Lasmap:

Desarrollo e implementación de utilidades de SLAM para un robot móvil utilizando un dispositivo LMS

Departamento de Informática y Automática

Universidad de Salamanca

Autor:

Carlos Fernández Caramés

Tutores:

Vidal Moreno Rodilla

Belén Curto Diego

Don Vidal Moreno Rodilla y Doña Belén Curto Diego, profesores del Departamento de Informática y Automática de la Universidad de Salamanca,

CERTIFICAN:

Que el trabajo titulado "LasMap: desarrollo e implementación de utilidades de SLAM para un robot móvil utilizando un dispositivo LMS" ha sido realizado por Carlos Fernández Caramés y constituye la memoria del trabajo realizado para la superación de la asignatura Proyecto Fin de Carrera de la titulación de Ingeniero Superior en Informática en la Universidad de Salamanca.

En Salamanca, a 11 de Julio de 2005

D. Vidal Moreno Rodilla	Dª. Belén Curto Diego
Dpto. Informática y Automática Universidad de Salamanca	Dpto Informática y Automática Universidad de Salamanca

Agradecimientos

Quisiera agradecer a mis tutores, Vidal Moreno Rodilla y Belén Curto Diego, su colaboración y ayuda a lo largo del proyecto. Tambíen deseo agradecer a Andrés Vicente Lober la indispensable ayuda que me ha prestado para construir el robot móvil.

A Noelia

Índice general

Índice de figuras

Índice de tablas

1

Introducción

La robótica móvil tiene en la autonomía uno de sus principales desafíos. Así, el elemento central a resolver es el problema de navegación, que debe permitir de forma autónoma alcanzar diferentes posiciones de acuerdo con alguna especificación, proporcionada por sí mismo o por otra entidad. Por esta razón, la planificación de caminos es una de las capacidades deseables en una primera aproximación. En el caso de un entorno conocido, y suponiendo que los datos de los sensores son perfectos, el criterio para la resolución de esta clase de problemas es la optimización del coste de viajar de una posición inicial hasta una final mientras se van evitando obstáculos.

Cuando se trata de entornos desconocidos, se plantea el problema de la exploración utilizando la odometría y suponiendo que los datos que se obtienen de los sensores no son perfectos. En este caso la cuestión que afronta el robot móvil es realizar una exploración completa del entorno e ir cons-

truyendo un mapa global a medida que el robot explora nuevas partes del entorno.

Cuando lo que se pretende es la ejecución de un camino en un entorno real, existe un problema en la elección de los sensores que se van a utilizar, para lograr ejecutar con precisión el camino preplanificado. Sin embargo, en este caso, el problema clave es la autolocalización del robot (problema habitualmente conocido como "¿Dónde estoy?"). Una de las cuestiones principales para resolver este problema consiste en cómo hacer coincidir los datos que obtienen los sensores del robot (visión, sónar, láser, infrarrojos) con los datos de un mapa modelo del entorno. El problema de la autolocalización puede resolverse mediante métricas, garantizando que la diferencia entre la posición real del robot y la posición en la que el robot piensa que está es arbitrariamente pequeña y limitada, o puede resolverse por medios cualitativos y topológicos. Basar el posicionamiento del robot en la odometría no es una técnica eficaz, puesto que es bien conocido que la odometría de un robot no es suficiente al conducir a errores ilimitados en la estimación de la posición.

Si el robot está equipado con cámara de visión, entonces es posible comparar las imágenes 2D de la cámara con un modelo a priori en 3D del entorno, pero este enfoque tiene unos enormes requisitos computacionales debido al procesamiento de datos de las imágenes. Por este motivo, el uso de escáneres láser de medición de distancia está más difundido últimamente, puesto que representa el entorno con bastante mejor precisión (el error relativo puede llegar a ser de ± 1 mm) que un sensor de tipo sónar, y el tiempo de procesamiento de los datos es muchísimo menor que el de las cámaras de visión.

Queda así planteado uno de los desafíos actuales de la robótica móvil, el SLAM (*Simultaneous Localization And Mapping*[1], autolocalización y construcción de mapas), que se puede definir como una técnica usada por robots y vehículos autónomos para construir un mapa en un entorno desconocido, mientras al mismo tiempo se mantiene actualizada posición del robot dentro

[1]En ocasiones se puede encontrar que este término se refiere a *Self-Localization And Mapping*.

del mapa. Esto no es tan sencillo como en un principio puede parecer, debido a varias causas. Una de ellas es la dificultad de calcular el desplazamiento relativo del robot usando unos sensores que no tienen precisión infinita o perfecta. Otra dificultad es que en el momento en que se comete un error en la estimación de un desplazamiento relativo, éste quedara acumulado, distorsionándose en gran medida el mapa que se está construyendo.

Este proyecto surge en el momento en el que en el laboratorio del Grupo de Robótica de la Universidad de Salamanca se me plantea desarrollar una solución práctica al problema del SLAM utilizando como elemento indispensable un escáner láser de medición de distancias, en concreto el modelo LMS 221 de la casa SICK, que se puede observar en la fig. 1.1. En el siguiente apartado de esta memoria se detallan los objetivos concretos marcados para la realización de este proyecto.

Figura 1.1: *Escáner láser sick lms 221 (en la derecha, montado en el robot móvil)*

Como plataforma de desarrollo, se ha optado por utilizar el entorno Linux/GNOME, por varias razones: es un sistema operativo de libre distribución, todas las herramientas y bibliotecas utilizadas son gratuitas y de fácil acceso. Además se necesita interactuar con el servidor *Player* y el simulador *Stage* [31] que están disponibles de momento exclusivamente en entornos Unix/Linux.

En las secciones posteriores de la memoria y en los anexos se detallará el trabajo realizado para llevar a cabo el proyecto. El contenido de la memoria tiene la siguiente estructura:

- **Objetivos del proyecto:** en este apartado se especificarán de forma concreta los objetivos que se persiguen con la realización del proyecto. Se distinguirán entre objetivos generales del proyecto y los objetivos técnicos impuestos por el desarrollador.
- **Conceptos teóricos:** capítulo donde se expondrán y explicarán ciertos aspectos sobre robótica móvil, así como los conceptos matemáticos y estadísticos acerca del análisis de componentes principales, con la finalidad de obtener una mayor comprensión para un mejor seguimiento del proyecto.
- **Técnicas:** apartado en el que se exponen las técnicas elegidas para dar una solución al problema del SLAM.
- **Herramientas:** aquí se expondrán las características fundamentales de las herramientas y bibliotecas que se han utilizado a lo largo del desarrollo del proyecto.
- **Aspectos relevantes del desarrollo:** se comentarán aquellos aspectos que puedan tener mayor importancia durante todo el desarrollo.
- **Trabajos relacionados:** comprende un breve resumen de las diversas técnicas existentes de localización y construcción de mapas relacionadas con este proyecto.
- **Conclusiones y líneas de trabajo futuro:** exposición de todas aquellas conclusiones a las que se ha llegado tras el desarrollo del proyecto así como indicaciones con vistas a mejoras y ampliaciones de la aplicación desarrollada.

La documentación técnica del proyecto está dividida en 6 anexos:

- **Anexo Hardware** - *Construcción del robot*. Aquí se comentan las características de los elementos (mecánicos y eléctricos) seleccionados para formar parte del robot móvil, y se describen los pasos más importantes realizados durante la construcción del mismo.

- **Anexo 1** - *Plan del proyecto software*. En él se recoge la planificación temporal del proyecto.

- **Anexo 2A** - *Especificación de requisitos del software*. Con este documento se intenta recoger todos los requerimientos del sistema software a construir, de forma que sirva como elemento contractual entre el cliente y los ingenieros del software.

- **Anexo 2B** - *Análisis de los requisitos del software*. En este documento se pretende realizar un análisis de estos requisitos para poder obtener un modelo inicial que sirva de punto de partida para el diseño.

- **Anexo 3** - *Especificación de diseño*. En este anexo se marcará el camino de la solución a seguir tomando importantes decisiones sobre la arquitectura, datos, interfaz o detalles procedimentales de la aplicación software.

- **Anexo 4** - *Documentación técnica de programación*. Documento en el que se recogerá más detalladamente el proceso de implementación de la aplicación.

- **Anexo 5** - *Manual del usuario*. Aquí se recoge la información necesaria para la instalación y el fácil aprendizaje y manejo del sistema desarrollado.

2

Objetivos

En este apartado se introducirán los distintos objetivos perseguidos durante la realización de este proyecto.

Se diferenciará entre dos tipos de objetivos: los objetivos generales del proyecto, y los de carácter técnico que se han planteado a la hora de llevarlo a la práctica.

2.1. Objetivos generales

Como se mencionaba en la introducción el principal objetivo de este trabajo es el planteamiento, diseño e implementación de una solución prototipo de autolocalización y construcción de mapas (SLAM) para un robot móvil utilizando como principal elemento sensor un escáner láser. Se trata de uno de los componentes que constituirán el prototipo del robot objetivo del pro-

yecto de investigación "Desarrollo de un sistema de transporte autónomo para el almacenaje paletizado" del Grupo de Robótica de la Universidad de Salamanca (GROUSAL http://gro.usal.es). No se trata de un trabajo de investigación básica sino de la propuesta de una solución práctica avanzada que utilice un elemento sensor de elevadas prestaciones (Láser Sick LMS 221) con unas restricciones en cuanto a la capacidad de comportamiento en tiempo real. Asimismo, ha de trabajar en entornos reales aunque también se utilizarán entornos simulados para determinados aspectos del proyecto. Por tanto, para el desarrollo de este trabajo se plantearon una serie de objetivos concretos que se enumeran a continuación:

1. **Integración del LMS.** Para ello se precisará de una herramienta de desarrollo software que permita la integración del mismo sin la necesidad de desarrollar de forma completa un manejador de dispositivo, lo que ya por sí solo constituiría un proyecto. Se plantea en este sentido la utilización de la plataforma *Player/Stage* [31].

2. **Análisis bibliográfico.** Como se ha señalado, se propone la búsqueda de técnicas de generación de mapas, así como de procedimientos de localización relativa y absoluta del robot dentro de un entorno. No se pretenden desarrollar nuevas técnicas de SLAM que se implementen aquellas que se adecuen más al tipo de sensor utilizado y que tengan en consideración de diferentes restricciones que impone el entorno. Así, en cuanto al tiempo del cálculo, se ha tener en cuenta que pueden aparecer necesidades de comportamiento en tiempo real en la ejecución y, por tanto, la sencillez de las propuestas puede ser uno de los criterios a tener en cuenta a la hora de realizar la selección de la técnica a utilizar.

3. **Implementación de técnicas de SLAM.** Se planteará inicialmente la utilización de herramientas de simulación como Matlab debido a que permiten tiempos de desarrollo más cortos para una rápida comprensión y validación de los métodos. No obstante, teniendo en cuenta que el objetivo es desarrollar un prototipo completo, será necesario reali-

zar una implementación optimizada de las técnicas seleccionadas sobre una plataforma hardware que permita su posterior validación. Además se planteará su integración dentro de la plataforma *Player/Stage* ya que se trata de una plataforma comúnmente aceptada en la comunidad científica de la robótica móvil, por su flexibilidad y capacidad para soportar problemas con múltiples robots.

4. **Construcción de una plataforma robótica.** Se construirá una plataforma real sobre la que se montará el sensor láser. Se trata de un equipo de elevadas prestaciones, pero se puede destacar que sus dimensiones y peso (más de 9 Kg.) no son despreciables. Es necesario, por tanto construir una plataforma con capacidad de movimiento (en caso contrario no sería un robot móvil) que permita explorar las capacidades de las soluciones planteadas. Asimismo, se incluirán los elementos de comunicación y procesamiento necesarios.

5. **Diseño, desarrollo e implementación de una herramienta gráfica** que permita la validación de las técnicas de SLAM utilizadas. Esta herramienta ha de permitir la visualización de los datos ofrecidos por el sensor y el posterior procesamiento de la señal obtenida para lograr el objetivo fundamental planteado. Se deberá construir una interfaz cómoda para la elaboración de los diferentes experimentos que permitan comprobar la validez de las técnicas utilizadas. Por último, deberá funcionar en entornos reales y también en entornos simulados, ya que estos permiten un mayor abanico de pruebas experimentales.

2.2. Objetivos de carácter técnico

Con la realización de este proyecto se pretende adquirir cierta experiencia en el desarrollo de proyectos más complejos que los realizados durante las asignaturas de la carrera, y profundizar en ciertos aspectos como pueden ser:

- Encontrar, analizar y afrontar los diversos problemas que surgen du-

rante las distintas etapas del desarrollo.

- Realizar un primer acercamiento y conocimiento del mundo de la robótica.
- Comprender el funcionamiento y manejo de dispositivos electrónicos y robóticos.
- Participar en la tarea de construcción de un robot móvil.
- Iniciarse en el mundo de la investigación al tratar de desarrollar nuevas alternativas e ideas no encontradas en las publicaciones consultadas, al tratar con campos de la informática todavía en fase de desarrollo.
- Contrastar las diferentes alternativas disponibles para realizar la misma tarea, y evaluar y seleccionar la mejor alternativa para un caso concreto.
- Mejorar el conocimiento ya adquirido en ciertas áreas de la informática como son el lenguaje C y los entornos y bibliotecas de libre distribución, como Linux/GNOME, BLAS, LAPACK, etc., observando las ventajas e inconvenientes (falta de documentación, software incompleto o con fallos ...) que presentan.

3

Conceptos Teóricos

3.1. Robótica Móvil

El problema de la navegación de robots móviles se puede resumir en tres preguntas [7]:

- ¿Dónde estoy?
- ¿A dónde voy?
- ¿Cómo debería llegar allí?

En este apartado de la memoria se abordarán los sensores, sistemas, métodos y tecnologías que tratan de responder la primera pregunta: posicionamiento de un robot en su entorno. La conclusión más importante de investigar la gran cantidad de artículos acerca del posicionamiento de los robots móviles es que hasta la fecha no hay una solución elegante y simple para

el problema. La gran cantidad de soluciones parciales de localización de un robot móvil en un entorno concreto se pueden clasificar en dos categorías:

- Sistemas que miden la posición de forma relativa.
- Sistemas que miden la posición de forma absoluta.

Debido a la carencia de un único método general, los desarrolladores de vehículos de navegación automática y de robots móviles, suelen combinar los dos tipos de métodos, uno de cada categoría. Las dos categorías pueden ser a su vez divididas en otros dos subgrupos: medidas de posición relativas (secc. 3.1.1) y medidas de posición absolutas (secc. 3.1.2). A continuación se explicarán dos métodos distintos para estimar la posición: la técnica conocida *dead reckoning* (secc. 3.1.3) y la odometría (secc. 3.1.4). Para finalizar se realizará una comparación entre los distintos tipos de sensores que existen en el mercado para realizar estimaciones de la posición (secc. 3.1.5), así como las ventajas e inconvenientes de la técnica de estimación de la posición basada en la comparación de mapas (secc. 3.1.6).

3.1.1. Medidas de posición relativas

En estos sistemas la posición viene dada en función de un origen. Si se perdiese ese punto de partida, el robot estaría completamente desorientado, sin poder ser capaz de volver a saber dónde se encuentra, salvo eligiendo un nuevo origen. Son principalmente dos: odometría y navegación inercial.

Odometría

Este método usa *encoders* para medir la rotación de las ruedas. A partir de las ecuaciones cinemáticas del robot es posible calcular la posición y orientación del mismo en todo momento. La ventaja de la odometría es que es un sistema contenido totalmente dentro del robot, no depende de su ambiente externo, y además siempre es capaz de proporcionar al vehículo una estimación de la posición. La desventaja que presenta es que los errores en

la posición aumentan ilimitadamente a menos que se utilice periódicamente una referencia independiente para reducir el error.

Navegación inercial

Este método utiliza giróscopos y algunas veces acelerómetros para medir la tasa de rotación y aceleración. Al igual que la odometría, estos sistemas también se encuentran completamente contenidos dentro del robot. Como inconveniente, cualquier pequeño error puede derivar con el tiempo en grandes diferencias con la realidad. Por lo tanto, los sensores inerciales no son adecuados para el posicionamiento exacto cuando se utiliza para largos tiempos de navegación. Otro inconveniente es el alto coste de los componentes. Por ejemplo, los giróscopos de alta precisión, que se usan en los aviones, tienen unos precios prohibitivos. Sin embargo, ya existen en el mercado desde hace poco tiempo unos giróscopos basados en láser, han caído de precio espectacularmente y se han convertido en una solución muy atractiva para la navegación de robots móviles.

3.1.2. Medidas de posición absolutas

Estos sistemas no dependen de un origen de referencia, y en todo momento el robot puede reencontrarse si hubiese perdido la orientación. A continuación se comentan los principales sistemas de posicionamiento absoluto.

Balizas activas

Este método calcula la posición absoluta del robot midiendo la dirección de incidencia de tres o más balizas activas distribuidas por la habitación. Las balizas transmiten radio frecuencia o luz láser para comunicarse con el robot. Estas balizas deberán estar en lugares bien localizados dentro del entorno de trabajo del robot.

Reconocimiento de marcas artificiales en el terreno

En este método se colocan marcas artificiales en posiciones conocidas en el entorno en el que se vaya a mover el robot. La ventaja de este método es que las marcas pueden ser diseñadas para que sean detectadas incluso bajo las circunstancias más adversas. Al igual que con las balizas activas, tres o más marcas deben ser reconocidas al mismo tiempo, para que se logre realizar una estimación de la posición. La ventaja de este sistema es que los errores en la posición no pueden crecer indefinidamente, pero puede que no siempre sea posible la detección de marcas junto con el ajuste de la posición en tiempo real. Este tipo de marcas pueden ser definidas como un conjunto de características, como por ejemplo una forma o un área, y a distinción del sistema anterior, no son las balizas las que emiten señales para comunicarse con el robot, sino que es el robot el que trata de localizarlas, ya que no emiten ningún tipo de señal.

Reconocimiento de marcas naturales

En este caso las marcas que se deben reconocer son características distintivas que pertenecen al propio entorno de trabajo. No es necesario preparar el entorno, pero éste debe ser conocido previamente. Si se tratase de una habitación podríamos tomar como referencia las esquinas, una puerta, una columna... La fiabilidad de este método no es tan alta como con las marcas artificiales.

Comparación de modelos

En este método la información adquirida mediante los sensores del robot es comparada con un mapa que previamente se ha obtenido del entorno. Si las características del mapa local obtenido por los sensores coinciden con algún fragmento del mapa global, entonces puede ser estimada la posición absoluta del vehículo. El posicionamiento basado en mapas suele ampliar los mapas globales utilizando los nuevos datos que los sensores van obtenien-

do. Para ello es necesario integrar los mapas locales en los mapas globales cubriendo áreas del entorno que no han sido exploradas con anterioridad. Los mapas utilizados en la navegación se pueden agrupar en dos tipos: mapas geométricos y mapas topológicos. Los mapas geométricos representan el mundo utilizando un sistema de coordenadas global, mientras que los mapas topológicos representan el mundo como una red de nodos y arcos.

3.1.3. *Dead Reckoning*

Dead reckoning significa en castellano algo así como "cálculo a ojo". A lo largo de esta memoria se ha optado por no traducir el término por no haber una clara traducción al castellano. *Dead reckoning* es una expresión derivada del término náutico *deduced reckoning* (cálculo basado en inferencia), y era un procedimiento matemático simple para inferir la ubicación actual de un navío haciendo cálculos basados en el rumbo y la velocidad de navegación a lo largo de un período de tiempo, sin usar el cielo y los astros como referencia. La amplia mayoría de sistemas de robots móviles terrestres que se usan hoy en día, utilizan la técnica *dead reckoning* como columna vertebral de su estrategia de navegación, y al igual que sus homólogos náuticos, necesitan eliminar los errores acumulados con continuos ajustes con varios sistemas de ayuda a la navegación.

La implementación más simple posible de la técnica *dead reckoning* es habitualmente llamada odometría, término que implica que el desplazamiento de un vehículo a lo largo de la trayectoria se deriva directamente de algún odómetro o cuentarrevoluciones a bordo del vehículo. Un técnica común para implementar la odometría consiste en utilizar *encoders* ópticos directamente acoplados a los motores o a los ejes de las ruedas.

Existen multitud de sensores para cuantificar el desplazamiento angular y la velocidad, como por ejemplo: potenciómetros, sincros, resolvers, *encoders* ópticos, *encoders* magnéticos, *encoders* inductivos, *encoders* de capacidad.

3.1.4. Odometría

La odometría es uno de los métodos más ampliamente usados para estimar la posición de un robot. Es bien sabido que la odometría proporciona una buena precisión a corto plazo, es barata de implantar, y permite tasas de muestro muy altas. Sin embargo la idea fundamental de la odometría es la integración de información incremental del movimiento a lo largo del tiempo, lo cual conlleva una inevitable acumulación de errores. En concreto, la acumulación de errores de orientación, causa grandes errores en la estimación de la posición, los cuales van aumentando proporcionalmente con la distancia recorrida por el robot. A pesar de estas limitaciones, muchos investigadores están de acuerdo en que la odometría es una parte importante del sistema de navegación de un robot, y que debe usarse con medidas del posicionamiento absolutas para proporcionar una estimación de la posición más fiable.

Errores sistemáticos y no sistemáticos en la odometría

La odometría se basa en ecuaciones simples que se pueden implementar fácilmente y que utilizan datos de *encoders* situados en las ruedas del robot. Sin embargo, la odometría también está basada en la suposición de que las revoluciones de las ruedas pueden ser traducidas en un desplazamiento lineal relativo al suelo. Esta suposición no tiene una validez absoluta. Un ejemplo extremo es cuando las ruedas patinan: si por ejemplo, un rueda patina sobre una mancha de aceite y la otra no, entonces el *encoder* asociado registrará revoluciones en la rueda, a pesar de que éstas no correspondan a un desplazamiento lineal de la rueda. Además de este ejemplo hay muchas otras razones más sutiles por las cuales se pueden producir imprecisiones en la traducción de las lecturas del *encoder* de la rueda a un desplazamiento lineal. Todos estos errores se pueden agrupar en dos categorías: errores sistemáticos, y errores no sistemáticos.

Entre los errores sistemáticos destacan:

- Los diámetros de las ruedas no son iguales.

- La media de los diámetros de las ruedas difieren del diámetro de fábrica de las ruedas.
- Mal alineamiento de las ruedas.
- Resolución discreta (no continua) del *encoder*.
- La tasa de muestreo del *encoder* es discreta.

Entre los errores no sistemáticos se encuentran:

- Desplazamiento en suelos desnivelados.
- Desplazamiento sobre objetos inesperados que se encuentren en el suelo.
- Patinaje de las ruedas debido a:
  - Suelos resbaladizos.
  - Sobre-aceleración.
  - Derrapes (debidos a una rotación excesivamente rápida).
  - Fuerzas externas (interacción con cuerpos externos).
  - No hay ningún punto de contacto con el suelo.

Una clara distinción entre errores sistemáticos y no sistemáticos es de gran importancia a la hora de reducir los errores en la odometría. Por ejemplo, los errores sistemáticos son específicamente graves, porque se acumulan constantemente. En muchas superficies no rugosas de entornos interiores, los errores sistemáticos contribuyen muchos más a los errores en la odometría que los errores no sistemáticos. Sin embargo, en superficies que agarran bien con irregularidades significativas, son los errores no sistemáticos los que predominan. El problema de los errores no sistemáticos es que pueden aparecer inesperadamente (por ejemplo cuando el robot pasa por encima de un objeto que se encuentra en el suelo), y pueden causar errores muy grandes en la estimación de la posición. Cabe destacar que muchos investigadores han desarrollado algoritmos para estimar la incertidumbre en la posición de

un robot que utiliza odometría (p.ej. [Tonouchi et al., 1994], Komoriya and Oyama, 1994].) Según estos enfoques, cada posición calculada por el robot está rodeada por una "elipse de error" característica, la cual indica la región de incertidumbre para la posición actual del robot, como se puede apreciar en la figura 3.1.

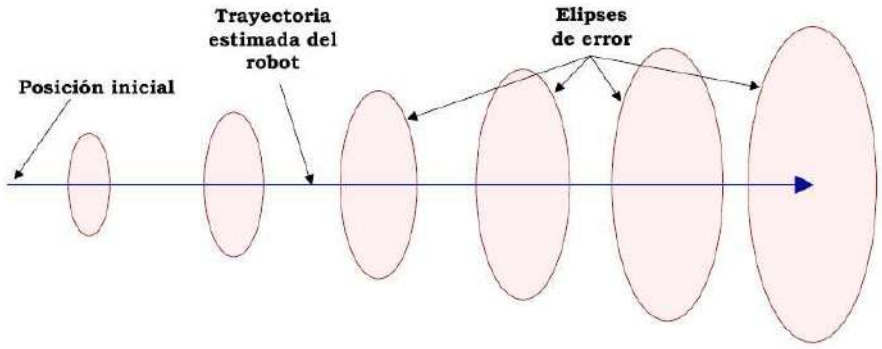

Figura 3.1: *Las "elipses de error" crecen, indicando esto que con la odometría crece la incertidumbre sobre cual es la posición real.*

Estas elipses crecen a medida que la distancia recorrida aumenta, a no ser que un sistema de estimación de la posición absoluto reduzca el crecimiento de la incertidumbre y por lo tanto ponga a cero el tamaño de la "elipse de error". Estas técnicas de estimación del error se basan en estimación de parámetros derivados de los errores sistemáticos, puesto que la magnitud de los errores no sistemáticos es siempre impredecible.

3.1.5. Sensores para el posicionamiento basado en mapas

La mayoría de sensores que se utilizan con el propósito de construir mapas suponen tener que medir distancias de un modo u otro. Básicamente existen tres tipos de técnicas para medir distancias:

- Sensores basados en la medición del tiempo de vuelo (conocido normalmente como TOF, *time of flight*) de un pulso de energía viajando hasta un objeto reflectante, el cual devuelve un eco hasta el emisor.

- Medida del cambio de fase (o detección de fase), en la que se transmiten continuamente ondas, a distinción de los pulsos cortos utilizados en los sistemas TOF.
- Sensores basados en un radar de frecuencia modulada (FM). Esta técnica está un tanto relacionada con la técnica de medición (basada en amplitud modulada) de cambio de fase.

Sensores de medición de tiempo de vuelo

Muchos de los sensores de hoy en día utilizan el método de tiempo de vuelo (TOF). Los pulsos utilizados típicamente provienen de una fuente de energía:

- Óptica
- Ultrasónica
- De radiofrecuencia.

Por lo tanto los parámetros relevantes que intervienen en el cálculo de la distancia son:

- La velocidad del sonido en el aire (aproximadamente 0,3m/ms).
- La velocidad de la luz. (aproximadamente 0,3m/ns)

Utilizando la física más elemental, la distancia es hallada multiplicando la velocidad de la onda de energía por el tiempo requerido para hacer el viaje de ida y vuelta hasta el objeto más cercano:

$d = vt$

donde:

d = distancia de ida y vuelta.
v = velocidad de propagación.
t = tiempo transcurrido.

El tiempo medido es el correspondiente a dos veces la distancia a la que se encuentra el objeto (esto es, la distancia de ida más la distancia de vuelta), y por lo tanto debe reducirse a la mitad para calcular la distancia real al objeto.

Las ventajas de los sistemas que utilizan la técnica del tiempo de vuelo están directamente relacionada con el hecho de que el sensor funciona en línea recta. El pulso de energía que refleja un determinado objeto sigue esencialmente el mismo camino de vuelta hacia un receptor situado en el mismo eje en el que el emisor envió el pulso, o a muy poca distancia de este. De hecho, es posible que en algunos casos tanto el transductor emisor como el transductor receptor estén integrados en el mismo dispositivo. La distancia absoluta hasta un punto observado está disponible directamente sin necesidad de realizar complejos análisis, y además esta técnica no se basa en ninguna suposición relativa a las propiedades o la orientación de la superficie objetivo. En este tipo de sistemas no aparecen problemas relacionados con la triangulación, como es el caso de los sistemas basados en balizas o marcas. Además la precisión de la medida se mantiene de modo lineal siempre que se realice una detección fiable del eco del pulso, mientras que en los esquemas que usan triangulación sufren una disminución de la precisión a medida que aumenta la distancia a las balizas o marcas.

No obstante los sistemas de medición de tiempo de vuelo, no quedan libres de errores. Los siguientes son algunos de los errores más habituales:

- Variaciones en la velocidad de propagación, particularmente en el caso de sistemas acústicos.
- Incertidumbres en la determinación del tiempo exacto de llegada del pulso reflejado
- Imprecisiones en el sistema de circuitos usado para medir el tiempo de vuelo de ida y vuelta.
- Interacciones de las ondas incidentes en la superficie objetivo.

Cada uno de estos problemas será tratado brevemente a continuación.

Velocidad de propagación: Para la mayoría de las aplicaciones de robótica móvil, los cambios en la velocidad de propagación de la energía electromagnética son intrascendentes, y pueden ser ignorados. Sin embargo, este no es el caso para los sistemas basados en acústica, donde la velocidad del sonido es influenciada de forma notable por los cambios en la temperatura, y en menor grado por la humedad. (La velocidad del sonido en realidad es proporcional a la raíz cuadrada de la temperatura en *grados Rankine*). Un cambio en la temperatura ambiente de solamente ($-1{,}1\,^{\circ}\mathrm{C}$) puede causar un error de 0.3 metros cuando se está midiendo un objeto a una distancia de 10 metros.

Incertidumbres en la detección: También se denomina habitualmente "errores en el tiempo de viaje". Son causados por el amplio rango de intensidades de la señal devuelta, debido a reflexiones variables en las superficies. Estas diferencias en la intensidad de la señal influyen en el tiempo de detección del pulso, y en el caso de un umbral fijo de detección, hará que los objetos más reflectantes aparezcan más cerca.

Consideraciones acerca del tiempo: Debido a la velocidad relativamente baja del sonido en el aire, en comparación con la luz, los sistemas basados en acústica tienen requisitos de tiempo menos rigurosos que sus homólogos basados en luz, y por lo tanto son menos caros. A la inversa, la velocidad de propagación de la energía electromagnética tiene unos requisitos más severos para el control asociado y los circuitos de medición, tanto en el caso de sistemas ópticos como en el caso de la radiofrecuencia. Como consecuencia, los sensores de tiempo de vuelo basados en la velocidad de la luz requieren sistemas de circuitos que respondan en tiempos de orden inferior al nanosegundo, para lograr medir distancias con una resolución inferior a 30 centímetros. Más específicamente, para conseguir precisión del orden del milímetro, se requiere una precisión en el sistema de circuitos de 3 pico segundos ($3 \times 10^{-12} s$). Esta alta precisión es un tanto cara de conseguir y puede que el coste no sea adecuado para ciertas aplicaciones, particularmente en distancias cortas, que

es precisamente donde se requiere una alta precisión.

Interacción con las superficies: Cuando la luz, el sonido, o las ondas de radio inciden en un objeto, cualquier eco detectable representa tan sólo una pequeña porción de la señal original. La energía restante se refleja en todas las direcciones y puede ser absorbida por el objeto o puede pasar a través de él, dependiendo de las características de la superficie, y del ángulo de incidencia del haz. Se pueden dar casos en los que no se recibe ninguna señal, debido a la reflexión especular de la superficie del objeto, especialmente en la región ultrasónica del espectro de energía. Si el ángulo de la fuente emisora es igual o mayor que un cierto valor crítico, la energía reflejada se verá desviada fuera de las dimensiones del sensor receptor. En determinados ambientes (normalmente llenos de objetos) las ondas de sonido pueden reflejarse en múltiples objetos, y pueden ser recibidas por otros sensores distintos del emisor. Este fenómeno es conocido como interferencia (*crosstalk*), y se puede apreciar en la figura 3.2.

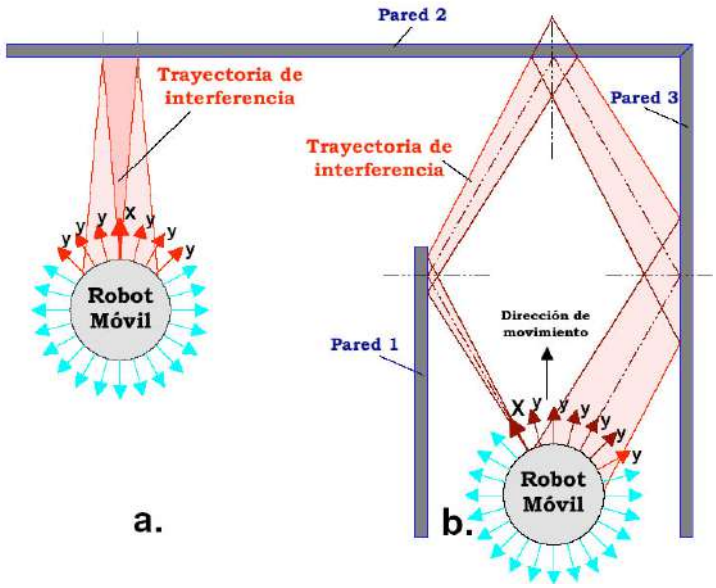

Figura 3.2: *La interferencia (crosstalk) es un fenómeno en el cual un sonar recibe el eco de otro. Se puede distinguir entre interferencia directa (a) e interferencia indirecta (b).*

Para compensar las interferencias, se hace la media de varias medidas con respecto al mismo objeto, para obtener la proporción entre señal y ruido dentro de unos niveles aceptables, pero con el gasto adicional del tiempo requerido para determinar una única medida de distancia.

Sistemas ultrasónicos basados en tiempo de vuelo

La medición de distancias utilizando técnicas TOF basadas en ultrasonidos es hoy en día el método más común empleado en sistemas de robots móviles para navegación en interiores, principalmente debido a su alta disponibilidad, costes reducidos y su facilidad para interactuar con ellos. En las últimas dos décadas se han realizado multitud de investigaciones para aplicar este tipo de sistemas en áreas como el modelado de mundos, evitación de obstáculos, estimación de la posición y detección de movimiento. Varios investigadores han comenzado hace poco tiempo a evaluar la efectividad de sensores de ultrasonidos en exteriores.

Sistemas láser basados en tiempo de vuelo

Los sistemas láser que utilizan técnicas TOF para la medición de distancias, se conocen también como radar láser, o *lidar* (*light detection and ranging*). Aparecieron por primera vez en trabajos realizados en el "Jet Propulsion Laboratory", en Pasadena, California, en la década de 1970. [Lewis y Jonson, 1977]. La energía del láser es emitida rápidamente en una secuencia muy rápida de pulsos cortos dirigidos directamente hacia el objeto que está siendo medido. El tiempo requerido para que un pulso dado se refleje en el objeto y vuelva hasta el foco emisor, es medido y usado para calcular la distancia al objeto, basándose este cálculo en la velocidad de la luz.

Diferencias entre *lidar* y radar: La principal diferencia entre lidar y radar es que en el primer sistema se usan longitudes de onda electromagnéticas mucho más pequeñas. En general es posible trabajar con objetos que sean del mismo tamaño que la longitud de onda, o mayores. Por lo tanto

cuanto menor sea la longitud de onda, menor será el tamaño de los objetos o partículas que se pueden llegar a detectar.

Para reflejar la onda transmitida, un objeto necesita producir una discontinuidad dieléctrica. Con las frecuencias que utiliza el radar (microondas de radio), un objeto metálico produce un reflejo significativo. Sin embargo, los objetos no metálicos, tales como las rocas por ejemplo, o incluso las gotas de lluvia, producen ecos más débiles, e incluso algunos materiales no producen ningún tipo de eco. Esto se traduce en que algunos objetos o características son realmente invisibles a las frecuencias del radar.

