Dialog, estilo plano y 12 pulgadas.

Pero para dibujar, nuestro programa necesita un contexto gráfico válido que vendrá dado a través de una instancia o ejemplar de la clase Graphics. El problema es que al ser una clase abstracta, ésta no se puede instanciar directamente por lo que lo que habrá que hacer es pasarle el contexto al programa a través de los métodos *paint()* o *update()* o bien, que el programa obtenga el contexto mediante el método *getGraphics()* de la clase *Component()* [ref 19].

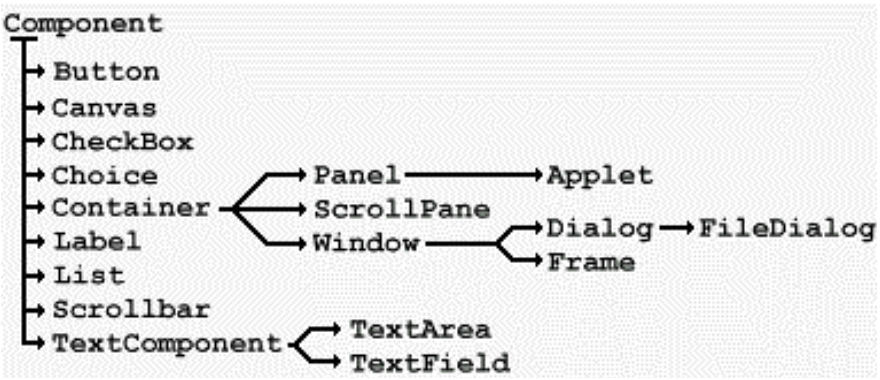


Figura 5.5: Jerarquía de la clase Component

Graphics nos debe proporcionar todos los métodos necesarios para poder dibujar formas básicas, texto con todas las fuentes que tiene el sistema así como permitirnos cargar imágenes.

Todo lo que portemos a la ventana del mapa se hará a través de estas funciones.

5.3.2 paint(), repaint() y update()

De entre todos los métodos, hay 3 de ellos esenciales y básicos para "pintar" con AWT: *paint(), repaint() y update()*. Estos 3 métodos son los encargados de mostrar los gráficos. Java nos proporciona una versión por defecto en la clase *Component* por lo que tendremos que sobrecargar *update()* y *paint()* para que nuestro programa pinte lo que deseemos y como queramos.

Cuando el usuario llama al método *repaint()* de un componente, el AWT (recondando que Swing es "*algo así*" como una extensión de AWT) llama al método *update()* de ese componente, que por defecto llama al método *paint()*.

Paint().

Su sintáxis es:

```
public void paint(Graphics g)
```

Es el responsable de dibujar las imágenes. Normalmente es el sistema operativo el que lo llama al dibujar la ventana por primera vez, volviéndolo a llamar cada vez que entiende que la ventana o una parte de ella debe ser redibujada (por ejemplo, por haber estado tapada por una ventana y quedar de nuevo a la vista). Esta llamada la hace a través del método update().

Update().

Su sintáxis es:

```
public void update(Graphics g)
```

Este método se llama en respuesta a una solicitud por parte del método repaint(), por el AWT o por el programador cuando ha realizado un cambio en la ventana de mapa y necesita que se dibuje de nuevo.

La implementación por defecto trabaja "borrando" la escena actual mediante un repintado del mismo color que el de background y luego llama al método paint() para pintar la nueva escena. Esta técnica provoca el molesto efecto llamado parpadeo (flickering) el cual se puede solucionar redefiniendo el método update() de manera que solo actualice la zona de pintado donde se han hecho modificaciones, o bien, como veremos en un artículo posterior, empleando la técnica del doble buffer.

Repaint().

Su sintáxis es:

```
public void repaint()

public void repaint(long tm)

public void repaint(int x, int y, int w, int h)

public void repaint(long tm, int x, int y, int w, int h)
```

Este método solicita que un componente sea repintado llamando lo más pronto posible al método *update()* del componente.

De las posibles sintáxis que disponemos se extrae que el repintado puede ser inmediato, transcurrido un tiempo (mseg) o repintar solo la zona rectangular especificada en los parámetros. Esta última técnica es de gran utilidad cuando el repintado de la escena consume mucho tiempo, así solo repintamos la franja que se haya modificado.

Para reducir el tiempo que se tarda en repintar, el AWT toma dos técnicas:

a) Repintar solo aquellos componentes que necesitan repintarse, bien porque se han cubierto por otro componente o bien porque el programador así lo solicitó.

b) Si un componente se tapa parcialmente por otro, cuando se le devuelve a primer plano, solo se repinta la zona cubierta.

5.4 Temporizadores

Ya que el mapa que vamos a programar, va a ser dinámico, es necesario un refresco constante de los cambios producidos por la odometría, por los puntos nuevos del entorno mientras se explora y los cambios en el entorno ya explorado. Para ello cada cierto tiempo el cliente (el PC que muestra el mapa en pantalla) pide los nuevos puntos obtenidos por la plataforma robótica nueva, y esta se los manda a través de red TCP/IP usando el middleware CORBA. Este evento se produce una vez cada 0.3 segundos y es iniciado por un temporizador de la clase *javax.swing.Timer*.

Puede pensar en Timer como un hilo único que Swing provee convenientemente para lanzar ActionEvents a intervalos. Los ActionListeners se pueden registrar para que reciban estos eventos tal y como los registramos en botones, o en otros componentes.

En primer lugar configuramos un ActionListener para que reciba ActionEvents. Entonces construimos un nuevo Timer pasando el tiempo entre eventos en milisegundos, el retraso (*delay*), y un ActionListener al que enviárselos. Finalmente llamamos al método *start()* de Timer para activarlo. Como no hay ejecutándose una GUI el programa saldrá inmediatamente, por lo tanto ponemos un bucle para permitir al Timer que siga con su trabajo indefinidamente que por defecto equivale al tiempo que se le pasa al constructor. Si queremos que el Timer lance un evento justo cuando se inicia debemos poner el retraso inicial a 0 usando su método *setInitialDelay().*

En cualquier momento podemos llamar a *stop()* para detener el Timer y *start()* para reiniciarlo (*start()* no hace nada si ya se está ejecutando). Podemos llamar a *restart()* en un Timer para que empiece de nuevo todo el proceso. El método *restart()* es sólo una abreviatura para llamar a *stop()* y *start()* secuencialmente.

Podemos poner el retraso de un Timer usando su método *setDelay()* y decirle si debe repetirse o no usando el método *setRepeats().* Una vez que hemos hecho que un Timer no se repita, sólo lanzará una acción cuando se inicie (o si ya se está ejecutando), y entonces se detendrá.

Los Timers son fáciles de usar y a menudo se pueden utilizar como una herramienta conveniente para construir nuestros propios hilos. Sin embargo, hay mucho más por detrás que merece un poco de atención. Antes de que veamos a fondo como trabajan los

Timers, echaremos un vistazo al servicio de mapeo de clases de Swing (SecurityContext-to-AppContext) para applets, así como a la forma en la que las aplicaciones manejan sus clases de servicio (también usando AppContext). Aunque nos referiremos de vez en cuando a AppContext, no significa que sea necesario para entender los detalles.

5.5 Interpretación de los datos

Los datos que el programa cliente recibe del servidor, tiene un significado que hay que conocer e interpretar, para dibujar en pantalla los dos mapas que queremos representar. Para ello hay que saber de qué forma nos llegan esos datos al cliente:

1º Para los puntos del laser, el cliente recibe 2 vectores de datos (también llamados String), uno con las distancias de los obstáculos respecto al robot en línea recta, y otro vector con las coordenadas angulares de estos obstáculos respecto la dirección hacia donde mira el robot. Es decir, nos está dando puntos en coordenadas muy parecidas a las polares. Según lo explicado, las distancias varían desde 0 metros hasta 20 metros, que es la distancia máxima de visión del laser. Mientras los ángulos varían desde - 90º hasta 90º, barriendo así los 180º máximos del laser. Normalmente los vectores tendrán una longitud de 180 valores, pero en algunos casos pueden ser menos

2º Para los puntos vistos del mapa del entorno, obtendremos un vector de 10000 valores que poseerán las coordenadas cartesianas de cada una de las esquinas: los valores impares son las coordenadas X y los valores pares son las coordenadas Y. A la hora de unir esquinas, dos puntos contiguos son esquinas unidas por una pared u obstáculo, de tal manera que cada cuatro valores del vector, tenemos una pared.

3º Hay que recordar también, que el cliente también nos brinda la posición del robot en cada momento y la dirección hacia la que está mirando. Es importante para determinar donde están los obstáculos.

A la hora de representar los datos del láser en pantalla, se necesitan ciertas transformaciones matemáticas relativamente simples que hay que conocer para un correcto visionado de los puntos obstáculos que ve en cada momento el láser del robot. De lo expuesto anteriormente obtenemos una serie de datos que es necesario definir para entender estas transformaciones:

1º Un ángulo θ, que el ángulo respecto a la horizontal de la dirección hacia donde está mirando el robot.

2º Un ángulo β que es el ángulo que hace el obstáculo respecto a la "horizontal" del robot.

3º Un ángulo α, que es la coordenada polar deseada para la representación del punto, es decir, es el ángulo respecto a la horizontal que hace la recta unión del origen de coordenada con el punto a tratar.

4º Una distancia D del punto obstáculo al origen de coordenadas, que en este caso es el propio robot.

5º Una posición Xr e Yr en coordenadas cartesianas de la posición actual del robot.

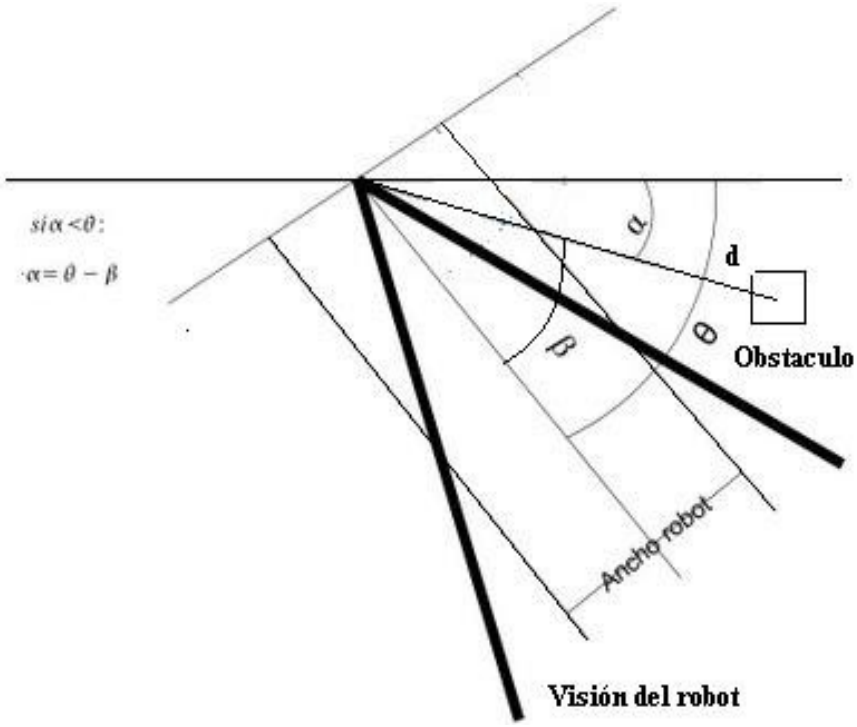


Figura 5.6: Datos geométricos de los puntos del entorno.

Entonces, la posición en coordenadas cartesianas de los puntos de los obstáculos Xt e Yt, se podría escribir como:

$$Xt = Dist * \cos(\theta - \beta)$$

$$Yt = Dist * \sin(\theta - \beta)$$

Con estas dos fórmulas ya tenemos las posiciones cartesianas de los puntos del laser SICK. En cambio para dibujar el mapa del entorno, los obstáculos son más difíciles de dibujar, y para ellos tenemos que hacer otras transformaciones matemáticas un poco más complejas, siendo Xe e Ye las posiciones en cartesianas de los obstáculos en el mapa de entorno. Las transformaciones matemáticas, serían:

$$Xe = Xr + Dist * \cos(\theta - \beta)$$

$$Ye = Yr + Dist * \sin(\theta - \beta)$$

5.6 Aspecto de los mapas

Los dos mapas estarán situados uno al lado del otro en dos porciones de ventanas rectangulares de 600 x 600 píxeles cada una, haciendo en total una ventana en pantalla de 1200x600 píxeles por pantalla. Además estará situado dentro de un marco con título y menú en la esquina superior derecha donde estarán los símbolos clásicos de minimizar, restaurar y cerrar. Estos botones hacen las funciones pertinentes sin tener que explicarlas, ya que su significado es trivial. Los dos mapas tienen un fondo gris que representa aquellos puntos que no se han explorados, y que no sabemos que hay, hasta que no lo exploremos, mientras que el robot está representado por un triángulo isósceles, cuyo vértice mayor es la dirección hacia donde está mirando el robot.