Los láseres ofrecen una solución a este problema. Las longitudes de onda que se pueden lograr con los láser son muchísimo más pequeñas que las que se utilizan en los sistemas de radio, y abarcan desde los 10 micrómetros hasta el ultravioleta (250 nm). Con estas longitudes de onda tan cortas, los sistemas lidar ofrecen una resolución mucho mayor que el radar. Gracias a la menor longitud de onda de los láseres, son más convenientes para utilizar en interiores, puesto que son capaces de lograr ecos más potentes, mientras que el radar en ese tipo de entornos recibiría ecos débiles de la mayoría de objetos, exceptuando los metálicos.

3.1.6. Posicionamiento basado en mapas

El posicionamiento basado en mapas, también conocido como **comparación de mapas** (*map matching*), es una técnica en la cual el robot usa sus sensores para crear un mapa de su entorno local. Este mapa local es comparado posteriormente con un mapa **global** previamente almacenado en la memoria. Si se encuentra un mapa bastante parecido al comparar, entonces el robot puede calcular su posición y orientación reales en su entorno. El mapa prealmacenado en memoria puede ser un modelo CAD del entorno o puede que se haya construido a partir de los datos previos de los sensores del robot.

El procedimiento básico para el posicionamiento basado en mapas se muestra figura 3.3.

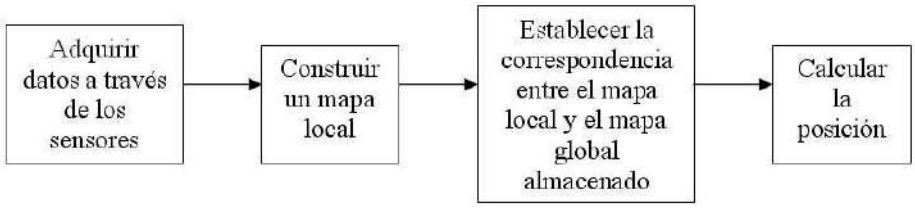

Figura 3.3: *Posicionamiento basado en mapas*

Las principales ventajas del posicionamiento basado en mapas son las siguientes:

- Es un método que usa la estructura natural típica en entornos interiores (dentro de los edificios) para calcular la posición del robot sin tener que modificar el ambiente (al contrario que si utilizamos balizas activas o marcas artificiales)
- El posicionamiento basado en mapas puede usarse para generar un mapa actualizado del entorno. Los mapas del entorno son muy importantes para otras tareas de robots móviles, como pueden ser el cálculo de trayectorias globales o la evitación de trampas debidas a mínimos locales en algunos métodos utilizados para evitar obstáculos locales.
- Permite a un robot aprender de forma autónoma partes del entorno que no conoce, y aumentar la precisión de su localización a través de la exploración.

Las desventajas del posicionamiento basado en mapas son los requisitos específicos para lograr una navegación aceptable. Por ejemplo, estos sistemas necesitan:

- Que haya suficientes características distinguibles y fijas que puedan usarse para establecer correspondencias entre el mapa local y el mapa global.
- Que el sensor que se utilice para crear los mapas sea suficientemente preciso (dependiendo de las tareas) para ser útil.

- Que esté disponible una importante cantidad de potencia de procesamiento.

3.2. Análisis de Componentes Principales

El análisis de componentes principales [13] es una técnica estadística, y por lo tanto utiliza como base la idea de que se tiene un gran conjunto de datos, y se pretende analizar ese conjunto basándose en las relaciones entre los datos individuales en ese conjunto de datos.

3.2.1. Media, Desviación estándar, Varianza

A continuación se comentarán diferentes medidas que podemos realizar sobre un conjunto de datos, y qué nos dicen acerca de ese conjunto de datos. Estas medidas son la desviación estándar, la media y la varianza.

En estadística se toma como base una muestra de la población. Por ejemplo, en los sondeos electorales, la población es toda la gente que habita el país, mientras que una muestra es un subconjunto de la población que se va a medir. Lo maravilloso de la estadística es que evaluando tan solo una muestra de la población, podemos obtener prácticamente los mismos resultados que si hubiéramos trabajado con la población total.

De aquí en adelante se asumirá que los conjuntos de datos que se tomen como ejemplo serán muestras de una población mayor.

Tomando como ejemplo la siguiente muestra de datos:

$$X = \left[\begin{array}{cccccccccccc} 1 & 2 & 4 & 6 & 12 & 15 & 25 & 45 & 68 & 67 & 65 & 98 \end{array} \right]$$

X_i se refiere al i-ésimo elemento de X, y n indica el número de elementos de X.

Media

La medida más sencilla que se puede calcular sobre este conjunto de datos es la media muestral, cuya fórmula es:

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

La media no dice mucho acerca de los datos, excepto obviamente cuál es el punto medio. Por ejemplo, los siguientes dos conjuntos de datos tienen la misma media (10), pero son muy diferentes:

$$Y = \begin{bmatrix} 0 & 8 & 12 & 20 \end{bmatrix}$$

$$Z = \begin{bmatrix} 8 & 9 & 11 & 12 \end{bmatrix}$$

Varianza

Lo que los diferencia es la dispersión de los datos. La desviación estándar de un conjunto de datos es una medida de cuánta dispersión tienen los datos. Se define como "el promedio de la distancia de cada punto con respecto a la media de la muestra". Su fórmula es:

$$s = \sqrt{\frac{\sum_{i=1}^{n} \left(X_i - \bar{X}\right)^2}{n-1}}$$

Como era de esperar, el conjunto Y tiene una desviación estándar mucho mayor (8.3266) que el conjunto Z (1.8257), debido a que los datos están mucho más dispersos respecto de la media.

Varianza

La varianza es otra medida de la dispersión de los datos de un conjunto. De hecho, es casi idéntica a la desviación estándar. Se calcula mediante la expresión:

$$s^2 = \frac{\sum_{i=1}^{n} \left(X_i - \bar{X}\right)^2}{n-1}$$

Tanto la desviación estándar como la varianza calculan la dispersión de los datos, y a pesar de que la primera suele ser la más utilizada, la varianza también es importante y juega un papel fundamental en el cálculo de la covarianza.

3.2.2. Covarianza y Matriz de Covarianza

Las medidas de la sección anterior anteriores son meramente unidimensionales. Conjuntos de datos de una sola dimensión podrían ser "alturas de todas las personas de una clase", "notas del último examen de ingeniería del software", etc. Sin embargo, muchos conjuntos de datos tienen varias dimensiones, y el propósito del análisis estadístico para este tipo de conjuntos es observar si existe alguna relación entre las dimensiones. Por ejemplo, se podría tomar como conjunto de datos la altura de las personas de una clase y la nota que han recibido en el examen. Luego se podría hacer un análisis estadístico para comprobar si la altura del estudiante tiene algún tipo de relación con su nota. La desviación estándar y la varianza operan solamente en una dimensión, de modo que solo podría calcularse la desviación estándar para cada dimensión de los datos, independientemente de las otras dimensiones. Sin embargo es útil tener una medida similar para averiguar cuanto varían las dimensiones entre ellas y con respecto a la media. La **covarianza** es esta medida, y siempre se mide entre dos dimensiones. Si se calcula la covarianza entre una dimensión y ella misma, obtendríamos la varianza. Dado un conjunto de datos tridimensional (x, y, z), se puede medir la covarianza entre x e y, entre x y z, y entre y y z.

La fórmula de la covarianza se expresa de la siguiente manera:

$$cov\,(X,Y) = \frac{\sum_{i=1}^{n}\left(X_i - \bar{X}\right)\cdot\left(Y_i - \bar{Y}\right)}{n-1}$$

Considérese el siguiente ejemplo. Se ha preguntado a cierto número de estudiantes el número de horas que han estudiado y la nota que han sacado en un examen de programación. Existen dos dimensiones, una el número de horas estudiadas H, y otra N, la nota recibida. La tabla 3.1 muestra unos datos ficticios así como los valores correspondientes de $cov(H, N)$, la covarianza entre las horas de estudio y la nota recibida.

El valor exacto de la covarianza no importa, lo importante es su signo.

- Si el valor es positivo, como es en el caso de la tabla 3.1, indica que

(a) Datos

	$Horas(H)$	$Nota(N)$
Datos	9	39
	15	56
	25	93
	14	61
	10	50
	18	75
	0	32
	16	85
	5	42
Total	112	533
Media	12.44	59.22

(b) Covarianza

H	N	$\left(H_i - \bar{H}\right)$	$\left(N_i - \bar{N}\right)$	$\left(H_i - \bar{H}\right) \cdot \left(N_i - \bar{N}\right)$
9	39	-4.92	-23.42	115.23
15	56	1.08	-6.42	-6.93
25	93	11.08	30.58	338.83
14	61	0.08	-1.42	-0.11
10	50	-3.92	-12.42	48.69
18	75	4.08	12.58	51.33
0	32	-13.92	-30.42	423.45
16	85	2.08	22.58	46.97
5	42	-8.92	-20.42	182.15
Total			1199.61	
Media			133.29	

Tabla 3.1: *Cálculo de la covarianza.*

ambas dimensiones crecen juntas, y por lo tanto cuantas más horas se haya estudiado, mejor es la nota recibida en el examen.

- Si el valor es negativo, significa que cuando una dimensión crece, la otra disminuye. Es decir, si se estudia más, se obtendría peor nota.
- Si la covarianza es cero, quiere decir que las dos dimensiones son totalmente independientes.

Puesto que la covarianza puede ser calculada entre dos dimensiones cualesquiera de un conjunto de datos, esta técnica se utiliza para encontrar relaciones entre conjuntos de datos de un gran número de dimensiones, en los cuales la visualización es compleja.

Matriz de covarianza

Para un conjunto de datos n-dimensional, se pueden calcular $\frac{n!}{(n-2)!\cdot 2}$ valores de covarianza diferentes, puesto que es posible calcular la covarianza entre dos dimensiones cualquiera.

Una manera útil de tener todas las covarianzas posibles entre las distintas dimensiones, es calcularlas todas juntas y disponerlas en una matriz. La definición para la matriz de covarianza de un conjunto de datos n-dimensional es:

$$C^{n\times n} = (c_i, c_j = cov(Dim_i, Dim_j)),$$

donde $C^{n\times n}$ es una matriz con n filas y n columnas y Dim_i es la i-ésima dimensión. Todo lo que nos dice esta fórmula es que el valor de la fila 2 columna 3, por ejemplo, es la covarianza entre la segunda y la tercera dimensión.

Por ejemplo, una matriz de covarianza para un conjunto de datos tridimensional (x, y, z), tendría el siguiente aspecto:

$$C^{n\times n} = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(y,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

A lo largo de la diagonal principal, el valor de la covarianza es entre una dimensión y ella misma, esto es equivalente a la varianza para dicha dimensión. Puesto que $cov(a, b) = cov(b, a)$, la matriz es simétrica con respecto a la diagonal principal.

3.2.3. Vectores propios

Consideremos el ejemplo de multiplicaciones entre una matriz y un vector que se puede apreciar en la figura 3.4.

$$\text{(a)} \quad \begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 11 \\ 5 \end{pmatrix}$$

$$\text{(b)} \quad \begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 8 \end{pmatrix} = 4 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Figura 3.4: *Ejemplo de vector no propio (a) y vector propio (b).*

En (a) el vector resultante no es un entero múltiplo del vector original, mientras que en (b), el resultado es exactamente cuatro veces el vector de partida. El vector $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ del apartado (b), se representa como una flecha desde el origen $(0, 0)$ al punto $(3, 2)$. La otra matriz que está delante, puede ser pensada como una matriz de transformación. Si se multiplica esta matriz a la izquierda por el vector, el resultado es otro vector que está transformado respecto de su posición original. De aquí es de donde surgen los vectores propios. Si se toma una matriz tal que, cuando la multiplicamos a la izquierda de un vector, nos da como resultado vectores en la recta $y = x$. Entonces, si se elige un vector en la recta $y = x$, su resultado es él mismo (con mayor longitud o menos, pero sobre la recta $y = x$). Este vector, junto con todos sus múltiplos, ya que no importa cual sea su longitud, sería un **vector propio** de esta matriz de transformación.

Los vectores propios tienen ciertas propiedades, como por ejemplo, que sólo se pueden encontrar en las matrices cuadradas (no en todas ellas). Para

una matriz de dimensiones $n \times n$ que sí tenga vectores propios, solamente hay n. Esto es, dada una matriz 3×3, habrá 3 vectores propios.

Otra propiedad de los vectores propios es que incluso si se modifica su longitud, antes de multiplicarlo, el resultado después de la multiplicación será exactamente el mismo, como se puede apreciar en la figura 3.5.

$$2 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 24 \\ 16 \end{pmatrix} = 4 \times \begin{pmatrix} 6 \\ 4 \end{pmatrix}$$

Figura 3.5: *Ejemplo de comó un vector propio escalado sigue siendo un vector propio.*

Esto ocurre porque si hacemos más largo o más corto el vector, solamente modificamos su longitud, pero no estamos cambiando su dirección.

Finalmente, cabe decir que todos los vectores propios de una matriz son perpendiculares u ortogonales, no importa cuántas dimensiones haya. Esto es muy importante, porque quiere decir, que se pueden expresar los datos en términos de estos vectores propios ortogonales, en lugar de expresarlos en términos de los ejes x e y. Esto se verá en un apartado posterior.

Una vez encontrados los vectores propios, se suelen normalizar, de modo que todos tengan longitud 1.

Sea el vector $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$,cuya longitud es $\sqrt{(3^2 + 2^2)} = \sqrt{(13)}$. entonces dividiendo el vector por su longitud , el vector resultante tiene longitud uno: está normalizado.

$$\begin{pmatrix} 3 \\ 2 \end{pmatrix} \div \sqrt{(13)} = \begin{pmatrix} 3 \div \sqrt{(13)} \\ 2 \div \sqrt{(13)} \end{pmatrix}$$

3.2.4. Valores propios

Los valores propios están muy relacionados con los vectores propios, de hecho, en la figura 3.4(b) aparece uno. En ambos ejemplos (figuras 3.4(b) y

3.5), la cantidad por la que es modificado el vector original es la misma: 4. Este valor es lo que se denomina **valor propio** asociado al vector propio. No importa qué múltiplo del vector propio tomemos antes de multiplicarlo por la matriz cuadrada, siempre se obtiene como resultado cuatro veces el vector original.

3.2.5. Método PCA

El análisis de componentes principales (PCA, *principal components analysis*) es un método para la identificación de patrones en los datos; además es una forma de expresar los datos de modo que se destaquen sus similitudes y diferencias. Puesto que los patrones están ocultos en los datos, pueden ser difíciles de encontrar cuando éstos tienen muchas dimensiones, y ya que no es fácil realizar una representación gráfica en más de tres dimensiones, el análisis de componentes principales es una herramienta muy potente para el análisis de datos. La otra ventaja principal de PCA es que una vez que se han encontrado los patrones en los datos, se puede reducir la dimensionalidad de los datos, sin tener gran pérdida de información. Esta posibilidad de compresión permite utilizar PCA para la compresión de imágenes. A continuación se describirá el modo de funcionamiento de esta técnica.

Paso1: Obtención de los datos

A lo largo de esta explicación se utilizará un conjunto de datos bidimensional, de modo que se pueda comprender mejor el funcionamiento de lo que está ocurriendo en cada paso de PCA. Los datos usados se presentan en la tabla 3.2, y en la figura 3.6 se muestra una gráfica de dichos datos.

Paso 2: Normalición

Para que PCA funcione correctamente, es necesario normalizar los datos, es decir, restar la media de cada dimensión de datos. Por lo tanto, a todos los valores x se les restará su media $\bar{x}$, y a todos los valores y se les restará su

(a) Sin normalizar

	x	y
	2.5	2.4
	0.5	0.7
	2.2	2.9
	1.9	2.2
	3.1	3.0
	2.3	2.7
	2	1.6
	1	1.1
	1.5	1.6
	1.1	0.9
Media	1.81	1.91

(b) Normalizados

x	y
0.69	0.49
-1.31	-1.21
0.39	0.99
0.09	0.29
1.29	1.09
0.49	0.79
0.19	-0.31
-0.81	-0.81
-0.31	-0.31
-0.71	-1.01

Tabla 3.2: *Datos de ejemplo para el análisis PCA*

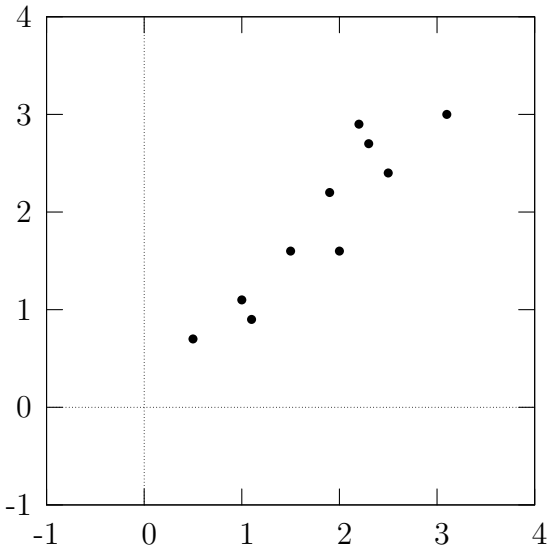

Figura 3.6: *Datos de ejemplo para el análisis PCA*

media $\bar{y}$. Esto produce un conjunto de datos cuya media es cero.

Paso 3: Cálculo de la matriz de covarianza

Esto se hace exactamente igual que como se expuso en la sección 3.2.2. Puesto que los datos son bidimensionales, la matriz de covarianza será 2×2. El resultado es el siguiente:

$$cov = \begin{pmatrix} 0{,}616555556 & 0615444444 \\ 0{,}615444444 & 0{,}716555556 \end{pmatrix}$$

Como los elementos que no están en la diagonal son positivos, es de esperar que las variables x e y aumenten juntas.

Paso 4: Cálculo de los vectores y valores propios de la matriz de covarianza

La matriz de covarianza es cuadrada, luego podemos calcular los vectores y valores propios. Son muy importantes, ya que proporcionan información importante acerca de los datos. Los resultados son los siguientes:

$$\text{valores propios} = \begin{pmatrix} 0{,}49083398 \\ 1{,}28402771 \end{pmatrix}$$

$$\text{vectores propios} = \begin{pmatrix} -0{,}735178656 & -0{,}677873399 \\ 0{,}677873399 & -0{,}735178656 \end{pmatrix}$$

La longitud de estos dos vectores propios es 1, es decir, son *unitarios*. En la figura 3.7, se puede ver que existe un patrón muy fuerte en los datos. Como se esperaba al analizar la matriz de covarianza, las dos variables aumentan a la vez.

Se puede apreciar que los vectores propios son perpendiculares entre sí, pero lo más importante es que proporcionan información acerca de los patrones en los datos. Si se prolonga el vector propio que apunta hacia abajo a la izquierda, se puede imaginar cómo atraviesa los puntos como si fuera una

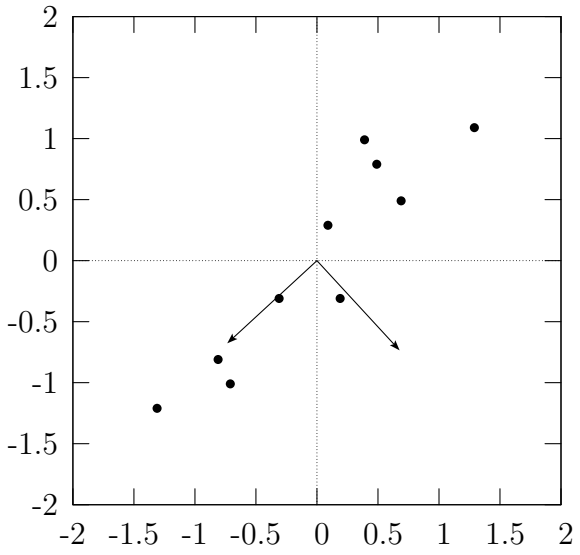

Figura 3.7: *Gráfico de los datos normalizados junto con los vectores propios de la matriz de covarianza.*

línea de ajuste. Dicho vector propio indica como los dos conjuntos de datos x e y están relacionados a lo largo de esa línea. El segundo vector propio (apuntando hacia abajo a la derecha) revela el otro patrón que existe en los datos, menos importante, pero que indica que todos los puntos están a lo largo de la línea del primer vector propio, pero se desvían de ella una cierta cantidad, siguiendo la dirección del segundo vector propio. Por lo tanto, con el proceso de cálculo de los vectores propios de la matriz de covarianza, es posible extraer líneas que caracterizan a los datos.

Paso 5: Elección de los componentes y formación de un vector característico

Si se observan los vectores y valores propios del paso anterior, es claro que los valores propios son muy diferentes. De hecho resulta que el vector propio con *mayor* valor propio es el *componente principal* del conjunto de datos. En el ejemplo anterior, el vector propio con mayor valor propio era el que apuntaba hacia abajo a la izquierda.

En general, una vez que se han encontrado los vectores propios de la matriz de covarianza, el siguiente paso es ordenarlos por su valor propio, de mayor a menor; de este modo se ordenan por importancia. Una vez hecho esto, se puede decidir si *ignorar* los componentes de menor importancia. Así se pierde algo de información, pero si los valores propios son pequeños, no se pierde mucha. En caso de prescindir de algunos componentes, el conjunto de datos final tendrá menos dimensiones que el original. Para ser precisos, si en un principio se tenían n dimensiones en los datos, y se calculan n vectores y valores propios, y luego se eligen solamente los primeros p vectores propios, entonces el conjunto de datos final tendrá solamente p dimensiones.

A continuación se necesita formar un *vector característico*, el cual consiste en una matriz de vectores. Se construye eligiendo los vectores propios que se desean conservar de la lista de vectores propios, para posteriormente formar una matriz con estos vectores propios como columnas.

$$\textit{Vector característico} = \begin{pmatrix} vp_1 & vp_2 & vp_3 & \dots & vp_5 \end{pmatrix}$$

En el ejemplo anterior, dado el hecho de que tenemos dos vectores propios, existen dos alternativas:

1. Formar un vector característico con ambos vectores propios:

$$\begin{pmatrix} -0{,}677873399 & -0{,}735178656 \\ -0{,}735178656 & 0{,}677873399 \end{pmatrix}$$

2. Eliminar el componente principal menos importante, y seleccionar una única columna:

$$\begin{pmatrix} -0{,}677873399 \\ -0{,}735178656 \end{pmatrix}$$

Más adelante se interpretará gráficamente el resultado de esta elección.

Paso 6: Derivación del nuevo conjunto de datos

Este es el último paso del PCA. Una vez que se han elegido los componentes principales (vectores propios) que queremos mantener, y una vez

formado el vector característico, simplemente hay que trasponer el vector y lo multiplicarlo a la izquierda del conjunto de datos original traspuesto.

$$DatosFinales = VectorCaracterístico^T \times DatosNormalizados^T$$

donde

- $VectorCaracterístico^T$ es la matriz con los vectores propios en las columnas *traspuestos* de modo que éstos estén ahora en las filas, con el vector propio más importante en la primera fila.
- $DatosNormalizados^T$ son los datos normalizados y traspuestos, de modo que cada fila sea ahora una dimensión.
- $DatosFinales$ es el conjunto de datos final, con las dimensiones a lo largo de las filas.

A partir de este cálculo, se tienen lo datos originales *únicamente en términos de los vectores que se hayan elegido*. El conjunto de datos original tenía dos ejes, x e y, y los datos estaban en términos de dichos ejes. Sin embargo, es posible expresar los datos en términos de cualquier par de ejes que se elijan. Si los ejes son perpendiculares, la expresión es la más eficiente posible. Es por esto que es importante el hecho de que los vectores propios siempre son perpendiculares entre ellos. Se ha logrado que los datos estén en términos de los dos vectores propios (o de un solo vector propio, en el caso de haber decidido eliminar un componente). En el caso de elegir los dos vectores propios para formar el vector característico, se obtienen los datos de la tabla 3.3.

El gráfico correspondiente a la tabla 3.3 se muestra en la figura 3.8, y consiste básicamente en los datos originales rotados de manera que los vectores propios son los nuevos ejes.

La otra transformación que se puede hacer es escoger únicamente el vector propio con el valor propio mayor. La tabla 3.4 contiene los datos resultantes. Como era de esperar, sólo tiene una dimensión. Si se compara este conjunto de datos con el resultante de utilizar ambos vectores propios, se puede observar

x	y
0.8279	-1.7511
1.7775	0.1428
0.9921	0.3843
-0.2742	0.1304
-1.6758	-0.2094
-0.9129	0.1752
-0.0991	-0.3498
1.1445	0.0464
0.4380	0.1776
1.2238	0.1626

Tabla 3.3: *Datos después de PCA usando dos vectores propios.*

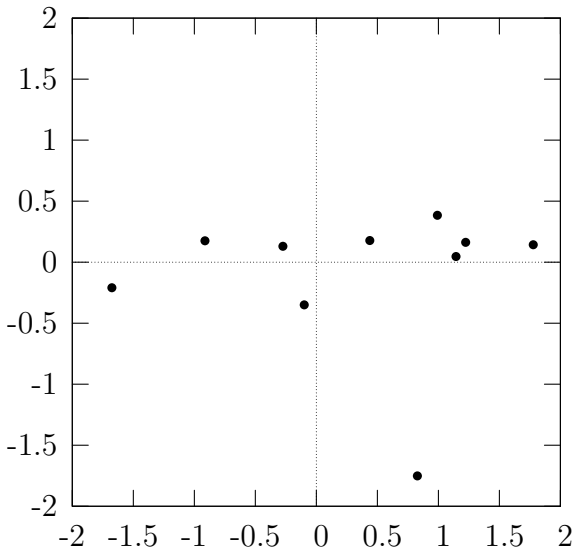

Figura 3.8: *Datos después de aplicar el análisis PCA usando dos vectores propios.*

que coincide exactamente con la primera columna. Por lo tanto si se dibuja este conjunto de datos, sería unidimensional, y los puntos estarían sobre el eje x en las posiciones x de los puntos del gráfico de la figura 3.3. Esto es así porque hemos descartado el otro eje, que es el otro vector propio.

x
0.8279
1.7775
0.9921
-0.2742
-1.6758
-0.9129
-0.0991
1.1445
0.4380
1.2238

Tabla 3.4: *Datos después de PCA usando un sólo vector propio.*

Resumiendo, básicamente hemos transformado los datos, de modo que se expresen en términos de los patrones que existen entre ellos, donde los patrones son las líneas que describen con mayor precisión las relaciones entre los datos. Esto es útil para tener clasificados los datos como una combinación de las contribuciones de cada una de esas líneas.

Recuperación de los datos originales

Recuperar los datos originales es de gran utilidad, sobre todo si se usa la transformación PCA para comprimir datos. Antes de explicar cómo lograrlo, es digno de mención el hecho de que únicamente si se han escogido todos los vectores propios para formar el vector característico, se conseguirán recuperar *exactamente* los datos originales. Si se ha reducido el número de vectores propios en la transformación final, entonces, los datos recuperados tendrán cierta pérdida de información.

La transformación final era:

$$DatosFinales = VectorCaracterístico^T \times DatosNormalizados^T$$

la cual se puede recolocar para recuperar los datos originales como:

$$DatosNormalizados^T = \left(VectorCaracterístico^T\right)^{-1} \times DatosFinales$$

Cuando se escogen todos los vectores propios, resulta que la inversa del vector característico es exactamente igual a la traspuesta de dicho vector. Esto es cierto porque los elementos de la matriz son los todos vectores propios unitarios del conjunto de datos. La ecuación se convierte en:

$$DatosNormalizados^T = \left(VectorCaracterístico^T\right)^T \times DatosFinales$$

Finalmente, para recuperar completamente los datos, se necesita añadir la media de los datos originales, ya que fue restada al normalizar los datos. Por lo tanto, para que quede completo:

$$DatosOriginales^T = (VectorCaracterístico \times DatosFinales) + MediaOriginal$$

Por último cabe destacar que esta fórmula también se aplica cuando no se tienen todos los vectores propios en el vector característico.

4

Técnicas

Los objetivos del SLAM son la construcción de mapas y la localización simultánea. Para estimar la localización de un robot existen dos enfoques diferentes. Por un lado se puede estimar de forma relativa; para ello se toma una posición de referencia inicial y se actualiza incrementalmente su valor realizando cálculos con respecto a la posición referencia a medida que el robot se mueve. Por otro lado, se puede estimar la posición de manera absoluta, teniendo previamente un mapa global del entorno deseado, y comparando la visión del entorno que tiene el robot en un momento dado con el mapa global.

En este apartado de la memoria, se presentan las dos técnicas seleccionadas por el autor de este proyecto para dar una solución al problema del SLAM basándose en el uso de un escáner láser de medición de distancias. Por una parte la técnica basada en la correlación de histogramas (apartado

4.1) permite construir mapas del entorno y resuelve el problema de la localización utilizando un enfoque relativo. Por otra parte, la técnica estadística del análisis de componentes principales (apartado 4.2) presenta una solución al problema de la localización estimando la posición de una manera absoluta.

La mayoría de los métodos que utilizan escáneres láser para estimar la posición de un robot, se basan en la comparación de los datos obtenidos por el sensor, siendo éstos expresados en un entorno de coordenadas cartesiano bidimensional. Tal es el caso de la técnica de construcción de mapas y localización relativa basada en histogramas. En el apartado 7 de esta memoria se comentan otros métodos de este tipo. No obstante, es posible utilizar espacios de coordenadas alternativos, como es el caso del uso del análisis de componentes principales aplicado a los datos de un escáner láser. En este caso, cada medida individual obtenida por láser puede ser considerada como una dimensión en un espacio. Como el escáner tiene una resolución de 0.5º y un campo de visión de 180º, habría un total de 361 medidas de distancia individuales y por tanto 361 dimensiones en lugar de las 2 habituales de un espacio bidimensional.

4.1. Correlación de laser-scans

La información que se presenta a continuación está extraida de las referencias [9], [10] y [8]. Antes de nada es necesario definir los términos *laserscan*[1] y correlación cruzada, para poder comprender claramente el resto de la sección.

Se considera un ***laserscan*** (ver figura 4.1) a una secuencia de números, donde cada elemento es un número que representa la distancia al obstáculo más cercano en la dirección en la que se emitió el haz de luz. El escáner con el que se ha trabajado en este proyecto es capaz de realizar un barrido de un máximo de 180°, y de emitir haces de luz espaciados uniformemente cada 0.5º o cada 1º. Esto nos da la posibilidad de tener secuencias de números

[1] A lo largo de esta sección se utilizará este término ya que resulta más conciso que su traducción al castellano

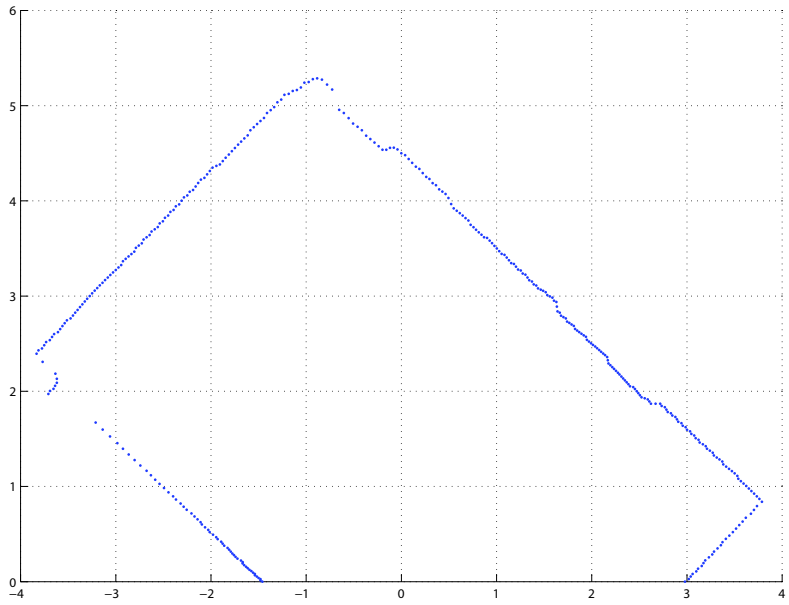

Figura 4.1: *Un ejemplo de laserscan*

(*laserscan*) de 181 ó 361 elementos. Estos números pueden venir dados en coordenadas polares, o en coordenadas cartesianas.

Una función de **correlación cruzada**, $c(y)$, (*crosscorrelation)* se puede definir como:

$$c(y) = \lim_{X \to \infty} \frac{1}{2X} \int_{-X}^{X} f(x) \cdot g(x+y) \cdot dx$$

y es una medida de la correlación entre dos funciones estocásticas, f y g con respecto al cambio de fase y. La correlación cruzada $c(y)$ tendrá un máximo absoluto en s si $f(x)$ es igual $g(x+s)$.

4.1.1. Calculando invariantes

Para hacer coincidir dos *laserscans* que fueron tomados en el mismo entorno y averiguar la rotación y traslación que los diferencia, se deben encontrar propiedades comunes a ambos que no varíen a pesar de que la posición y la orientación en la que se tomó cada uno fuera diferente. Tales propiedades constituyen una marca común que se utilizará para intentar hacer coincidir

los dos *laserscans*. Estas marcas se calculan como histogramas a partir de la distribución de ángulos y distancias medidas por el escáner láser. Mediante la correlación de los histogramas de los *laserscans*, se puede calcular los desplazamientos de rotación y traslación que se han efectuado. Para poder efectuar la correlación de dos histogramas es necesario que se puedan superponer, esto es, que los *laserscans* correspondientes tengan partes en común del entorno (invariantes).

4.1.2. Histogramas angulares

Desafortunadamente, a pesar de que la correlación cruzada encuentra fácilmente la rotación entre dos *laserscans* tomados en la misma posición, no funciona correctamente cuando se aplica a dos *laserscans* tomados en posiciones diferentes. Esto sucede porque una traslación en un *laserscan* produce desplazamientos que no son necesariamente simétricos, de modo que con la correlación cruzada se interpretaría erróneamente esto como una rotación. Por lo tanto necesitamos una función para representar un *laserscan* que sea robusta incluso aunque haya traslación, pero que también funcione con rotación y además ésta aparezca como un cambio de fase proporcional.

En un *laserscan* las medidas individuales de distancia y los ángulos obtenidos a partir de dichas medidas (en el sistema de coordenadas del sensor) pueden interpretarse como vectores, que corresponden a los haces de luz que el escáner láser ha tomado en el entorno.

Si se toma un *laserscan* en un entorno en el que paredes u otras superficies planas tales como armarios son visibles para el sensor, muchos de los puntos que aparezcan en los vectores serán el resultado de medir las superficies de los armarios y las paredes. Todos los puntos que están en la misma superficie describen una línea recta común que representa esta superficie. Si conectamos mediante líneas rectas cada par de puntos consecutivos, todas las líneas que comparten la misma superficie tendrán orientaciones similares. Sin embargo, debido a las imprecisiones de medida propias del sensor, cada punto tendrá pequeñas desviaciones laterales con respecto a la línea rec-

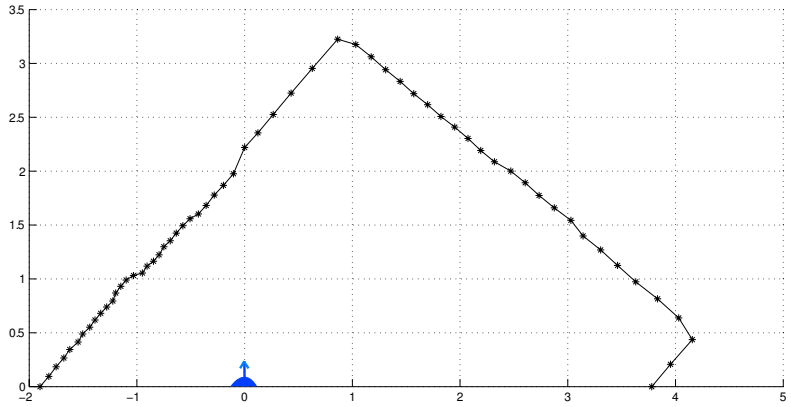

Figura 4.2: *Laserscan con puntos conectados por líneas*

ta correspondiente a una superficie plana (en el caso del escáner empleado, aproximadamente ± 1 cm).

En la figura 4.2 se ilustra un *laserscan* con líneas que conectan cada par de puntos vecinos. El escáner láser está situado en el punto $(0, 0)$. Aparecen representados unos 60 puntos en lugar de los 361 o 181 que da el escáner con el que se ha trabajado, con la única pretensión de que no se mezclen los asteriscos unos con otros (en los cálculos posteriores con histogramas se utilizan los 361 puntos). Tal y como se puede observar, existen dos direcciones principales, una podría ser cercana a los $+45^{o}$ y la otra -45^{o}, aproximadamente. También se aprecia que no todas las pequeñas líneas que unen cada par de puntos siguen exactamente una de las dos direcciones principales, sino que los ángulos que estas líneas forman con el eje de las x oscilan por encima y por debajo de las direcciones principales una pequeña cantidad de grados. Ésta es la verdadera utilidad de los histogramas: gracias a la acumulación de ocurrencias de los ángulos de cada línea en el histograma, aparecerán máximos locales que indican las direcciones principales, como se verá a continuación. (ver figura 4.4).

Los ángulos entre el eje x y las líneas que conectan cada par de puntos vecinos aproximadamente corresponden a las orientaciones de las superficies del entorno, pero desde referenciados en el sistema de coordenadas del escáner

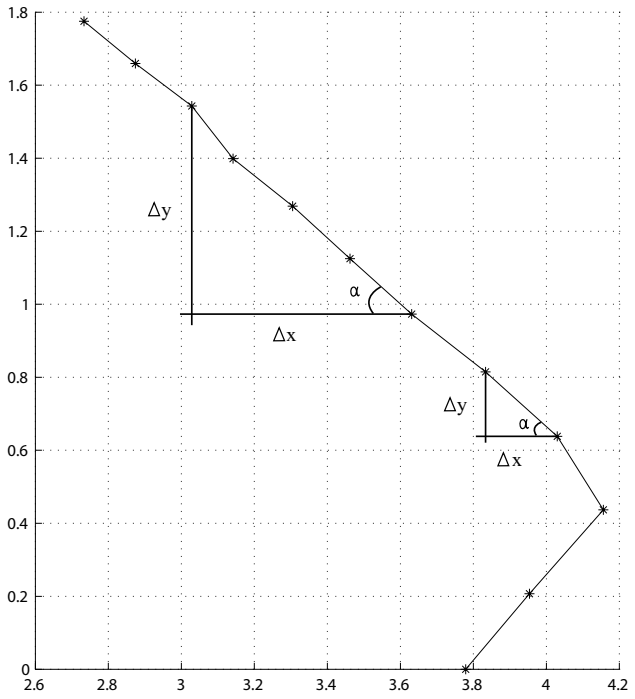

Figura 4.3: *Cálculo de ángulos para el histograma*

láser. En la figura 4.3 se muestra gráficamente cómo calcular cada uno de estos ángulos. Conociendo las coordenadas x e y de cada uno de los puntos, se obtiene inmediatamente el ángulo con las expresiones siguientes:

$\Delta x = x_{i+1} - x_i$
$\Delta y = y_{i+1} - y_i$
$\alpha = \arctan\left(\frac{\Delta x}{\Delta y}\right)$

En caso de querer reducir el ruido en la medida, como se comenta más adelante, deberá sustituirse x_{i+1} e y_{i+1} por x_{i+k} e y_{i+k}, donde k es una cierta cantidad arbitraria que indica que se calculará el ángulo entre el punto i-ésimo y el punto situado k posiciones más adelante.

Una distribución estadística de la orientación de todas estas pequeñas líneas produce lo que se denomina como **histograma angular**. A partir de este histograma, puede determinarse con qué frecuencia aparece cada orientación en el trozo de entorno en el que se acaban de tomar las medidas. Si

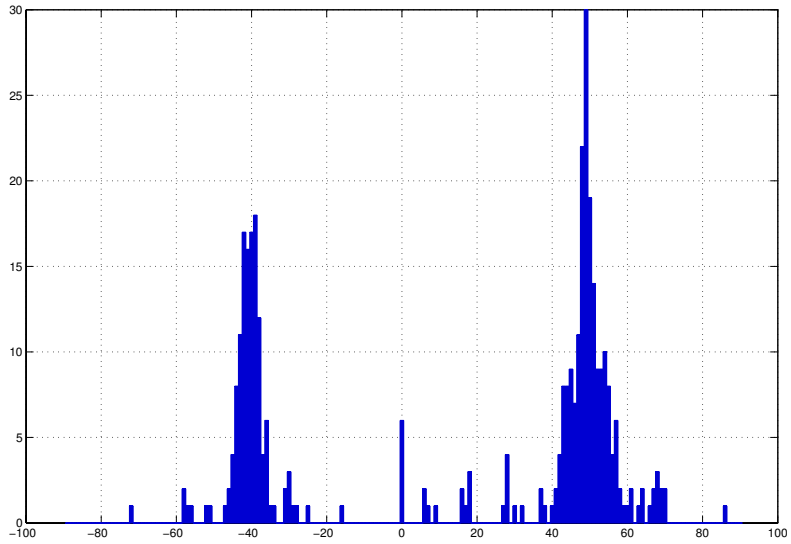

Figura 4.4: *Histograma angular*

muchos puntos son el resultado de medir superficies planas con la misma orientación para un cierto ángulo, se acumulará un máximo local en el histograma. En la figura 4.4 se muestra un histograma angular generado a partir del *laserscan* de la figura 4.2. Los números que aparecen en el eje de abscisas corresponden a valores de ángulos, mientras que en el eje de ordenadas se puede contar el número de ocurrencias de un determinado ángulo. En dicho histograma, se han tomado pares de puntos situados a una distancia $k = 6$ para calcular los ángulos. Los máximos locales se encuentran aproximadamente en -40º y en +45º, y se diferencian nítidamente.

Un problema que surge en la construcción de histogramas es el ruido en la medida de cada punto, debido a la imprecisión del escáner láser. Si midiéramos los ángulos entre un punto y él punto inmediatamente siguiente, y los acumuláramos formando un histograma, la calidad de éste sería muy pobre, puesto que aparecerían valores más o menos similares en cada barra del histograma. Éste fenómeno se puede apreciar en la figura 4.5, donde se han tomado pares de puntos situados a una distancia $k = 1$ para calcular cada ángulo del *laserscan* de la figura 4.2. Aún así se pueden diferenciar dos máximos locales aproximadamente en +50º y en -40º, pero no se observan

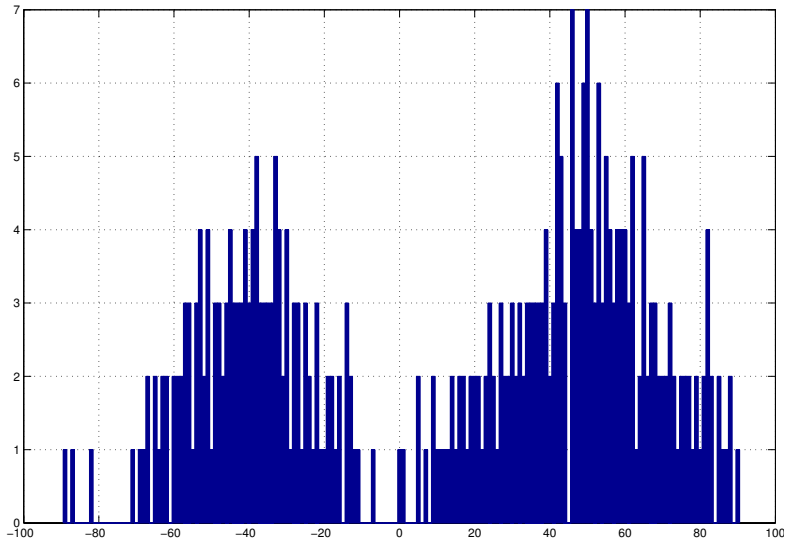

Figura 4.5: *Histograma con mucho ruido*

con tanta nitidez como en el histograma de la fig.4.4.

Este ruido puede ser reducido si en lugar de utilizar puntos consecutivos, se usa un punto y su k-ésimo sucesor. De este modo se suavizará el ángulo calculado para cada recta, y alrededor de los máximos locales, en las barras del histograma, se acumulará una variedad de ángulos menor, con lo cual los máximos se diferenciarán de manera más nítida. Tras varias pruebas realizadas se ha comprobado cómo usando k entre 4 y 10, como k-ésimo sucesor de un punto, se obtienen buenos resultados (p.ej., en el histograma de la figura 4.4 se han tomado $k = 6$ para calcular los ángulos de cada recta).

4.1.3. Rotación

Supóngase que se toma un segundo *laserscan* en una posición girada con respecto a la del primer histograma. En la fig 4.6 el escáner láser se ha girado en sentido horario, y el resultado del nuevo *laserscan* aparece en línea discontinua (la línea continua es el mismo *laserscan* de la fig. 4.2).

El nuevo histograma angular contendrá máximos que cubren distancias similares y ángulos que ya aparecieron en el primer histograma, sin em-

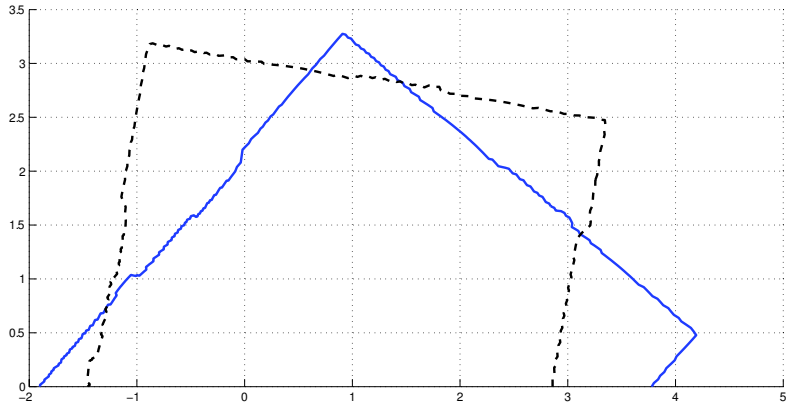

Figura 4.6: *Comparación de dos laserscans*

bargo, los nuevos máximos tendrán un cierto desplazamiento de fase. Este desplazamiento es el resultado de que el escáner ha sufrido una rotación y las paredes u otros objetos ahora forman ángulos distintos con respecto al sistema de referencia del escáner. No obstante, la distancia entre máximos locales será equivalente a la que apareció en el primer histograma, independientemente de la rotación, puesto que el ángulo que forman (entre ellas) las paredes del entorno no ha variado.

Quizás el valor de algunos máximos haya aumentado o disminuido, dependiendo de si nos hemos acercado a una superficie plana o nos hemos alejado, respectivamente. Si el escáner está más cerca de una superficie plana, más cantidad de líneas tendrán la misma dirección, puesto que el escáner tomará un mayor número de puntos de esta superficie plana, y de modo opuesto ocurre cuando el escáner está lejos de la superficie. Gracias a que la distancia entre máximos locales no resulta modificada, el desplazamiento de fase entre los histogramas podrá ser calculado por medio de una correlación cruzada, y esto permitirá conocer la rotación entre las posiciones en las que se tomaron los *laserscans*.

En la figura 4.7 se muestra el mismo histograma de la figura 4.4 (en línea continua) junto con un histograma (en línea discontinua) correspondiente al *laserscan* en línea discontinua mostrado en la fig. 4.6. Los dos histogramas

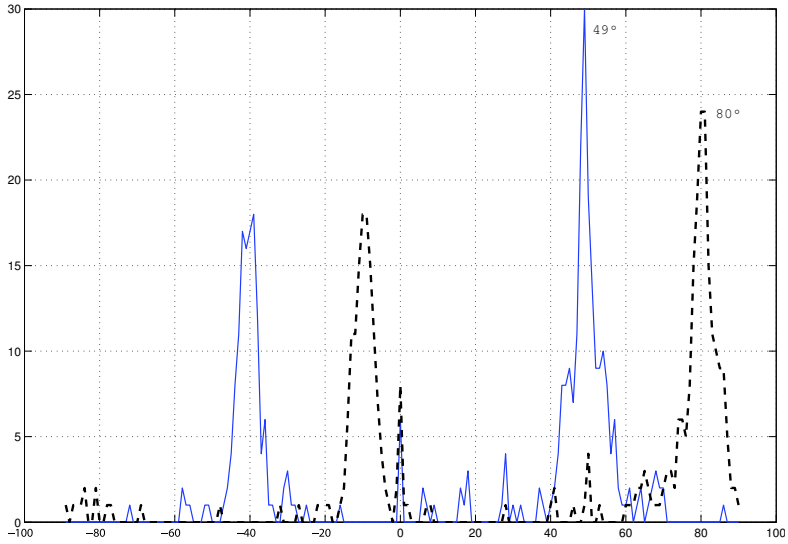

Figura 4.7: *Comparación de dos histogramas*

son muy parecidos, aunque no exactamente iguales, y además hay ciertas características que son invariantes con respecto a la rotación: los máximos locales. La distancia entre el máximo absoluto del primer histograma y el máximo absoluto del segundo, es aproximadamente de unos 30º.

El cambio de fase entre dos histogramas correspondientes a dos *laserscans* rotados puede ser estimado mediante la búsqueda del máximo de la correlación cruzada de ambos *laserscans*. Puesto que un *laserscan* dado es una secuencia discreta, la expresión de la correlación cruzada necesita ser transformada a una forma discreta:

$$c(j) = \sum_{i=1}^{n} h_1(i) \cdot h_2(i+j)$$

donde h_1 y h_2 son dos *laserscans*. Puesto que el número de posibles valores para j es finito, el máximo de k y su correspondiente j puede encontrarse fácilmente. Básicamente lo que hace esta fórmula es multiplicar los valores discretos del primer histograma por los del segundo, que se va desplazando sucesivamente. Si en uno de los dos histogramas hay ceros, el resultado será cero; sin embargo si en los dos histogramas hay valores altos, el resultado será muy alto (la multiplicación de los dos). El cambio de fase encontrado de

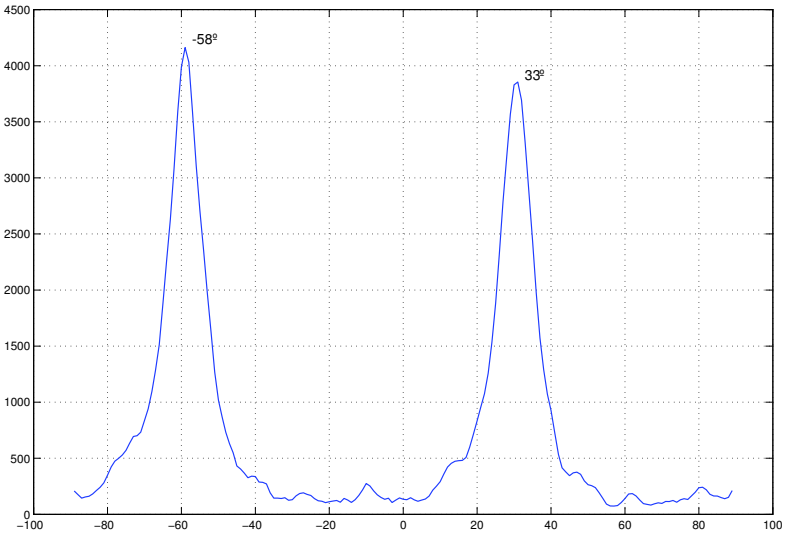

Figura 4.8: *Correlación cruzada*

este modo (uno de los máximos locales) entre los dos histogramas puede ser interpretado como la rotación entre ellos.

La representación de la correlación cruzada para los dos histogramas presentados, se muestra en la fig. 4.8. Aparecen dos máximos locales, uno en -58º y otro en 33º. Quizá resulte sencillo en este ejemplo darse cuenta de que el giro correcto del segundo *laserscan* son 33º, sin embargo, es difícil proponer un algoritmo para distinguirlo, porque -58º es el máximo absoluto. Este error en la correlación cruzada ocurre porque en el segundo *laserscan* han aparecido nuevas partes del entorno que no existían en el primero. Para solucionar este problema es necesario utilizar algún tipo de estimación odométrica mediante otros sensores del robot.

Gracias a la estimación odométrica, si se conoce que se ha rotado el escáner láser en el sentido de las agujas del reloj, el resultado obviamente será 33º, puesto que es el máximo local que se encuentra a la derecha del cero. Si en lugar de esto se supiera que la rotación se ha producido en sentido antihorario, es claro que la rotación realizada hubiera sido -58º.

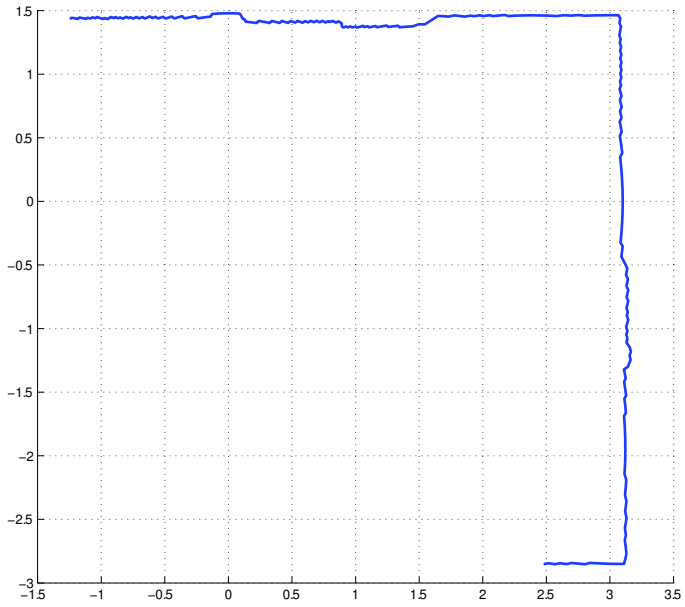

Figura 4.9: *Laserscan alineado con los ejes de coordenadas*

4.1.4. Traslación

Una vez encontrada la rotación entre los dos *laserscans*, el siguiente paso es encontrar el desplazamiento de traslación que se ha realizado, esto es, el movimiento que ha sufrido el escáner láser en los ejes x e y.

Una traslación (en los ejes x e y) puede calcularse generando dos histogramas, denominados **histograma-X** e **histograma-Y**, que representen la distribución de las distancias de los puntos del *laserscan* con respecto a los ejes x e y. En un histograma de este tipo, las acumulaciones de distancias iguales ocurren en las posiciones donde estructuras planas, tales como paredes, están orientadas perpendicularmente con respecto a la dirección del eje correspondiente, esto es, el eje x para el histograma-X y el eje y para el histograma-X. Puesto que las paredes no estarán orientadas perpendicularmente al eje que pretendemos utilizar, los *laserscans* deben ser rotados.

Para lograr esto, se debe buscar el ángulo más frecuente en el histograma de ángulos, esto es, el ángulo con el que la mayoría de las superficies están alineadas. Dicho valor se obtiene al calcular el máximo absoluto de cada

uno de los histogramas. Según se indica en la fig. 4.7, el máximo del primer histograma es 49º. Si el vector que contiene los puntos del primer *laserscan* se gira el ángulo correspondiente a este máximo absoluto, la superficie principal de dicho *laserscan* estará entonces orientada en paralelo al eje de las x. Por lo tanto, las coordenadas y de todos los puntos que resulten de medir esta superficie principal son similares, y darán un máximo en el histograma-Y. Al acumular los valores de las coordenadas x se producirá un máximo en el histograma x correspondiente a la segunda superficie principal.

Por lo tanto, y a modo de ejemplo, para realizar el histograma-X del primer *laserscan*, necesitaremos girarlo 49º, para alinear la dirección principal con el eje de las x (fig 4.9). A continuación se acumulan los valores de las coordenadas y dando lugar a la fig. 4.10. Para el segundo *laserscan* se repite este mismo proceso, y posteriormente se efectúa una correlación cruzada en los dos histogramas-X, de modo similar a los histogramas angulares, obteniendose el valor que los dos *laserscans* están desplazados en el eje x. El mismo proceso con los histogramas-Y permitirá averiguar el desplazamiento en el eje y.

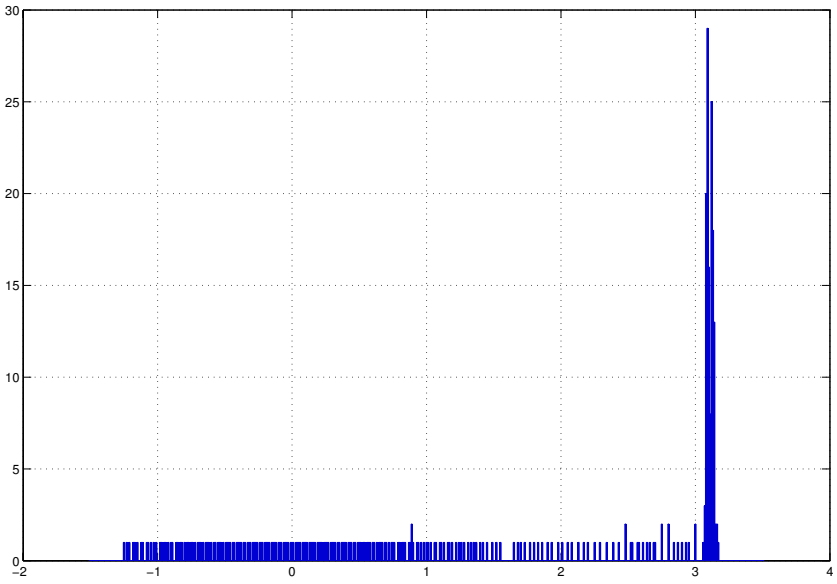

Figura 4.10: *Histograma-X*

4.2. Estimación de la Posición usando PCA

Un análisis de los datos de un escáner láser de medición de distancias, muestra que los datos de medidas de distancia adyacentes están altamente correlacionados. Esto es lógico, puesto que la mayoría de entornos, especialmente los construidos por el hombre tienen una estructura regular. Por lo tanto es posible utilizar el análisis de componentes principales (PCA) para determinar un subespacio lineal con un número mínimo de dimensiones para representar un entorno mediante un sensor de medición de distancias. Este enfoque tiene una sólida base matemática, puesto que se basa en la utilización de un método estadístico. La utilización de PCA servirá para producir un sistema fiable, robusto ante oclusiones en el entorno y computacionalmente simple.

La aplicación del análisis PCA a los datos de un escáner láser se describe en [11] y [12] y es un enfoque basado en la apariencia. Métodos similares han sido adaptados para la navegación mediante visión artificial, para el reconocimiento de objetos y para la interpretación de objetos deformables tales como caras.

En las secciones siguientes se comentará como utilizar el análisis de componentes principales aplicado a los datos de distancia que nos proporciona un escáner láser, cómo simular una cuadrícula de posiciones y orientaciones del escáner láser y finalmente cómo realizar una búsqueda de hipótesis de posicionamiento en el espacio vectorial.

4.2.1. PCA aplicado a los datos de distancia del láser

Con un escáner láser de medición de distancias, cada medida de distancia individual puede considerarse como una dimensión independiente. Esta interpretación no es muy habitual debido a la obvia correlación que existe entre las medidas adyacentes. Esta redundancia significa que las dimensiones en tal espacio no son ortogonales, y que el espacio no es mínimo. El análisis de componentes principales proporciona un método para determinar

automáticamente un subespacio lineal con un número mínimo de dimensiones. Tal representación es óptima teniendo en cuenta el método de mínimos cuadrados.

Los vectores que constituyen los componentes principales extraídos de la matriz de covarianza de los datos de distancia forman una base ortogonal, la cual se puede interpretar como los *scans* propios (o *eigenscans*) del sensor medidor de distancias en el entorno en el cual se encuentra. Usando únicamente los vectores propios más importantes, es decir, los que tienen los valores propios con un mayor valor, la dimensión del espacio vectorial puede reducirse significativamente, con una pérdida mínima en precisión y fiabilidad. Esta reducción puede disminuir bastante los requisitos de memoria y procesamiento.

4.2.2. Creación de una cuadrícula simulada de laserscans

El análisis del componentes principales solamente resulta útil para la estimación de la posición si representa de una manera fiable todas las posiciones posibles en las que se puede situar el sensor láser. Cuando se calculan los componentes principales a partir del laserscans tomados en posiciones aleatorias (esto es posiciones que el robot ha visitado), las características del entorno que se hayan observado con mayor frecuencia, serán las dominantes. Para lograr obtener unos componentes principales que incluyan todas las características posibles del entorno, las posiciones donde se han obtenido los datos deben formar una cuadrícula muy densa, esto es, deben haberse tomado muchos *scans* en todo el entorno. Éste es uno de los principales problemas del análisis PCA aplicado a los datos de un escáner láser, para que funcione adecuadamente se necesitan grandes muestras de datos de ejemplo.

Obtener una cuadrícula de datos muy densa en posiciones y orientaciones precisas presenta dificultades prácticas. El esfuerzo de tomar todos estos datos de entrenamiento puede reducirse significativamente si estos datos son generados mediante una simulación por ordenador. Esto puede lograrse me-

diante el ensamblaje de una serie de laserscans que se superpongan, esto es, tengan características en común. Esta técnica se describe en la sección 4.1, y permite obtener un mapa del entorno. Una vez obtenido este mapa del entorno, se puede utilizar para predecir nuevos laserscans para formar una cuadrícula de posiciones y orientaciones del sensor láser tan densa como sea necesario. Este proceso es bastante robusto puesto que sólo se utilizan datos en bruto del escáner láser.

Dado un conjunto de laserscans en diferentes posiciones y orientaciones, aplicando una correlación cruzada será posible determinar la rotación relativa y el desplazamiento relativo entre los laserscans, y girando y trasladando los laserscans a un sistema de coordenadas fijo y común a todos, se obtendrá un laserscan compuesto a partir de todos los laserscans parciales y que servirá a modo de mapa.

Una vez que tenemos un mapa del entorno, es posible generar laserscans simulados en cualquier posición y orientación del entorno. Cada medida del laserscan simulado corresponde a la intersección de un haz de luz (emitido por el sensor láser en la posición y orientación deseada) con las líneas del mapa obtenido.

Para obtener una gran cantidad de laserscans simulados, basta con colocar el generador de laserscans en una cuadrícula de 50 cm de ancho. Esto hace un total de 4 posiciones diferentes por metro cuadrado. Además, en cada posición el sensor láser se irá girando cada 0.5 grados, lo cual significa que se simularán 720 laserscans en cada posición. Como además tenemos 4 posiciones por metro cuadrado, el total son 2880 laserscans por metro cuadrado. Una vez hecho esto, se calcula la media y la covarianza para el conjunto de laserscans simulados en todo el entorno, y se obtiene una matriz de covarianza de 360 por 360; a continuación se realiza un análisis de componentes principales. Éste cálculo sólo es necesario realizarlo una vez, por lo tanto no importa el hecho de que requiera gran cantidad de tiempo de procesamiento (sobre todo el cálculo de la matriz de covarianza), puesto que una vez obtenidos los componentes principales, se podrán utilizar directamente para estimar la

posición del sensor láser.

El número de vectores propios que se utilice dependerá del entorno en concreto. En general, según los autores de le técnica, si el entorno tiene muchas superficies planas, bastará con utilizar 3 o 4 vectores propios, ya que el resto serán suficientemente pequeños como para no contener demasiada información acerca del entorno. Sin embargo en un entorno en el que no hayan demasiadas paredes o superficies planas, será necesario utilizar un mayor número de vectores propios para que recojan toda la información del entorno.

4.2.3. Uso de *eigenscans* para estimar la posición

Supóngase que, utilizando la técnica del apartado anterior, se han obtenido una gran densidad de *scans* simulados y que éstos están uniformemente distribuidos a lo largo y ancho del entorno del que se dispone el mapa. Lo que se pretende con la obtención de todo el conjunto de *scans* simulados es poder tomar un *scan* en un determinado instante de tiempo y compararlo con todos los *scans* simulados para observar cual es el que más se parece. Esta comparación se debe realizar una vez que todos los datos están proyectados en el espacio vectorial calculado, para que tenga lugar una gran reducción de los datos.

Por lo tanto, para poder estimar la posición, es necesario seguir los siguientes pasos. En primer lugar, proyectar todos los *scans* simulados en el espacio vectorial. En segundo lugar, obtener los datos del escáner láser, que estará situado en una determinada posición y orientación dentro del entorno, y proyectar el *scan* obtenido en el espacio vectorial. A continuación se debe comparar el *scan* actual con todos y cada uno de los *scans* simulados. Puesto que la dimensión de los datos está reducida al estar proyectados en el espacio vectorial que se obtuvo con el análisis PCA, esta comparación tendrá unos costes computacionales mucho más reducidos que si los datos no hubieran sido proyectados. Por último queda decidir cual de todas las comparaciones realizadas es la mejor, obteniendo de este modo la localización más probable.

5

Herramientas

5.1. Player/Stage

El proyecto *Player/Stage* [31] fue fundado por Brian Gerkey (Stanford), Richard Vaughan (SFU) y Andrew Howard (NASA JPL) en el año 2000, basándose en software que habían desarrollado con Kasper Stoy y otros en los Laboratorios de Investigación Robótica de la Universidad de California del Sur. La versión actual de *Player* es la 1.6.4, y la de *Stage* es la 1.6.2 (Mayo 2005). El nombre del proyecto fue tomado de una célebre frase de Shakespeare *"All the world's a stage, And all the men and women merely players"* (El mundo entero es un escenario, y los hombres y mujeres son simplemente actores).

Este proyecto está compuesto por tres componentes bien diferenciados:

- *Player*, un servidor de dispositivos robóticos.

- *Stage*, un simulador multi-robot en 2D.
- *Gazebo*, un simulador multi-robot en 3D.

A continuación se explicarán los componentes *Player* y *Stage* únicamente, puesto que el componente Gazebo no ha sido utilizado en el desarrollo del proyecto.

5.1.1. Player

Player es un servidor en red multihilo para el control de robots. Ejecutándose en un robot, *Player* proporciona una interfaz simple y clara para comunicarse con los sensores y actuadores del robot a través de la red IP. El programa cliente habla con *Player* a través de un *socket* TCP, leyendo datos de los sensores, escribiendo órdenes en los actuadores y configurando dispositivos al vuelo.

Player soporta una amplia variedad de hardware robótico. La plataforma original de *Player* es la familia ActivMedia Pioneer 2, pero muchos otros robots y muchos sensores comunes son soportados actualmente. La arquitectura modular de *Player* hace que sea muy fácil añadir soporte para nuevo hardware, además de que existe una comunidad de usuarios y desarrolladores que contribuyen con nuevos *drivers*.

Player se puede ejecutar en la mayoría de las plataformas compatibles con POSIX, incluyendo sistemas empotrados. Los requisitos que necesita *Player* son:

- Entorno de desarrollo POSIX, con *threads* (`pthreads`).
- Pila de protocolos TCP/IP.
- Una versión reciente de GNU gcc, con soporte para C y C++.

Trabajos anteriores a *Player* en el área de programación de interfaces para robots se habían centrado principalmente en proporcionar un entorno de desarrollo que utilizara una determinada filosofía de control. A pesar de

que esto es bastante útil, los desarrolladores de *Player* creyeron que la implementación a bajo nivel impondría restricciones innecesarias al programador, el cual debería tener la oportunidad de construir cualquier clase de sistema de control a la vez que se mantiene la abstracción y el encapsulamiento.

Por lo tanto *Player* hace una clara distinción entre la interfaz de programación y la estructura de control, optando por una interfaz de programación general, con la creencia de que los usuarios desarrollarán sus propias herramientas para construir sistemas de control. Además, muchas interfaces de programación de robots permiten al programador utilizar un único lenguaje de programación, proporcionando una biblioteca (normalmente que no es de código abierto) con la cual el usuario debe enlazar sus programas. En contraste, la abstracción de *Player* mediante *sockets* TCP permite el uso potencial de cualquier lenguaje de programación. De este modo, es mucho menos restringido que otras interfaces de programación de robots.

Por tanto, *Player* está diseñado para ser independiente del lenguaje de programación y de la plataforma. Un programa cliente puede ejecutarse en cualquier máquina que tenga una conexión TCP con el robot en el que se ejecuta *Player*, y puede ser escrito en cualquier lenguaje que soporte *sockets* TCP. Actualmente, existen clientes en C++, Tcl, Java y Python. Además, *Player* no impone restricciones en cuanto a cómo estructurar los programas de control del robot. Si se desea que el cliente sea multi-hilo y altamente concurrente, se puede hacer así. Si se desea que será un simple bucle lectura-actuación, también se puede hacer así.

Otra característica muy interesante es que no tiene capacidad para soportar un gran número de clientes (dependiendo de las capacidades de la máquina en la que se ejecute) y permite que varios clientes accedan simultáneamente a dispositivos comunes de uno o varios robots. Esto hace que la creación de clientes multi-robot sea muy sencilla.

Por último, pero no menos importante, es que *Player* es código fuente abierto, liberado bajo la licencia pública GNU. Este hecho ha dado la posibilidad de que A. Morales [19] desarrollara, en el Departamento de Informática

y Automática de la Universidad de Salamanca, una modificación de la arquitectura y el código del servidor *Player*. De esta modificación se hablará en el apartado 5.1.6.

Un ejemplo sencillo del funcionamiento de *Player* es el mostrado en la figura 5.1. El cliente establece una conexión con el servidor (*Player*) por medio de un *socket* TCP, y envía un conjunto de mensajes al servidor para abrir los dispositivos a los que desea acceder. Tras este proceso, *Player* enviará continuamente datos desde los dispositivos al cliente y éste, por su parte, realizará el control sobre el robot, enviado las órdenes apropiadas al servidor *Player*.

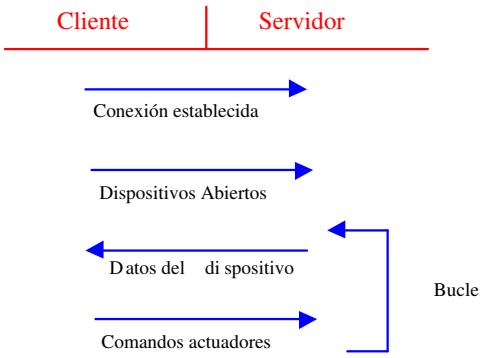

Figura 5.1: *Interacción entre cliente y Player*

5.1.2. Stage

Stage simula una población de robots móviles, sensores y objetos en un entorno bidimensional. *Stage* está diseñado para soportar investigación en sistemas autónomos multi-agente, de modo que proporciona modelos computacionalmente simples de gran cantidad de dispositivos en lugar de intentar emular cualquier dispositivo pero con alta fidelidad. Los desarrolladores consideran que este enfoque es de gran utilidad.

Stage proporciona poblaciones de dispositivos virtuales para *Player*. Los usuarios escriben programas clientes para el servidor *Player*. Normalmente

los clientes no pueden notar la diferencia entre robots reales y sus equivalentes simulados en *Stage*. La experiencia ha demostrado que los clientes desarrollados utilizando *Stage* funcionan con ninguna o escasa modificación con los robots reales y viceversa. Por lo tanto *Stage* permite desarrollar prototipos rápidos de controladores destinados para robots reales, así como experimentar con dispositivos reales que el usuario no tiene en su posesión, o no dispone de ellos en un momento dado. Además proporciona la simulación de una amplia variedad de sensores y actuadores, tales como sonar, medidores de distancia por láser, dispositivos de detección de color por visión, odometría, etc...

En la figura se puede observar una imagen de la última versión (1.6) de *Stage*, simulando distintos tipos de dispositivos.