Los obstáculos están representados por puntos azules, y los puntos blancos son aquellos puntos que están libres de obstáculos. Evidentemente, hay que evitar que el triángulo rojo (que representa el robot) nunca toque los puntos azules (obstáculos) y se mantenga siempre en los puntos blancos (puntos libres).

La escala elegida para representar los puntos es de 1 metro por cada 20 píxeles para poder ver de forma nítida los diferentes puntos del mapa. En el mapa de entorno, cuando el robot se acerca al borde del mapa, este se centra otra vez en el robot como en el principio, y los puntos fijos, tanto blancos como azules, se desplazan en el mapa la distancia movida por el robot. Si algún punto queda fuera de los límites, no se mostrará en el mapa, pero será visible otra vez si el robot entra en una zona donde haya suficiente campo de visión.

Capítulo 6

Interfaz de control

En éste capítulo veremos en qué consiste la última parte del software que falta por explicar para la completa teleoperarión de la plataforma robótica móvil y como se ha desarrollada hasta cumplir todos los requisitos propuestos. Además se ha intentado construir de tal manera que sea agradable para la vista y que todos los datos mostrados sean claros y precisos para el teleoperador.

6.1 Funciones del programa

Para poder entender este apartado de la mejor forma posible, es necesario saber cómo trabaja el programa, y como funciona. El programa cliente tiene como función la muestra en pantalla de los datos más importantes del estado y uso de la plataforma robótica:

1º) Capacidad para encender y apagar tanto el motor de Higgs como sus sensores sonars, ya que si no los necesitamos los apagamos para ahorrar batería.

2º) Monitorizar el estado del robot.

3º) Control del movimiento a través de las teclas de dirección del teclado

4º) Variación de las velocidades del rotbot

5º) Monitorizado de la posición y ángulo de la plataforma robótica

6º) Lectura en pantalla de los datos que recoge los sonars en cada momento.

Al igual que en todas las demás partes del sistema, existe un cliente y un servidor, pero ahora tanto el servidor hará veces de cliente y el cliente hará veces de servidor, ya que el software interfaz interactúa con el software de control dentro de Higgs y viceversa, por ejemplo, la velocidad de traslación a la que debe moverse Higgs parte de la interfaz de control y las lecturas del laser son enviadas por el software de control de Higgs hasta la interfaz.

De todas formas se ha intentado seguir unas pautas de garantías, para obtener un resultado de una interfaz de fácil uso, elegante y muy intuitiva:Nuestro reto es diseñar una GUI que responda a estos tres objetivos:

1. **Ser una interfaz fácil de aprender y utilizar:** los usuarios menos familiarizados con sistemas de recuperación de imágenes mediante QbE habrán de ser capaces de manejar sin problemas la aplicación después de una breve sesión de demostración.

2. **Cumplir las necesidades de interacción entre usuario y sistema de búsqueda:** la interfaz es la pieza clave entre el diálogo usuario – máquina, y por tanto, deberá ser capaz de transmitir la información necesaria para que ambos interlocutores entiendan los datos que se intercambian.

3. **Ofrecer los elementos apropiados para realizar las tareas requeridas:** la presentación de la información, tanto a la hora de recoger los datos de entrada como de mostrar los datos de salida, debe ser adecuada y facilitar el trabajo en todo momento al usuario.

La consecución de estos tres objetivos ya nos garantiza una buena herramienta en términos de usabilidad, pero además, queremos que nuestra interfaz sea estéticamente atractiva, que agrade al usuario y que posea un estilo que la identifique y la diferencie de otros productos similares.

6.2 Java Swing para el diseño de interfaces gráficas

Swing es un extenso conjunto de componentes que van desde los más simples, como etiquetas, hasta los más complejos, como tablas, árboles, y documentos de texto con estilo. Casi todos los componentes

Swing descienden de un mismo padre llamado JComponent que desciende de la clase de AWT Container. Es por ello que Swing es más una capa encima de AWT que una sustitución del mismo.

Si la compara con la jerarquía de Component de swing notará que para cada componente AWT hay otro equivalente en Swing que empieza con "J". La única excepción es la clase de AWT Canvas, que se puede reemplazar con JComponent,

JLabel, o JPanel. Asimismo se percatará de que existen algunas clases Swing sin su correspondiente homólogo.

Los pasos seguidos para la creación de una interfaz gráfica de usuario, han sido los siguientes [ref 20]:

1º) El diseñó y composición de la apariencia de la interfaz g´rafica de la aplicación. Normalmente, esto implica la elección de una ventana principal (contenedor), la elección de contenedores para la jerarquía de componentes, y la elección de los administradores de disposición para cada uno de los contenedores y los lelmentos gráficos de interacción (componentes primitivos).

2º) Se escribe el código que crea todos los componentes, crea la jerarquía de componentes contenedores y componentes primitivos y d la apariencia a la interfaz.

3º) Escribir el código que proporciona el comportamiento de dicha interfaz como respuesta a las interacciones de los usuarios para gestionar los eventos de interés para la aplicación de control de Higgs.

4º) La visualización de la interfaz gráfica. Una vez visualizada la interfaz, la aplicación queda a la espera de las interacciones del usuario que provocarán la ejecución de los gestores correspondientes. Además cada 0,3 milisegundos, se refrescan los datos que se muestran en pantalla.

6.2.1 Orden Z

A los componentes Swing se les denomina *ligeros* mientras que a los componentes AWT se les denominados *pesados*. La diferencia entre componentes ligeros y pesados es su *orden*: la noción de profundidad. Cada componente pesado ocupa su propia capa de orden Z. Todos los componentes ligeros se encuentran dentro de componentes pesados y mantienen su propio esquema de capas definido por Swing. Cuando colocamos un componente pesado dentro de un contenedor que también lo es, se superpondrá por definición a todos los componentes ligeros del contenedor.

Lo que esto significa es que debemos intentar evitar el uso de componentes ligeros y pesados en un mismo contenedor siempre que sea posible. Esto no significa que no podamos mezclar nunca con éxito componentes AWT y Swing, sólo que tenemos que tener cuidado y saber qué situaciones son seguras y cuáles no. Puesto que probablemente no seremos capaces de prescindir completamente del uso de

componentes pesados en un breve espacio de tiempo, debemos encontrar formas de que las dos tecnologías trabajen juntas de manera aceptable.

La regla más importante a seguir es que no deberíamos colocar componentes pesados dentro de contenedores ligeros, que comúnmente soportan hijos que se superponen. Algunos ejemplos de este tipo de contenedores son JInternalFrame, JScrollPane, JLayeredPane, y JDesktopPane.

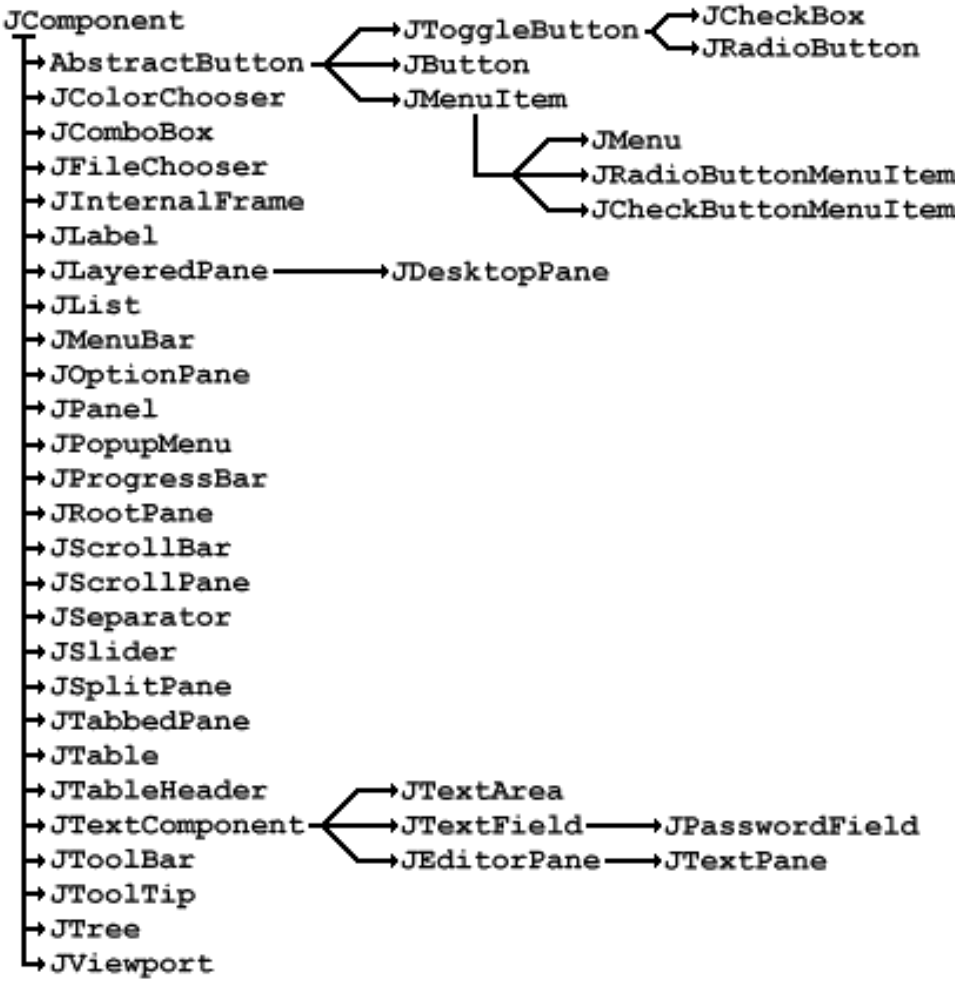


Figura 6.1 Parte de la jerarquía de JComponent de Swing

6.2.2 Vistazo al paquete Swing

javax.swing

Contiene la mayor parte de los componentes básicos de Swing, modelos de componente por defecto, e interfaces. (La mayoría de las clases mostradas en la Figura 1.2 se encuentran en este 4 paquete.)

javax.swing.border

Clases e interfaces que se usan para definir estilos de bordes específicos. Observe que los bordes pueden ser compartidos por cualquier número de componentes Swing, ya que no son componentes por si mismos.

javax.swing.colorchooser

Clases e interfaces que dan soporte al componente JColorChooser, usado para selección de colores. (Este paquete también contiene alguna clase privada interesante sin documentar.)

javax.swing.event

El paquete contiene todos los oyentes y eventos específicos de Swing. Los componentes Swing también soportan eventos y oyentes definidos en java.awt.event y java.beans.

javax.swing.filechooser

Clases e interfaces que dan soporte al componente JFileChooser, usado para selección de ficheros.

javax.swing.plaf

Contiene el API del comportamiento y aspecto conectable usado para definir componentes de interfaz de usuario personalizados. La mayoría de las clases de este paquete son abstractas. Las implementaciones de look-and-feel, como metal, motif y basic, crean subclases e implementan las clases de este paquete. Éstas están orientadas a desarrolladores que, por una razón u otra, no pueden usar uno de los look-and-feel existentes.

javax.swing.plaf.basic

Consiste en la implementación del Basic look-and-feel, encima del cual se construyen los lookand- feels que provee Swing. Normalmente deberemos usar las clases de este paquete si queremos crear nuestro look-and-feel personal.

javax.swing.plaf.metal

Metal es el look-and-feel por defecto de los componentes Swing. Es el único look-and-feel que viene con Swing y que no está diseñado para ser consistente con una plataforma específica.

javax.swing.plaf.multi

Este es el Multiplexing look-and-feel. No se trata de una implementación normal de look-andfeel ya que no define ni el aspecto ni el comportamiento de ningún componente. Más bien ofrece la capacidad de combinar varios look-and-feels para usarlos simultáneamente. Un ejemplo típico podría ser un look-and-feel de audio combinado con metal o motif. Actualmente

Java 2 no viene con ninguna implementación de multiplexing look-and-feel (de todos modos, se rumorea que el equipo de Swing esta trabajando en un audio look-and-feel mientras escribimos estas líneas).

javax.swing.table

Clases e interfaces para dar soporte al control de JTable. Este componente se usa para manejar datos en forma de hoja de cálculo. Soporta un alto grado de personalización sin requerir mejoras de look-and-feel.

javax.swing.text

Clases e interfaces usadas por los componentes de texto, incluyendo soporte para documentos con o sin estilo, las vistas de estos documentos, resaltado, acciones de editor y personalización del teclado.

javax.swing.tree

Clases e interfaces que dan soporte al componente JTree. Este componente se usa para mostrar y manejar datos que guardan alguna jerarquía. Soporta un alto grado de personalización sin requerir mejoras de look-and-feel.

javax.swing.undo

El paquete undo contiene soporte para implementar y manejar la funcionalidad deshacer/rehacer.