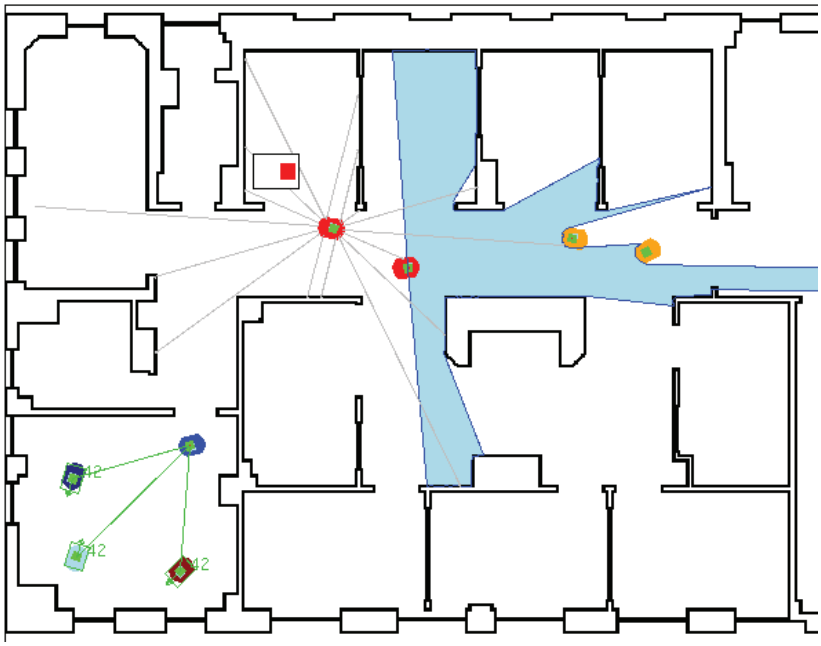

Figura 5.2: *Simulador de robots Stage*

5.1.3. Dispositivos, Interfaces y *Drivers*

En *Player* se utiliza el concepto de dispositivo para referirse a una entidad abstracta que proporciona una interfaz estándar con algún servicio, de modo

que es posible leer y escribir de y en ellos. Un dispositivo puede ser, por ejemplo, un sonar o un láser. *Player* no implementa dispositivos de acceso individual, sino que varios clientes pueden acceder de forma concurrente a un dispositivo dado, de modo que el sistema es más flexible.

Por otra parte, *Player* establece una distinción entre la interfaz de un dispositivo y el *driver* del mismo. La interfaz de un dispositivo especifica el formato de los datos, órdenes y posibilidades de configuración que el dispositivo soporta. El *driver* de un dispositivo implementa el control a bajo nivel sobre el dispositivo. Según esto más de un *driver* puede soportar una única interfaz de dispositivo, pero no al contrario.

Todas las interacciones entre los clientes y el servidor (*Player*) se realizan a través de las *interfaces*, sin hacer referencia al *driver* que se está ejecutando por debajo. La asociación entre los dispositivos que va a controlar *Player* y el *driver* que va a utilizar con cada uno de ellos se hace en un fichero de configuración que *Player* lee al arrancar. Por ejemplo podríamos configurar *Player* para usarse con un dispositivo `position`, al que asociamos el índice 0 y el *driver* `p2os_position`, y otro dispositivo del mismo tipo al que asociamos el índice 1 y el *driver* `rwi_position`. De esta forma un programa cliente podría controlar la posición de dos robots distintos mediante una única *interfaz* sin preocuparse del *driver* asociado al dispositivo de cada robot, simplemente debe decidir si accede al dispositivo con índice 0 ó 1.

En *Player* se encuentran implementadas *interfaces* para los siguientes dispositivos (además de otras muchas):

- **player.** Representa al propio servidor. Se utiliza para configurar el comportamiento del mismo. Este dispositivo puede ser leído pero no escrito. Las peticiones de configuración incluyen apertura y cierre de dispositivos, etc.

- **position.** Controla la posición de robot. Se pueden obtener las coordenadas y el ángulo del robot, su velocidad angular y tangencial. Se emplea para indicar nuevas velocidades.

- **sonar.** Proporciona las lecturas realizadas por los sonars del robot. Solo puede ser configurado para activar o desactivar los sonars.

- **laser.** Controla el láser SICK LMS-200, obteniendo las lecturas del mismo. Puede ser configurado para cambiar la apertura de lectura y activar o desactivar la lectura de los valores de reflexión. No puede escribirse.

- **vision.** Permite interactuar con el dispositivo de visión en color ACTS ActivMedia. Sólo permite lectura.

- **ptz.** Controla la cámara Sony EVID30. Al leer de este dispositivo obtenemos la panorámica actual, inclinación y zoom. Estos parámetros pueden modificarse escribiendo en el mismo.

- **laserbeacon.** Procesa las lecturas del láser, buscando la señales emitidas por un conjunto de balizas. Al leer del dispositivo se obtiene el identificador, el rango, la posición y la orientación de las balizas.

- **broadcast.** Permite a los clientes comunicarse con varios servidores *Player*. Si un cliente escribe en este dispositivo todos los servidores recibirán lo escrito. Al leer se obtendrían los últimos datos escritos por cada cliente.

- **gps.** Sólo funciona en el simulador, proporciona información de la posición y el ángulo del robot en coordenadas globales.

- **bps.** Proporciona la misma información que el gps pero basándose en el dispositivo laserbeacon.

Es posible implementar nuevas interfaces para otros dispositivos, o múltiples *interfaces* para un dispositivo, aunque esto último no es aconsejable ya que es preferible modificar una interfaz existente ampliando su funcionalidad que definir varias, cada una para realizar ciertas tareas sobre un mismo dispositivo. Es importante tratar de mantener una única *interfaz* para cada

tipo de dispositivo, ya que esto permite a los usuarios el manejo de distintos modelos de dispositivos de un mismo modo.

En *Player* se encuentran implementados, entre otros, *drivers* para el control de los siguientes sistemas y dispositivos robóticos:

- Robots Pioner 1, Pioneer 2, AmigoBot, PeopleBot de ActivMedia: posición (motores), sonars, *bumpers*, voltaje de la batería y brújula.
- Robots RWI de la serie B: posición, sonars, *bumpers*, voltaje de la batería y láser.
- Robots K-Team, Kameleon 376SBC: posición, infrarrojos y voltaje de la batería.
- Cámara Sony EVID30 PTZ.
- Láser SICK LMS-200.
- Tarjetas de red wireless.
- Tarjetas de sonido.

Para poder controlar desde *Player* los dispositivos de un robot que no se encuentre en la lista de los implementados, es necesario crear los *drivers* para el manejo de los mismos, adaptándolos a las *interfaces* existentes (siempre y cuando exista una asociada a ese tipo de dispositivos, sino, será necesario crearla también).

5.1.4. Arquitectura de *Player*

Player ha sido diseñado con el propósito de que resulte sencillo añadir nuevos dispositivos, como un sistema asíncrono y para ser independiente de lenguaje y plataforma. Está implementado en C y C++, y hace uso de hilos POSIX para crear programas multihilo.

Dentro de *Player* hay un hilo servidor y un hilo por cada dispositivo abierto. El hilo servidor atiende las conexiones de nuevos clientes a través de

un *socket* TCP, recibe las órdenes de todos los clientes conectados y envía los datos y respuestas de los dispositivos a cada cliente. Cuando *Player* recibe una petición de un cliente sobre un dispositivo con el que aún no se tiene comunicación, crea un nuevo hilo que realiza la comunicación con dicho dispositivo. El funcionamiento de *Player* se resume en la figura 5.3.

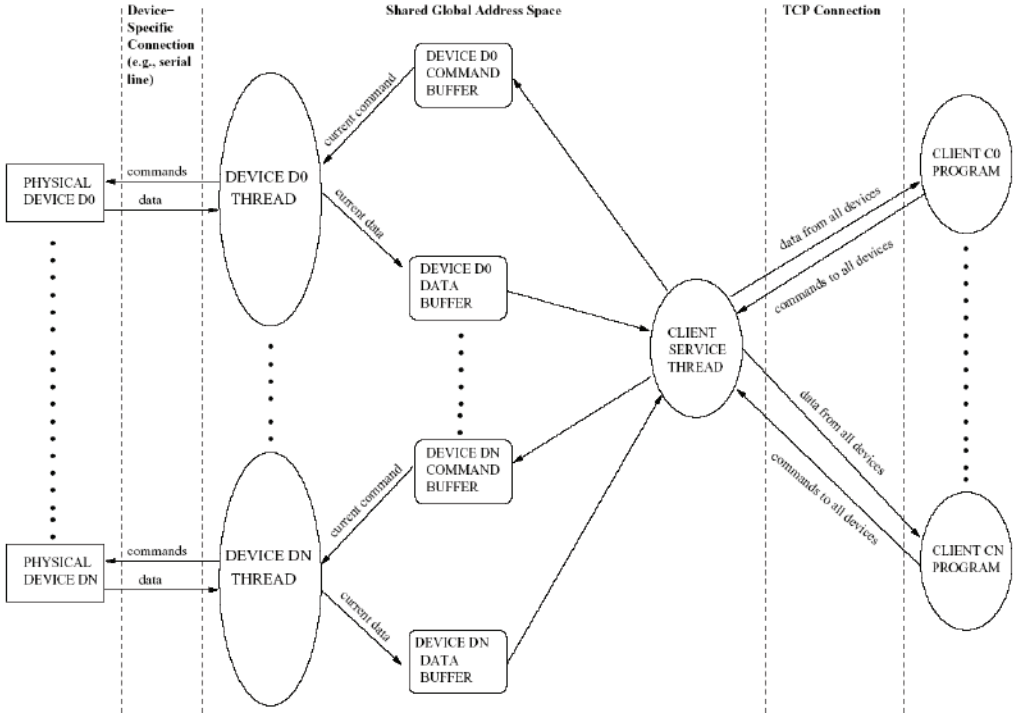

Figura 5.3: *Arquitectura del servidor Player*

La comunicación entre los hilos de los dispositivos se realiza a través de un espacio de direcciones global compartido. Cada dispositivo tiene asociado un *buffer* de órdenes y un *buffer* de datos, protegidos por secciones *mutex*. Estos *buffers* proporcionan un canal de comunicación asíncrona entre los hilos de los dispositivos y el hilo que atiende a los clientes. Así cuando este hilo recibe una orden de un cliente para un dispositivo, escribe la orden en el *buffer* de órdenes correspondiente, y cuando el hilo del dispositivo está preparado para enviar una nueva orden, lo lee de su *buffer* de órdenes y lo envía al dispositivo. Del mismo modo cuando el hilo del dispositivo recibe un dato del mismo, lo

escribe en el *buffer* de datos, y después cuando el hilo que atiende a los clientes está preparado para enviar datos, lo lee del *buffer* de datos y se lo pasa a un cliente en particular.

El cliente puede ejecutarse en el mismo *host* que *Player* o en otro *host* que tenga conexión con el de *Player* a través de la red TCP/IP.

Cuando un cliente lee de un dispositivo, el cliente recibe el estado del dispositivo en el instante en el que es leído. Este estado suele ser enviado al cliente con una frecuencia de 10 Hz o menos (es configurable). Por su parte si un cliente escribe en un dispositivo, está enviando una orden al mismo. Esta orden suele incluir los parámetros del dispositivo que más frecuentemente cambian durante el uso del mismo. Por ejemplo en el dispositivo *position* que controla el robot móvil, la orden es un conjunto de velocidades.

Finalmente los dispositivos pueden ser configurados desde *Player*. Mientras que la lectura y escritura son asíncronas, la configuración es un proceso de petición-respuesta síncrona entre cliente y dispositivo. Cuando un cliente envía una petición de configuración al servidor, la petición se añade a una cola de entrada para el dispositivo correspondiente, y cuando el dispositivo sirve la petición genera una respuesta que se añade a la cola de salida del dispositivo. Esta respuesta será transmitida por el servidor al cliente que se encuentra esperando. Otro aspecto que lo diferencia de la lectura y escritura, es que las peticiones de configuración no se sobrescriben. Normalmente la configuración se emplea para asignar o solicitar algún aspecto del estado del dispositivo, siendo un proceso menos frecuente que la lectura y escritura.

5.1.5. Valoración general

Se puede decir que *Player* es un servidor multi-robot que constituye una capa de abstracción entre los dispositivos robóticos y los programas clientes que acceden a ellos. De modo que facilita en gran medida la creación de programas clientes, en varios lenguajes de programación, para el manejo de los robots.

Presenta numerosas ventajas, como son:

- Código abierto: permite adaptar el código a nuestras necesidades, integrar nuevos dispositivos, crear nuevas interfaces...

- Ha sido usado en múltiples proyectos de investigación, por lo que su fiabilidad y eficiencia están más que probadas.

- Constituye una herramienta muy útil para la investigación en robótica, ya que permite desarrollar programas clientes sin tener que adaptarlos a dispositivos o arquitecturas particulares.

- Permite acceder a múltiples dispositivos simultáneamente, incluso de robots distintos.

- Soporta múltiples programas clientes, los cuales pueden estar escritos en distintos lenguajes de programación (C, C++, Java, Python ...)

No obstante también posee algún aspecto un poco menos ventajoso, como por ejemplo, el hecho de que está pensado para trabajar con dispositivos a través de conexiones serie únicamente. Es decir, *Player* contiene *drivers* para el control de determinados modelos de dispositivos, de modo que en la implementación de esos *drivers* se establece una comunicación directa con el dispositivo, el cuál posee una *interfaz* o protocolo de comunicación, que es respetado por el *driver*.

Esta forma de trabajar permite que dicho *driver* sirva para cualquier ejemplar de ese modelo de dispositivo, siempre y cuando exista una conexión serie con él. Esto implica dos limitaciones:

- Cuando *Player* se ejecuta en un PC, la cantidad de dispositivos que pueden ser controlados a través de él es bastante escasa, ya que se limita a un dispositivo por conexión serie del PC.

- El radio de acción para una conexión serie es bastante limitado, sobre todo si lo comparamos con conexiones TCP/IP inalámbricas.

5.1.6. Integración de dispositivos en Player

Uno de los aspectos interesantes que presenta *Player*, es que el conjunto de *drivers* que tiene implementados, permiten controlar dispositivos de algunos robots comerciales (posee *drivers* para controlar todos los dispositivos de: Pioneer 1, Pioneer 2, AmigoBot, RWI B-series robots...). Esto implica que, si quiere usarse con robots de fabricación propia (como ocurre en el caso del presente proyecto), u otros que no estén entre los implementados, es necesario desarrollar un conjunto de *drivers* para los dispositivos de estos robots. *Player* ofrece la posibilidad a sus usuarios de añadir al proyecto nuevos *drivers* de dispositivos, de modo que dichos dispositivos puedan pasar a ser controlados a través de él.

Otro problema adicional en el caso de los robots de fabricación propia es que no es posible controlar algunos tipos de dispositivos a través del puerto serie de un PC, como es el caso de los actuadores empleados en el robot construido en este proyecto. (Ver Anexo Hardware).

El uso de *Player* para el control de dispositivos robóticos a través de conexiones serie, como se comentó en el apartado anterior, permite un radio de acción bastante limitado, debido a la longitud finita de los cables. A diferencia de las conexiones serie, el uso de conexiones TCP/IP inalámbricas puede ser muy ventajoso, ya que a través de este tipo de comunicación se puede alcanzar un mayor radio de acción. Sin embargo, el control de dispositivos a través de conexiones TCP/IP presenta el problema de que a través de este tipo de conexiones no se puede acceder directamente a los dispositivos, sino que se necesita que el robot tenga una aplicación que atienda las peticiones que *Player* realiza sobre los dispositivos del robot.

A. Morales [19] presenta una solución para el problema de las comunicaciones TCP/IP inalámbricas entre *Player* y los dispositivos robóticos, en su Trabajo de Grado, realizado en el Departamento de Informática y Automática de la Universidad de Salamanca.

La idea que presenta es que los *drivers* de *Player* no accedan directamente a los dispositivos, sino que se comuniquen con una aplicación que se ejecute en

el robot móvil a través de TCP/IP inalámbrico. Por tanto ya no es necesario hacer que cada *driver* sea particular para un modelo de dispositivo. Se pueden crear una serie de *drivers* genéricos para cada tipo de dispositivo, de forma que sea la aplicación que se ejecuta en el robot la que se adapte a estos *drivers*. Esta forma de trabajar evita tener que crear un *driver* específico para cada modelo de dispositivo e incluso permite crear un conjunto de *drivers* genéricos que eviten tener que modificar una sola línea del código de *Player*.

La arquitectura propuesta en [19] consiste en añadir una capa más a la arquitectura cliente-servidor que emplea *Player*, de manera que en el robot móvil se ejecute una aplicación servidora que se encargue de acceder a los dispositivos a petición de *Player*, el cual hará de intermediario entre los programas clientes y la aplicación servidora del robot móvil. De esta forma es posible crear una serie de *drivers* genéricos (uno por cada tipo de dispositivo: uno para sonar otro para infrarrojos,...) que se comunicarán con la aplicación servidora del robot móvil a través de TCP/IP inalámbrico, en lugar de con los propios dispositivos directamente a través del puerto serie. Para que los *drivers* genéricos sean válidos con cualquier aplicación servidora de cualquier robot móvil, es necesario establecer un protocolo de comunicación entre ambas partes, el cual deberá ser respetado por la aplicación servidora del robot que se quiera integrar en *Player*. Así se evitará tener que crear un nuevo *driver* por cada modelo particular de dispositivo.

La arquitectura en concreto consta tres capas (ver figura 5.4) en la que *Player* actúa como servidor de los programas clientes y al mismo tiempo como cliente de una aplicación servidora que se ejecuta en el robot y que se encarga de acceder a los dispositivos de éste.

La capa más baja está formada por una aplicación servidora necesaria para acceder desde los *drivers* de *Player* a los dispositivos, ya que no es posible acceder a través de comunicaciones TCP/IP directamente a los dispositivos. La aplicación servidora que compone esta capa más baja accederá, por un lado, a los dispositivos a través de conexiones serie o de otro tipo y, por otro, se comunicará a través de un protocolo propio sobre TCP/IP, con los *drivers*

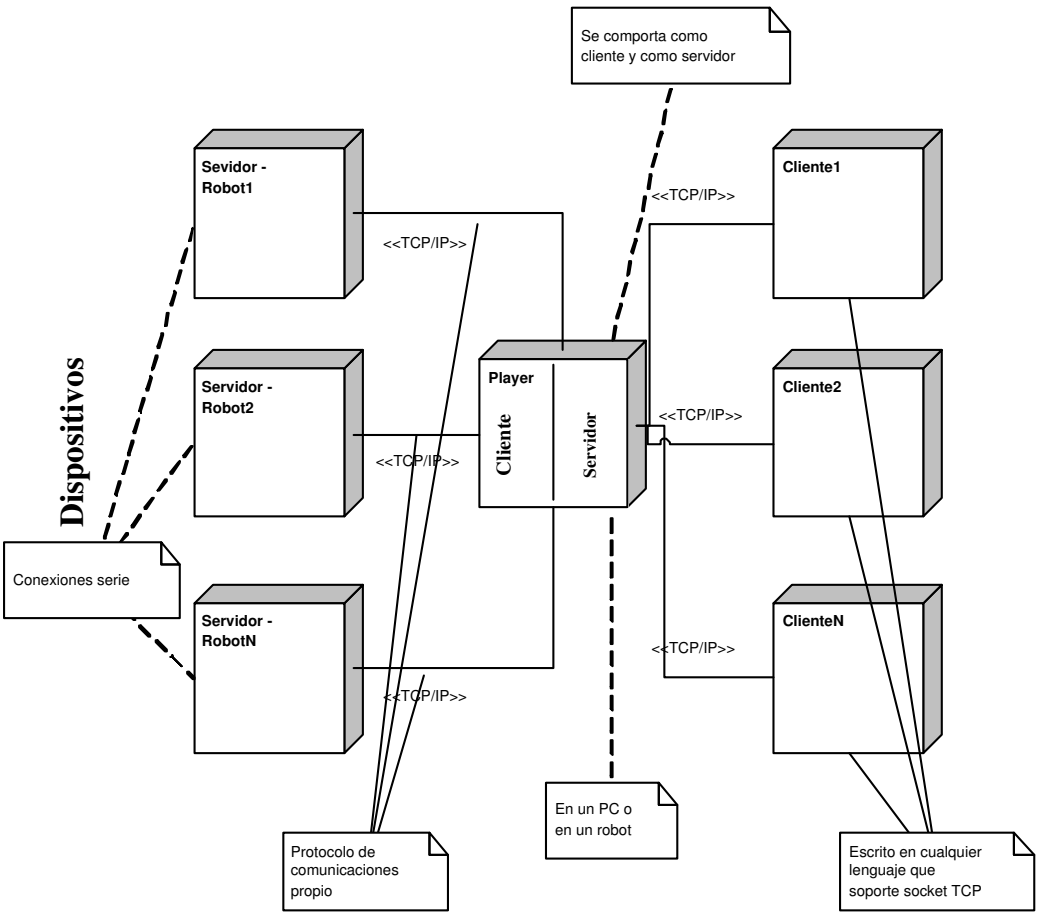

Figura 5.4: *Arquitectura de Player modificada por A. Morales.*

genéricos que forman parte de *Player*.

La capa intermedia la constituye el servidor *Player* modificado con los drivers genéricos, el cual ahora, además de comportarse como servidor atendiendo las peticiones de los programas clientes, se comportará como cliente de la capa más baja (aplicación servidora en el robot móvil). El acceso a los dispositivos ya no se realiza directamente a través de los *drivers* implementados en *Player*, sino que se crearán unos *drivers* genéricos, que se comportarán como clientes de la aplicación servidora que se ejecuta en el robot. La comunicación entre los *drivers* genéricos (clientes) y el servidor se realizará a través de un protocolo propio.

La creación de una serie de *drivers* genéricos (clientes) para cada tipo de dispositivo (uno para sonar, otro para infrarrojos, otro para *position*, otro para *batery*...), evita tener que crear un *driver* específico para cada dispositivo concreto del robot y permite usar estos sin necesidad de modificar código de *Player*. Es decir un robot para cuyos dispositivos existan *drivers* genéricos ya implementados en *Player* y que posea el servidor que proponemos, podrá controlarse a través de *Player* sin modificar su código.

La capa más alta está formada por los programas clientes, los cuales pueden estar escritos en cualquier lenguaje que soporte *sockets* TCP y hacer uso de la API (*Aplication Programming Interface*) que *Player* les ofrece para acceder a distintos tipos de dispositivos. Esta capa no se verá afectada en esta nueva arquitectura.

Finalmente, cabe destacar que esta nueva arquitectura de *Player* ha sido de gran ayuda para lograr controlar los actuadores del robot, permitiendo que éste fuera totalmente inalámbrico.

5.2. Bibliotecas de cálculo científico

Cuando se utilizan algoritmos para la resolución de problemas de álgebra lineal y cálculo numérico, la mayor parte de los recursos computacionales utilizados dependen del método de resolución del problema y no de los detalles

de construcción de alto nivel del algoritmo. El hecho es que el rendimiento de muchos de estos algoritmos depende en gran parte del rendimiento de los cálculos que se hacen con las matrices.

Un análisis teórico de complejidad puede proporcionar cierta estimación del rendimiento, sin embargo, una comprobación real del rendimiento se obtiene a través de una técnica conocida como *profiling*[1]. Esta técnica consiste en modificar ligeramente un programa de tal modo que su ejecución pueda ser monitorizada detalladamente: por ejemplo, cuántas veces se llama cada subrutina y cuánto tiempo se gasta en cada una. En general el uso de *profiling* revela que el rendimiento de un programa está dominado por unas pocas subrutinas clave.

Para poder construir software de cálculo numérico muy optimizado es necesario dominar muy a fondo el álgebra lineal, puesto que un análisis basado en *profiling* así lo determina.

Por lo tanto, si se desea construir software numérico optimizado, y que se ejecute con la mayor rapidez posible, nadie debería implementar código para realizar tareas simples como el producto vectorial, la multiplicación de dos matrices, el cálculo de los vectores y valores propios, etc., siempre que las dimensiones de los vectores y matrices implicados sea grande. Lo que se debería hacer es utilizar alguna de las múltiples librerías gratuitas existentes puesto que son mucho más maduras y realizadas por un amplio número de expertos en la materia. Otras razones son las siguientes:

- Centrarse en otras tareas que también sean importantes.
- La cantidad de código que será necesaria escribir se verá reducida.
- Se reducirá, por tanto, el número de errores en el código del programa.
- No hay que tratar con aspectos de optimización numérica a veces realmente difíciles.

[1]No se traducirá este vocablo al castellano debido a la carencia de una traducción adecuada.

- El código se ejecutará mucho más rápido, en ocasiones de manera espectacular.

A continuación se comentarán dos bibliotecas ampliamente utilizadas en el cálculo numérico: BLAS y LAPACK, que se pueden obtener en la web Netlib [28]. En Netlib se encuentran disponibles la mayoría de las bibliotecas más populares para el cálculo numérico, especialmente para el álgebra lineal, y es un sitio mantenido por la Universidad de *Tennesse*, el *Oak Ridge National Laboratory*, y por colaboradores de todo el mundo. Las bibliotecas BLAS y LAPACK en concreto son de tal prestigio que son usadas por importantes paquetes de cálculo numérico como Mathematica y Matlab.

5.2.1. *Numerical Recipes*

Antes de comenzar con BLAS y LAPACK es necesario hablar sobre otra referencia importante en el mundo del cálculo numérico informatizado: la serie de libros *Numerical Recipes* ([26]), que consta de cuatro variantes:

- *Numerical Recipes in C*
- *Numerical Recipes in Fortran 77*
- *Numerical Recipes in Fortran 90*
- *Numerical Recipes in C++*

Estos libros contienen implementaciones de una gran cantidad de algoritmos para resolver problemas de cálculo numérico de álgebra lineal, incluidos el cálculo de la matriz de covarianza y el cálculo de los vectores y valores propios. La serie *Numerical Recipes* se distingue por dos aspectos. El primero es que son unos libros muy accesibles: se encuentran *on-line* y el código puede descargarse rápidamente de la web de manera gratuita. El segundo es que en esta serie de libros no se presentan implementaciones altamente optimizadas, sino que por el contrario su valor es más bien *pedagógico* que *práctico*; se enfatiza en la comprensión de las técnicas. En otras palabras, si se pretende

aprender cómo funciona un cierto algoritmo, estos libros pueden ser de gran ayuda; pero si lo que realmente se desea es *utilizar* un algoritmo y que se ejecute lo más rápidamente posible, lo más recomendable es utilizar bibliotecas como BLAS y LAPACK.

5.2.2. BLAS

La biblioteca BLAS (*Basic Linear Algebra Subroutines*, Subrutinas Básicas de Álgebra Lineal) es un conjunto de funciones que implementan operaciones fundamentales del álgebra lineal tales como productos escalares entre vectores, multiplicaciones matriz-vector y operaciones matriz-matriz. Se puede obtener en el sitio web [21].

Las funciones implementadas en BLAS son consideradas como bloques de construcción de alta calidad para realizar operaciones básicas de matrices y vectores. Son muy eficientes, portables y ampliamente disponibles, por lo cual son usadas habitualmente como base para construir otros paquetes software de álgebra lineal de más alta calidad, como por ejemplo LINPACK o LAPACK.

Existen versiones propias optimizadas por ciertos fabricantes, como por ejemplo: Compaq, HP, IBM, Intel, Silicon Graphics y SUN.

En sus orígenes la biblioteca BLAS se escribió en Fortran 66 y Fortran 77, pero se pueden utilizar desde otros lenguajes como C y Java, directamente, o a través de una interfaz que sirve de envoltorio (*wrapper interface*). En concreto la interfaz C se denomina CBLAS, y se ha utilizado en este proyecto. Se puede obtener en [22].

Niveles de BLAS

Los requisitos y el desarrollo de la biblioteca BLAS se realizaron en tres etapas o niveles: Nivel 1 (1973-77), Nivel 2 (1984-86) y Nivel 3 (1987-1990). Cada uno de estos niveles corresponde a un cierto nivel de complejidad teórica:

- *Nivel 1:* operaciones entre vectores de complejidad *O(n)*, como por ejemplo:

$$y \leftarrow \alpha x + y$$

así como productos escalares y módulos.

- *Nivel 2:* operaciones entre matrices y vectores de complejidad $O(n)^2$, como por ejemplo:

$$y \leftarrow \alpha Ax + \beta y$$

Además ofrece, entre otras, la resolución de sistemas del tipo:

$$Ax = y$$

para x.

- *Nivel 3:* operaciones entre matrices de complejidad $O(n)^3$, de la forma:

$$C \leftarrow \alpha AB + \beta C$$

También resuelve entre otras cosas, por ejemplo la ecuación:

$$B \leftarrow \alpha A^{-1}B$$

Para obtener el máximo rendimiento de la biblioteca BLAS, lo más recomendable es realizar los cálculos de tal modo que se maximice el uso de las operaciones de más alto nivel que sea posible, especialmente si se puede lo mejor es utilizar operaciones de Nivel 3. Esto es debido a que las operaciones de Nivel 3 ejercen un mejor control de los accesos a memoria que las funciones equivalentes de Nivel 2, y son precisamente los accesos a memoria los que dictan el rendimiento del algoritmo.

5.2.3. LAPACK

LAPACK [23] responde a las siglas *Linear Algebra PACKage* (en castellano, Paquete de Álgebra Lineal), e implementa una serie de cálculos

algebráicos más avanzados que BLAS, diseñados para resolver una amplia variedad de tipos de sistemas lineales y para realizar descomposiciones y factorizaciones de matrices. LAPACK fue lanzado por primera vez en febrero de 1992, y la última versión (3.0) en Mayo de 2000. Sustituye a sus predecesores, EISPACK y LINPACK, proporcionando una mayor funcionalidad, mejor precisión y mejor rendimiento.

Esta biblioteca está construida utilizando como base las rutinas BLAS, lo que significa que su rendimiento depende en gran medida del rendimiento de la biblioteca BLAS que se esté usando. Está escrita en Fortran 77 y proporciona funciones para resolver sistemas de ecuaciones lineales simultáneos, soluciones de mínimos cuadrados en sistemas de ecuaciones lineales, problemas de vectores y valores propios, así como para realizar factorizaciones de matrices (LU, Cholesky, QR, SVD, Schur...) y otras muchas operaciones.

El objetivo original de LAPACK fue hacer que las ampliamente utilizadas bibliotecas EISPACK y LINPACK se ejecutaran de manera eficiente en procesadores vectoriales y paralelos de memoria compartida. En este tipo de máquinas, LINPACK y EISPACK eran ineficientes porque los patrones de acceso a memoria que utilizaban ignoraban las jerarquías de memoria multicapa de dichas máquinas, gastando más tiempo moviendo los datos en lugar de gastarlo en hacer operaciones en punto flotante. LAPACK soluciona este problema reorganizando los algoritmos de modo que usen operaciones de bloques de matrices, tales como multiplicación de matrices en los bucles más internos. Estas operaciones de bloque pueden ser optimizadas para cada arquitectura de tal modo que tengan en cuenta la jerarquía de memoria y también proporcionan un modo "transportable" de lograr una alta eficiencia en diversas máquinas modernas.

La guía del usuario de la biblioteca LAPACK está disponible electrónicamente en [25].

Categorías de funciones

Las funciones de la biblioteca LAPACK están divididas en tres categorías: **auxiliares**, **de cálculo** y ***drivers***. Las funciones auxiliares realizan una serie de tareas de bajo nivel, y su intención primordial es servir de soporte a otras funciones. Las funciones de cálculo están diseñadas para realizar tareas simples y específicas:

- factorizaciones: $LU, LL^T/LL^H, LDL^T/LDL^H, QR, \ldots$
- cálculo de valores y vectores propios para matrices simétricas y no simétricas.
- cálculo generalizado de vectores y valores propios.

Finalmente, las funciones *drivers* combinan los dos tipos de funciones anteriores en el orden adecuado para lograr resolver una serie de problemas de álgebra lineal más complejos que los anteriores:

- Ecuaciones lineales: $AX = B$;
- Problemas lineales de cuadrados mínimos como por ejemplo:

$$minimizar_x \|b - Ax\|_2$$

- Problemas lineales generalizados de mínimos cuadrados:

$$minimizar_x \|c - Ax\|_2$$

$$\text{sujeto a } Bx = d$$

- Problemas generalizados de vectores y valores propios:

$$AZ = \Lambda Z, A = U\Sigma V^T, Z = \Lambda BZ$$

En general se obtiene más rendimiento usando las funciones *drivers*, siempre que encajen dentro de lo que se necesita. Sin embargo, acceder a las rutinas de cálculo individuales proporciona flexibilidad adicional, especialmente cuando se trata de resolver problemas no estándar o cuando se necesita acceder a resultados intermedios.

5.3. GNOME

GNOME [33] es un acrónimo de *"GNU Network Object Model Environment"*, entorno de trabajo en red orientado a objetos, por lo que forma parte del más amplio proyecto GNU (*GNU's Not Unix*). El proyecto GNU [34] nació en 1984 con el objetivo de desarrollar un sistema operativo tipo UNIX completamente libre.

GNOME es un entorno gráfico (escritorio de trabajo) amigable que permite a los usuarios configurar y usar sus ordenadores de una forma sencilla. Incluye un panel (para arrancar aplicaciones y presentar el estado de funcionamiento), un escritorio (donde se pueden situar los datos y las aplicaciones), un conjunto estándar de aplicaciones y herramientas de escritorio, y un conjunto de convenciones que facilitan la operación y consistencia de las aplicaciones entre sí.

GNOME es en su totalidad código abierto (software libre), con el código fuente disponible libremente y desarrollado por cientos de ingenieros en todo el mundo. GNOME tiene una serie de ventajas para los usuarios. Facilita el uso y la configuración de aplicaciones usando una simple pero potente interfaz gráfica (*console*).

Es altamente configurable, permitiendo ajustar su escritorio con el aspecto que se desee. El gestor de sesiones de GNOME recuerda la configuración previa, de manera que una vez que se hayan configurado las cosas, las mantendrá así. Además, soporta muchos idiomas, y se pueden añadir más sin hacer modificaciones en el software. Soporta incluso varios protocolos de arrastrar y soltar (*drag and drop D&D*) para una máxima interoperabilidad con aplicaciones que no sean compatibles con GNOME.

Posee además una serie de ventajas para los desarrolladores que indirectamente benefician también a los usuarios. Los desarrolladores no necesitan comprar una cara licencia de software para hacer compatibles sus aplicaciones comerciales. De hecho, GNOME es independiente de cualquier compañía. Ningún componente del interfaz está controlado únicamente por una compañía o tiene restringidas la modificación o redistribución. Las aplicaciones

GNOME pueden desarrollarse en una gran variedad de lenguajes de programación, por lo que el programador no se ve limitado a un solo lenguaje de programación. GNOME usa la arquitectura común de petición de servicios a objetos (CORBA) para permitir que los componentes software interactúen perfectamente, independientemente del lenguaje en el que estén implementados, o incluso de la máquina en la que se estén ejecutando. Finalmente, GNOME funciona en numerosos sistemas operativos de tipo UNIX.

Actualmente se distribuye la versión 2.6 de GNOME. En esta versión se han mejorado el rendimiento, la usabilidad, las fuentes y gráficos, así como otra serie de características.

5.4. GLIB

GLIB [35] es una biblioteca escrita en C que ofrece un conjunto de funciones comunes ampliamente utilizadas en la biblioteca GTK+ [35] y en el entorno GNOME. En ella se ofrecen los servicios para tipos de datos, tanto estándares como tipos abstractos (árboles, listas, tablas hash...) así como un conjunto de funciones de tipo general.

Tipos de Datos

En lugar de los tipos de datos estándar de C, GLIB utiliza su propio conjunto de tipos. Este enfoque facilita la portabilidad hacia otras plataformas de cualquier aplicación que utilice esta biblioteca, ya que el cambio de plataforma estribaría en la necesidad de portar GLIB solamente. Dado que GTK+ utiliza los tipos proporcionados por GLIB, portar GTK+ también sería, en teoría, sólo cuestión de portar GLIB y GDK.

Los tipos simples de datos que proporciona GLIB y sus equivalentes en lenguaje C se muestran en la tabla 5.1.

Se puede ver que algunos de los tipos, como `gint`, no tienen sentido ya que simplemente es cuestión de añadir el prefijo `g` al tipo, pero el sentido de esto es el poder conseguir un código más consistente y limpio ya que facilitan

Nombre GLIB	Equivalente en C	Descripción
`gchar`	`char`	Carácter
`guchar`	`unsigned char`	Carácter sin signo
`gshort`	`short`	Entero corto
`gushort`	`unsigned short`	Entero corto sin signo
`glong`	`long`	Entero largo
`gulong`	`unsigned long`	Entero largo sin signo
`gint`	`int`	Entero
`guint`	`unsigned int`	Entero sin signo
`gfloat`	`float`	Número real
`gdouble`	`double`	Número real de doble precisión
`gboolean`	`int`	Almacena TRUE/FALSE
`gpointer`	`void *`	Punteros a objetos
`gconstpointer`	`const void *`	Punteros a objetos constantes

Tabla 5.1: *Tipos de datos simples GLIB.*

la portabilidad de la aplicación como se ha dicho antes. Algunos tipos como `gboolean` sirven para aclarar el código ya que son más expresivos que su homólogo en C. Bien es verdad que el tipo `int` o `gint` podrían ser utilizados con la misma finalidad, pero se ha introducido este tipo para aumentar la legibilidad y limpieza del código.

Tipos Abstractos

También llamados contenedores en GLIB. Sirven de gran ayuda, ya que corresponden a la implementación de tipos de datos abstractos, muy comúnmente utilizados. Son los mostrados en la tabla 5.2.

Nombre	**Descripción**
`GList`	Lista doblemente enlazada
`GSList`	Lista enlazada simple
`GHashTable`	Tabla hash
`GCache`	Caché
`GTree`	Árbol binario balanceado
`GNode`	Árbol n-ario
`GString`	Cadena de caracteres de tamaño dinámico
`GArray`	Array de tamaño dinámico
`GByteArray`	Array de tipos de 8 bits de tamaño dinámico

Tabla 5.2: *Tipos de datos abstractos GLIB.*

Miscelánea

GLIB, a parte de una amplia variedad de tipos, como se ha visto anteriormente, también ofrece un conjunto de funciones de propósito general que se puede agrupar según su funcionalidad:

Mensajes: conjunto de cuatro funciones para mostrar información y cada una de ellas puede ser ampliada para desarrollar interfaces gráficas y no gráfi-

cas. Estas cuatro diferentes funciones tienen cuatro niveles de tratamiento de mensajes, que van desde el error no recuperable hasta la salida estándar. Cada función muestra un tipo diferente de mensaje y admite un número variable de parámetros.

```
g_error(‘‘Esto es un error,\n’’);
```

Depuración: conjunto de macros que se usan durante el desarrollo de aplicaciones para verificar suposiciones hechas en el código. Estas suposiciones suelen estar asociadas a zonas "peligrosas"del programa cuyo error podría producir el fallo de la aplicación o incluso del sistema.

```
g_return_if_fail(i > 0);
```

Multiproceso: realmente es una simulación de multihilo, más que de multiproceso, ya que todos los hilos creados comparten la misma memoria. Esto facilita la comunicación entre los hilos mediante esta memoria compartida, aunque tiene la complicación de los problemas de concurrencia, exclusión mutua, sincronización... Las funciones que proporciona GLIB son un medio para la escritura de un código multihilo seguro.

Gestión de memoria

Para reemplazar las funciones estándar `malloc` y `free`, así como `new` y otras funciones de tratamiento de memoria.

Funciones de Cadena: GLIB proporciona también un conjunto de funciones que mejoran el uso de cadenas simples en lenguaje C, aunque se recomienda el uso del tipo de datos que proporciona `GString`.

```
cadena = g_strdup(‘‘Cadena a copiar.\n’’);
```

Canales de Entrada/Salida: un conjunto de métodos utilizado para facilitar el uso de descriptores de archivos, tuberías y *sockets*, los canales de entrada/salida se utilizan de forma transparente mediante el uso de las funciones que aporta la biblioteca GLIB y se manejan de forma asíncrona, de forma que es un método factible de utilizarlos como medios de sincronización entre procesos o como medio de comunicación entre máquinas.

Temporizadores: conjunto de funciones que se usan para el manejo de temporizadores, que son elementos que avisan del paso de un cierto tiempo.

5.5. GDK

GDK (*Graphics Drawing Kit* - Kit de Dibujo Gráfico) es una capa de bajo nivel para dibujo, situada entre GTK+ y la interfaz de programación de aplicaciones (API) específica del sistema operativo: Xlib en el caso de Linux. Puesto que GTK+ no tiene interacción directa con la API de la máquina, el portar GTK+ a otro sistema, es cuestión de portar GDK y GLIB. GDK proporciona la capacidad de dibujar hasta el nivel de píxel, y proporciona funciones de bajo nivel para la creación y manipulación de ventanas.

Eventos GDK

Los eventos GDK son un conjunto de señales que intercambian los objetos GDK indicando algún cambio en su estado o alguna interacción con el usuario que pueda ser interceptado para ampliar la funcionalidad del objeto. Se asocian también a los objetos GTK+ de forma que ciertos *widgets* creados con GTK+ pueden interceptar eventos GDK. También se asocian a los ítems incluidos en el *canvas* de GNOME.

Eventos de botón: Existen cuatro tipos diferentes:

- `GDK_BUTTON_PRESS`: el botón ha sido presionado.

- `GDK_BUTTON_RELEASE`: el botón ha sido soltado después de haber sido presionado.
- `GDK_2BUTTON_PRESS`: se ha realizado un doble click sobre el botón.
- `GDK_3BUTTON_PRESS`: se ha realizado un triple click sobre el botón.

La forma de recibir los eventos por parte de los objetos GDK consiste en una cola de eventos de forma que, por ejemplo, la recepción de un evento de doble click implica la llegada de dos eventos `GDK_BUTTON_PRESS` y dos eventos `GDK_BUTTON_RELEASE`.

Eventos de teclado: Existen dos tipos diferentes:

- `GDK_KEY_PRESS`: se ha pulsado una tecla.
- `GDK_KEY_RELEASED`: se ha soltado la tecla.

Eventos de ratón: Estos eventos se producen cuando el usuario desplaza el cursor del ratón en la pantalla, de modo que se producen los siguientes eventos GDK para un objeto `GdkWindow`:

- `GDK_MOTION_NOTIFY`: Se produce mientras el ratón se mueve dentro del objeto
- `GDK_ENTER_NOTIFY`: se produce cuando el ratón entra en la zona visible de un objeto.
- `GDK_LEAVE_NOTIFY`: se produce cuando el ratón sale de la zona visible del objeto.
- `GDK_CROSSING_NOTIFY`: se produce cuando el ratón entra o sale de la zona visible del objeto.

Eventos de exposición Estos eventos se producen cuando un objeto que estaba superpuesto a otro desaparece o cambia su posición, dejando una nueva área visible del objeto colocado en el fondo, así como cuando una ventana aparece por primera vez en la pantalla.

Eventos de ventana Estos eventos se producen cuando una ventana se muestra, se esconde, es redimensionada o destruida.

5.6. GTK

GTK+ [35] es una biblioteca diseñada mediante una visión orientada al objeto que fue implementada en el lenguaje de programación C con el propósito de facilitar al desarrollador la tarea de diseñar una interfaz gráfica de usuario de una forma independiente del sistema de ventanas que se utilice.

Uno de los aspectos más importantes que GTK+ buscó desde su inicio fue la independencia de plataforma, para ello se basa en las bibliotecas GLIB, a nivel lógico, y GDK como intermediario entre la interfaz gráfica y el sistema de ventanas en el que se desarrolle, debido a ello, portar GTK+ de una plataforma a otra "sólo" requiere portar GLIB y GDK. Esta dependencia se puede apreciar en la figura 5.5

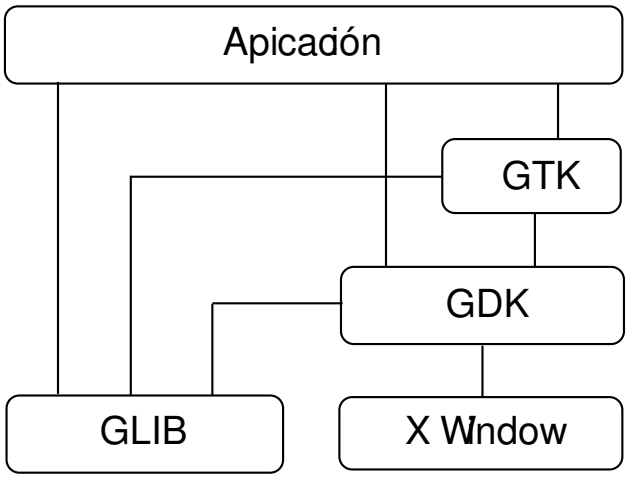

Figura 5.5: *Esquema de dependencias de GTK+*

Para crear aplicaciones con una interfaz gráfica se usan los *widget*. Un *widget* de GTK+ es un componente de interfaz gráfica de usuario. Es una estructura `GtkWidget` que define un tipo de datos genérico utilizado por todos los *widget* y ventanas en GTK+.

A la hora de crear todos los *widgets* de una aplicación GTK, estos deben tener una relación padre/hijo en la que el padre es el contenedor y el hijo es incluido dentro de dicho contenedor.

Los *widget* de GTK+ derivan de otros *widget*. Por ejemplo `GtkButton` deriva de un contenedor `GtkContainer` que a su vez deriva de un objeto GTK+ (`GtkObject`). Todas las funciones de creación de *widgets* devuelven un puntero a un tipo de dato `GtkWidget`, que es un *widget* genérico del que derivan para ahorrar esfuerzos en la implementación de funcionalidades que ya tienen otros *widgets*.

Aunque al derivar de otros *widgets* se puede pensar que GTK+ se basa en C++, se eligió C por varias razones:

- Es el lenguaje principal para aquéllos que desarrollan programas en UNIX/GNOME.
- Es más portable que C++. Adoptado como estándar desde hace más tiempo (cuando GTK+ empezó C++ no estaba estandarizado).
- C++ presenta problemas a la hora de trabajar con diferentes compiladores.

Las señales son necesarias en la programación de aplicaciones debido a que el programa debe ser capaz de responder a las acciones que el usuario realice y pueda enviarse una señal a una función de *callback*[2] de la aplicación. En GTK+ se están generando señales continuamente, pero la mayoría de ellas son ignoradas. Cuando es necesario tratar una señal, hay que registrar la función de *callback* y asociarla con un *widget*.

Los *widgets* pueden registrar *callbacks* y una misma *callback* puede registrarse ante múltiples *widgets*. Cuando se genera una señal destinada a un *widget*, se ejecuta la función de *callback* del *widget* para dicha señal.

[2]El autor del proyecto prefiere utilizar el término inglés *callback*, en lugar de rellamada o retrollamada

5.7. GnomeCanvas

El canvas de GNOME es un motor para gráficos estructurados que ofrece un *renderizado* de alto rendimiento y una API de alto nivel. Permite la elección de dos modos de renderizado, uno basado en **Xlib** (a través de GDK) para una visualización extremadamente rápida, y otro basado en **Libart**, más sofisticado, con capacidad de *antialiasing* y manejo de transparencias.

Un *canvas* es una ventana con una colección de ítems gráficos en su interior. El *canvas* está diseñado para funcionar como un motor visualizador de propósito general para las aplicaciones, de modo que los desarrolladores no tengan que construir el suyo propio usando GDK. Proporciona ítems simples de tipo rudimentario como son rectángulos, elipses y texto, para ser usados por las aplicaciones con necesidades gráficas modestas. Sin embargo también proporciona la posibilidad de que el desarrollador puede crear ítems más sofisticados.

Una de las cualidades más destacadas del *canvas* de cara al rendimiento es que maneja de manera interna la técnica del doble buffer, de modo que la superficie de dibujo nunca parpadea. Los ítems del *canvas* son objetos GTK+ normales, y por lo tanto emiten señales basadas en los eventos del ratón y el teclado. Esto permite al programador implementar en los ítems un comportamiento interactivo con el usuario, al igual que el resto de *widgets* de GTK+.

El *canvas* maneja automáticamente los eventos de exposición GDK además de los eventos de ventana. Sin embargo, los eventos de ratón y de teclado los ha de manejar explícitamente el desarrollador para cada ítem del *canvas*.

5.8. Matlab

Acrónimo de *"**Mat**rix **Lab**oratory"*, el lenguaje MATLAB [36] fue inventado a finales de la década de 1970 por *Cleve Moler*, director del departamento de Ciencias de la Computación de la Universidad de Nuevo Méjico. Lo diseñó para permitir a sus estudiantes acceder a *LINPACK* y *EISPACK*

sin necesidad de aprender Fortran. Pronto se difundió a otras universidades y se encontró con un gran público en la comunidad de matemática aplicada.

El ingeniero *Jack Little* conoció MATLAB mientras *Cleve Moler* realizaba una visita a la Universidad de Stanford en 1983. Rápidamente reconoció su potencial comercial, y se unió a *Cleve Moler* y *Steve Bangert*. Reescribieron MATLAB en C, y fundaron *The MathWorks* en 1984 para continuar su desarrollo. En un inicio MATLAB se adaptó más a la ingeniería de control (especialidad de *Jack Little*), pero rápidamente se expandió a otros dominios.

Perspectiva general

Matlab es un lenguaje de programación de alto nivel, para el desarrollo de algoritmos, visualización de datos, análisis de datos y cálculo numérico. Usando Matlab, se pueden resolver problemas técnicos de computación más rápido que con los lenguajes tradicionales de programación tales como *C*, *C++* y *Fortran*. Está disponible para la mayoría de sistemas operativos más utilizados: UNIX/Linux, Apple Macintosh y Microsoft Windows.

Matlab está diseñado para ser utilizado en un amplio rango de aplicaciones, como procesado de imágenes y de señales, comunicaciones, ingeniería de control, análisis y modelado financiero, biología computacional, etc. Dispone de *toolboxes* (cajas de herramientas), que son colecciones de funciones MATLAB de propósito específico para una tarea determinada, y extienden el entorno MATLAB para posibilitar la resolución de ciertos problemas en áreas específicas.

Características principales:

- Lenguaje de programación de alto nivel para computación técnica.
- Entorno integrado de desarrollo.
- Herramientes interactivas para diseño y solución de problemas.
- Funciones matemáticas para álgebra lineal, estadística, análisis de Fourier, filtros, optimización e integración numérica.

- Gráficos 2D y 3D para visualización de datos.
- Herramientas para la construcción de interfaces gráficas de usuario.
- Posibilidad de integrar algoritmos basados en matlab con aplicaciones y lenguajes externos, tales como *C, C++, Fortran, Java, COM y Microsoft Excel*.

Las principales razones por las que se ha utilizado MATLAB en este proyecto son porque permite la manipulación de matrices de modo realmente sencillo, tiene incorporadas funciones de gran utilidad para el análisis de componentes principales (PCA) así como para realizar histogramas, ofrece funciones para el cálculo de vectores y valores propios y finalmente, es posible realizar gráficos de funciones o de datos muy rápidamente tanto en dos como en tres dimensiones. Ésta última posibilidad ha sido de gran utilidad en el desarrollo del algoritmo de cálculo de componentes principales, ya que continuamente fue necesario visualizar resultados gráficos para comprender realmente el funcionamiento del algoritmo y el significado de los cálculos implementados. En definitiva, gracias a Matlab se pudieron realizar prototipos rápidos, permitiendo tiempos de desarrollo más cortos.

5.9. Autocad

Autocad [37] es un paquete software de diseño asistido por computador (CAD, *Computer Aided Design*) para el diseño y creación de proyectos en 2D y 3D. Está desarrollado por la compañía *Autodesk*. Está disponible exclusivamente para sistemas operativos *Microsoft Windows*. Existieron versiones para UNIX y Macintosh, pero se encontraron con una baja aceptación en el mercado y fueron desechadas.

Inicialmente fue desarrollado para el mercado de ingenieros mecánicos, pero posteriormente se extendió y ahora es ampliamente usado por arquitectos y otros profesionales del diseño. En concreto se usa en los siguientes entornos: industria, cartografía, electrónica, arquitectura y mecánica.

La primera versión (1.0) se lanzó en diciembre de 1982. La versión actual es AutoCad 2006 (la actualización número 20), que fue lanzada en marzo de 2005.

Su uso está tan extendido que sus formatos de fichero, DWG (***d**ra**w**in**g**)* en binario y DXF *(Drawing eXchange Format)* en ASCII, se han convertido en el estándar de facto para los paquetes de desarrollo CAD.

A lo largo del desarrollo de proyecto se ha utilizado AutoCad para realizar el diseño de la pieza que sirve para sujetar firmemente al láser a la plataforma que constituye el robot móvil, y que ha sido encargada a los talleres *"Rectificados del Oeste"*.

5.10. Gnuplot

Gnuplot [38] es un programa de trazado de gráficos interactivo basado en el uso de una *shell* de comandos. Puede usarse para hacer gráficos de funciones y de conjuntos de datos tanto en dos como en tres dimensiones. Está diseñado principalmente para visualizar gráficamente datos científicos, pudiendo producir la salida en la pantalla pero también como un fichero gráfico en multitud de formatos, entre ellos *postscript* encapsulado (eps, *encapsulated postscript*) que se integra a la perfección con LaTeX.

Está disponible para la mayoría de plataformas, entre ellas Unix, Linux, VMS, OS/2, MS-DOS, Amiga, Ms-Windows, OS-9/68k, Atari ST, BeOS y Macintosh. La versión oficial actual es la 4.0, lanzada en abril de 2004.

Gnuplot tiene *copyright*, pero se puede distribuir libremente bajo una licencia de código abierto que permite la copia y modificación del código fuente. El programa no tiene ningún tipo de conexión con el proyecto GNU y no usa la licencia *copyleft* GPL.

Sus principales características son:

- Gráficos bi y tridimensionales de funciones matemáticas y ficheros de datos.
- Cálculos algebraicos en aritmética entera, flotante y compleja.

- Posibilidad de crear funciones definidas por el usuario.
- Soporte de un gran número de sistemas operativos, formatos gráficos y dispositivos de salida.
- Amplia ayuda *on-line*.
- Formato estilo TEX de etiquetas, títulos, ejes y puntos.
- Edición interactiva de la línea de comandos y el historial.

5.11. LATEX

TEX

TEX es una aplicación creada por *Donald E.Knuth*. Su propósito era servir para la composición de textos y fórmulas. *Knuth* empezó a escribirlo en 1977 porque se sentía molesto con la decreciente calidad de la tipografía en los volúmenes I-III de su obra "El arte de programar ordenadores". En una manifestación del típico impulso *hacker* de resolver los problemas por sí mismo y de una vez por todas, empezó a diseñar su propio lenguaje de tipografía. Pensó que podía acabarlo en su año sabático, 1978; sin embargo no fue terminado hasta 1982. La fama de TEX reside en su gran estabilidad, además de que es posible ejecutarlo en la mayoría de sistemas operativos, y porque prácticamente no contienen errores.

LATEX

LATEX (pronunciado látej, en referencia al sonido alemán de ch como en *Ich*) es un conjunto de macros de TEX, escritas por *Leslie Lamport* (***La****mport* ***TeX***) en 1984, con la intención de facilitar el uso del lenguaje creado por *Donald Knuth*, (TEX), al cual no modifica sino complementa.

LATEX se basa en la idea de que los autores deberían ser capaces de concentrarse en ordenar su texto, basándose en la estructura de su documento, en lugar de gastar su tiempo en detalles acerca de cómo darle un formato

de aspecto adecuado. Fomenta la separación del formato respecto al contenido, a la vez que permite realizar ajustes tipográficos donde sea necesario. Mantener los detalles del formato del texto en un fichero separado del texto, a menudo es considerado como algo superior a los procesadores de textos y otros sistemas de publicación, los cuales permiten realizar pequeños cambios en el formato visual pero tienden a entrelazar tanto el contenido y la forma que a veces es difícil mantener la consistencia del formato del texto. LaTeX también proporciona gran flexibilidad en el formato mientras se mantiene la estructura del texto, algo que sistemas puramente estructurales, como SGML y XML no permiten realizar directamente.

En el entorno LaTeX, éste tiene el papel de escritor de libros y utiliza TeX como herramienta tipográfica. No obstante, LaTeX es solamente un programa y por lo tanto necesita asistencia humana. El autor necesita proporcionar información adicional para describir la estructura lógica de su trabajo, y esto se introduce en el texto a base de "comandos LaTeX".

Este es un enfoque muy diferente al enfoque WYSIWYG que se utilizan en la mayoría de procesadores de textos actuales, como *Microsoft Word* o *Corel WordPerfect*. Con estas aplicaciones se puede especificar el formato visual del documento a la par que se escribe, pudiendo ver la apariencia final del documento en pantalla.

Cuando se usa LaTeX, no es posible ver el aspecto final del documento mientras se escribe el texto, pero es posible previsualizarlo en pantalla y se pueden hacer las correcciones necesarias antes de imprimirlo. Existe un entorno integrado de desarrollo, llamado TeXnicCenter [41], de codigo fuente abierto, que permite editar fácilmente documentos LaTeX en sistemas Microsoft Windows.

Es muy utilizado para la composición de tesis y libros técnicos dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea; de hecho se ha convertido en el estándar de facto de publicaciones científicas. LaTeX es software libre bajo licencia LPPL.

LaTeX es menos popular en ambientes externos a las comunidades científicas por varias razones, siendo la principal que es considerado difícil de aprender para gente que no tiene experiencia en lenguajes de marcas de texto.

5.12. Elicitación de requisitos

Una parte fundamental de cualquier proyecto, bien sea estructurado u orientado a objetos es la denominada elicitación de requisitos.

Para la realización de esta fase del ciclo de vida se ha optado por utilizar la Metodología para la Elicitación de Requisitos de Sistemas Software versión 2.1 [32], que define las siguientes tareas:

- Tarea 1: Obtener información sobre el dominio del problema.
- Tarea 2: Preparar y realizar reuniones.
- Tarea 3: Identificar los objetivos del sistema
- Tarea 4: Identificar los requisitos de almacenamiento de información
- Tarea 5: Identificar los requisitos funcionales.
- Tarea 6: Identificar los requisitos no funcionales.

Los autores de la metodología proponen un orden 1 - 6 en la realización de estas tareas, aunque también indican la posibilidad de que las tareas 4, 5 y 6 se pueden realizar simultáneamente o en cualquier orden. Para cada tarea, los autores proponen distintas técnicas. Se destacan las siguientes:

- Casos de uso
- Plantillas

5.12.1. Utilización de casos de uso

Casos de uso

Técnica para la especificación de requisitos funcionales. Es la descripción de una secuencia de interacciones entre el sistema y uno o más actores en la que se considera al sistema como una caja negra. Los actores son personas u otros sistemas que interactúan con el sistema cuyos requisitos se están describiendo.

Diagramas de casos de uso

Los casos de uso tienen una representación gráfica en los denominados diagramas de casos de uso. En estos diagramas, los actores se presentan en forma de pequeños personajes denominados *stick man* que participan en los casos de uso. Los casos de uso se representan por elipses contenidas dentro de un rectángulo que representa al sistema. La participación de los actores en los casos de uso se indica por una flecha entre el actor y el caso de uso que apunta en la dirección en la que fluye la información.

5.12.2. Plantillas

La notación utilizada en las plantillas es la siguiente:

<texto> ⇒ comentario aclaratorio al apartado de la plantilla al que pertenece.
[texto] ⇒ este texto debe ser substituido por el analista.
{texto, texto2, ...} ⇒ selección entre las diferentes opciones.

El resto del texto debe aparecer tal cual en el resto de los requisitos. Para la descripción correcta de los casos de uso, requisitos funcionales, requisitos de almacenamiento y otros requisitos del sistema, se proponen un conjunto plantillas:

- Plantilla general para los requisitos de sistema.

- Plantilla para los requisitos de información.
- Plantilla para los requisitos funcionales.
- Plantilla para los requisitos no funcionales.
- Plantilla para otros requisitos.

En las tablas 5.3, 5.4, 5.5 y 5.6 se muestra el formato con el que se deben realizar cada una de las plantillas mencionadas.

<**identificador**>	<**nombre descriptivo**>
Descripción	<Descripción del requisito. Depende del tipo de requisito>
Importancia	{vital, importante, quedaría bien}
Urgencia	{inmediatamente, hay presión, puede esperar}
Comentarios	<Comentarios adicionales sobre el requisito en formato libre>

Tabla 5.3: *Plantilla general para requisitos de sistema.*

Una breve explicación de los campos de cada plantilla sería la siguiente:

- **Identificador:** código único que identifica al requisito y que podrá utilizarse como referencia.
- **Nombre descriptivo:** nombre asociado al requisito.
- **Descripción:** variará dependiendo del tipo de requisito.
- **Importancia:** importancia del requisito para el cliente.
- **Urgencia:** urgencia de implantación de la funcionalidad descrita por el requisito para el cliente.
- **Comentario:** indicaciones no especificadas en apartados anteriores.
- **Datos concretos:** datos específicos de la estructura a almacenar.
- **Intervalo temporal:** representa la validez temporal de la información para el sistema.

- **Secuencia normal:** sucesión de interacciones que se realizan en el caso de uso entre el actor y el sistema.
- **Excepciones:** representa comportamientos anómalos o no previstos en alguno de los pasos de la secuencia normal.

<**identificador**>	<**nombre descriptivo**>
Descripción	<Descripción del requisito. Depende del tipo de requisito>
Datos concretos	<Datos relevantes sobre el concepto>
Intervalo temporal	{pasado y presente, sólo presente}
Importancia	{vital, importante, quedaría bien}
Urgencia	{inmediatamente, hay presión, puede esperar}
Comentarios	<Comentarios adicionales sobre el requisito en formato libre>

Tabla 5.4: *Plantilla para requisitos de almacenamiento de información.*

<**identificador**>	<**nombre descriptivo**>
Descripción	El sistema deberá [descripción en formato libre]
Importancia	{vital, importante, quedaría bien}
Urgencia	{inmediatamente, hay presión, puede esperar}
Comentarios	<Comentarios adicionales sobre el requisito en formato libre>

Tabla 5.5: *Plantilla para requisitos no funcionales.*

<identificador>	**<nombre descriptivo>**		
Descripción	El sistema deberá permitir a [lista de actores] en [instante en el que se puede realizar el caso de uso] [funcionalidad que define el caso de uso] según se describe en el siguiente caso de uso:		
Secuencia Normal	**Paso**	**Acción**	
	1	{Acción a realizar} realizar el caso de uso [caso de uso]	
	2	<Situación que produce una alternativa>	
		2a	Si [situación que produce esta alternativa] el sistema deberá {<acción a realizar>, realizar el caso de uso [caso de uso]}
		2b	...
		...	...
	n	...	
Excepciones	**Paso**	**Acción**	
	p	En el caso de que [situación que produce esta alternativa] el sistema deberá {<acción a realizar>, realizar el caso de uso [caso de uso]}	
		...	...
	n	...	
Rendimiento	El sistema deberá realizar la/s acción/es descrita/s en {los pasos [primer paso] a [último paso], el paso [número de paso]} en un máximo de [cota de tiempo]		
Frecuencia	Este caso de uso se espera que se lleve a cabo una media de [número de veces] por [unidad temporal]		
Importancia	{vital, importante, quedaría bien}		
Urgencia	{inmediatamente, hay presión, puede esperar}		
Comentarios	<Otras consideraciones en formato libre>		

Tabla 5.6: *Plantilla para requisitos funcionales.*

6

Aspectos Relevantes del Desarrollo

6.1. Estructura general del sistema

Este proyecto consta de dos partes muy distintas, pero que a su vez están muy relacionadas: por un lado, la construcción de un robot móvil, y por otro, el desarrollo de un sistema software que interactúe con el robot y además ofrezca una solución práctica al problema del SLAM. Los aspectos relacionados con el desarrollo del sistema software se tratan en los anexos 2, 3, y 4, mientras que la construcción del robot móvil se describe en un anexo especial denominado "Anexo Hardware".

En este apartado se describe la estructura de comunicación desarrollada para relacionar el sistema software con el robot móvil a través del servidor *Player*, realizando una modificación de éste.

En cuanto al robot móvil, un requisito que se decide imponer a la hora su

construcción es que éste sea totalmente inalámbrico, de modo que disponga de total libertad de movimientos. Para lograrlo hay que eliminar los dos tipos de cable que utiliza el escáner láser, uno de alimentación eléctrica y otro de datos para transferir las medidas de distancia que obtiene del entorno a un PC de sobremesa[1]. Esta transferencia de datos se lleva a cabo por medio del servidor de dispositivos *Player*.

Eliminar el cable de alimentación es muy sencillo, basta con utilizar una batería y colocarla a bordo del robot. Sin embargo conseguir eliminar el cable de datos es más complejo, como se explica a continuación. El servidor *Player* se puede colocar en un ordenador que tenga conexión a la red, y en otro ordenador diferente, a través de *sockets* TCP se puede ejecutar un cliente que interactúe con el servidor *Player*. Este modo de trabajar de *Player* se ha aprovechado para colocar un mini-PC portátil en el robot conectado a la red mediante un punto de acceso inalámbrico. Así, en el mini-PC portátil se ejecuta el servidor *Player*, mientras que en un PC de sobremesa, se puede lanzar un cliente para obtener los datos del láser. De este modo queda zanjado el problema de obtención de datos del láser sin conectarlo mediante cables a un PC.

Por otra parte, para poder mover el robot, se necesita controlar desde un PC de sobremesa los motores instalados en el robot. Éstos tienen que se manejarse a través de un bus I2C, que no puede conectarse a un PC, ya que no dispone de este puerto. Sin embargo, es posible actuar sobre los motores a través de un módulo controlador denominado DK-40. Aquí es donde entra en juego la modificación del servidor *Player* que propone A. Morales, descrita en [19] y realizada en el Departamento de Informática y Automática de la Universidad de Salamanca. La modificación está descrita con detalle en el apartado 5.1.6 de esta memoria.

Si los motores pudieran controlarse a través del puerto serie, estos se hubieran conectado al PC instalado en el portátil, al igual que se ha hecho

[1]En este caso se entiende que un PC de sobremesa es un PC que no forma parte del robot móvil, ya sea portátil o no.

con el láser, y acto seguido se podrían manejar desde un PC de sobremesa. Sin embargo, al tener que ser controlados por el DK-40, se necesita una aplicación servidora en el DK-40 que envíe órdenes a los actuadores. Ésta sería la capa más baja propuesta por Morales. El servidor instalado en el DK-40 a la vez que maneja los actuadores se comunica con el servidor *Player* **modificado**, y lo hace a través de TCP/IP en lugar de por el puerto serie. Finalmente un cliente de *Player* en el ordenador de sobremesa se comunica con *Player* de manera normal, a través de TCP/IP, sin conocer lo que ocurre en las capas inferiores.

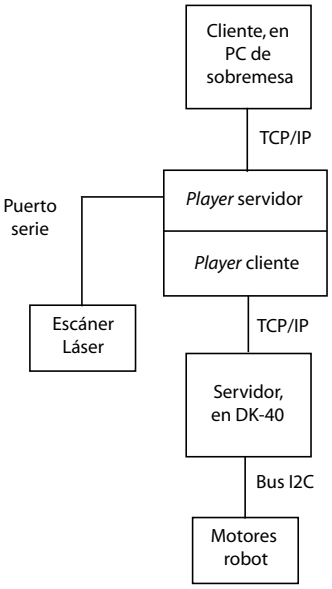

Figura 6.1: *Arquitectura actual del servidor Player*

En la fig. 6.1 se presenta la arquitectura utilizada, y a continuación se describe un ejemplo de su funcionamiento: un cliente en un PC de sobremesa pide al servidor *Player* modificado que actúe sobre los motores. El servidor *Player* modificado, en lugar de actuar directamente sobre los motores (como haría el servidor *Player* sin modificar), transfiere la orden a un servidor colocado en el DK-40, el cual ya puede actuar sobre los motores. En el caso de leer datos del láser, se leen los datos sin necesitar el servidor DK-40, es decir, no se hace uso de la modificación de *Player*.

Al optar por la solución propuesta, se han presentado ciertos problemas, debido sobre todo a las constantes modificaciones que ha sufrido el proyecto *Player/Stage*, ya que continuamente los desarrolladores están intentando mejorarlo. Morales realizó una modificación del servidor *Player* cuando se distribuía la versión 1.3, e implementó un servidor en el módulo DK-40 que permitía manejar motores, infrarrojos, relés y sónar. Además construyó unos *drivers genéricos* para estos dispositivos, y que constituyen la parte denominada como *Player cliente* en la fig. 6.1. Estos *drivers genéricos* se diferencian de un *driver* normal de *Player* en que se comunican vía TCP con el servidor instalado en el DK-40 en lugar de comunicarse vía puerto serie directamente con el dispositivo.

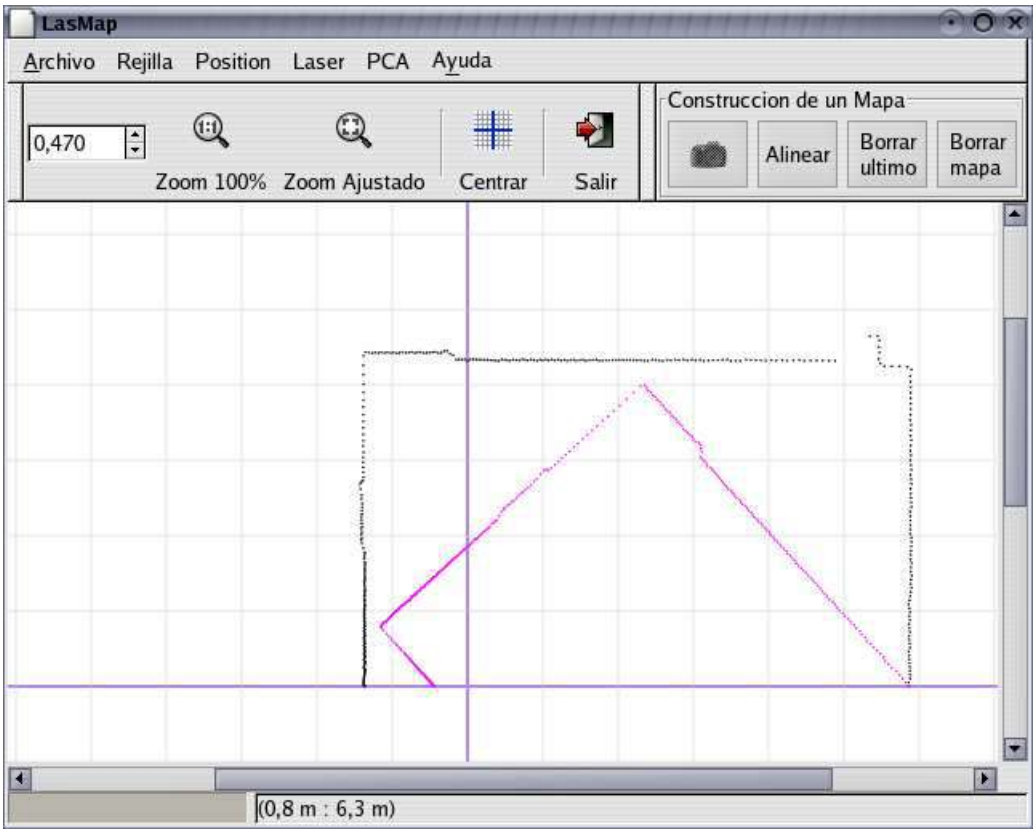

Figura 6.2: *Aplicación alineando dos photoscans.*

De todo esto, el primer problema que surge es que el *driver genérico* para

los motores del servidor *Player* modificado, implementado por Morales, no se puede usar con el bus I2C, así que hay que reescribirlo. Una vez reescrito el *driver genérico* de los motores para la versión 1.3 de *Player*, aparece un segundo problema. Los*drivers* (incluidos en la distribución estándar de *Player)* del láser no funcionan correctamente, a no ser con la versión 1.5 de *Player* o superior. Con esta situación, finalmente se decide portar la modificación de *Player* realizada por Morales hacia la versión 1.5 de Player, actualizando para ello los *drivers genéricos* de los motores.