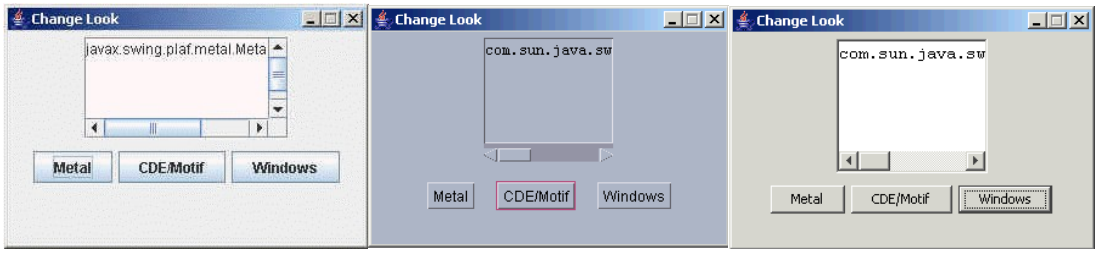


Figura 6.2 Distintos "look-and-feels" de java swing, de izquierda a derecha Metal, Motif y Windows

6.2.3 Manejo y lanzamiento de eventos

Los eventos ocurren en cualquier momento que se pulsa una tecla o un botón del ratón. La forma en la que los componentes reciben y procesan los eventos no ha cambiado desde el JDK1.1. Los componentes

Swing pueden generar diferentes tipos de eventos, incluyendo los de java.awt.event y por supuesto, los de javax.swing.event. Algunos de estos nuevos tipos de eventos de Swing son específicos del componente. Todos los tipos de eventos se representan por un objeto, que como mínimo, identifica la fuente del evento, y a menudo lleva información adicional acerca de la clase específica de evento que se trata, e información acerca del estado de la fuente del evento antes y después de que éste se generase.

Las fuentes de eventos son normalmente componentes o modelos, pero hay también clases de objetos diferentes que pueden generar eventos.

Debemos registrar oyentes en el objeto destino. Un oyente es una implementación de alguna de las clases XXListener (donde XX es un tipo de evento) definidas en los paquetes java.awt.event, java.beans, y javax.swing.event. Como mínimo, siempre hay un método definido en cada interface al que se le pasa el XXEvent correspondiente como parámetro. Las clases que soportan la notificación de XXEvents implementan

generalmente el interface XXListener, y tienen soporte para registrar y cancelar el registro de estos oyentes a través del uso de los métodos addXXListener() y removeXXListener() respectivamente. La mayoría de los destinos de eventos permiten tener registrados cualquier número de oyentes. Igualmente, cualquier instancia de un oyente se puede registrar para recibir eventos de cualquier número de fuentes de éstos. Normalmente, las clases que soportan XXEvents ofrecen métodos fireXX() protegidos (protected) que se usan para construir objetos de eventos y para enviarlos a los manejadores de eventos para su proceso.

La clase javax.swing.event.EventListenerList

EventListenerList es un vector de pares XXEvent/XXListener. JComponent y cada uno de sus descendientes usa una EventListenerList para mantener sus oyentes. Todos los modelos por defecto mantienen también oyentes y una EventListenerList. Cuando se añade un oyente a un componente Swing o a un modelo, la instancia de Class asociada al evento (usada para identificar el tipo de evento) se añade a un vector EventListenerList, seguida del oyente. Como estos pares se guardan en un vector en lugar de en una colección modificable (por eficiencia), se crea un nuevo vector usando el método System.arrayCopy() en cada adición o borrado. Cuando se reciben eventos, se recorre la lista y se envían eventos a todos los oyentes de un tipo adecuado. Como el vector está ordenado de la forma XXEvent, XXListener, YYEvent, YYListener, etc., un oyente correspondiente a un determinador tipo de evento está siempre el siguiente en el vector. Esta estrategia permite unas rutinas de manejo de eventos muy eficientes (ver sección 2.7.7). Para seguridad entre procesos, los métodos para añadir y borrar oyentes de una EventListenerList sincronizan el acceso al vector cuando lo manipulamos.

JComponent define sus EventListenerList como un campo protegido llamado listenerList, así que todas sus subclases lo heredan. Los componentes Swing manejan la mayoría de sus oyentes directamente a través de listenerList.

Hilo de despacho de eventos (Event-dispatching thread)

Todos los eventos se procesan por los oyentes que los reciben dentro del hilo de despacho de eventos

(una instancia de java.awt.EventDispatchThread). Todo el dibujo y posicionamiento de componentes debería llevarse a cabo en este hilo. El hilo de despacho de eventos es de vital importancia en Swing y AWT, y juega un papel principal manteniendo actualizado el estado y la visualización de un componente en una aplicación bajo control.

Asociada con este hilo hay una cola FIFO (primero que entró - primero que sale) de eventos: la cola de eventos del sistema (una instancia de java.awt.EventQueue). Esta cola se rellena, como cualquier cola FIFO, en serie. Cada petición toma su turno para ejecutar el código de manejo de eventos, que puede ser para actualizar las propiedades, el posicionamiento o el repintado de un componente. Todos los eventos se procesan en serie para evitar situaciones tales como modificar el estado de un componente en mitad de un repintado. Sabiendo esto, tenemos que ser cuidadosos de no despachar eventos fuera del hilo de despacho de eventos. Por ejemplo, llamar directamente a un método fireXX() desde un hilo de ejecución separado es inseguro. Tenemos que estar seguros también de que el código de manejo de eventos se puede ejecutar rápidamente. En otro caso toda la cola de eventos del sistema se bloquearía esperando a que terminase el proceso de un evento, el repintado, o el posicionamiento, y nuestra aplicación parecería bloqueada o congelada.

6.2.4 Elementos de Java Swing

En este apartado explicaremos los diferentes elementos que hemos usados para la generación de la interfaz gráfica de control.

6.2.4.1 Contenedores de alto nivel

JFrame

JFrame se emplea para crear la ventana principal de una aplicaión. Es una ventana con marco que incluye los controles habituales de cambio de tamaño y cierr (por emeplo, cerrar, iconizar, maximizar).

JDialog

JDialog es la clase raíz de las ventanas secundarias que implementan cuadros de diálogo en Swing. Estas ventanas depende de una ventana principal, normalmente JFrame, y si la ventana principal se cierra, se iconiza o se desiconiza, las ventanas segudarias hacen la misma operación de forma automática.

JOptionPane

Esta clase se utiliza para crear los tipos de cuadros de diálogos más habituales, como los que permiten pedir un valor, mostrar un mensaje de error o advertencia, solicitar una confirmación…

Estos paneles son modales (es decir bloquean la interacción del usuario con toras ventanas).

6.2.4.2 Contenedores intermedios

JPane

JPane es un contenedor simple de propósito general que sirve para agrupar a otros componentes. Habitualmente se utiliza para agrupar componentes a los que se aplica un gestor de disposición adecuado. Permiten añadirles bordes o personalizar su presentación gráfica.

JMenuBar

Esta clase implementa una barrar de herramientas, formada principalmente por botones con menús despegables, estas barras se sitúan normalmente en la parte superior de la ventana principal haciendo de menú y de apoyo a las otras clases gráficas de botones.

6.2.4.2 Componentes atómicos

Iconos

Un icono es una imagen gráfica de tamaño fijo que habitualmente se utiliza para presentarla dentro de otro componente, como un botón, o una etiquita. Puede crearse a partir de imágenes .gif o .jpg.

JLabel

Esta clase implementa una etiqueta que puede contener una cadena de texto, un icono o ambos. En una etiqueta se puede espificar donde aparece su contenido indicando el alineamiento vertical y horizontal. Por defecto, las etiquetas se muestran centradas verticalmente, y si sólo tienen texto, ajustadas a la izquierda. Si sólo tienen una imagen gráfica por defecto se muestran centradas tanto vertical como horizontalmente.

JButton

Esta clase implementa la forma más habitual de botón gráfico de interacción que sirve para ejecutar una acción haciendo clic sobre él. También se puede activar mediante el teclado se se le ha asociado una combinación de teclas.

JRadioButton

Esta clase implementa los botones de radio o de opción que son una especialización de un botón con estado o conmutador y se caracterizan porque en un grupo de botones de radio sólo uno de ellos puede estar seleccionado. Para crear un grupo de botones de radio hay que añadirlos en un objeto ButtonGroup, que no tiene representación gráfica.

6.3 Java Beans

Los **JavaBeans** son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.

Se usan para encapsular varios objetos en un único objeto (la vaina), para hacer uso de un sólo objeto en lugar de varios más simples.

La especificación de JavaBeans de Sun Microsystems los define como "componentes de software reutilizables que se puedan manipular visualmente en una herramienta de construcción".

A pesar de haber muchas semejanzas, los JavaBeans no deben confundirse con los Enterprise JavaBeans (EJB), una tecnología de componentes del lado servidor que es parte de Java EE.

Aunque los beans individuales pueden variar ampliamente en funcionalidad desde los más simples a los más complejos, todos ellos comparten las siguientes características:

- **Introspection**: Permite analizar a la herramienta de programación o IDE como trabaja el bean
- **Customization**: El programador puede alterar la apariencia y la conducta del bean.
- **Events**: Informa al IDE de los sucesos que puede generar en respuesta a las acciones del usuario o del sistema, y también los sucesos que puede manejar.
- **Properties**: Permite cambiar los valores de las propiedades del bean para personalizarlo (customization).
- **Persistence**: Se puede guardar el estado de los beans que han sido personalizados por el programador, cambiando los valores de sus propiedades.

En general, un bean es una clase que obedece ciertas reglas:

- Un bean tiene que tener un constructor por defecto (sin argumentos)

- Un bean tiene que tener persistencia, es decir, implementar el interface *Serializable*.

- Un bean tiene que tener introspección (**instrospection**). Los IDE reconocen ciertas pautas de diseño, nombres de las funciones miembros o métodos y definiciones de las clases, que permiten a la herramienta de programación mirar dentro del bean y conocer sus propiedades y su conducta.

Una propiedad es un atributo del JavaBean que afecta a su apariencia o a su conducta. Por ejemplo, un botón puede tener las siguintes propiedades: el tamaño, la posición, el título, el color de fondo, el color del texto, si está o no habilitado, etc.

Las propiedades de un bean pueden examinarse y modificarse mediante métodos o funciones miembro, que acceden a dicha propiedad, y pueden ser de dos tipos:

- **get**ter method: lee el valor de la propiedad
- **set**ter method: cambia el valor de la propiedad.

6.4 Diseño de la interfaz

Una vez expuesto lo que queremos hacer y como lo vamos hacer, lo primero es poner los elementos que se han usado para la interfaz:

a) Un JFrame que hace de contenedor superior, tiene un tamaño fijo y posee el titulo de "*GUI for Higgs*".

Figura 6.3: Contenedor superior de la interfaz gráfica

b) Un JPanel dentro del JFrame anterior que hace de soporte de la mayoría de los objetos incluidos en la interfaz.

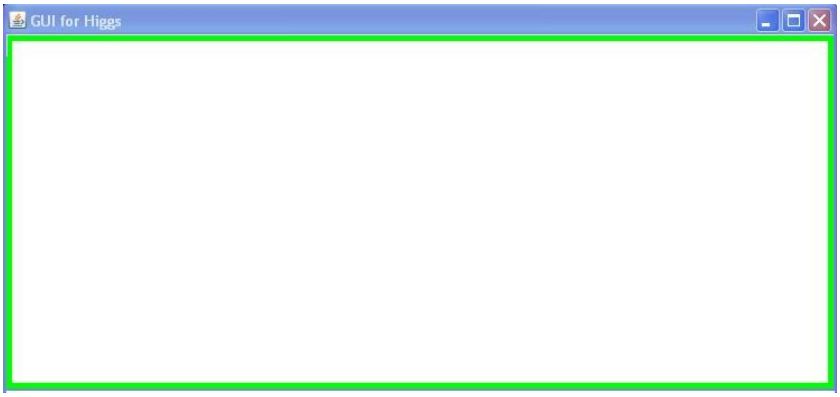


Figura 6.4 JPane de la interfaz con borde verde

C) Un conjunto de elementos que están dentro del JPane anterior, estos elementos se son:

- 47 JLabel (*etiqueta*) que sirven tanto como de iconos como de etiquetas de control.