En este punto, se dispone de un robot móvil totalmente inalámbrico y que porta un escáner láser. Desde un PC de sobremesa es posible leer datos del escáner láser y trasladar el robot al punto que se desee, mediante el sistema software desarrollado. En la fig. 6.2 se observa el aspecto general de la aplicación en el momento de la alineación de dos *photoscans*.

6.2. Problemas relevantes de implementación

En esta sección se comentan los problemas más importantes que surgieron durante la implementacíón del software de la aplicación.

6.2.1. Análisis de componentes principales

Como se explica en el apartado 4.2 de esta memoria, en [11] se describe como utilizar el análisis de componentes principales aplicado a la localización absoluta en entornos interiores utilizando un escáner láser de medición de distancias. Al ser una técnica basada en el análisis estadístico el algoritmo es muy robusto ante los fallos. Un ejemplo es que aunque el escáner detecte personas en el entorno en el que se va a intentar la autolocalización, esto no influye en gran medida en el resultado del análisis. Desde un principio se puso un gran empeño por parte del autor de este proyecto en intentar llevar a la práctica la técnica descrita en el artículo, puesto que las ideas que se proponían en él resultaban muy interesantes.

Sin embargo el artículo resulta difícil de comprender, puesto que es eminentemente teórico y no se presenta ejemplo alguno de como llevar a la práctica el análisis de componentes principales, técnica que resulta ciertamente compleja de comprender (el análisis PCA se puede estudiar en el apartado 3.2 de esta memoria). Para comprender mejor la técnica hubo que estudiar detenidamente otros artículos en los que estaba basado o que tenían cierta relación, como por ejemplo [17] y [18].

Por lo tanto en primer lugar hubo que estudiar el análisis PCA, lo cual requirió una cantidad importante de tiempo, y en segundo lugar hubo que realizar multitud de pruebas para aplicar de forma satisfactoria el análisis PCA a los datos obtenidos por el escáner láser.

El escáner láser al realizar una lectura del entorno proporciona 361 distancias en formato polar; también puede dar 181, pero siempre se ha trabajado con 361 porque ofrece una mayor precisión en la reconstrucción del entorno. Para comenzar, el primer paso debe ser obtener una cuadrícula de scans simulados, como se describe en el apartado 4.2.2 del proyecto. El primer problema surge al intentar calcular la matriz de covarianza para un conjunto de scans simulados. Parece lógico utilizar una lectura completa del escáner láser como una dimensión independiente y colocarlo por tanto como una columna de la matriz de covarianza. A lo largo del artículo este dato permanece bastante ambiguo, por lo cual no hay más remedio que aclararlo de manera práctica. Al final se comprobó que cada scan completo se debía colocar en una fila, con lo cual solo existían 361 dimensiones correspondientes a cada haz de luz que emite el láser durante la realización de un barrido.

Una vez solventado este problema, y realizado el análisis de componentes principales, se presentan nuevos inconvenientes, no aclarados en el artículo. ¿Cuantos vectores propios seleccionar de entre los 361 que se han calculado en el análisis PCA? La utilidad subyacente en el análisis PCA es reducir la dimensionalidad de los datos sin perder información del entorno, y por tanto no sería una buena opción elegir todos los vectores propios, ya que no se reduciría la dimensión. Si se escogen tan sólo 3 o 4 vectores propios tal y como

se indica en el artículo, la autolocalización en la práctica no resulta de gran calidad, puesto que se pierde demasiada información sobre el entorno. Tras multitud de pruebas prácticas se observó que si se tomaban los suficientes vectores propios para que la varianza contenida en los vectores propios usados superara el 99 %, la autolocalización era suficientemente buena, y el número de vectores utilizados rondaba los 15.

Finalmente, apareció una última dificultad. La pretensión de utilizar una cuadrícula con *scans* simulados es poder comparar el *scan* actual con todos los scans simulados para observar cual es el que más se parece. Como medida de comparación se decidió utilizar la distancia euclídea debido a su simplicidad y buenos resultado que ofrecía (ésto se obtuvo de [12], ya que en [11] no aparece). El problema fundamental de la comparación de *scans* es cómo proyectar los *scans* simulados y el *scan* actual en el espacio vectorial calculado, ya que en ningún punto del artículo se comenta cómo hacerlo, al darlo por supuesto. Tras varias pruebas, finalmente se resolvió este problema utilizando la información obtenida en [13].

6.2.2. Histogramas

Para la puesta en práctica de la técnica comentada en el apartado 4.1 uno de los primeros problemas encontrados fue la necesidad de implementar histogramas para la acumulación de ángulos, distancias y su respectiva correlación cruzada. Para ello se investigó el código fuente de una biblioteca científica gratuita [39] lo cual sirvió de gran ayuda y se modificó ligeramente para adaptarse a las necesidades del proyecto.

Como es bien sabido por todos, un histograma consiste en una serie de barras que dividen un intervalo en un subconjunto de intervalos que pueden ser o no uniforme. El valor de cada barra representa el número de ocurrencias de un determinado suceso que recaen en el subintervalo correspondiente.

En los histogramas normales, el valor de cada barra no está centrado en cada número, como se puede apreciar en la fig. 6.3. Ésto, para la acumulación de ángulos comentada en el apartado 4.1 es de escasa utilidad, porque lo que

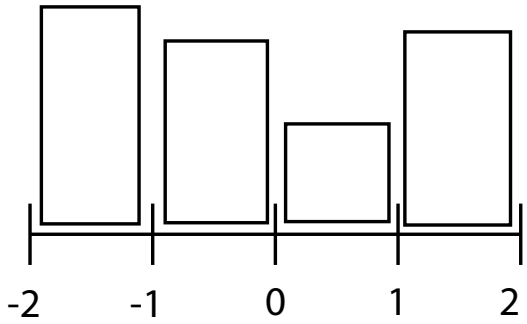

Figura 6.3: *Histograma normal*

se pretende es saber cuantos ángulos han tomado el valor cero, el valor 45, etc. Se puede pensar que centrar las barras en cada número solucionaría el problema, pero solamente se soluciona en parte, porque existe otro pequeño problema. Los ángulos calculados entre cada par de puntos se expresan como un número del intervalo que va de -90º a 90º, ya que la pendiente de una recta de 91º sería exactamente la misma que la de una recta de 89º. Por lo tanto con el intervalo $[-90, 90]$ se pueden expresar todas las pendientes. Una recta con una inclinación de 90º y otra con una inclinación de -90º son equivalentes, por lo tanto en el histograma no deberían considerarse dos barras diferentes, una para 90º grados y otra para -90º, y para ello se ha optado por programar los histogramas de manera que sean circulares, como se observa en la figura 6.4. En este caso el histograma representa el intervalo $[-2, 2]$, las barras están centradas en cada número y además las barras en gris en realidad son la misma barra, pero divida en dos, para otorgar al histograma la cualidad de ser circular.

6.2.3. Utilización de BLAS y LAPACK

Una de las dificultades de utilizar las bibliotecas de álgebra lineal BLAS y LAPACK, es que están implementadas en Fortran. Manejar estas bibliotecas desde C requiere manejar ciertos aspectos de interoperabilidad. Esta ha sido la principal dificultad de su uso, puesto que en la distribución oficial no vienen

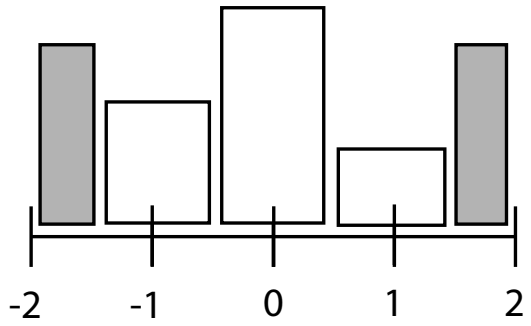

Figura 6.4: *Histograma circular y con las barras centradas*

ejemplos de utilización y no ha sido fácil encontrar ejemplos en los que se demuestre el uso de estas bibliotecas desde el lenguaje C.

En algunos casos es posible utilizar interfaces-envoltorio que simplifican la utilización de la biblioteca desde C, como es el caso de CBLAS, mientras que en otros casos como el de LAPACK, no es posible. Si bien es cierto que existe una versión de LAPACK denominada CLAPACK, disponible en [24], ésta no es equivalente a CBLAS. Con CBLAS es posible implementar código sin tener que manejar los detalles de interoperabilidad entre C y Fortran. Sin embargo con CLAPACK sí es necesario manejar las estructuras de datos y las llamadas a funciones según las convenciones de Fortran. CLAPACK está construido utilizando una utilidad de conversión de Fortran a C llamada `f2c`, y su objetivo es permitir utilizar LAPACK a alguien que no tenga acceso a un compilador de Fortran. CBLAS forma parte de un proyecto denominado *BLAST-FORUM* y su objetivo es proporcionar una interfaz C para la biblioteca BLAS, de modo que se pueda manejar directamente desde C.

A continuación se presentan los diferentes aspectos que es necesario conocer para manejar directamente desde C una biblioteca escrita en Fortran, como es LAPACK, sin cometer errores debidos a las diferencias entre ambos lenguajes.

Tipos de Datos

Al declarar variables en C que se van a pasar a funciones Fortran es necesario que se usen tipos de datos equivalentes a los de Fortran. En la tabla 6.1 se presentan algunas equivalencias habituales.

Fortran	C
`INTEGER`	`int`
`REAL*4`	`float`
`REAL*8`	`double`
`CHARACTER`	`char`
`DOUBLE PRECISION`	`double`

Tabla 6.1: *Equivalencias de tipos de datos Fortran-C.*

Nombres de funciones y paso de argumentos

Los compiladores de Fortran a menudo modifican los nombres de las funciones una vez que han compilado. Por eso, al llamar a una función Fortran desde C, debe ser llamada por su nombre modificado. La modificación consiste en convertir el nombre de la función a letras minúsculas y añadir un guión bajo.

Otro aspecto importante al llamar a una función Fortran desde C es que se deben pasar *punteros* para todos los parámetros que sean escalares (esto es, no *arrays*). La razón es que Fortran pasa los argumentos escalares *por referencia*, mientras que C los pasa *por valor*. Los *arrays* en C se pasan por referencia, por lo cual quedan exentos de esta regla.

Por ejemplo, una función definida en Fortran como:

```
DOUBLE PRECISION FUNCTION DOT_PRODUCT ( N, X, Y)
DOUBLE PRECISION X(*), Y(*)
INTEGER N
```

tendría como prototipo ANSI C:

```
double dot_product_ ( int* n, double* x, double* y );
```

Y para usarla sería necesario hacer lo siguiente:

```
double x[10], y[10], ans;
int ten = 10;
/* Llenamos x e y con datos */
ans = dot_product_(&ten, x, y);
```

El entero `10` necesita ser almacenado en una variable temporal de modo que al llamar a la función se le pueda pasar como puntero.

Indexación de *arrays* bidimensionales

Puesto que C almacena los arrays *por filas*, esto significa que los elementos al ser guardados en memoria quedarán así:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

Fortran, por otra parte, almacena los arrays *por columnas*, lo que significa que los elementos serán guardados del siguiente modo:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 4 & 7 & 2 & 5 & 8 & 3 & 6 & 9 \end{bmatrix}$$

Teniendo en cuenta estos tres aspectos mencionados: conversión de tipos de datos, nombres de funciones y paso de argumentos, y la indexación de *arrays* bidimensionales, es posible utilizar directamente la biblioteca LAPACK desde un programa en C.

6.2.4. Transformar un mapa de entorno en imagen

A pesar de que el *canvas* de GNOME ofrece una serie de características que facilitan la tarea del manejo de gráficos al programador, como por ejem-

plo un doble *buffer* para la visualización, gráficos suavizados (*antialiased*), etc., presenta un gran inconveniente a la hora de guardar los gráficos.

Este inconveniente es que no existe la posibilidad de almacenar el *canvas* y sus contenidos en memoria secundaria. Esto se debe a la manera en que está programado. Cada elemento representado lo trata como un objeto independiente. Por ejemplo, si se quiere dibujar una línea, se añade un objeto línea al *canvas*. Lo mismo ocurre si añadimos un círculo y un cuadrado, los dos serían objetos colocados en el *canvas*. Esto facilita algunas tareas, como mover objetos independientemente y hacer que respondan a los eventos del ratón, entre otras cosas. Sin embargo, al utilizar este modo de programación, el *canvas* no es tratado como un **mapa de bits**, que es básicamente un buffer tridimensional en el que cada punto de la imagen tiene un color RGB. De este modo, cada vez que el *canvas* se necesita redibujar o hacer zoom, no se repinta el buffer con el mapa de bits, sino que se va repintando objeto por objeto (solamente los que se necesiten repintar).

Un requisito de la aplicación consiste en almacenar el mapa global del entorno construido por el usuario en una imagen PNM. Como no es posible hacer esto directamente con el *canvas* de GNOME, se ha necesitado suplir esta capacidad de otro modo. Después de estudiar diferentes posibilidades de conversiones entre formatos gráficos se optó por escribir cada punto que formaba el mapa global como un punto en un formato de imagen denominado *Postscript Encapsulado* (EPS), que es un tipo de gráfico vectorial.

La ventaja de los gráficos vectoriales es que los elementos dibujados se definen como entidades matemáticas, por ejemplo un punto, una recta, un círculo, etc., y además no pierden calidad al hacer zoom en la imagen. Esto hace sencillo en algunos casos (como el formato EPS) definir los elementos que van a constituir la imagen. Existen multitud de formatos de gráficos vectoriales, como por ejemplo SVG (*Scalable Vector Graphics)*, CMG (*Computer Graphics Metafile*), WMF (*Windows MetaFile*), AI (*Adobe Illustrator*), FH (*FreeHand*), etc., pero todos ellos son muy complejos de utilizar.

Sin embargo, el formato EPS, a pesar de que puede ser muy complejo

en algunos aspectos, es un tipo de gráfico vectorial en el que resulta muy sencillo definir círculos, líneas y rectángulos. Además el formato EPS presenta otra notable ventaja: mediante el interprete *ghostcript* [40] es muy sencillo convertir una imagen EPS a cualquier tipo de formato gráfico de mapa de bits (jpeg, png, gif, pnm, ...). Por estas dos razones se tomó la decisión de utilizar este formato.

Por lo tanto la solución para convertir un mapa constituido por puntos en un mapa de bits es muy sencilla. Se toma cada punto que forma el mapa global del entorno y se escribe como un círculo en un archivo EPS. A continuación utilizando las bibliotecas proporcionadas por *ghostscript* para interactuar con el lenguaje C, se transforma el archivo en formato EPS en un archivo en formato PNM. De este modo ya esta listo para ser usado como un mapa de entorno en el simulador *Stage*.

6.2.5. *Locale* y los problemas de la internacionalización

A lo largo del proyecto se ha trabajado con Fedora Core 2 como sistema operativo. Como requisito del proyecto se estableció controlar el escáner láser y los motores del robot a través del servidor *Player/Stage*. El servidor *Player* funcionaba correctamente, pero al intentar utilizar el simulador *Stage*, resultaba imposible hacerlo funcionar. Tras una larga investigación se llegó a la conclusión de que *Stage* cometía un error al interpretar los puntos que servían de separador decimal en los números en formato de punto flotante que aparecían en el fichero de configuración que procesaba *Stage* al iniciarse.

Otro problema similar al anterior se presentó al utilizar la aplicación *ghostscript* para transformar una imagen en formato EPS a una imagen PNM. Al escribir un fichero en formato EPS la aplicación desarrollada utilizaba una coma como separador decimal, y sin embargo el intérprete *ghostcript* estaba esperando un punto como separador decimal.

Estos dos problemas están relacionados con la internacionalización, la localización y el entorno *locale* en los sistemas linux, como se explicará a continuación.

Se entiende por "internacionalización" (abreviado "i18n") la operación por medio de la cual se modifica un programa, o conjunto de programas en un paquete, para que pueda adecuarse a múltiples idiomas y convenciones culturales.

Por "localización" ("l10n"), nos referimos a la operación por la que, sobre un conjunto de programas que ya han sido internacionalizados, se le proporciona al programa toda la información necesaria para que pueda manejar su entrada y su salida de un modo que sea correcto respecto a determinados hábitos lingüísticos y culturales (por ejemplo el signo de la moneda de un país, el orden en que se expresan mes, día y año en una fecha).

Por otra parte, *Locale* es un concepto básico introducido en ISO C (ISO/IEC 9899:1990), el estándar internacional definido para el lenguaje de programación C. En el modelo *Locale*, el comportamiento de algunas funciónes del lenguaje C dependen del entorno *locale* definido en el ordenador. El entorno *locale* está dividido en seis categorías y cada una de ellas puede configurarse independientemente usando la función `setlocale()`. Las categorías son `LC_TYPE`, `LC_COLLATE`, `LC_MESSAGES`, `LC_MONETARY`, `LC_NUMERIC` y `LC_TIME`. Hay una categoría que engloba a todas: `LC_ALL`.De todas estas categorías la única relevante para el problema mencionado resultó ser `LC_NUMERIC`, que es la que se ocupa de mostrar el formato de los números. La categoría `LC_ALL` se puede controlar por medio de la variable de entorno `$LANG`.

La sintaxis para especificar un *locale* determinado es la siguiente: idioma_[territorio].[*codeset*][@modificador]. Por ejemplo es_ES.UTF-8@euro indicaría idioma español, territorio España, codificación UTF-8 y modificador zona euro.

El problema de porqué *Stage* y *ghostcript* no podían comprender una coma como separador de los números en formato de punto flotante, reside en que ambos programas están desarrollados en un entorno de habla inglesa. El uso del punto y la coma como separador en los números es totalmente opuesto en el Español que en el Inglés. Los ingleses escriben 1,000.00 para indicar "mil" con dos decimales. Sin embargo los españoles escribiríamos

1.000,00 para indicar la misma cifra. No obstante existen algunos tipos de español, como el de Méjico o Guatemala, por ejemplo, en los que el formato de los números es el inglés.

Este parece un problema trivial, pero no lo es, ya que las funciones `printf()`, `scanf()` y derivadas de estas utilizan el *locale* que este activado en el sistema para leer y escribir números. Una instalación de Fedora o Redhat en español tendrá la variable `$LANG` con el valor `es_ES.UTF-8` por defecto. Esto significa que para entender números en punto flotante las funciones `printf()` y `scanf()` utilizarán comas como separador decimal. Por el contrario, los programas *Stage* y *ghostscript* al utilizar `scanf()` o una función similar para leer los archivos no pueden comprender que haya una coma separando números.

Por lo tanto una buena solución para solventar el problema, consiste en establecer la variable de entorno `$LANG` con el valor `es_MX.UTF-8` (español de Méjico), `es_US.UTF-8` (español de Estados Unidos) o `en_US.UTF-8` (inglés de Estados Unidos), por ejemplo. De este modo se logra que todas las aplicaciones utilizadas en el sistema lean y escriban los números con un **punto** como separador decimal y no comentan errores en el procesamiento de los archivos.

6.3. Ciclo de Vida

Un factor importante en la elección del ciclo de vida ha sido la necesidad de recoger los requisitos de una manera certera, de modo que el sistema satisficiera plenamente las necesidades del cliente. Para ello se decidió ir presentado una serie de prototipos mediante los cuales el cliente pudiera observar la evolución del sistema desde sus orígenes, consiguiendo así involucrar al cliente y evitar posibles sorpresas indeseadas.

Según lo expuesto, se determinó seguir un ciclo de vida iterativo basado en prototipos. En este ciclo de vida cada iteración es como un mini-ciclo de vida, en el que se tienen todas las fases del ciclo de vida clásico, con la

diferencia de que el ciclo iterativo es progresivo. Además se tiene la ventaja de que las fases del mini-ciclo de vida se pueden solapar, no hace falta que sean secuenciales.

El software se elabora parcialmente construyendo prototipos sobre una determinada funcionalidad y a continuación se van añadiendo funciones. Estos prototipos se entregan de manera periódica. Los requisitos del usuario están mejor definidos, puesto que los redefine sobre algo que está viendo como funciona (el prototipo). El rechazo del usuario es bastante bajo al parrticipar constantemente en la realización. Otra ventaja de las entregas de prototipos es que permiten detectar errores en una iteración y corregir los errores en una próxima iteración.

Dentro de cada iteración se pueden establecer puntos de control, que sirven para evaluar las iteraciones y para fijar los objetivos. Se pueden establecer dos puntos básicos. Uno es la revisión inicial, para fijiar los objetivos y fijar criterios de evaluación de objetivos. El otro sería la revisión de evaluación, que se realiza al final de una iteración y su función es ver si se han cumplido los objetivos fijados.

7

Trabajos relacionados

Como se ha comentado anteriormente, para llevar a cabo determinadas tareas, los robots móviles autónomos deben ser capaces de determinar su posición y orientación mientras se están moviendo. Lograr una autolocalización precisa y estable es uno de los requisitos más importantes para que un robot pueda actuar adecuadamente en cualquier entorno.

En principio existen dos enfoques diferentes para estimar la posición y orientación de un robot:

- **Relativo:** dada una posición inicial arbitraria $p_0 = (x, y, \theta)$, la posición relativa del robot con respecto a p_0 es actualizada incrementalmente a medida que el robot se mueve. Para lograr esto, se necesitan comparar los datos adquiridos consecutivamente por el sensor láser. Sea R el scan de referencia y S el scan actual. Si los dos scans se han adqui-

rido en diferentes ubicaciones, se necesita calcular una transformación que haga coincidir S en R. La transformación calculada corresponde al movimiento que ha realizado el robot entre los dos scans.

- **Absoluto:** en lugar de actualizar la posición y orientación incrementalmente, se calculan utilizando un mapa previamente adquirido de un entorno

Los algoritmos de localización relativa basados en la utilización de sensores de proximidad se pueden distinguir por la manera en que calculan el cambio de posición y orientación entre dos tomas de datos de los sensores. El presente proyecto se ha centrado en métodos que utilizan un escáner láser de medición de distancias, puesto que presenta ventajas de precisión y rapidez de procesamiento de datos con respecto a una cámara de visión estéreo o un sonar.

Aunque el campo de la localización relativa utilizando escáners láser está todavía en fases de desarrollo, se pueden distinguir tres principales enfoques sobre los que agrupar las técnicas existentes que tratan sobre la comparación de scans:

- Comparación de los scans punto a punto, sin extraer características de los scans.
- Comparación de scans basada en la extracción de líneas.
- Comparación de los scans utilizando histogramas.

El primer y segundo enfoques son iterativos, y por lo tanto requieren más tiempo de procesamiento que el tercer enfoque, que ha sido utilizado en este proyecto debido a su menor tiempo de procesamiento y sencillez de cálculos. En las secciones 7.1 y 7.2 se comentarán artículos pertenecientes a los dos primeros enfoques; en el apartado 4.1 de esta memoria se encuentra información detallada acerca de la técnica basada en histogramas.

Con respecto a la localización absoluta parece que existe una menor cantidad de alternativas y concretamente el autor de este proyecto solamente ha

encontrado una técnica para realizar la autolocalización de un robot móvil basada en un modelo del entorno sin tener en cuenta el método de obtener dicho modelo. Esta técnica se conoce como localización absoluta basada en el análisis de componentes principales, y está desarrollada por *J.L. Crowley* en [11]. En el apartado 4.2 de esta memoria se aborda con detalle. Cabe destacar que el análisis PCA aplicado a *laserscans* está basado en una técnica muy conocida y utilizada en el campo de la visión artificial conocida como análisis de *eigenfaces*. En [17] se trata a fondo este tema y su lectura puede aclarar ciertos aspectos al lector acerca del análisis PCA que no son explicados con demasiada claridad en el artículo de *Crowley*.

7.1. Comparación de scans punto a punto

En [14] se proponen dos algoritmos que hacen coincidir uno a uno los puntos de cada scan. El primer algoritmo propuesto está basado en hacer coincidir puntos con direcciones tangentes en los dos scans. El segundo algoritmo se basa en una solución de cuadrados mínimos iterativa, usando correspondencias punto a punto entre cada punto de los scans. Ambos algoritmos son bastante complejos, pero son ejemplos representativos de comparación de scans punto a punto, por lo cual solo se ofrecerá una idea general de cada algoritmo, sin entrar en detalles.

Algoritmo de búsqueda de mínimos cuadrados

Este enfoque de comparación de scans consiste en definir una medida de la distancia entre los dos scans y buscar una transformación que minimice la distancia. Aunque el espacio de búsqueda es esencialmente tridimensional (rotación y traslación bidimensional), se intenta reducir el problema a una búsqueda unidimensional más una solución de mínimos cuadrados, formulando cuidadosamente la medida de la distancia.

La idea principal del algoritmo se describe a continuación. Primero, se calculan las direcciones tangentes en los scans, buscando las líneas que más

se ajustan a los conjuntos de puntos de cada scan. Después se asocian correspondencias aproximadas entre los puntos de cada scan, asumiendo un ángulo de rotación conocido ω (pero no la traslación T). Para cada par de puntos asociado, se formula una ecuación lineal acerca de la traslación desconocida T. Después, usando los pares de correspondencia establecidos, se define un modelo de mínimos cuadrados para T, que también representará una comparación de la distancia como función de ω. El último paso consiste en buscar una rotación ω que minimice la función de distancia. La traslación T se resuelve a través del método de mínimos cuadrados.

Algoritmo de correspondencia de puntos

Este otro algoritmo para alinear dos scans es un algoritmo iterativo basado en asociar una correspondencia punto a punto. La idea que subyace en el método es la siguiente. Sea S_ref el scan de referencia y S_new el nuevo scan. Para cada punto P_i en S_new, se usa una regla simple (independiente de la rotación y traslación reales) para determinar un punto correspondiente P'_i en S_ref. Después, para todos los pares de correspondencias establecidos entre los puntos, se calcula una solución de mínimos cuadrados de la rotación y traslación relativa. Esta solución se aplica para reducir el error de posición entre los dos scans. Finalmente se repite este proceso hasta que converge. El asunto central del algoritmo es la definición de una regla para determinar correspondencias entre puntos sin conocer la rotación y traslación reales.

7.2. Comparación de scans basada en la extracción de líneas

En este apartado se comentarán dos artículos que servirán de representantes de este tipo de algoritmos. El primero está detallado en [15] y presenta una técnica para construir un mapa global y autolocalizarse posteriormente en el utilizando el para ello un método denominado CLS (*Complete Line*

Segments). El segundo, [16], utiliza una técnica tomada de la visión artificial conocida como ICP (*Iterative Closest Point*).

7.2.1. *CLS: Complete Line Segments*

En primer lugar se analizan los datos del entorno obtenidos continuamente mediante un escáner láser y se construye un mapa local mediante una fase de división - unión - división - unión, como se indica a continuación. Cada vez que se supera una cierta distancia umbral entre dos puntos consecutivos, el scan se divide en dos partes. De este modo el scan queda dividido en una serie de regiones. A continuación las regiones formadas por un número pequeño de puntos son eliminadas, y las regiones restantes se vuelven a unir si la distancia que las separa es arbitrariamente pequeña. A continuación para cada region se traza una línea recta entre el primer y el último punto y se divide en dos partes por el punto cuya distancia ortogonal a la línea sea mayor. Finalmente se vuelven a unir las regiones cuya distancia sea menor que un determinado umbral. De este modo el scan que constituye el mapa local ha quedado dividido en una serie de segmentos.

Para añadir el mapa local a un mapa global, se necesita resolver un problema de correspondencia entre los segmentos del mapa local y el mapa global. Los segmentos en el mapa local que tengan un segmento correspondiente en el mapa global, se añadirán al mapa complementando al segmento correspondiente. Los segmentos del mapa local que no tengan un segmento en el mapa global se añadirán en la posición adecuada con respecto a la correspondencia establecida.

Una vez dividido el mapa local en segmentos habrá algunos que sean segmentos incompletos y representen parcialmente un objeto real, y habrá otros que sean completos y representen completamente a un objeto real. Los segmentos completos (CLS) serán estos últimos, y se caracterizan porque otros objetos no pueden obstaculizar su detección, como se puede observar en la FIGURA. Para comprobar si un segmento es CLS se comparan los puntos de sus dos extremos con los puntos de sus extremos vecinos. Si por ejemplo en

la FIGURA se cumple que $\mid OP_{70} \mid \leq \mid OP_{61} \mid$ y $\mid OP_{71} \mid \leq \mid OP_{80} \mid$ el segmento será CLS. Si no se cumple esta relación será un segmento incompleto.

Finalmente para conseguir la autolocalización se utiliza la idea de que los segmentos CLS, al ser completos pueden cambiar de posición y orientación, pero no de longitud. Por lo tanto usando la longitud de varios CLS, se buscan posibles alternativas haciendo coincidir los segmentos CLS de un mapa local con los posibles candidatos en el mapa global.

7.2.2. *ICP: Iterative Closest Point*

Cox [16] presenta un sistema de estimación de la positión que consiste en un mapa *a priori* del entorno, obtenido a mano, y representado como una colección de segmentos discretos en el plano, y un algoritmo de comparación que trata de confrontar los datos obtenidos por el escáner láser con algún segmento del mapa.

Cabe destacar que el sistema de posicionamiento utiliza datos odométricos y supone constantemente que la posición estimada no es muy distinta de la real, esto es, supone que la imagen tomada por el escáner láser estará levemente desplazada con respecto al modelo. La odometría se irá corrigiendo con los valores encontrados por el algoritmo, con lo cual el error no puede crecer ilimitadamente.

Los pasos del algoritmo son los siguientes:

1. Para cada punto en el scan, encontrar el segmento en el modelo que está más cerca del punto. Este segmento recibe el nombre de *objetivo*.

2. Encontrar el desplazamiento y rotación que minimiza la distancia total al cuadrado entre los puntos del scan y los segmentos objetivo.

3. Desplazar y rotar los puntos del scan los valores encontrados en el paso anterior.

4. Repetir los tres pasos anteriores hasta que el procedimiento converja.

El segundo paso es el más complejo computacionalmente, y en [16] se describe un método para mejorar la eficiencia de cálculos.

8

Conclusiones y líneas de trabajo futuro

8.1. Resultados del Proyecto

A continuación, se presentan las principales conclusiones de este trabajo fin de carrera donde se muestra cómo se han completado con éxito los objetivos planteados.

Se ha desarrollado un procedimiento de construcción de mapas basado en la obtención de diferentes fotografías del entorno tomadas mediante el escáner láser. A partir de una fotografía de referencia, se implementa un procedimiento de alineación de pares de *scans* que permiten la construcción un mapa global del entorno de forma incremental. Este procedimiento permite, además, realizar una tarea de localización relativa.

Para resolver el problema de autolocalización absoluta de un robot móvil en un entorno conocido del que dispone un mapa se ha implementado un pro-

cedimiento basado en el Análisis de Componentes Principales (PCA). Considerando los datos que el escáner láser esté percibiendo en un determinado instante de tiempo, el sistema ofrece una estimación de la posición en la que se encuentra en dicho entorno conocido, utilizando para ello la técnica PCA. Se ha optimizado su cálculo utilizando las bibliotecas BLAS y LACPACK y se han definido los procedimientos de almacenamiento necesarios para su procesamiento posterior.

Con objeto de validar estos procedimientos, se ha desarrollado una plataforma software que interacciona con el servidor *Player*, y que permite poner de manifiesto el cumplimiento de las características fijadas en los objetivos:

- De forma fundamental, permite poner en práctica los procedimientos de SLAM desarrollados en este proyecto.
- Manejo del láser y de los motores del robot. Se ha hecho necesaria la implementación del dispositivo que controla los motores utilizando las herramientas hardware y software desarrolladas en el Grupo de Robótica de la Universidad de Salamanca.
- Control del robot móvil. El usuario puede elegir si va a manejar el robot móvil físico o si por el contrario hará uso de un robot virtual simulado por *Stage*.
- Visualización de datos: una vez que el usuario se conecta al dispositivo láser, se inicia la comunicación en tiempo real con él y se muestra la representación gráfica de los datos recibidos sobre un área de dibujo. Asimismo, muestra el mapa global del entorno a medida que el usuario alinea los *scans* fotografiados y lo va incrementando paulatinamente.

La aplicación desarrollada se caracteriza por su facilidad de manejo y dispone de una interfaz gráfica amigable e intuitiva. En este sentido, incorpora diferentes funcionalidades que incrementan su utilidad:

- Operaciones sobre el área de dibujo. El usuario puede realizar un zoom de acercamiento o alejamiento, para observar con más o menos detalle

los elementos presentados en la imagen (*scans* fotografiados, datos en tiempo real y mapa global). Además tiene la posibilidad de mostrar o no una cuadrícula que divida en partes proporcionales el área de dibujo, de modo que en todo momento se pueda estimar las distancias y dimensiones de los datos con un simple vistazo. El tamaño de la cuadrícula es configurable por el usuario.

- Transformación de un mapa en imagen PNM. Los datos leídos por escáner láser consisten en una serie de distancias únicamente, y por lo tanto el mapa global construido está constituido por un conjunto de puntos en el espacio euclídeo bidimensional. Con esta utilidad, el usuario obtiene el mapa en formato de imagen PNM, que es exactamente el formato que necesita para poderlo utilizar como modelo de mapa en el simulador *Stage* y escribir un cliente que simule un conjunto de posiciones y orientaciones del escáner láser.

8.2. Trabajo Futuro

Algunos de los aspectos que se proponen para continuar el proyecto actual y perfeccionar sus características son:

- Mejorar el algoritmo de correlación cruzada para la alineación de *scans*, investigando soluciones que permitan incrementar su precisión.
- Estudiar procedimientos para que el robot móvil sea capaz de llevar a cabo la construcción de un mapa del entorno de manera automática, sin necesidad de la colaboración del usuario.
- Desarrollar una utilidad de trazado de rayos en el mapa, para simular de manera automática una cuadrícula de posiciones y orientaciones del escáner láser en el interior del entorno.
- Estudiar las prestaciones mecánicas del robot móvil y proponer nuevas alternativas.

Anexo Hardware

Construcción del robot móvil

Proyecto LasMap

Construcción del robot móvil

Versión 1.0

Julio 2005

Realizado Por	Tutores
Carlos Fernández Caramés	*Vidal Moreno Rodilla* *Belén Curto Diego*

Para

Departamento de Informática y Automática

Universidad de Salamanca

Índice general

Índice de figuras

1

Introducción

El principal sensor que se suele emplear en la construcción de mapas, tanto bidimensionales como tridimensionales suele consistir en un escáner láser de barrido. Para este proyecto el escáner láser ha sido desde el principio el sensor central del proyecto y concretamente se ha empleado el modelo LMS 221 de la casa SICK (fig. 1.1 a)). Este modelo escanea su entorno en un plano 2D y está especialmente diseñado para poder trabajar, además de en ambientes interiores, en ambientes exteriores con condiciones extremas (lluvia, nieve, niebla, frío).