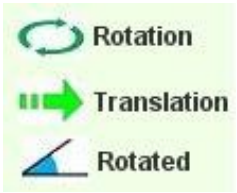


Figura 6.5: Ejemplo de etiquetas que funcionan como iconos y como texto

- 24 JTextFeld (*campo de texto*) que sirven para mostrar los datos que recibe de robot.

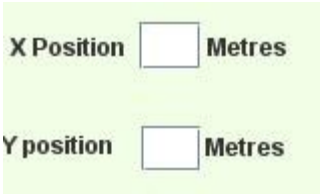


Figura 6.6: Ejemplos de campos de texto para mostrar la posiciónes cartesianas de Higgs

- Una JProgressBar (*Barra de progreso*) que sirve para mostrar la batería que le queda a Higgs de forma porcentual.

Figura 6.7 Barra de progreso de la batería

- Dos JSlider (*Deslizadores*) que sirven para ajustar la velocidad de traslación y rotación a la que va Higgs cuando pulsas los botones de dirección. Los valores van de 0% a 100% de la velocidad máxima.

Figura 6.8 Deslizadores de la interfaz gráfica

- 8 JButton que son: Conectar a Higgs, desconectar a Higgs, encender motores, encender sonars y 4 flechas de dirección.

Figura 6.9: Los botones de conectar, finalizar, encender motores y encender sonars

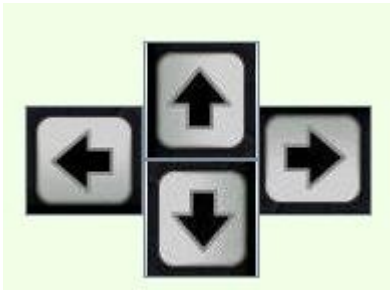


Figura 6.10: Los 4 botones que están asociados a las teclas de dirección del teclado

d) Un JMenuBar que depende del JFrame principal del punto a) que sirve como contenedor del menú desplegable de control de la interfaz gráfica. De este elemento derivan 3 JMenus que contiene a su vez botones de control en menús desplegables solo visibles si se pinchan en ellos.

Submenú "Edit" clásico

Submenú para cerrar el programa y para eleguir la apariencia

Menú "Help" para conocer datos sobre la interfaz

Figura 6.11: Menús contextuales

Para aclarar conceptos, la jerarquía de los elementos del programa interfaz es el siguiente:

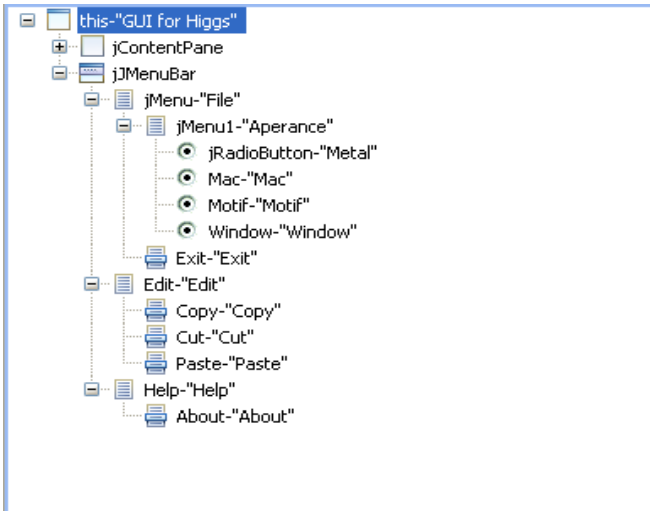


Figura 6.12: Jerarquía de elementos del programa interfaz

No desplegamos el menú "jContentPane" debido a la gran cantidad de elementos que tiene y acabaría de mostrarse dos páginas más tarde. De forma general, se puede apreciar la jerarquía de clases y la forma de estructuración de JavaBeans que nos facilita enormemente el trabajo a realizar.

Juntando cada uno de los elementos anteriormente comentados y haciendo una programación de eventos y de temporizadores, se ha conseguido una interfaz gráfica intuitiva y muy elegante, con todas las funciones necesarias para la correcta teleoperación de Higgs.

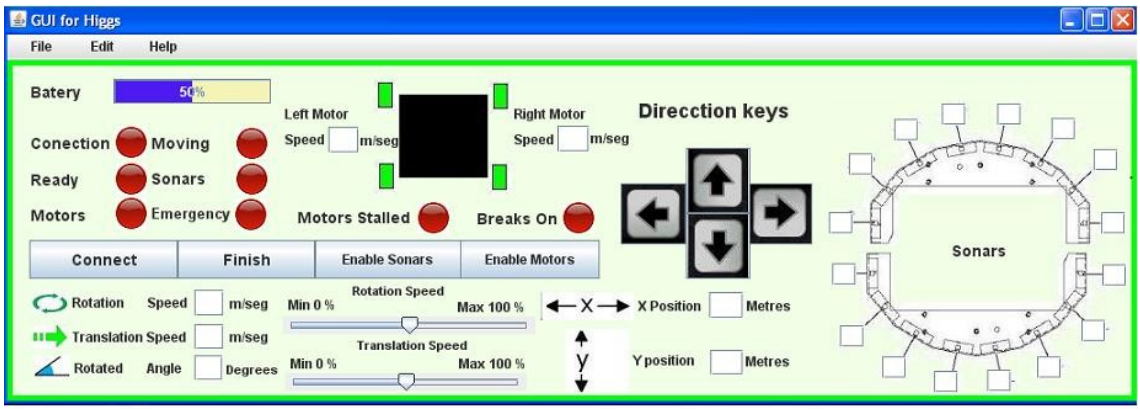


Figura 6.13: Apariencia final de la interfaz gráfica

6.5 Funciones de la interfaz gráfica

Las funciones respecto que podemos realizar gracias a la interfaz gráfica son:

- Cambio de apariencia de la interfaz para hacerla lo más atractiva posible al operario. Se accede a ella a través del menú desplegable de control.
- Copiar y pegar los datos recibidos en un momento de la odometría del robot.
- Visualización de la batería que le queda al robot antes de apagarse.
- Saber el estado de la conexión con la plataforma robótica móvil.
- Desconectar o conectar el robot de la interfaz.
- Saber el estado de disposición del robot: Listo o preparándose.
- Saber el estado de disposición del motor del robot: Apagado o encendido.
- Encender o apagar los motores del robot.
- Estado de disposición de los sónars: Apagados o encendidos.
- Encender o apagar los sónars.
- Saber si el robot está en movimiento o si está parado.
- Conocer si se ha pulsado el botón de emergencia del robot.
- Conocer si los motores se han atorado, en algún lugar.
- Conocer la velocidad angular de rotación de cada motor.
- Conocer la velocidad de translación del robot en cada momento.
- Conocer la velocidad de rotación del robot en cada momento.
- Saber las coordenadas cartesianas del robot. Se elige como punto de inicio, el punto donde se situaba el momento de iniciarse el robot.

- Controlar la velociada máxima a la que se puede mover el robot.
- Conocer los valores de lectura de los sónars
- Controlar el movimiento del robot con las teclas de dirección del teclado.

Capítulo 7

Entorno de desarrollo

7.1 Linux

Linux denominado también GNU/Linux, es uno de los términos empleados para referirse al sistema operativo libre similar a Unix que utiliza el núcleo Linux y herramientas de sistema GNU. Su desarrollo es uno de los ejemplos más prominentes de software libre; todo el código fuente puede ser utilizado, modificado y redistribuido libremente por cualquiera bajo los términos de la GPL (Licencia Pública General de GNU) y otras licencias libres.

A pesar de que Linux sólo es el núcleo de este sistema operativo una parte significativa de la comunidad, así como muchos medios generales y especializados, prefieren utilizar dicho término. Para más información consulte la sección *"Denominación GNU/Linux"* o el artículo *"Controversia por la denominación GNU/Linux"*.

Las variantes de este sistema se denominan distribuciones y su objetivo es ofrecer una edición que cumpla con las necesidades de determinado grupo de usuarios.

Algunas distribuciones son especialmente conocidas por su uso en servidores y supercomputadoras. No obstante, es posible instalar GNU/Linux en una amplia variedad de hardware como computadoras de escritorio y portátiles.

En el caso de computadoras de bolsillo, teléfonos móviles, dispositivos empotrados, videoconsolas y otros, puede darse el caso de que las partes de GNU se remplacen por alternativas más adecuadas en caso.

Dentro de los sistemas operativos GNU/Linux, tiene una seríe de encapsulado del nucleo de Linux que hacen completar un sistema operativo completo llamado distribución. Una distribución Linux o distribución GNU/Linux (coloquialmente llamadas *distros o sabores*) es cada una de las variantes de este sistema operativo que incorpora determinados paquetes de software para satisfacer las necesidades de un grupo específico de usuarios, dando así origen a ediciones domésticas, empresariales y

para servidores. Por lo general están compuestas, total o mayoritariamente, de software libre, aunque a menudo incorporan aplicaciones o controladores propietarios.

Dentro de este proyecto se decidió usar el sistema operativo GNU/Linux como base de ejecución de los programas de desarrollo y de los programas desarrollados. Y como distribución elegimos la Fedora 9.

7.2 Fedora

Fedora es una distribución Linux para propósitos generales basada en RPM, que se mantiene gracias a una comunidad internacional de ingenieros, diseñadores gráficos y usuarios que informan de fallos y prueban nuevas tecnologías. Cuenta con el respaldo y la promoción de Red Hat.

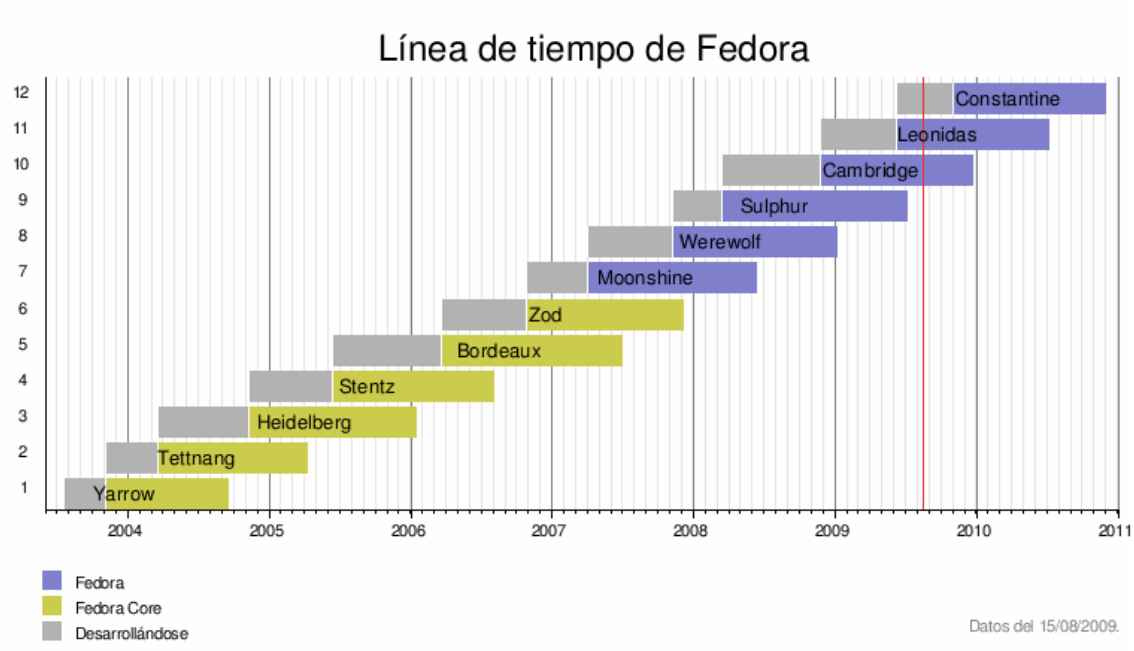
El proyecto no busca sólo incluir software libre y de código abierto, sino ser el líder en ese ámbito tecnológico. Algo que hay que destacar es que los desarrolladores de Fedora prefieren hacer cambios en las fuentes originales en lugar de aplicar los parches específicos en su distribución, de esta forma se asegura que las actualizaciones estén disponibles para todas las variantes de GNU/Linux. Max Spevack en una entrevista afirmó que: "Hablar de Fedora es hablar del rápido progreso del Software Libre y de Código Abierto." Durante sus primeras 6 versiones se llamó *Fedora Core*, debido a que solo incluía los paquetes más importantes del sistema operativo [ref 20].

Las características más importantes de fedora son:

- Incluye DVD’s**,** CD’s**,** LiveCD’s**,** USB’s para instalar y CD’s y USB’s de rescate por si el sistema falla o tiene que ser reparado.
- Fedora trata de ser el líder en el ámbito de usar software libre **y** código abierto.
- Prefieren hacer cambios en las fuentes que aplicar un parche específico para su distribución, de esta forma se asegura que las actualizaciones estén disponibles para todas las variantes de Linux.
- Yum es el administrador de paquetes del sistema. Una alternativa para este administrador es apt-rpm (comparable con APT pero maneja RPM) que podria ser más familiar para personas que hayan usado Debian o derivados.
- Al igual que Mandriva y Ubuntu, cuenta con repositorios de los cuales Fedora recomienda usar solo los de código abierto o software libre.
- Fedora se destaca en seguridad y utiliza SELinux entre otras medidas de seguridad.

Nombre del proyecto	Versión	Nombre en código	Fecha de liberación	Versión del núcleo Linux
Fedora Core	1	Yarrow	6 de noviembre de 2003	2.4.19
	2	Tettnang	18 de mayo de 2004	2.6.5
	3	Heidelberg	8 de noviembre de 2004	2.6.9
	4	Stentz	13 de junio de 2005	2.6.11
	5	Bordeaux	20 de marzo de 2006	2.6.15
	6	Zod	24 de octubre de 2006	2.6.18
Fedora	7	Moonshine	31 de mayo de 2007	2.6.21
	8	Werewolf	8 de noviembre de 2007	2.6.23.1
	9	Sulphur	13 de mayo de 2008	2.6.25
	10	Cambridge	25 de noviembre de 2008	2.6.27
	11	Leonidas	9 de junio de 2009	2.6.29-4
	12	Constantine	Programada para el día: 3 de noviembre de 2009	

Figura 7.1: Versiones de Fedora



Diagramas de desarrollo de fedora

7.2.1 Instalar soporte para cámara firewire en fedora 9

Para poder usar nuestra cámara firewire hay que seguir los siguientes pasos:

1º Habilitar los repositorios ATrpms, creando un fichero /etc/Yum.repos.d/atrpms, con las líneas siguientes:

```
[atrpms]
name=Fedora $releasever - $basearch - ATrpms
baseurl=http://dl.atrpms.net/f$releasever-$basearch/atrpms/stable
gpgkey=http://ATrpms.net/RPM-GPG-KEY.atrpms
enabled=1
gpgcheck=1
includepkgs=ieee1394*,libraw1394*
```

2º Instalar los paquetes ieee1394, ieee1394-kmdl: yum install ieee1394{,-kmdl}.

3º Desabilitar los packetes libraw1394 instalados por defecto: rpm –e –nodeps libraw1394

4º Instalar los paquetes libraw1394, libraw1394_8: yum install libraw1394{,_8}.

5º Añadir las líneas siguientes a /etc/modprobe.d/blacklist:

```
blacklist firewire_core
blacklist firewire_ohci
```

6º Reiniciar el sistema

7º Dar permisos a /dev/raw1394 creando el fichero /etc/udev/rules.d/45-firewire.rules y incluyendo la línea: KERNEL=="raw1394", GROUP="disk"

8º Añadir tu usuario al grupo “disk”. Abrir el fichero /etc/group con tu editor de texto y añadir disk: x: 6: root.

9º Para cargar en cada inicio el modulo raw1394, incluir la línea /sbin/modprobe raw1394 en /etc/rc.local.

7.2.2 Instalación de JDK para desarrollar aplicaciones en Java

- Primero bajar el paquete de la pag de SUN

http://java.sun.com/javase/downloads/index.jsp

- Luego elegir el Java Runtime Environment (JRE) 6 Update 6, debido a que el paquete JDK es para desarrolladores.

- En la página siguiente escoger "Linux" como Plataforma y como lenguaje (Multi-language). Luego aceptar la licencia y presionar "Continue".
- En la siguiente pagina, seleccionar:

Linux self-extracting file jre-6u6-linux-i586.bin 18.84 MB

Para instalar:

```
[root@localhost ~]# sh jre-6u6-linux-i586.bin
```

- Revise la licencia hasta la parte de abajo y luego digite 'yes'
- Luego colocar

```
[root@localhost ~]# mv -f jre1.6* /opt/jre1.6
```

- Simplemente se debe borrar el directorio /opt/jre1.6 y copiar la nueva descarga en /opt/jre1.6.

```
[root@localhost ~]# rm -rf /opt/jre1.6
[root@localhost ~]# mv -f jre1.6* /opt/jre1.6
```

7.3 Entorno de desarrollo Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Casi todo el código desarrollado en este proyecto ha sido a través de eclipse como modo de apoyo [ref 21].

Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse,

una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

La base para Eclipse es la Plataforma de cliente enriquecido (del Inglés Rich Client Platform RCP). Los siguientes componentes constituyen la plataforma de cliente enriquecido:

- Plataforma principal - inicio de Eclipse, ejecución de plugins
- OSGi - una plataforma para bundling estándar.
- El Standard Widget Toolkit (SWT) - Un widget toolkit portable.
- JFace - manejo de archivos, manejo de texto, editores de texto
- El Workbench de Eclipse - vistas, editores, perspectivas, asistentes

Los widgets de Eclipse están implementados por una herramienta de widget para Java llamada SWT, a diferencia de la mayoría de las aplicaciones Java, que usan las opciones estándar Abstract Window Toolkit (AWT) o Swing. La interfaz de usuario de Eclipse también tiene una capa GUI intermedia llamada JFace, la cual simplifica la construcción de aplicaciones basada en SWT.

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés *plug-in*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente para soportar otros lenguajes de programación.

La definición que da el proyecto Eclipse acerca de su software es: "*una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular*".

En cuanto a las aplicaciones clientes, eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Por ejemplo, GEF (Graphic Editing Framework - Framework para la edición gráfica) es un plugin de Eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto wysiwyg hasta

editores de diagramas UML, interfaces gráficas para el usuario (GUI), etc. Dado que los editores realizados con GEF "viven" dentro de Eclipse, además de poder ser usados conjuntamente con otros plugins, hacen uso de su interfaz gráfica personalizable y profesional.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java.

Esto permite técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

El 28 de junio de 2005 fue liberada la versión 3.1 que incluye mejoras en el rendimiento, el soporte de Java 5.0, mejor integración con Ant (incluido *debugger*) y CVS.

Eclipse comenzó como un proyecto de IBM Canadá. Fue desarrollado por OTI (Object Technology International) como reemplazo de VisualAge también desarrollado por OTI. En noviembre del 2001, se formó un consorcio para el desarrollo futuro de Eclipse como código abierto. En 2003, la fundación independiente de IBM fue creada.

Eclipse 3.0 (2003) seleccionó las especificaciones de la plataforma OSGi como la arquitectura de tiempo de ejecución. Las versiones de eclipse son:

Callisto

En 2006 la fundación Eclipse coordinó sus 10 proyectos de código abierto, incluyendo la Platafoma 3.2, para que sean liberados el mismo día. Esta liberación simultánea fue conocida como la liberación **Callisto**.

Europa

La versión consecutiva a Callisto es Europa, que corresponde a la versión 3.3 de Eclipse, salió el 29 de Junio del 2007.

Ganymede

La versión consecutiva a Europa es Ganymede, que corresponde a la versión 3.4 de Eclipse, salió el 25 de Junio del 2008.

Galileo

La versión consecutiva a Ganymede es Galileo, que corresponde a la versión 3.5 de Eclipse, salió el 24 de Junio del 2009.

Gracias al uso de eclipse, y de su editor de líneas de programación, se pudo ahorrar mucho tiempo de programación, ya que en cada momento sabías si la sintaxis escrita en tu programa tenía algún fallo de compilación sin tener que estar compilando todo el rato que quisieras comprobar los resultados.

Para instalar Eclipse, introducirse en una ventana de comando y teclear *sudo yum install eclipse*.

7.3.1 Editor de texto de eclipse

Eclipse dispone de un Editor de texto con resaltado de sintaxis. La compilación es en tiempo real. Tiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (*wizards*) para creación de proyectos, clases, tests, etc… También tiene la función de refactorización. El editor de texto de eclipse es muy intuitivo, elegante y es fácil de encontrar los elementos gracias a su motor de búsqueda. Además da sugerencias de programación y puede dar soluciones a posibles fallos de programación.

7.3.2 Visual Editor de Eclipse: Uso e instalación

El proyecto de Visual Editor de Eclipse es una plataforma de desarrollo independiente, de código abierto suministrando herramientas para la creación de interfaces gráficas en e implementaciones de programación amigable y potente para Swing/JFC y SWT/RCP. El propósito del proyecto editor visual de Eclipse es la creación avanzada, evolución, y para cultivar una comunidad de código abierto y un sistema de productos complementarios, capacidades y servicios.

En particular, el proyecto de de editor visual pretende ser útil para crear constructores de GUI para otros idiomas como C/C++ y conjuntos de widget alternativo, las que no son compatibles con las incluidas en Java.

Dentro del proyecto de creación de la interfaz de control de Higgs, fue muy útil el uso de este añadido de eclipse, debido a que ahorras muchas horas de programación escrita, ya que a través de un atractivo entorno se puede generar muchas líneas de código de forma automática, además de que era capaz de mostrar en cada momento la apariencia de la interfaz sin tener que estar ejecutando la aplicación en cada momento [ref 22].

Para instalar el Visual Editor en Eclipse Ganymede:

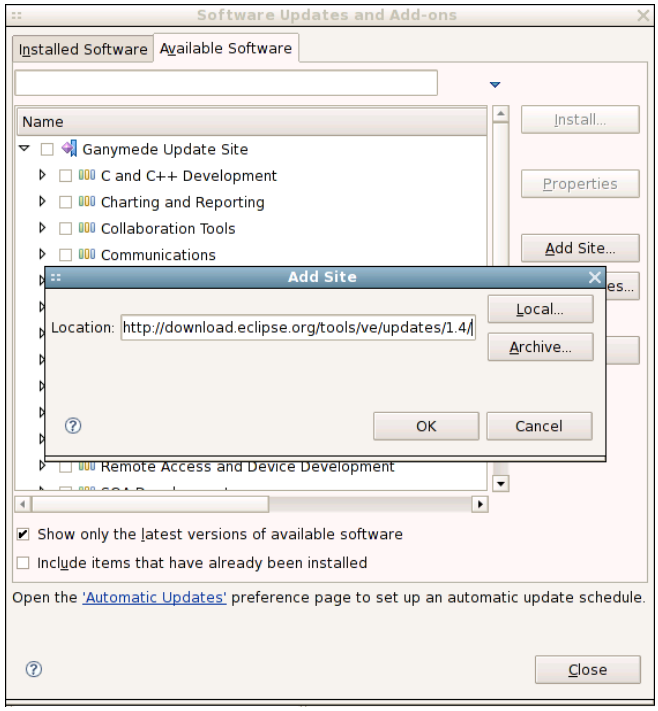
1º Descargar e instalar una de las siguiente verisones para java de eclipse desde su pagina principal

- "Eclipse IDE for Java EE Developers"
- "Eclipse IDE for Java Developers"
- "Eclipse for RCP/Plug-in Developers"
- "Eclipse IDE for Java and Report Developers"

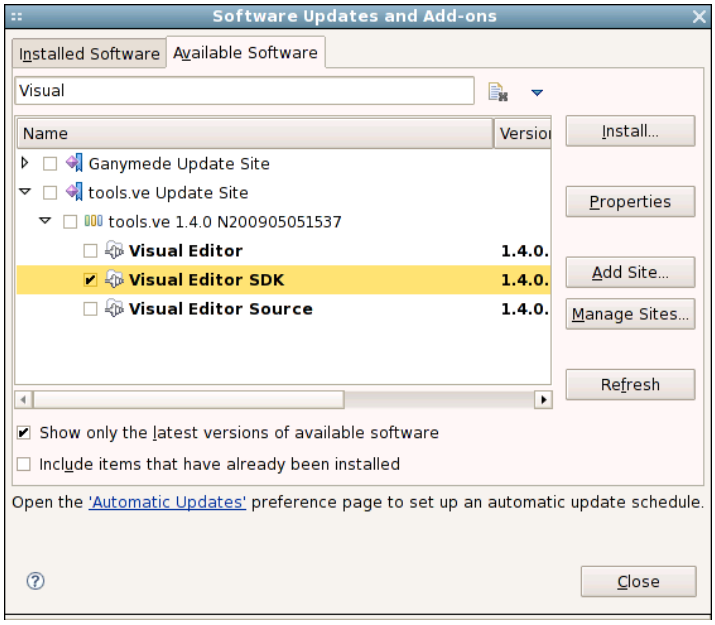
2º Iniciar eclipse.