El Sick LMS 221 es un escáner láser que basa su funcionamiento en la medida del tiempo de vuelo. Como se muestra en la fig. 1.1 b), el escáner emite un pulso de rayo láser infrarrojo, que rebotará si se encuentra un objeto en su camino, llegando de nuevo al receptor del escáner. Éste, mide el tiempo que ha tardado el pulso de luz en ir y volver. Como la velocidad a la que se

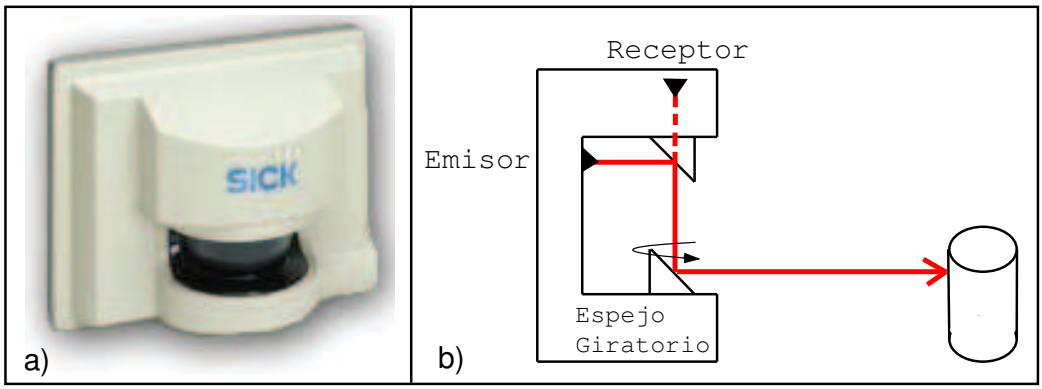

Figura 1.1: *a) Escáner láser LMS 221 de SICK. b)Sistema de funcionamiento interno*

propaga la luz láser es conocida, la electrónica del escáner hace internamente el cálculo de la distancia a la que se encuentra el objeto.

El modelo LMS 221 posee una resolución de 1º, 0.5º ó 0.25º a elegir por el usuario. Su ángulo de apertura es de 180º. Los datos obtenidos por el láser son transmitidos al PC por medio del puerto serie utilizando el protocolo RS-232. La tasa de transmisión podrá ser elegida también por el usuario de entre la siguientes: 9.6, 19.2, 38.4 ó 500 Kbaudios (en este último caso se necesita una interfaz RS-422).

En la tabla 1.1 se muestran las caraterísticas técnicas del SICK LMS 221.

Amplitud de barrido	180º
Resolución angular:	de 0,25º a 1º
Tiempo de respuesta:	de 13 a 52 ms
Resolución:	10 mm
Alcance del láser:	80 m
Interfaz de datos:	RS-422 o RS-232
Velocidad de transmisión de datos:	9.6, 19.2, 38.4 ó 500 Kbaudios
Suministro de voltaje:	24 V DC
Consumo de potencia:	20 W
Peso:	9 kg
Dimensiones:	196 x 352 x 266 mm

Tabla 1.1: *Datos técnicos del láser*

2

Diseño mecánico del robot

En este apartado se describe el diseño de un soporte fundamental para lograr que la lectura de los datos se realice en el plano horizontal. Además se comentan las dos plataformas para transportar el láser que se construyeron. La primera es muy básica, sin motorizar, con una fuente de alimentación conectada a la red eléctrica y con un ordenador portátil en el que se ejecutaban los programas de lectura de datos. La segunda plataforma construida es muucho más compleja, se puede decir que es un robot móvil completamente autónomo. Lleva baterías portátiles para eliminar la necesidad de cables y es posible controlar su movimiento y la lectura de datos de manera totalmente inalámbrica desde un ordenador de sobremesa.

2.1. Soporte para el escáner láser

El láser toma datos del entorno de manera bidimensional, esto es, en un plano. Si el plano en el que toma los datos el láser no es el mismo que el plano en el que el láser se traslada y gira, será imposible establecer una correlación entre los datos de un scan y el siguiente, ya que los datos de cada scan tomado pertenecerán a un plano diferente.

En la fig. 2.1 se puede apreciar este fenómeno. La plataforma en la que está colocado el escáner avanza en el plano horizontal en el sentido que indica la flecha. Sin embargo el láser se encuentra girado ligeramente con respecto al plano horizontal. Por tanto, los datos obtenidos por el láser estarían en un plano que formaría un ángulo α con respecto a la horizontal y los datos de cada scan estarían en planos paralelos separados por cierta distancia vertical.

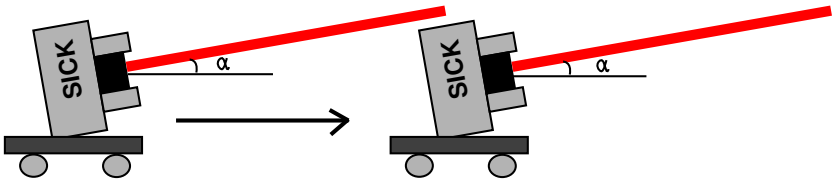

Figura 2.1: *Laser colocado incorrectamente*

Por lo tanto se necesita algún mecanismo para sujetar el láser a la plataforma de manera que sea muy difícil, prácticamente imposible que el plano en el que toma datos el láser no sea el plano horizontal. Así, todos los datos que obtenga el láser corresponderán a un sólo plano (ver fig. 2.2), y gracias a esto un scan se podrá correlacionar con el siguiente y calcular su desplazamiento y rotación.

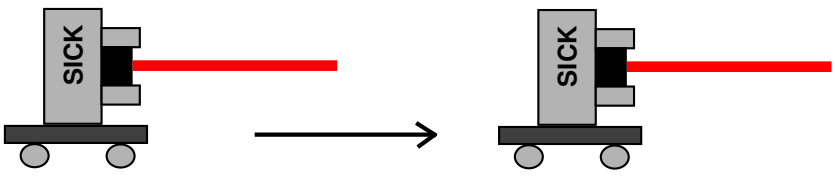

Figura 2.2: *Laser colocado correctamente*

Debido al peso y dimensiones del robot (aprox. 9 Kg), no se ha podido utilizar ningún tipo de soporte estándar comercial. Aprovechando que en la trasera del escáner se encuentran cuatro agujeros M8, es decir de un diámetro de 8 mm, se decidió diseñar un soporte de aluminio en forma de L o en forma de T invertida en el cual sujetar el escáner láser mediante cuatro tornillos M8 (de gran robustez). Un soporte en forma de L ocupa menos espacio. Sin embargo el soporte en forma de T invertida debería ser más estable, puesto que la sujeción a la plataforma móvil se realizaría mediante una superficie el doble de amplia.

El diseño se realizó mediante la herramienta AutoCad, y la construcción de la pieza la llevó a cabo el taller mecánico *"Rectificados del Oeste"*. En la fig. 2.3 se observa el diseño de la pieza en 3 dimensiones, y en las páginas siguientes se muestran el alzado, planta el perfil, y la pieza una vez construida. Todas las unidades vienen dadas en milímetros.

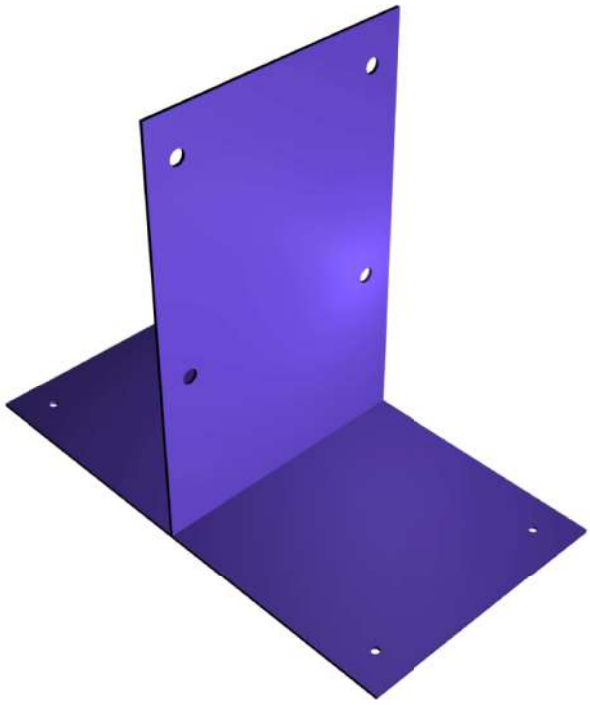

Figura 2.3: *Pieza en 3D*

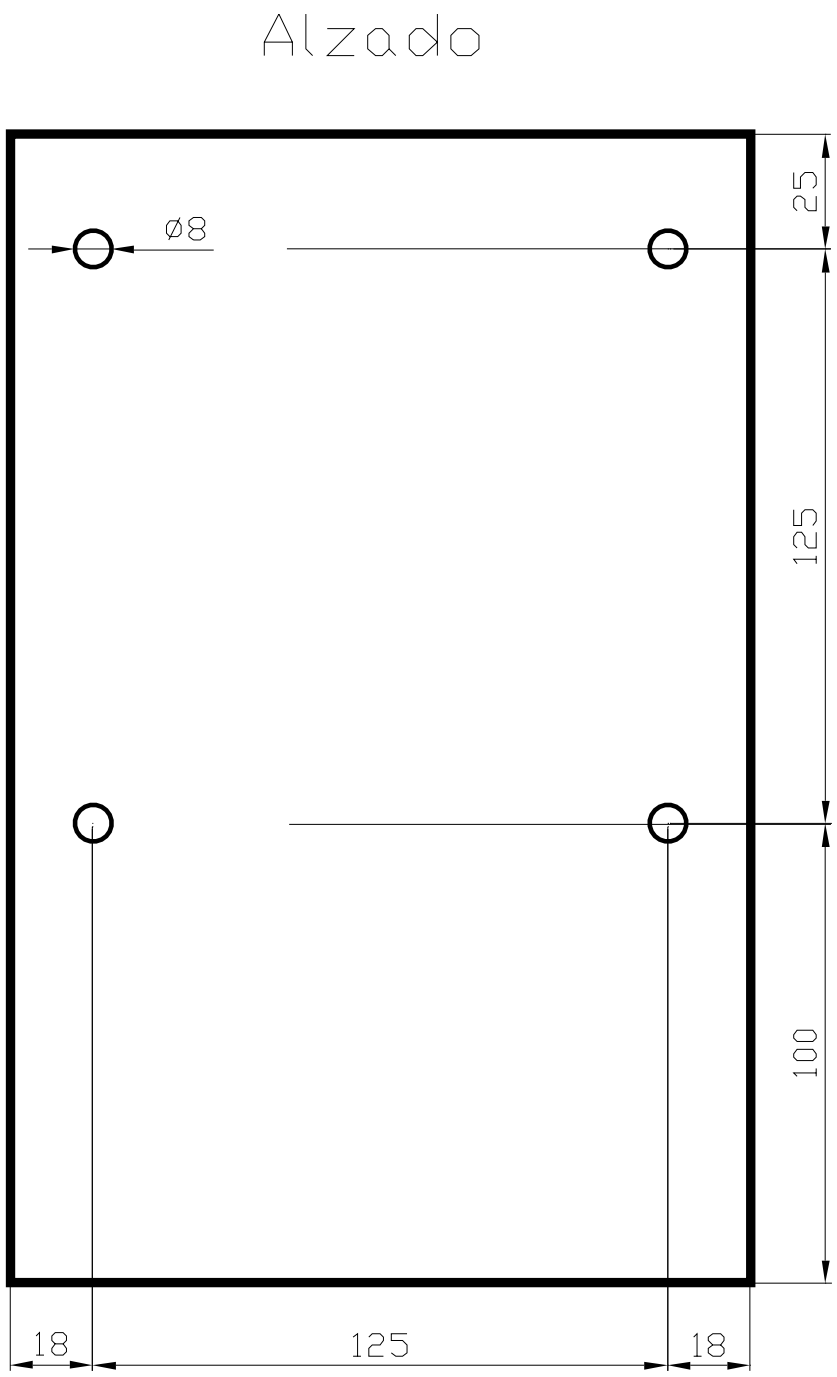

Figura 2.4: *Alzado de la pieza*

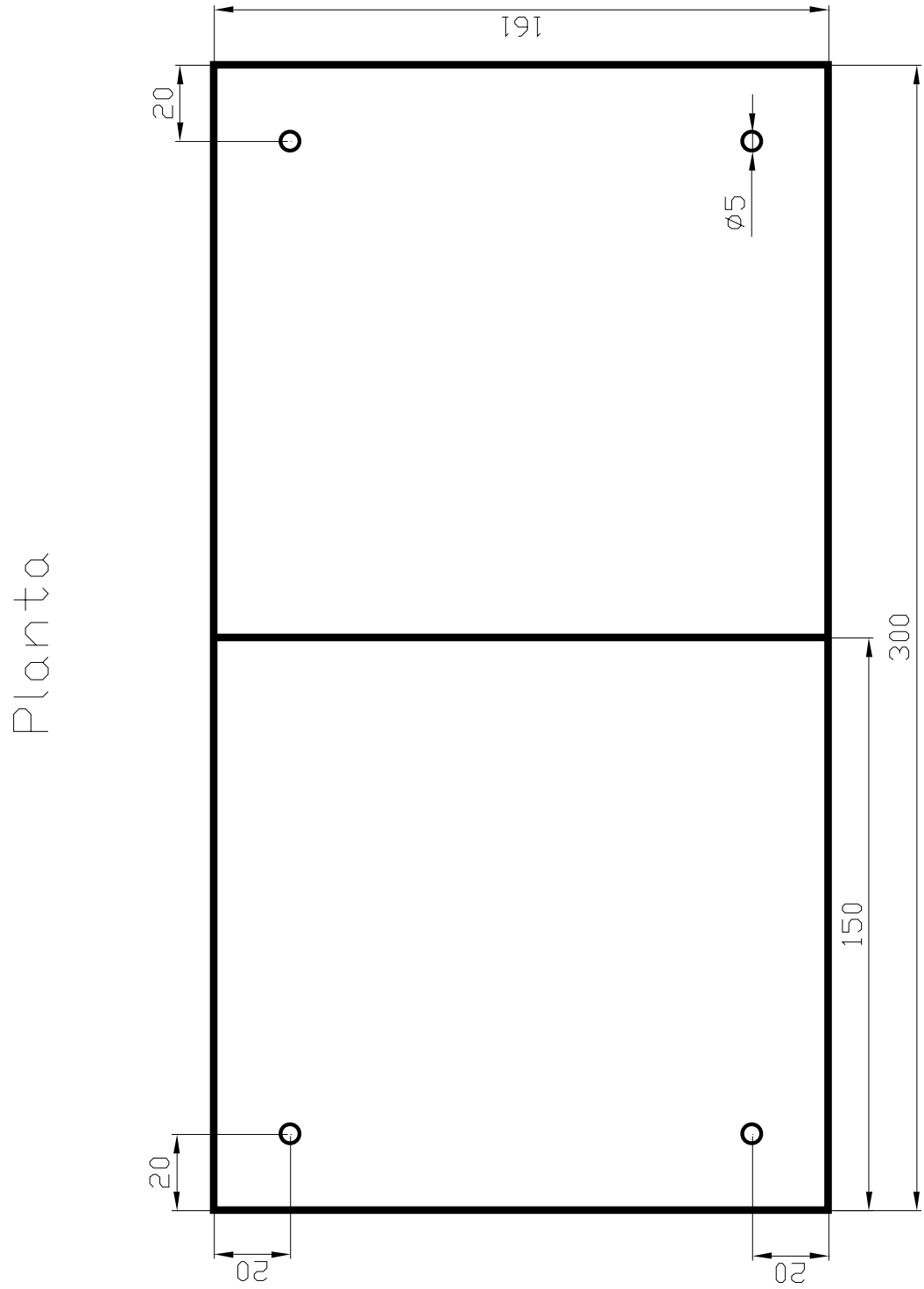

Figura 2.5: *Planta de la pieza*

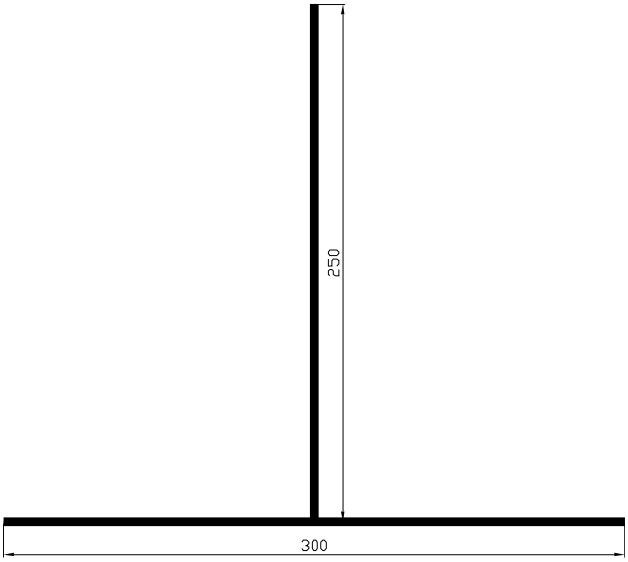

Figura 2.6: *Perfil de la pieza*

Figura 2.7: *Pieza final una vez construida*

2.2. Diseño de la plataforma inicial

Antes de decidir construir un robot móvil autónomo se opto por construir una plataforma muy rudimentaria, en la que poder transportar el escáner láser junto con dos elementos imprescindibles para poder utilizarlo.

Uno de ellos es una fuente de alimentación que sea capaz de proporcionar los 24 V de corriente continua que el láser necesita. El otro elemento es un ordenador portátil, para poder obtener datos del láser fuera del laboratorio de robótica, ya que el éste necesita una conexión directa a un PC a través del puerto serie. La plataforma esta soportada por cuatro ruedas locas situadas cerca de las esquinas y es lo suficientemente grande para que quepan en ella los tres elementos.

El escáner esta sujeto a la plataforma mediante el soporte a medida descrito en el apartado anterior (2.1). En la fig. 2.8 se puede apreciar que el artilugio construido es realmente rudimentario, sin embargo, para realizar las primeras pruebas de toma de datos fuera del laboratorio de robótica era más que suficiente.

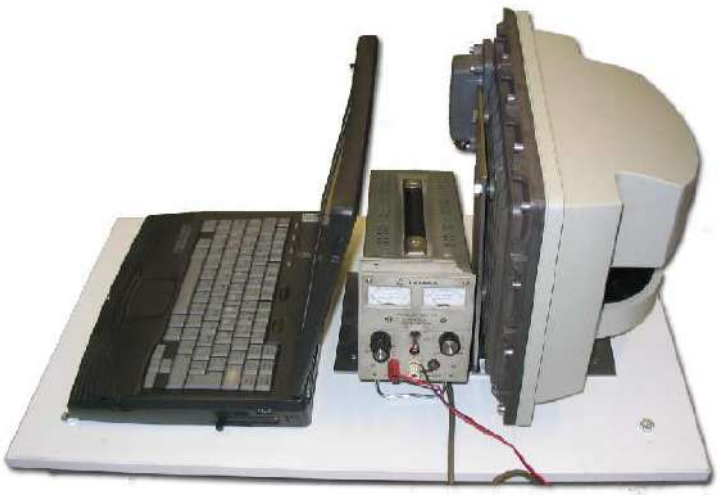

Figura 2.8: *Primera plataforma para el escáner láser*

2.3. Diseño del robot móvil autónomo

En la fig. 2.9 se muestra el aspecto que presenta el robot móvil que se ha construido.

Figura 2.9: *Aspecto final del robot móvil*

El primer paso que se da en la construcción del robot es la elección de su configuración, esto es, definir cómo se van a distribuir los principales elementos que lo componen: plataforma, motores y ruedas.

Como plataforma se ha escogido una tabla de madera de $37 \times 37 \times 1{,}5$ cm. Estas dimensiones hacen que la tabla sea del tamaño mínimo indispensable para poder alojar el escáner láser, las baterías y el mini-pc. Éstos elementos irán encima de la tabla al ser más grandes, mientras que la electrónica irá acoplada por debajo de la tabla, ya que la componen elementos de tamaño más reducido.

En relación a las ruedas existen distintas configuraciones típicamente uti-

lizadas en robótica móvil: diferencial, triciclo, Ackerman, omnidireccional, con múltiples grados de libertad y movimiento mediante orugas. Se ha elegido la configuración diferencial por ser la más sencilla.

Configuración diferencial

Para distribuir las ruedas hay que tener en cuenta que la plataforma es cuadrada y el peso del láser estará colocado íntegramente en la mitad delantera de la tabla. Sabiendo esto se ha optado por colocar una rueda loca en la mitad delantera de la tabla (ya que puede soportar mucho más peso que las ruedas fijas) y cuatro ruedas fijas en la mitad trasera de la tabla, como se puede ver en la fig. 2.10. Las ruedas locas se caracterizan porque no llevan asociadas ningún motor, y giran libremente según la velocidad del robot. Además pueden orientarse según la dirección del movimiento.

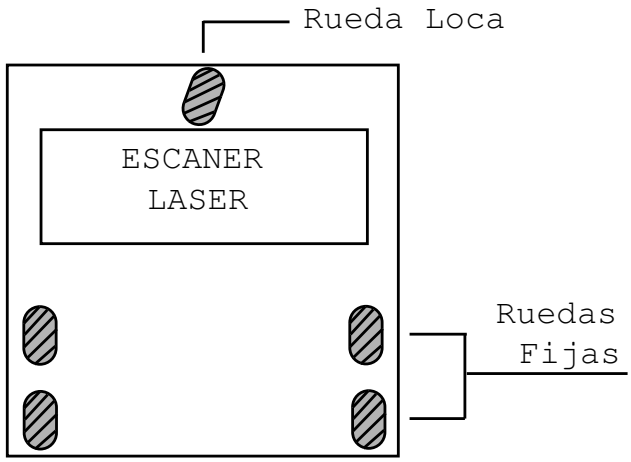

Figura 2.10: *Configuración diferencial de las ruedas*

Las cuatro ruedas fijas están situadas diametralmente opuestas en un eje perpendicular a la dirección de desplazamiento del robot. A cada lado del eje central de la plataforma irán dos ruedas fijas, cada una de ellas dotada de un motor, de forma que los giros se realizan dándoles diferentes velocidades. Para girar hacia la izquierda se dará mayor velocidad al motor derecho, mientras que para girar a la derecha se dará más velocidad al motor izquierdo. Para

rotar sobre si mismo, se deberá dar la misma velocidad a cada motor, pero en sentido contrario.

Una vez elegida la plataforma y la distribución de los componentes, se procede a elegir los motores, las ruedas y otros componentes para fijar las ruedas al motor y los motores a la plataforma.

Motores

Para la tracción del robot se ha elegido el motor de la fig. 2.11 a). Es un motor que necesita una alimentación de 12 V. Tiene un consumo de 6mA y una fuerza de 8,8 Kg/cm. Como máximo puede alcanzar 120 revoluciones por minuto. Sin embargo, y gracias a que es un motor con una posibilidad de reducción de 50:1, se ha utilizado esta característica. La reducción 50:1 significa que el motor tiene un mecanismo interno que le permite hacer que por cada 50 revoluciones internas, el eje del motor solamente de una vuelta. Gracias a esta reducción se consigue una potencia mucho mayor.

Ruedas

Se han elegido ruedas como las de la fig. 2.11 b), porque están especialmente indicadas para robots que necesiten gran adherencia y alto nivel de tracción. Este tipo de rueda es apta para superficies de todo tipo, presentando un alto nivel de adherencia gracias al material compuesto que la forma y que similar al neopreno. Debido al gran peso del láser la adherencia de las ruedas resultará de gran ayuda para aprovechar toda la potencia del motor.

Elementos de fijación

Para asegurar las ruedas al eje del motor se han seleccionado los casquillos de fijación de la fig. 2.12 a), realizados en aluminio, y para sujetar el motor a la plataforma de madera, se han utilizado soportes de aluminio como el de la fig. 2.12 b). El único inconveniente al realizar el montaje es que los soportes

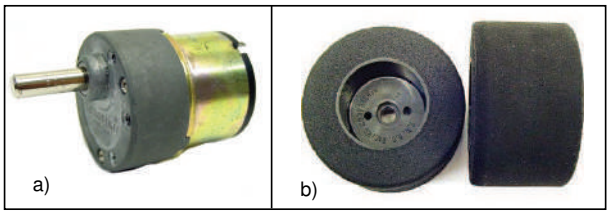

Figura 2.11: ***a)*** *Motores del robot.* ***b)*** *Ruedas del robot.*

no fueron tan fuertes como se esperaba, aunque de todas maneras soportan bien el peso del láser.

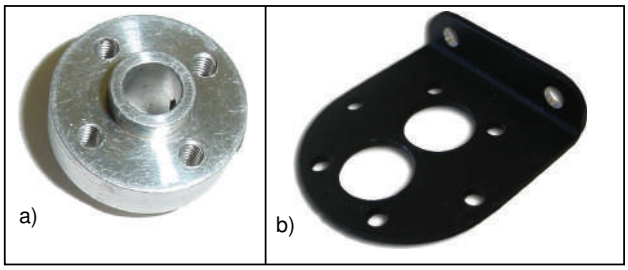

Figura 2.12: *Elementos de fijación*

3

Diseño eléctrico del robot

3.1. Diagrama eléctrico general

En la figura 3.1 se representa el diagráma eléctrico global que controla el funcionamiento del robot. En los apartados siguientes se comentará cada uno de los elementos que aparecen en el diagrama.

3.2. Sistema de alimentación

Como sistema electrónico que es, el robot necesita una fuente de alimentación. Uno de los requisitos que se decide imponer a la hora del diseño es que el robot sea completamente autónomo, para que no haya limitación en cuanto a su movimiento debido a las restricciones de movimiento que le impondrían los cables. Para ello en el robot móvil se situará una batería de plomo/calcio

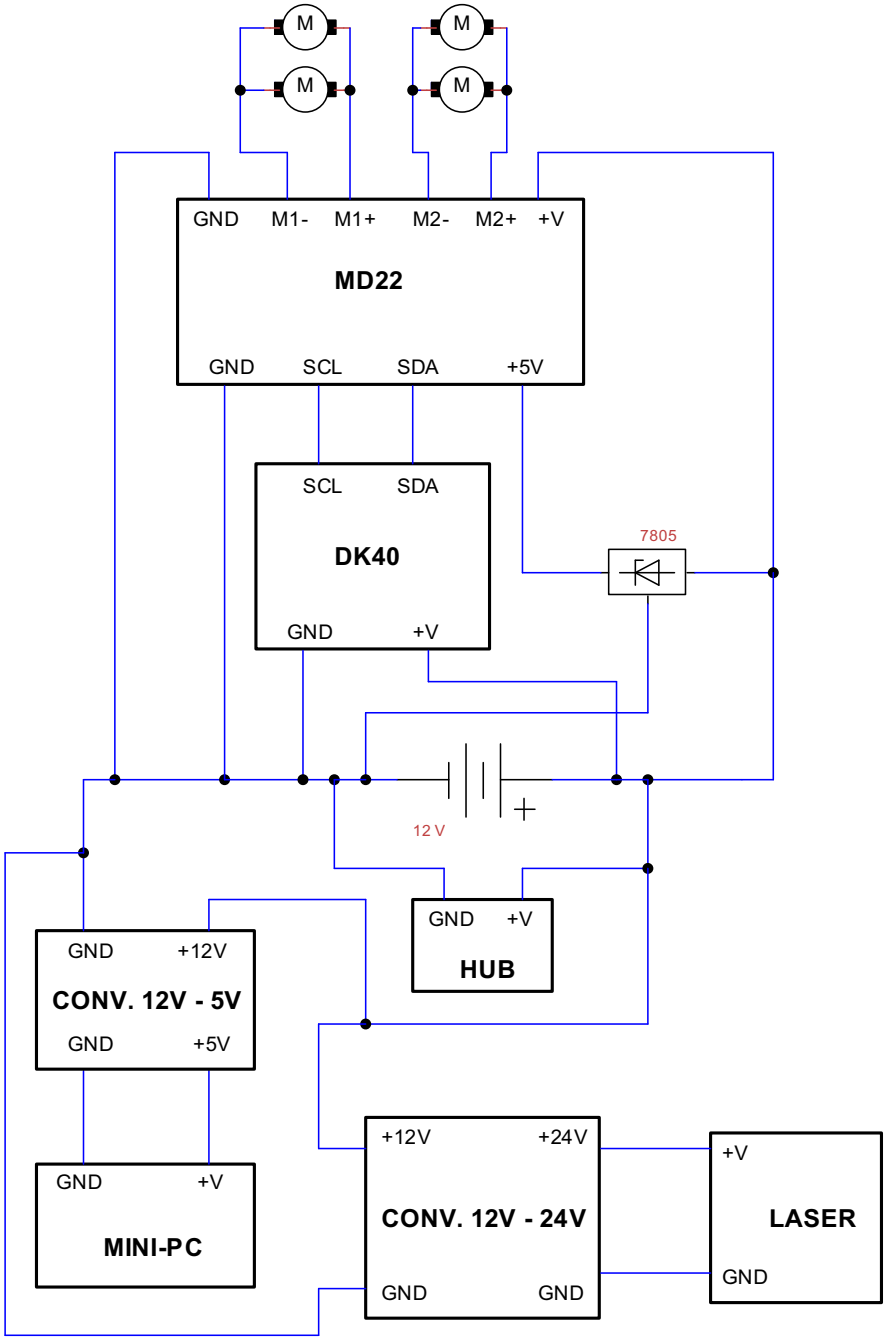

Figura 3.1: *Esquema eléctrico de las conexiones del robot*

de 12 Voltios y 7 Amperios/Hora. Puesto que el láser necesita $24V$ de alimentación y el mini-pc necesita $5V$, se han colocado dos conversores de voltaje, uno de $12V$ a $24V$ y otro de $12V$ a $5V$, respectivamente.

3.3. Control de los motores: MD22

La configuración electrica de los motores se ha establecido de manera que los cuatro motores actúen como si sólo fueran dos. Esto es, los motores que están situados del mismo lado, se han conectado en paralelo, de modo que al enviarle un determinado voltaje a un motor para que gire, el motor contiguo gire en el mismo sentido.

Los motores no pueden controlarse en directo desde el servidor *Player* ni desde un puerto serie. Para hacerlos funcionar correctamente hay que hacerlo a través de un driver de potencia. En este caso se ha elegido el módulo controlador MD22 [42] (fig. 3.2), que permite controlar dos motores de corriente continua de mediana potencia, y está diseñado para proporcionar mas potencia que los controladores basados en un único circuito integrado.

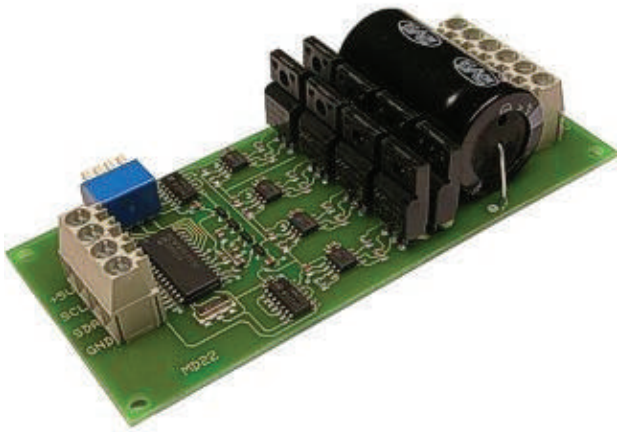

Figura 3.2: *Controlador de motores MD22*

Los 15V de la tensión de control del MOSFET se genera en el mismo circuito mediante una bomba de carga, por lo que solo se requieren 5V a 50

mA para la alimentación del circuito, además de la alimentación del motor, que en este caso son 12V a 6mA. El modulo puede controlarse de 5 formas diferentes:

1. Modo bus I2C. Hasta ocho módulos MD22 con direcciones seleccionables mediante micro interruptores y 4 modos de funcionamiento incluyendo el control de dirección (modo diferencial).

2. Dos entradas analógicas independientes de 0V - 2.5V - 5V. 0V un sentido, 2,5V parado y 5V la otra dirección.

3. Una entrada analógica 0v-2,5v-5v. para el control de la velocidad y la otra para el control de dirección.

4. Modo RC de canales independientes. Para controlarlo directamente desde un receptor de radio control estándar. Cada canal controla un motor de forma independiente.

5. Modo RC con control de dirección. Un stick controla la velocidad, mientras que el otro controla la dirección.

De entre todos estos modos sólamente nos interesa el primero, porque utiliza el bus I2C para controlar los motores, y esto los comandos necesarios se pueden enviar por este bus por medio del chip DK40.

Circuito regulador de tensión

Para utilizar el controlador MD22 en el modo I2C es necesario proporcionar +5V de tensión en la parte de control de la placa. Para ello se emplea un regulador de tensión 7805 (ver fig. 3.1), el cual acepta tensiones de entrada entre 7 y 25 voltios y proporciona una salida fija de +5 voltios.

3.4. Módulo controlador DK-40

El módulo controlador DK-40 (fig. 3.3)es un controlador de propósito general de la casa BECK [43]. Contiene un chip IPC@CHIP SC12, cuyas

características hardware más importantes son:

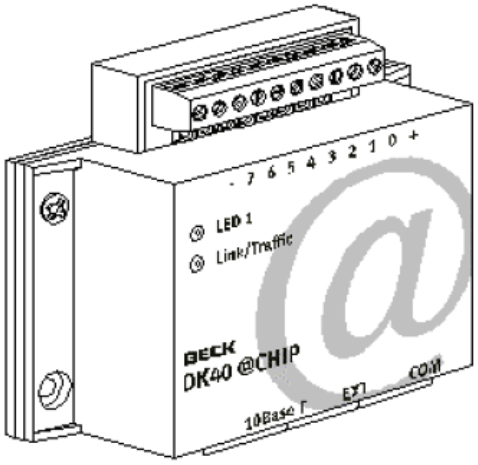

Figura 3.3: *Controlador DK40*

- Procesador a 186 Mhz.
- 256 Mb de memoria RAM.
- Flash Disk.
- Dos puertos serie.
- Tarjeta de red.

Entre las características software más relevantes del DK-40 se pueden destacar:

- Servidor Web.
- Servidor FTP.
- Permite conexiones por *socket* TCP y UDP.
- Posee una API para el control del puerto serie, los *sockets* y la creación de procesos.

- Soporta semáforos.
- Incorpora una shell de órdenes tipo DOS y permite ejecutar aplicaciones compiladas para MS-DOS

La importancia de este módulo controlador es que es donde se ejecuta el programa servidor que accede a los motores a petición de *Player*, utilizando el servidor *Player* modificado según se indica en [19].

El diseño original presentado en [19] no permite controlar los motores a través del bus I2C, sin embargo mediante una pequeña modificación del código controlador de los motores, funciona a la perfección. Hubiera sido deseable disponer de un controlador para leer los datos del escáner láser directamente desde el DK-40, sin embargo esto no ha sido posible ya que la implementación de un driver controlador del láser podría constituir prácticamente un proyecto por si sólo. Por lo tanto fue necesario utilizar un mini-pc para leer los datos del láser a través del puerto serie.

3.5. Mini-pc y hub inalámbrico

Estos dos dispositivos juegan un papel clave en la autonomía del robot. El *hub* inalámbrico consta de dos puertos ethernet. El mini-pc es un pc reducido al máximo, de tal modo que sus dimensiones son de unos $10 \times 15 \times 4$ cm. Sus características principales son:

- Placa base ICOP-6075.
- 40 Gb de disco duro.
- 128 Mb de RAM.
- CPU Vortex a 166 Mhz.
- Necesita una alimentación 5 Voltios.

El mini-pc como se ha comentado lee los datos del láser a través del puerto serie, y para ello debe estar ejecutando el servidor *Player* modificado tal

y como se indica en [19]. A su vez El DK-40 ejecutará un servidor que se comunicará con el servidor *Player* modificado. Gracias a esta comunicación, el servidor *Player* modificado enviar comandos de velocidad al servidor instalado en el DK-40 y éste a su vez a los motores. Finalmente, tanto el DK-40 como el mini-pc se conectarán a un hub inalámbrico, el cual hará que un cliente desde un PC de sobremesa pueda manejar por completo el robot móvil, tanto leyendo del láser, como actuando sobre los motores.