3º Presionar Help > Software Updates > Available software

4º Añadir este sitio: *http://download.eclipse.org/tools/ve/updates/1.4*, tal y como aparece en la imagen.



5º Expadir la vista y seleccionar “Visual Editor” o “Visual Editor SDK”



6º Presionar install

Capítulo 8

Resultados, Conclusiones y líneas futuras

En este capítulo se analizan las conclusiones alcanzadas tras la realización de este proyecto, los resultados obtenidos en el mismo, así como las líneas futuras que pueden surgir a raíz del mismo.

8.1 Resultados

Tras la realización de las tareas propuestas en este proyecto y la integración de las aplicaciones descritas, la plataforma de desarrollo Higgs de ASLab está provista de nuevas funcionalidades y mayores recursos para la consecución de sus objetivos:

- Higgs es ahora un robot teleoperable de forma mucho más sencilla e intuitiva. A través de una interfaz gráfica podemos saber en cada momento dónde está y cual es su estado. Del robot podemos conocer en cada momeno:

  - El estado de las baterías de Higgs, así sabemos cuanta energía queda y cuanta autonomía tiene.
  - Podemos activar y desactivar el motor y los sonars del robot, para así ahorrar batería si es posible.
  - Obtener una lectura en cada momento de los valores tomados por los 16 sonars que posee el robot.
  - La velocidad de traslación y de rotación del robot en cada momento.
  - La posición en coordenadas cartesianas respecto al punto de inicio del robot.
  - Podremos saber si los motores se han bloqueado o si se ha pulsado el botón de emergencia del robot.

- Higgs ahora es un robot teleoperable de forma remota. Todas las especificaciones técnicas son compatibles con tecnologías inalámbricas, haciendo posible un movimiento libre por parte del robot.

- Se ha instalado una cámara de visión estereoscópica sobre una muñeca mecánica que está montada a su vez sobre el robot. Esta cámara se alimenta a partir de las baterías del robot.
- Ahora, gracias a la implementación de la cámara firewire y la recepción en el PC de control de las imágenes en tiempo real, la teleoperación de Higgs es mucho más sencilla; así, cuando Higgs esté siendo manejado de forma manual, el teleoperador podrá ver y distinguir obstáculos de forma mucho más intuitiva que a través de datos de sensores, que puede no llegar a interpretar bien.
- El software de control es totalmente portable entre sistemas gracias al uso del lenguaje de programación Java.
- Se ha implementado de forma satisfactoria el uso de la tecnología CORBA tanto en el robot, como en el PC de control. Una de las grandes ventajas que nos da CORBA es la interoperabilidad de entornos de programación heterogéneos, gracias a ser un sistema distribuido.

8.2 Conclusiones

Tras la realización de este proyecto se ha creado un sistema que nos permite visualizar remotamente y en tiempo real la imagen obtenida por una cámara estereoscópica. Se ha creado un sistema que permite controlar a Higgs remotamente de forma sencilla, dinámica e intuitiva, de tal manera que sabemos todo lo necesario para la teleoperación completa.

Las principales conclusiones obtenidas son:

1º **Teleoperación completa**: Gracias al conjunto de imágenes en tiempo real, junto con un visionado de la información del láser y de un mapa del entorno del robot, es muy fácil conocer como es el entorno y cómo manejar el robot en los consecutivos movimientos. Hacer los movimientos del robot ahora mismo es muy fácil gracias a la interfaz de control: Podemos manejar el robot con las pulsaciones de las teclas de dirección como si de un coche teledirigido se tratase. Además las ordenes llegan de forma fluida y ordenada.

De todas formas, hay momentos que al depender de la tecnología wireless, es posible que se pierda la comunicación con el robot, haciendo necesario un sistema de emergencia en caso de que no podamos manejar más el robot, ya que puede darse el caso de que colisione contra una pared.

2º **Visión estereoscópica**: esta tecnología ofrece nuevas funcionalidades y características al Higgs de gran utilidad. Se pueden desarrollar muchas aplicaciones innovadoras por el grupo ASLab para aumentar las capacidades de Higgs.

La parte de la cámara fue la más laboriosa de todas, ya que había que implementar dos librerías y tecnologías para poder enviar la señal de video a través de la red inalámbrica: CORBA y las librerías dc1394. El hecho de transformar la imagen en un string de datos a un puntero que apuntaba a una sección de memoria donde se alojaba la cámara fue una característica de diseño muy difícil de implementar. Además al principio teníamos unos problemas enormes por causa de un "delay" de 10 segundos, haciendo inviable un control óptimo del robot. La tecnología CORBA funcionaba muy bien, pero el ancho de banda de la red wireless era muy pequeña para lo que queríamos hacer. Por lo tanto, ese fue uno de los mayores retos de este proyecto. Al final conseguimos hacer que el "delay" bajara a menos de un segundo mejorando las capacidades de red y sobretodo haciendo transformaciones a la imagen para hacer que ocupara menos, de todas formas intentábamos que se disminuyera el tamaño de las imágenes sin sacrificar apenas nada de calidad, con complejos algoritmos de transformación de imágenes.

3º Uso de tecnología Java para la interoperabilidad El uso de Java como principal lenguaje de programación ha sido una decisión acertada a la hora de obtener una compatibilidad plena con casi todos los sistemas operativos en la actualidad, dándonos una portabilidad extrema, incluso haciendo que 2 operarios puedan controlar el mismo robot.

4º Sistema basado en CORBA: la instalación de nuevas aplicaciones o funciones en un sistema se vuelve extremadamente sencilla, además de proveer de una gran organización y transparencia. Se recomienda el uso de tecnologías de este estilo, transparentes al usuario, que permita la compartición de software, aprovechar y compartir el trabajo de otros, para así no tener que realizar varias veces la misma aplicación, una para cada lenguaje.

Al tener un estado inicial basado en CORBA ha sido relativamente sencillo acceder y conocer las capacidades iniciales de las que disponía Higgs y trabajar con ellas.

5º Uso de software libre: las aplicaciones se han hecho portables a cualquier tipo de sistema. Si bien es verdad que al principio el hecho de trabajar con software libre pudo ralentizar los avances, al final ha sido productivo.

El problema inicial consistió, como en muchas cosas, en aprender. La costumbre que se tenía era trabajar con software propietario, desconociendo por completo lo que ocurría bajo éste.

Las desventajas fundamentales del software propietario son que se conoce menos el sistema que uno tiene entre manos o nos da menos libertad y puede resultar más restringido. Las ventajas que ofrece, como puede ser mejor información, se van equiparando con el software libre día a día. Ahora bien, el hecho de poder utilizar un programa ya compilado, y modificarlo si las necesidades no son exactamente las mismas que las del anterior usuario es una ventaja fundamental del software libre.

De esta manera se favorece la investigación conjunta, fomentando el hecho de compartir el software, de no cerrarlo a nadie, buscando siempre un objetivo común. Todo esto se completa con Internet, que posibilita contactar con la persona que ha escrito el software e intercambiar opiniones.

8.3 Líneas futuras

Una de los objetivos “no oficiales” era implementar un joystick como modo de teleoperación, pero debido a que el desarrollo de controladores de joystick para java está todavía sin desarrollar, era necesario usar librerías en C++ y después portarlas a Java a través de JNI (sistema para el uso de librerías C++ con código Java), Esto, eliminaría la portabilidad de la interfaz, haciéndolo compatible con uno de los obejtivos principales del trabajo. Cuando el estado del arte esté más desarrollado, se podrá volver a retomar este punto.

Uno de los proyectos futuros será la implementación de la interfaz a dispositivos móviles como pueden ser PDAs o teléfonos móviles. Debido al uso de sistemas operativos basados en linux o a gracias a Windows Mobile, se podría portar el código C++ a ellos, y por supuesto el código JAVA.

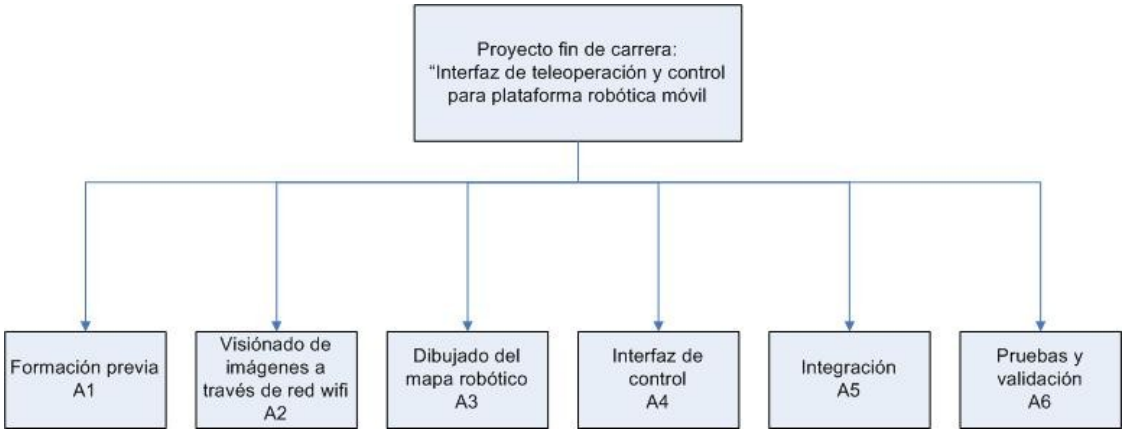
Otra opción interesante sería Implementar todas las 3 funciones principales en una ejecución única a través de programación multihilos, así la coordinación y control sería más precisa y controlable.

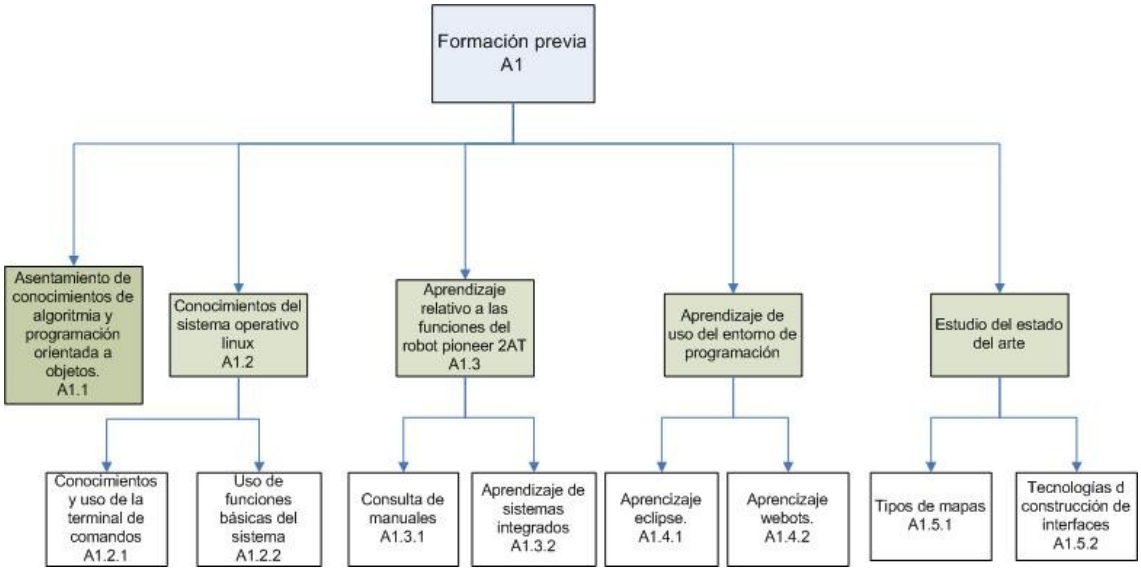
Otra función posible hubiera sido la inclusión de un control de trayectorias para el robo: haciendo click en una parte del mapa, podríamos indicarle al robot hacia donde debe de ir.

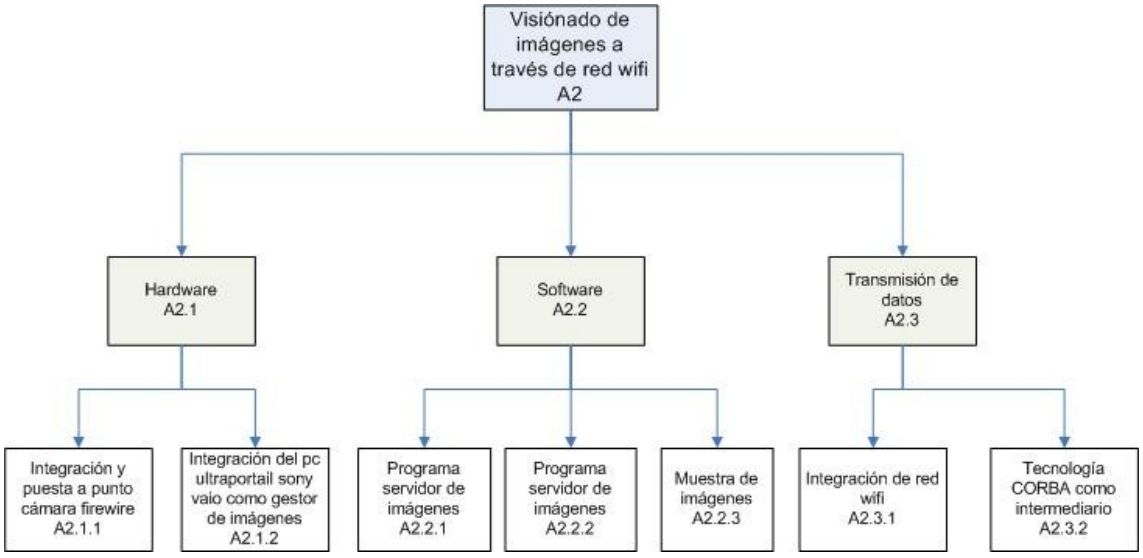
Extensión de los modelos geométricos 2D a 3D mediante la utilización de una muñeca que incline el láser. En una primera etapa esto serviría para aumentar la seguridad con la que se mueve el robot, posteriormente se podrán desarrollar modelos que utilicen estos datos para mejorar la localización.

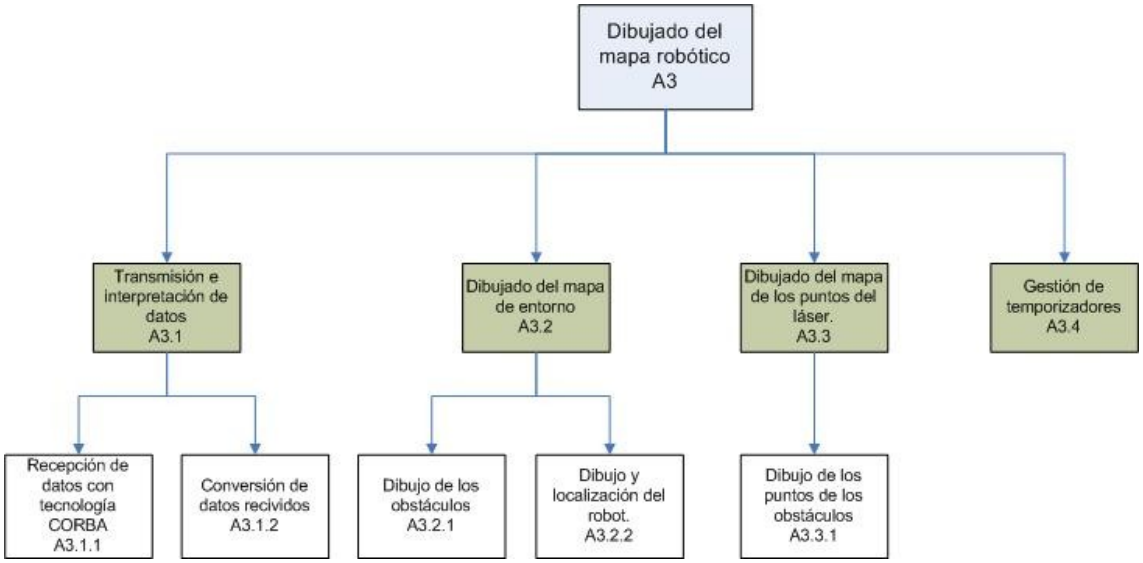
III INFORMACIÓN COMPLEMENTARIA

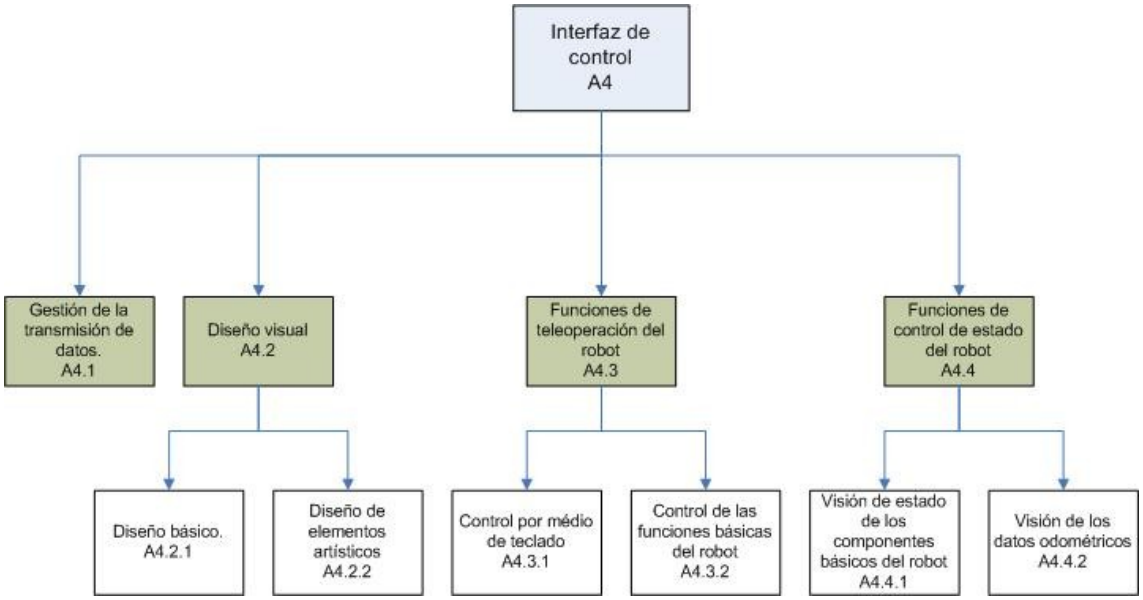
Estructura de Descomposición del Proyecto (EDP)

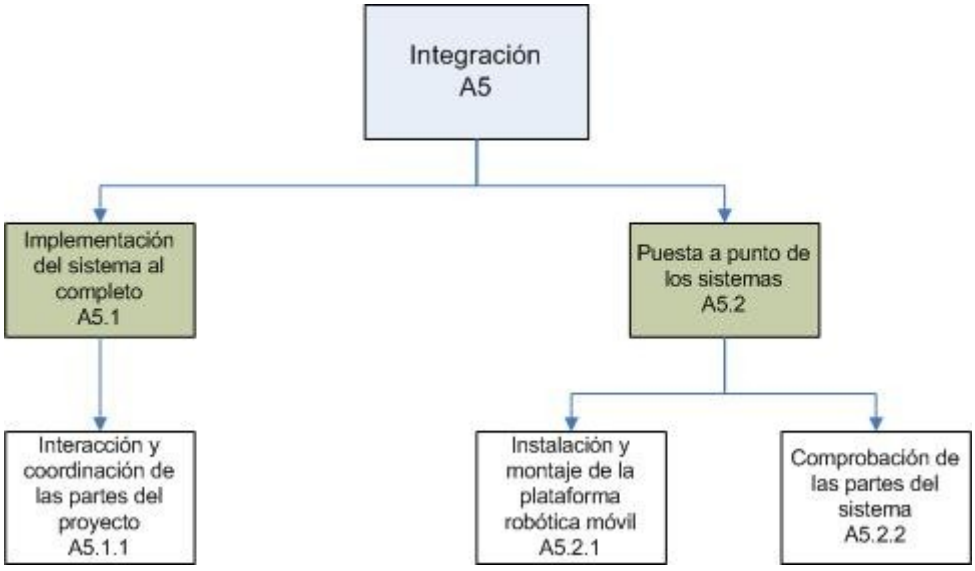


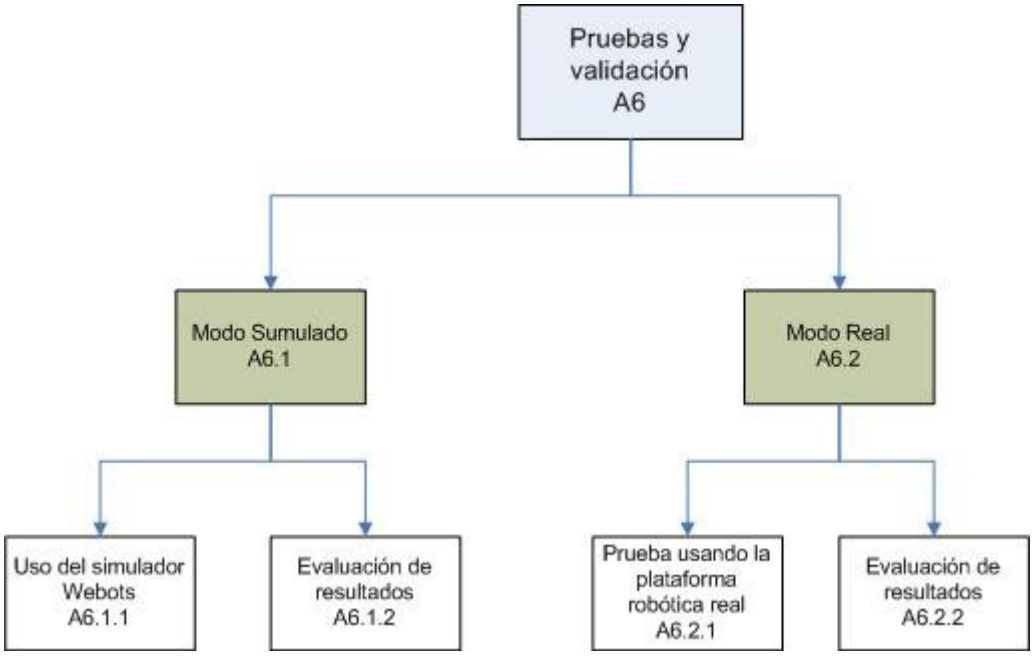



Visiónado de imágenes a través de red wifi A2
Hardware A2.1
Software A2.2
Transmisión de datos A2.3
Integración y puesta a punto cámara firewire A2.1.1
Integración del pc ultraportail sony vaio como gestor de imágenes A2.1.2
Programa servidor de imágenes A2.2.1
Programa servidor de imágenes A2.2.2
Muestra de imágenes A2.2.3
Integración de red wifi A2.3.1
Tecnología CORBA como intermediario A2.3.2


Dibujado del mapa robótico A3
Transmisión e interpretación de datos A3.1
Dibujado del mapa de entorno A3.2
Dibujado del mapa de los puntos del láser. A3.3
Gestión de temporizadores A3.4
Recepción de datos con tecnología CORBA A3.1.1
Conversión de datos recividos A3.1.2
Dibujo de los obstáculos A3.2.1
Dibujo y localización del robot. A3.2.2
Dibujo de los puntos de los obstáculos A3.3.1


Interfaz de control A4
Gestión de la transmisión de datos. A4.1
Diseño visual A4.2
Funciones de teleoperación del robot A4.3
Funciones de control de estado del robot A4.4
Diseño básico. A4.2.1
Diseño de elementos artísticos A4.2.2
Control por médio de teclado A4.3.1
Control de las funciones básicas del robot A4.3.2
Visión de estado de los componentes básicos del robot A4.4.1
Visión de los datos odométricos A4.4.2


Integración A5
Implementación del sistema al completo A5.1
Puesta a punto de los sistemas A5.2
Interacción y coordinación de las partes del proyecto A5.1.1
Instalación y montaje de la plataforma robótica móvil A5.2.1
Comprobación de las partes del sistema A5.2.2


Pruebas y validación A6
Modo Sumulado A6.1
Modo Real A6.2
Uso del simulador Webots A6.1.1
Evaluación de resultados A6.1.2
Prueba usando la plataforma robótica real A6.2.1
Evaluación de resultados A6.2.2

Planificación. Diagrama de Gantt

En la siguiente figura F.1 se muestra el diagrama de Gantt de la planificación del proyecto. El desarrollo del mismo ha tenido una duración de 12 meses, incluyendo la fase inicial de formación. El comienzo tuvo lugar en septiembre de 2008 y la finalización se establece a mediados de septiembre de 2009.

Una etapa muy importante del proyecto, viene dada por el aprendizaje de las múltiples tecnologías que componen la base tecnológica del trabajo. Este estudio se iba alternando con el desarrollo de aplicaciones y etapas donde los conocimientos básicos para el desarrollo ya se había pasado, optimizando el tiempo de ejecución. Del aprendizaje la parte que más tiempo llevó, fue la familiarización con la tecnología CORBA debido a su alta curva de aprendizaje. También el uso de Linux como sistema operativo base, supuso un breve aprendizaje del mismo, ya que no es un sistema tan extendido como el que puede ser Microsoft Windows.

Una vez dominado CORBA y el sistema operativo Linux fedora, se desarrolló el servidor de imágenes de la cámara firewire, la cual necesito un periodo más de aprendizaje para el uso de la librería dc1394 para el control de cámaras en Linux. Después de ver que era viable la transmisión de imágenes a través de red con CORBA y esta librería, se desarrolló el software cliente que recoge las imágenes que envía el servidor. El hecho de que fuera posible trabajar en estos dos proyectos a la vez suponía el ahorro de mucho tiempo que se empezó a invertir en el aprendizaje de las librerías de diseño gráfico AWT y Swing.

Una vez terminado con e desarrollo del programa de visión de imágenes, se pudo dedicar todo el tiempo a la compresión y dominación de las librerías AWT y swing para poder diseñar la interfaz propiamente dicha.

Una vez terminado el aprendizaje, la programación de los mapas con AWT y CORBA fue relativamente fácil y no se tardó mucho en conseguir desarrollar tanto el mapa del láser como el mapa del entorno.

El caso de la interfaz de teleoperación en Swing supuso un problema algo mayor, ya que requería de un desarrollo artístico y poco ingenieril. Primero se desarrolló el sistema

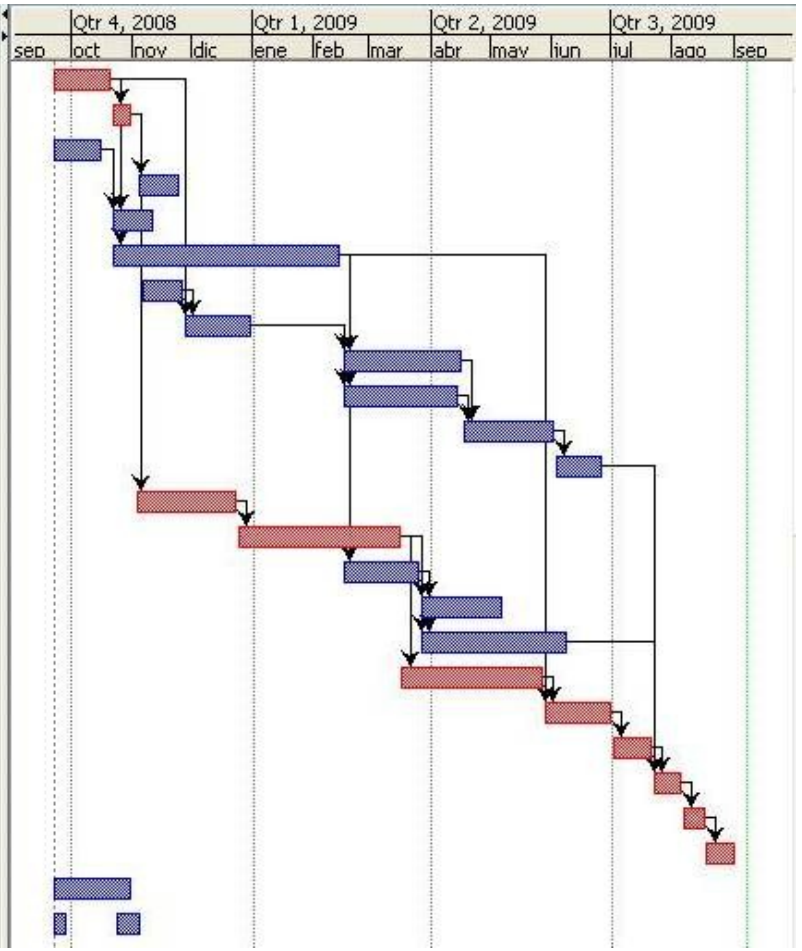
básico, y después el artístico, era un problema que no se pudieran hacer los dos a la vez, ya que uno dependía del otro.

Una vez se tenían las 3 partes (visión remota, dibujado de mapas e interfaz de teleoperación), se integrarían en un mismo sistema, a la vez que probábamos que funcionaban bien.

Cuando todo estaba preparado hicimos pruebas con el simulador Webots, con el que se obtuvieron resultados satisfactorios de las pruebas, al ver que la interfaz respondía de forma coherente y rápida.

Una vez terminadas las pruebas virtuales, se emperazon a hacer pruebas ya con el robot real, para pulir y ver que todo andaba en ordn

	Nombre	Duración	Inicio	Terminado	Predece...
1	Aprendizaje de algoritmia	22 days	22/09/08 8:00	21/10/08 17:00	
2	Aprendizaje programación	8 days	22/10/08 8:00	31/10/08 17:00	1
3	Familiarización con SO Fedora	19 days	22/09/08 8:00	16/10/08 17:00	
4	Aprendizaje Webots	15 days	4/11/08 8:00	24/11/08 17:00	2
5	Aprendizaje del entorno programación	15 days	22/10/08 8:00	11/11/08 17:00	1;3
6	Aprendizaje Corba	83 days	22/10/08 8:00	13/02/09 17:00	1
7	Familiarización con cámara firewire	15 days	6/11/08 8:00	26/11/08 17:00	
8	Aprendizaje uso libreria dc1394	25 days	27/11/08 8:00	31/12/08 17:00	1;7
9	Programación cliente emisión imágenes	44 days	16/02/09 8:00	16/04/09 17:00	8;6
10	Programación servidor recepción imágenes	42 days	16/02/09 8:00	14/04/09 17:00	8;6
11	Algoritmo de muestra de imagenes	33 days	17/04/09 8:00	2/06/09 17:00	9;10
12	Pruebas de transmisión de imágenes	18 days	3/06/09 8:00	26/06/09 17:00	11
13	Familiarización con librerias gráficas	37 days	3/11/08 8:00	23/12/08 17:00	2
14	Aprendizaje de libreria grafica AWT	59 days	24/12/08 8:00	16/03/09 17:00	13
15	Diseño algoritmo de recepción de datos	29 days	16/02/09 8:00	26/03/09 17:00	6
16	Programación del visionado del mapa láser	30 days	27/03/09 8:00	7/05/09 17:00	14;15
17	Prgramación del mapa del entorno	52 days	27/03/09 8:00	8/06/09 17:00	14;15
18	Aprendizaje de la libreria gráfica SWING	52 days	17/03/09 8:00	27/05/09 17:00	14
19	Diseño básico de la interfaz	25 days	28/05/09 8:00	1/07/09 17:00	18;6
20	Diseño artistico de la interfaz	14 days	2/07/09 8:00	21/07/09 17:00	19
21	Implementación del sistema	11 days	22/07/09 8:00	5/08/09 17:00	12;17;20
22	Pruebas de simulación	8 days	6/08/09 8:00	17/08/09 17:00	21
23	Pruebas reales	11 days	18/08/09 8:00	1/09/09 17:00	22
24	Lectura de manuales	30 days	22/09/08 8:00	31/10/08 17:00	
25	Estudio del estado del arte	32 days	22/09/08 12:00	5/11/08 13:00	



Presupuesto

En el siguiente cuadro se presenta el presupuesto desglosado del proyecto. Hay que tener en cuenta que se trata de una aproximación claramente sobreestimada, ya que se ha asignado el coste total del hardware a este proyecto. Con el robot Higgs, actualmente se están realizando otros nuevos y en el futuro se seguirá trabajando sobre él. Respecto a la cámara esteroscópica, ya se han hecho anteriormente más proyectos. Además el ordenador sony vaio no está solo asignado a este trabajo. En los componentes del inmovilizado sí se ha hecho una valoración de la parte correspondiente a este proyecto.

CONCEPTO	VALOR
Sistema robótico completo	
Robot pioneer 2AT	6.500 €
Sensor láser SICK LMS-200	860 €
PC interno	1.000 €
Cámara STH-MDCS-C	876 €
Cable firewire 6 pines - 4 pines	6 €
Sony VAIO 11"	1.200 €
Baterías adicionales	300 €
Transformador 24V – 12V	200 €
Subtotal	**10.942 €**
Inmuebles	
Laboratorios	5.000 €
Mantenimiento y servicios	1.500 €
Subtotal	**6.500 €**
Equipos hardware	
PCs	1.500 €
Comunicaciones	300 €
Mantenimiento	300 €
Subtotal	**2.100 €**
Software	
Licencia Webots	1.200 €
Subtotal	**1.200 €**
Personal	
Proyectante 1000 horas x 5 euros/hora	5.000 €
Tutores 100 horas x 40 euros/hora	4.000 €
Subtotal	**9.000 €**
Total del proyecto	**29.752 €**

Diagrama de colaboración UML del sistema completo

Debido al enorme tamaño de éste proyecto, es necesario la explicación total de cómo funciona el sistema al completo, para así dar una idea al lector de cómo está funcionando todo de una forma mucho más amena que leyendo todo lo expuesto en el punto de desarrollo. De todas formas el diagrama es aclarativo y a todas luces, insuficiente, para comprender todo el sistema hay que leerse el documento completo.

Para crear este diagrama se han usado los conocimientos que se imparten en la asignatura de proyectos de quinto de Ingeniería industrial (plan 2000) [ref 23].

Vaio + Cámara
Camara Firewire
ENVIAR: Datos Imagen
CONTROL: Cámara
Cpu
CONTROL: Lecturas
Laser SICK
CONTROL: Sistema
Sistema operativo
ENVIAR: Datos imagen, láser
ENVIAR: Lecturas
ENVIAR: Datos laser, estado y navegación
ENVIAR: Ordenes, Petición de datos
Higgs
ENVIAR: Lecturas
Control de navegación
ENVIAR:Datos navegación
CONTROL: Lecturas
ENVIAR: Lecturas
Cpu de abordo
Sonars
ENVIAR: Ordenes, Petición de datos
ENVIAR: Datos laser, estado y navegación
NOTIFICACIÓN: Estado
Control de estado
Operario Higgs
Pc de control
Control de datos y tecleo
GUI Higgs
ENVIO: Datos imagen
CPU
ORDEN: Mostrar imágenes
CONTROL: Interfaz
NOTIFICACIÓN: Imagen procesada
Visión
Monitor
Sistema operativo

Bibliografía

[1] The International Federation of Robotics, http://www.ifr.org/

[[2] Aníbal Ollero: Manipuladores Y Robots Móviles.

[3] Daniel Audí Piera: Cómo Y Cuándo Aplicar Un Robot Industrial.

[4] López, Ignacio, Sanz, Ricardo: Towards a Theory of General Autonomous Systems, UPM ASLab Seminars on Autonomous Systems I, 2007.

[5] Jesús Sánchez Allende, Gabriel huecas Fernández-Toribio, Baltasar Fernández Manjón, Pilar Moreno Díaz, Antonio José Reinoso Pinado, Ricardo Sosa Sánchez-Cortés: Programación en java 2.

[6] Sun Microsystems. RMI Documentation Beta Draft. Diciembre 2004.

[7] Michi Hening, Steve Vinoski: Advanced CORBA(R) Programming with C++

[8] Rodríguez Manuel: RCS for process control: Is there anything new under the sun?,UPM ASLab, 2007.

[9] Ronan Geraghty et al.COM-CORBA Interoperability.Prentice Hall, 1997

[10] Barrientos Antonio: Nuevas Aplicaciones de la Robótica. Robots de Servicio. (Departamento de Automática, Ingeniería electrónica e Informática industrial DISAMUPM.)

[11] Hernando, Adolfo, Versión RT-CORBA del servidor Pioneer 2AT-8, UPM ASLab, 2007.

[12] Hernández, Carlos; Hernando, Adolfo, Manual de Higgs, UPM ASLab, 2008.

[13] STH-MDCS/-C Stereo Head User's Manual. http://www.videredesign.com/docs/sthmdcs.pdf

[14] Apple Computer developer group. http://developer.apple.com/hardwaredrivers/firewire/index.html

[15] S.N. y Pollefeys. M. Synchronization and Calibration of Camera Networks from Silhouettes. In Proc. of 17th ICPR, 2004.

[16] libdc1394 Homepage. http://damien.douxchamps.net/ieee1394/libdc1394/

[17] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In Proc. of the International Joint Conference on Artificial Intelligence, 1995.

[18] The Java Technology Tutorial, http://java.sun.com/docs/books/tutorial/

[19] Holzner, Steven La biblia en Java 2.

[20] Wikipedia sobre fedora http://es.wikipedia.org/wiki/Fedora_(distribuci
%C3%B3n_Linux)

[21] The eclipse project. www.eclipse.org/

[22] Visual editor project. http://www.eclipse.org/vep/

[23] Jose Luis Fernández Sánchez. Documentación de la asignatura Proyectos de la especialidad de Automática y Electrónica de la ETSII-UPM, 2007.