Anexo 1

PLAN DEL PROYECTO SOFTWARE

Proyecto LasMap

Plan Del Proyecto Software

Versión 1.0

Julio 2005

Realizado Por	Tutores
Carlos Fernández Caramés	*Vidal Moreno Rodilla*
	Belén Curto Diego

Para

Departamento de Informática y Automática

Universidad de Salamanca

Índice general

1 Introducción

A continuación se presentan, en forma de diagrama de Gantt, la distribución en el tiempo de cada una de las tareas que se han llevado a cabo durante la realización del proyecto. Un diagrama de barras o de Gantt es una representación gráfica de las actividades del proyecto sobre una escala de tiempos. Las actividades se representan en forma de barra sobre dicha escala manteniendo la relación de proporcionalidad entre sus duraciones y su representación gráfica, y su posición respecto del punto origen del proyecto.

1.1. Calendario de trabajo

La duración estimada del proyecto es la de un curso académico, esto es, 9 meses, proponiendo como fecha de comienzo el 20 de Septiembre de 2004, y como fecha final, Junio de 2005.

Para la jornada laboral se ha establecido un horario consistente en 7 horas dia-

rias, de lunes a jueves, de 10:00 a 14:00 h y de 16:00 a 19:00h. Los viernes se ha determinado un horario especial de 6 horas, de 10:00 a 14:00 h y de 16:00 a 18:00 h. El fin de semana quedará como horario libre.

2

Fases del proyecto

El proyecto ha sido descompuesto en las siguientes fases.

- Preliminares
- Estudio del servidor Player/Stage
- Estudio del funcionamiento del láser
- Estudio de técnicas SLAM
- Especificación de requisitos software
- Análisis
- Diseño
- Implementación

- Construcción del robot
- Documentación.

A continuación se describe cada una de las fases.

2.1. Preliminares

Desde el 20 de Septiembre, fecha de comienzo del proyecto, hasta finales de ese mismo mes, se realizará la tarea de captación de requisitos iniciales, para lo cual será necesario acordar los diferentes objetivos del proyecto efectuando una serie de reuniones con el cliente.

2.2. Estudio de *Player/Stage*

Uno de los objetivos es manejar el servidor *Player*. Para lograr esta tarea se estima una duración de dos meses, los de octubre y noviembre. Una vez concluida esta fase se supone que se dominará completamente el servidor *Player* y el simulador Stage.

2.3. Estudio del funcionamiento del láser

Otro objetivo imprescindible es manejar el escáner láser en conjunto con el servidor *Player*. Ésta tarea se realizará simultáneamente con la anterior, estimándose que durante los meses de octubre y noviembre se dispondrá del suficiente tiempo para dominar por completo el funcionamiento del láser, así como sus características y peculiaridades.

2.4. Estudio de técnicas SLAM

Una vez terminado el estudio del láser y del servidor *Player*, se dominarán por completo estos dos elementos. Con ésto se estará en disposición de comenzar, en el mes de Diciembre, el estudio de los diferentes métodos para dar una solución al

problema de la construcción de mapas y localización simultánea mediante el uso de un escáner láser. Puesto que no se conoce nada en absoluto de estas técnicas, se estima una duración bastante larga, de cinco meses, concluyendo a finales del mes de Abril.

2.5. Especificación de requisitos, análisis, diseño e implementación

A partir del mes de Noviembre, en el que se estima que se tendrán unos ciertos conocimientos acerca del servidor *Player* y el escáner láser, se podrá comenzar una primera iteración de estas fases. Cuando realmente cambiarán los requisitos, será a lo largo del estudio de las técnicas de SLAM, y es donde se realizará más hincapié en la realización de estas fases. Se estima que la duración de las cuatro fases será de unos 7 meses, prácticamente igual a la duración de todo el proyecto, exceptuando el primer mes, dedicado al aprendizaje de *Player* y del funcionamiento del escáner láser en exclusiva, y el último mes, en el que se espera tener que realizar los últimos esfuerzos de documentación únicamente.

Puesto que se ha seleccionado un ciclo de vida iterativo, basado en prototipos, estas cuatro fases, no pueden efectuarse secuencialmente, una detrás de otra. Por el contrario, al irse realizando prototipos sobre una determinada funcionalidad, se realizarán varios mini-ciclos de vida, que constarán, cada uno de ellos de las fases habituales del ciclo de vida clásico: especificación de requisitos, análisis, diseño y documentación. Cada uno de estos mini-ciclos de vida constituirá una iteración, y por este motivo, en el diagrama de Gantt aparecen estas cuatro fases con la misma duración en el tiempo, si bien en cada iteración las fases se podrán solapar, ya que no es necesario que sean secuenciales.

2.6. Construcción del robot

En cuanto se disponga de los conocimientos mínimos para manejar el láser en conjunción con el servidor *Player* se llevará a cabo la construcción de una plataforma

móvil que no esté motorizada, para poder realizar las primeras pruebas de lectura de datos en el entorno. Esto se representa con una barra con una duración de un mes, desde principios de noviembre hasta principios de Diciembre, si bien la duración será ciertamente variable debido a la necesidad de encargar una pieza a unos talleres mecánicos.

Por otra parte, la verdadera construcción del robot móvil comenzará una vez que se dispongan de los conocimientos acerca de las técnicas de SLAM y una vez que se hayan desarrollado prototipos prácticos de estas técnicas. Debido a que no se dispone de ningún conocimiento práctico de robótica, se estimará una duración de tres meses, comenzando en marzo, para tener suficiente tiempo de estudiar los diferentes elementos que formarán el robot móvil, tanto mecánicos como eléctricos, y posteriormente efectuar el esfuerzo de construcción.

Esta fase no se necesita comenzar demasiado pronto, ya que el robot móvil se utilizará solamente para comprobar en un entorno real el funcionamiento de las técnicas SLAM implementadas, y por lo tanto se puede retrasar su comienzo hacia los meses finales.

2.7. Documentación

En esta fase se recogen los aspectos más destacados de las otras fases para plasmarlos en documentos que contengan todo el proceso de análisis, diseño, desarrollo y construcción del robot. También se genera aquí el manual del usuario y la memoria teórica del proyecto. Se estima que esta fase será la más larga, comenzando en el mes de noviembre, ya que durante el mes de octubre no se dispondrá aún de aspectos teóricos o prácticos para documentar, debido a que se estará realizando una primera aproximación al servidor *Player* y al manejo del láser. Finalmente, se espera haber terminado el resto de tareas y poder realizar los últimos ajustes en la documentación.

Anexo 2

Especificación de requisitos software

Este anexo recoge tanto los requisitos software que debe cumplir el sistema a desarrollar, como un análisis de éstos para poder obtener un modelo preliminar que sirva como punto de partida del diseño del sistema.

Debido a esto el anexo 2 se encuentra dividido a su vez en dos nuevos anexos:

- Anexo 2A: Este documento es el encargado de recoger todos los requisitos que el sistema debe cumplir de forma que se convertirá en un elemento contractual entre la parte cliente y el desarrollador.
- Anexo 2B: Documento donde se realiza el análisis de los requisitos obtenidos en el anexo 2A.

Anexo 2A

DOCUMENTO DE REQUISITOS DEL SISTEMA

Proyecto LasMap

Documento de Requisitos del Sistema

Versión 1.0

Julio 2005

Realizado Por	Tutores
Carlos Fernández Caramés	*Vidal Moreno Rodilla* *Belén Curto Diego*

Para

Departamento de Informática y Automática

Universidad de Salamanca

Índice general

Índice de figuras

Índice de tablas

1 Introducción

Para la especificación de requisitos del software se ha optado por la *Metodología para la Elicitación de Requisitos de Sistemas Software V 2.1.* [32]. En esta metodología se recomienda la utilización de casos de uso para la identificación de los requisitos que deberá cumplir el sistema. Todos estos requisitos estarán recogidos en un documento denominado DRS (Documento de Requisitos del Sistema) cuyo esquema será seguido a lo largo de este anexo. De esta forma la organización de este anexo será la siguiente:

- Objetivos del sistema
- Catálogo de requisitos del sistema
  - Requisitos de almacenamiento de información
  - Requisitos funcionales

  - Definición de actores
  - Diagramas de casos de uso
  - Casos de uso del sistema
- Requisitos no funcionales

2

Objetivos del Sistema

El objetivo de este proyecto es la creación de una aplicación de fácil uso cuya función sea ayudar en la construcción de mapas de entornos interiores, esto es, creados por el hombre, en los que predominen estructuras rectilíneas, como por ejemplo paredes, armarios, etc, y además ofrecer una función de autolocalización en un entorno que previamente haya sido explorado. Para ambas tareas se utilizará como dispositivo imprescindible un escáner láser de la casa *Sick*, y como dispositivo auxiliar una plataforma motorizada que transporte al escáner láser.

El escáner láser es capaz de emitir haces de luz con un radio de barrido de 180 grados y 0.5 o 1 grado de distancia entre cada haz .Proporciona un conjunto de datos consistentes en la distancia medida hasta el objeto más cercano que se encontraba al emitir el pulso de luz. Este barrido del entorno por sí solo no ofrece mucha información, hay que analizar los datos del láser

en tiempo real para proporcionar algún tipo de utilidad. Típicos usos de este tipo de escáner son por ejemplo detectar personas en áreas peligrosas o medir volúmenes de objetos en áreas industriales.

La aplicación que se pretende desarrollar presenta una utilidad distinta a las habituales para este tipo de escaners industriales: construir mapas y autolocalizarse posteriormente en ellos. Para la primera parte, la aplicación dispondrá de un lienzo de dibujo en el que realizar todas las operaciones necesarias para construir un mapa del entorno. Por ejemplo se podrán tomar "fotografías" (en dos dimensiones) de la parte del entorno que está digitalizando el escáner en un momento dado. Al tomar dos de estas fotografías, se ofrecerá al usuario la posibilidad de hacerlas coincidir, mostrando el resultado de esta alineación en el área de dibujo. De este modo el usuario podrá decidir si la alineación es correcta y convertir los resultados en un mapa global. Repitiendo este proceso se podrán añadir más resultados al mapa, incrementando por tanto su tamaño y precisión. La aplicación también facilitará el borrado del mapa global así como eliminar la última fotografía tomada, en caso de haber cometido un error.

Por otra parte se debe proporcionar una interfaz para controlar el robot móvil en el que irá montado el escáner láser, ofreciendo la posibilidad de controlar el sentido de la marcha, el giro y la velocidad del robot en todo momento, para decidir en qué punto se desea tomar una nueva fotografía del entorno. Puesto que los motores del robot físico se controlarán de modo distinto a los simulados por *Stage*, será necesario ofrecer esta alternativa en la aplicación; el láser físico y el simulado por *Stage* funcionan exactamente igual, con lo cual no habrá que establecer una diferenciación.

Para controlar el escáner láser y los motores del robot hay que hacerlo a traves de *Player/Stage*, y por lo tanto habrá que permitir al usuario conectarse y desconectarse (suscribirse / cancelar la suscripción) a estos dos dispositivos (láser y *position* en la terminología *Player*)

Una vez que el mapa esté construido de manera aceptable, el usuario podrá almacenar el conjunto de puntos que lo forman como una imagen en

formato *PNM*. Esto será necesario hacerlo para interactuar con el simulador *Stage*. El usuario, al margen del sistema propuesto en este proyecto, deberá obtener un conjunto de scans simulados en todas las posiciones y giros que desee para el entorno del que acaba de obtener el mapa, utilizando el mencionado simulador *Stage*. Se recomienda simular que el escáner láser se situe en una rejilla imaginaria con 0.5 metros de anchura en el entorno del que se posee el mapa. En cada punto de la rejilla, se deberá girar el escaner 1 grado hasta regresar a la posicion original, obeniendo 359 posiciones distintas para cada punto.

Teniendo un conjunto de scans simulados, el sistema permitirá realizar un análisis de componentes principales a éste conjunto, permitiendo reducir en gran medida la dimensionalidad de los datos, sin perder las características geométricas que definen al entorno. El análisis PCA puede requerir gran cantidad de tiempo de procesamiento si el conjunto de scans simulados es de gran tamaño. Por esto, el usuario deberá poder almacenar en memoria secundaria un análisis PCA para posteriormente volver a leerlo. Finalmente, al tener un análisis PCA en la memoria y el escáner láser conectado, se ofrecerá al usuario la posibilidad de que el sistema analice los datos que está percibiendo el láser en un instante determinado y los compare con el análisis PCA que realizó previamente, eligiendo la aplicación las coordenadas más probables en las que se encuentra el escáner.

Para facilitar el uso del área de dibujo se incluirán ciertas características, como el manejo de la herramienta zoom variable, que permitirá acercar y alejar el área de dibujo; una rejilla que dividirá el espacio en cuadrados de igual tamaño, facilitando la tarea de medir distancias y una utilidad para centrar el área de dibujo en el medio de la ventana que la contiene (el área normalmente será más grande que la ventana contenedora).

Por último, la aplicación contará con un sistema de ayuda que proporcione al usuario indicaciones acerca de cómo interactuar con la aplicación, así como avisos en caso de ocurrir algún error.

3

Catálogo de requisitos del sistema

El catálogo de requisitos del sistema se ha dividido en tres secciones que clasifican dichos requisitos en tres grandes grupos: requisitos de almacenamiento de información, requisitios funcionales y requisitos no funcionales. Se han elegido las plantillas según las recomendaciones de Durán y Bernárdez [32] para la especificación de cada tipo de requisito.

3.1. Requisitos de almacenamiento de información

RI - 01	**Información sobre análisis PCA**
Descripción	El sistema deberá ser capaz de almacenar información correspondiente a un análisis de componentes principales que se encuentre en la memoria principal. En concreto:
Datos Concretos	▪ Dimensión de la matriz PCA (entero) ▪ Número de vectores propios a almacenar (entero) ▪ Número de scans sintéticos (entero largo) ▪ Datos del vector media ▪ Datos del vector característico ▪ Matriz de datos almacenada en el espacio vectorial
Importancia	importante
Urgencia	hay presión
Comentarios	Ninguno.

Tabla 3.1: *RI - 01*

RI - 02	**Información sobre análisis PCA**
Descripción	El sistema deberá ser capaz de almacenar información correspondiente a al mapa global del entorno que se encuentre en la memoria principal. En concreto:
Datos Concretos	Se deberá transformar cada uno de los puntos que forman el mapa en un punto de un mapa de bits en formato PNM (portable anymap file). El fondo de la imagen debe ser negro, y cada punto de la imagen que sea un obstaculo real se dibujará en blanco.
Importancia	vital
Urgencia	inmediatamente
Comentarios	Se necesita almacenar una imagen en formato PNM porque es el formato que usa el simulador Stage para interpretar mapas del entorno.

Tabla 3.2: *RI - 02*

RI - 03	**Información sobre análisis PCA**
Descripción	El sistema deberá ser capaz de leer información correspondiente a una serie de scans en formato polar correspondientes a la simulación de haber situado el dispositivo láser en diferentes posiciones y orientaciones en un entorno determinado (del que se dispone un mapa). En concreto:
Datos Concretos	▪ Número de datos de cada scan en formato polar. ▪ Número de scans en formato polar. ▪ Conjunto de scans en formato polar.
Importancia	vital
Urgencia	inmediatamente
Comentarios	Ninguno.

Tabla 3.3: *RI - 03*

3.2. Requisitos funcionales

Se incluyen a continuación los requisitos funcionales del sistema que describen las funcionalidades implementadas. Este apartado se divide en definición de los actores del sistema, diagramas de casos de uso y especificación de los casos de uso del sistema.

3.2.1. Definición de los actores del sistema

Usuario: será toda aquella persona que utilice la aplicación.

3.2.2. Diagramas de casos de uso

- Diagrama de casos de uso para el manejo de gráficos:

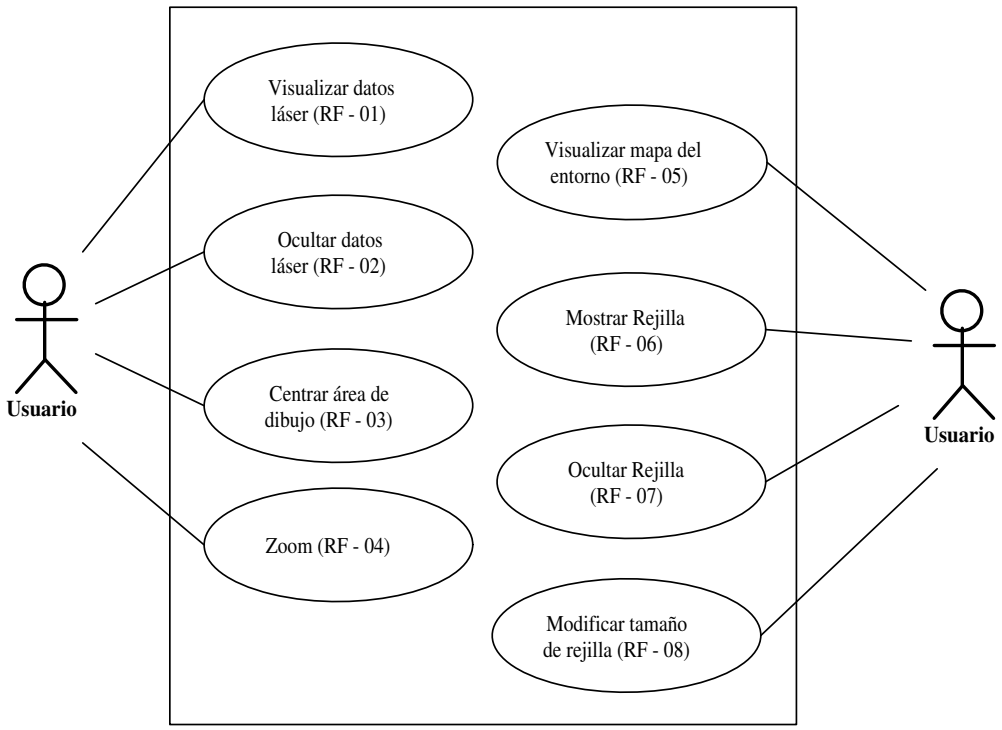

Figura 3.1: *Casos de uso para el manejo de gráficos*

- Diagrama de casos de uso para la construcción de un mapa del entorno:

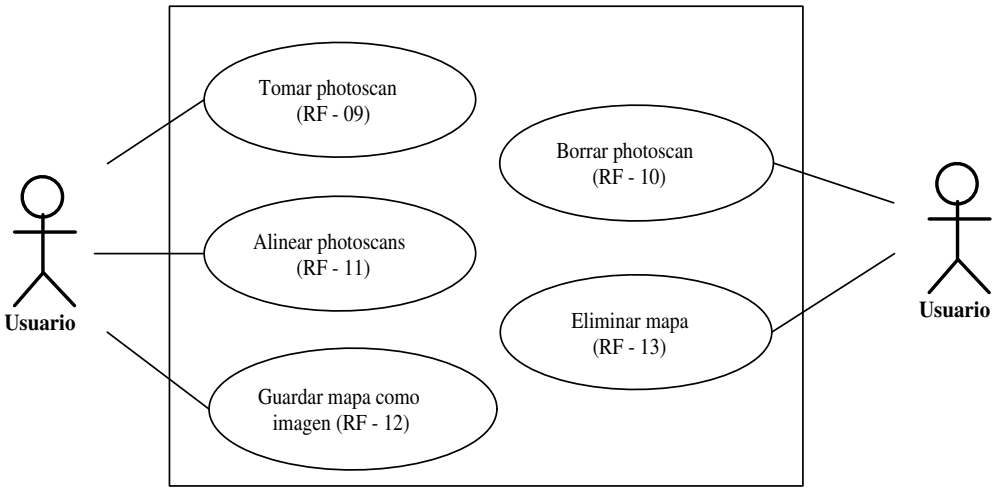

Figura 3.2: *Casos de uso para la construcción de mapas*

- Diagrama de casos de uso para la conexión a dispositivos:

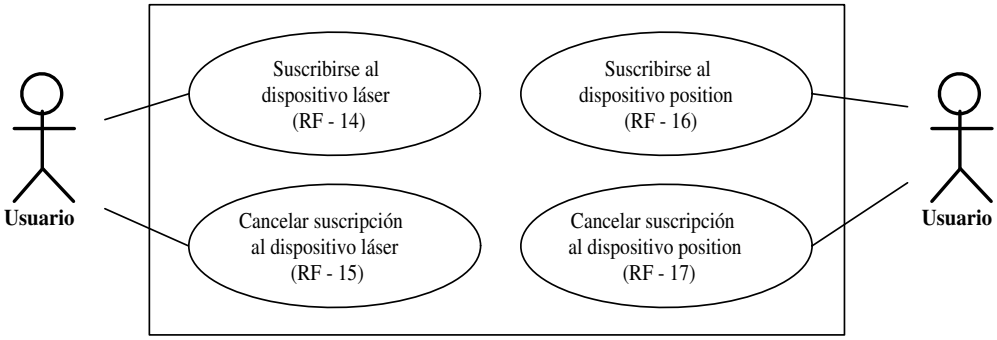

Figura 3.3: *Casos de uso para la conexión de dispositivos*

- Diagrama de casos de uso para el control del movimiento del robot:

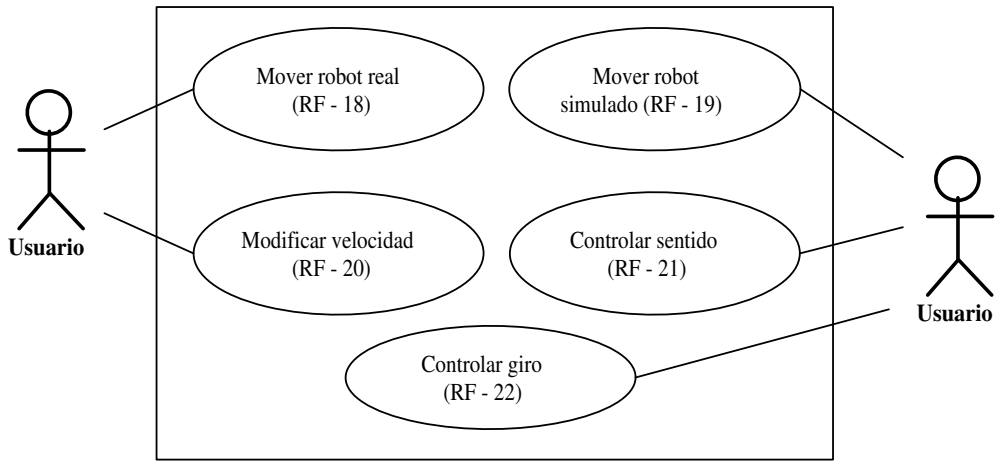

Figura 3.4: *Casos de uso para la construcción el control del robot*

- Diagrama de casos de uso para el cálculo de componentes principales:

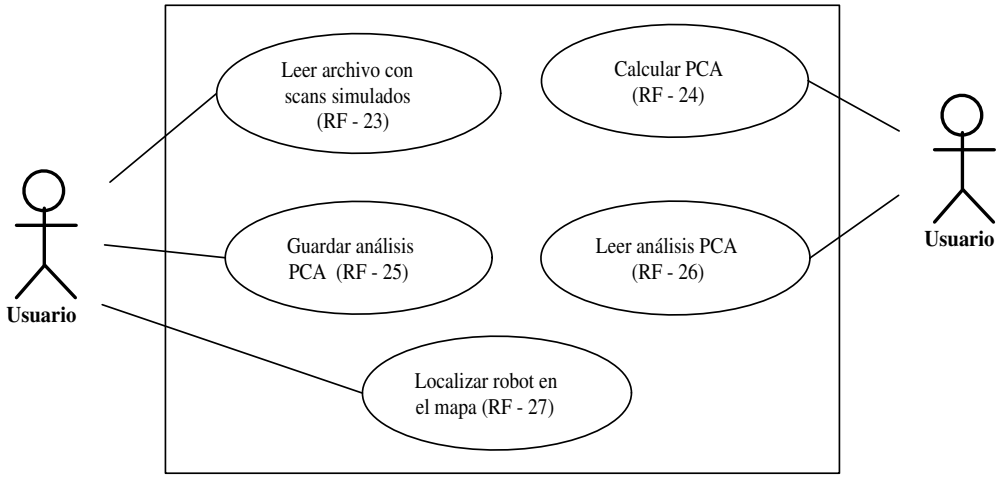

Figura 3.5: *Casos de uso para el análisis PCA*

3.2.3. Casos de uso del sistema

RF - 01	**Visualizar datos del láser**	
Descripción	El sistema deberá permitir al usuario en cualquier momento visualizar los datos obtenidos en tiempo real del dispositivo láser en el área de dibujo según se describe en el siguiente caso de uso:	
Secuencia	**Paso**	**Acción**
Normal	1	El usuario solicita al sistema comenzar el proceso para visualizar los datos leídos del dispositivo láser.
Normal	2	El sistema muestra los datos recibidos del dispositivo láser centrados en el origen de los ejes de coordenadas del área de dibujo.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Cada vez que lleguen nuevos datos del láser se eleminarán del área de dibujo los anteriores datos que estuvieran representados, de tal manera que sólo se pueda ver por pantalla lo que se encuentra viendo el láser en tiempo real.	

Tabla 3.4: *RF - 01*

RF - 02	**Ocultar datos del láser**	
Descripción	El sistema deberá permitir al usuario en cualquier momento hacer desaparecer la visualización de los datos en tiempo real del dispositivo láser según se describe en el siguiente caso de uso:	
Secuencia	**Paso**	**Acción**
Normal	1	El usuario solicita al sistema comenzar el proceso para ocultar los datos obtenidos en tiempo real del láser y representados en el área de dibujo.
Normal	2	El sistema hace desaparecer del área de dibujo los datos en tiempo real obtenidos del láser.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno	

Tabla 3.5: *RF - 02*

RF - 03	**Centrar el área de dibujo**	
Descripción	El sistema deberá permitir al usuario en cualquier momento centrar el área de dibujo de la aplicación según se describe en el siguiente caso de uso:	
Secuencia	**Paso**	**Acción**
Normal	1	El usuario solicita al sistema comenzar el proceso para centrar el área de dibujo.
Normal	2	El sistema muestra en la ventana de la aplicación el centro del área de dibujo.
Importancia	quedaría bien	
Urgencia	puede esperar	
Comentarios	El área de dibujo debe constar de unos ejes de coordenadas cartesianos que la dividan en cuatro cuadrantes. El área de dibujo puede ser más grande que la ventana en la que se muestre, por lo que debe permitirse la opción de centrar el origen de coordenadas (que coincide con el centro del área de dibujo).	

Tabla 3.6: *RF - 03*

RF - 04	**Zoom**	
Descripción	El sistema deberá permitir al usuario en cualquier momento modificar el valor del zoom de la imagen según se describe en el siguiente caso de uso:	
Secuencia	**Paso**	**Acción**
Normal	1	El usuario solicita al sistema modificar el valor del zoom del área de dibujo.
Normal	2	El sistema muestra el resultado de la modificación del valor del zoom.
Importancia	importante	
Urgencia	hay presión	
Comentarios	Ninguno	

Tabla 3.7: *RF - 04*

RF - 05	**Visualizar mapa del entorno**	
Descripción	El sistema deberá permitir al usuario en el momento a partir del cual se hayan alineado dos photoscans, visualizar el resultado de la alineación de dichos photoscans y de los posteriores que se logren alinear, formando progresivamente un mapa, según se describe en el siguiente caso de uso:	
Secuencia	**Paso**	**Acción**
Normal	1	El usuario confirma al sistema que la alineación de dos photoscans ha sido satisfactoria.
Normal	2	El sistema hace coindidir en la posición correspondiente en el mapa el último photoscan del que se dispone aumentando así el número de scans que componen el mapa.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno	

Tabla 3.8: *RF - 05*

RF - 06	**Mostrar rejilla**	
Descripción	El sistema deberá permitir al usuario en cualquier momento visualizar una rejilla que divida en porciones simétricas el área de dibujo según se describe en el siguiente caso de uso:	
Secuencia	**Paso**	**Acción**
Normal	1	El usuario solicita al sistema que visualize una rejilla a lo largo y ancho de todo el área de dibujo.
Normal	2	El sistema muestra la rejilla en el área de dibujo.
Importancia	quedaría bien	
Urgencia	puede esperar	
Comentarios	Ninguno	

Tabla 3.9: *RF - 06*

RF - 07	**Ocultar rejilla**	
Descripción	El sistema deberá permitir al usuario en cualquier momento hacer desaparecer la rejilla que divide en porciones simétricas el área de dibujo según se describe en el siguiente caso de uso:	
Secuencia	**Paso**	**Acción**
Normal	1	El usuario solicita al sistema comenzar el proceso de borrado de la rejilla del área de dibujo.
Normal	2	El sistema hace desaparecer la rejilla del área de dibujo.
Importancia	quedaría bien	
Urgencia	puede esperar	
Comentarios	Ninguno	

Tabla 3.10: *RF - 07*

RF - 08	**Modificar tamaño de la rejilla**		
Descripción	El sistema deberá permitir al usuario en cualquier momento modificar el tamaño de la rejilla del área de dibujo según se describe en el siguiente caso de uso:		
Secuencia Normal	**Paso**	**Acción**	
	1	El usuario solicita al sistema comenzar el proceso de modificación del tamaño de la rejilla del área de dibujo.	
	2	El sistema solicita el nuevo tamaño de la separación entre cada línea de la rejilla.	
	3	El sistema comprueba si la rejilla está visible o si está oculta.	
		3a	Si la rejilla está visible el sistema hará desaparecer la rejilla existente, creará una nueva con el tamaño indicado por el usuario, y la visualizará en el área de dibujo.
		3b	Si la rejilla está oculta, el sistema creará una nueva rejilla con el tamaño indicado por el usuario, pero no la visualizará en el área de dibujo.
Importancia	quedaría bien		
Urgencia	puede esperar		
Comentarios	Ninguno		

Tabla 3.11: *RF - 08*

RF - 09	**Tomar photoscan**	
Descripción	El sistema deberá permitir al usuario, siempre que se haya establecido previamente una conexión con el dispositivo láser, hacer una "fotografía" del scan en tiempo real (photoscan) según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema fotografiar el entorno que el escáner láser está percibiendo.
	2	El sistema almacena de manera fija los datos del entorno que está percibiendo el escáner láser y lo visualizará en el área de dibujo de la aplicación, centrado en el origen de coordenadas.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	El sistema debe permitir tomar dos photoscans para poder realizar posteriormente una correlación cruzada y calcular de este modo el giro y el desplazamiento que diferencia al segundo scan del primero. Cada phostoscan se debe representar con un color distinto para que sea más fácil para el usuario comprobar después cúal ha sido la bondad de la alineación.	

Tabla 3.12: *RF - 09*

RF - 10	**Borrar photoscan**	
Descripción	El sistema deberá permitir al usuario, siempre que se hayan fotografiado dos scans, borrar el segundo de los photoscans del área de dibujo según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema borrar el último photoscan tomado al decidir que no quería haberlo fotografiado.
	2	El sistema hace desaparecer del área de dibujo el segundo photoscan tomado.
Importancia	importante	
Urgencia	hay presión	
Comentarios	Ninguno	

Tabla 3.13: *RF - 10*

RF - 11	**Alinear photoscans**		
Descripción	El sistema deberá permitir al usuario, siempre que se hayan fotografiado dos scans, alinear el segundo photoscan con respecto al primero, haciendo coincidir las partes que tengan en común, según se describe en el siguiente caso de uso:		
Secuencia Normal	**Paso**	**Acción**	
	1	El usuario solicita al sistema comenzar el proceso de alineación de los dos photoscans que ha tomado.	
	2	El sistema solicita al usuario una estimación del desplazamiento y giro de la posición en la que se tomó el segundo photoscan con respecto a la posición en la que se tomó el primero.	
	3	El usuario introduce una estimación del desplazamiento en el eje x, el desplazamiento en el eje y, y el giro que piensa que se han efectuado.	
	4	El sistema calcula los desplazamientos y el giro del segundo photoscan con respecto al primero, basándose en la estimación que ha introducido el usuario, y le ofrece al usuario dichos resultados. Además redibuja el segundo photoscan en la posición que resulta de colocar proyectar el segundo photoscan en el sistema de coordenadas del primer photoscan.	
	5	El sistema pregunta al usuario si acepta la alineación de los scans.	
		5a	Si el usuario acepta, se añade el segundo photoscan al mapa global y se dibuja en el lugar apropiado en el área de dibujo.
		5b	Si el usuario no acepta, se elimina del área de dibujo el segundo photoscan.
Importancia	vital		
Urgencia	inmediatamente		
Comentarios	Ninguno		

Tabla 3.14: *RF - 11*

RF - 12	**Eliminar mapa**	
Descripción	El sistema deberá permitir al usuario, siempre que un mapa del entorno esté en proceso de construcción, elminar el mapa global y los photoscans del área de dibujo, según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema deshacerse del mapa que está construyendo.
	2	El sistema elimina de la memoria principal el mapa global y hace desaparecer del área de dibujo el mapa global y los photoscans, en caso de que estén presentes.
Importancia	importante	
Urgencia	hay presión	
Comentarios	Ninguno	

Tabla 3.15: *RF - 12*

RF - 13	**Guardar mapa como imagen**	
Descripción	El sistema deberá permitir al usuario, siempre que un mapa del entorno esté en proceso de construcción, guardar dicho mapa con el formato de imagen PNM, según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema comenzar el proceso de transformación del mapa en una imagen con el formato PNM.
	2	El sistema solicita un nombre y una ubicación para almacenar la imagen.
	3	El usuario introduce los datos pedidos.
	4	El sistema almacenará la imagen en un archivo con la ubicación y el nombre asignados.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	El formato de la imagen debe ser en concreto PNM tipo 5 (también conocido como PGM con escritura de datos en formato binario). Esto es así porque el simulador Stage necesita este tipo de imágenes para usarlas como mapa.	

Tabla 3.16: *RF - 13*

RF - 14	**Suscribirse al dispositivo láser**	
Descripción	El sistema deberá permitir al usuario en cualquier momento comenzar a obtener datos en tiempo real del dispositivo láser según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema comenzar el proceso de obtención de datos en tiempo real del dispositivo láser.
	2 3	El sistema inicia el proceso de obtención de datos del láser en tiempo real. El sistema visualizará en el área de dibujo, centrado en el origen de coordenadas, la porción del entorno que el dispositivo láser está obteniendo en un instante dado, cada vez que estén disponibles nuevos datos.
Excepciones	**Paso**	**Acción**
	2	Si el dispositivo láser no está conectado debidamente, el sistema informará al usuario de esto y cancelará la operación.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno	

Tabla 3.17: *RF - 14*

RF - 15	**Cancelar suscripción al láser**	
Descripción	El sistema deberá permitir al usuario en cualquier momento detener la obtención de datos en tiempo real del dispositivo láser según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema detener el proceso de obtención de datos en tiempo real del dispositivo láser.
	2	El sistema detiene el proceso de obtención de datos del láser en tiempo real.
Excepciones	**Paso**	**Acción**
	2	Si el dispositivo láser no está conectado debidamente, el sistema informará al usuario de esto y cancelará la operación.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno	

Tabla 3.18: *RF - 15*

RF - 16	**Suscribirse al dispositivo *position***	
Descripción	El sistema deberá permitir al usuario en cualquier establecer una conexión con un dispositivo *position* de tal manera que posteriormente se puedan enviar comandos de velocidad a los motores, según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema comenzar el proceso de conexión con el dispositivo *position*.
	2	El sistema establece la comunicación con el dispositivo *position* quedando abierta una vía de comunicación con él.
Excepciones	**Paso**	**Acción**
	2	Si el dispositivo *position* no está conectado debidamente, el sistema informará al usuario de esto y cancelará la operación.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno	

Tabla 3.19: *RF - 16*

RF - 17	**Cancelar suscripción a *position***	
Descripción	El sistema deberá permitir al usuario en cualquier momento cortar la vía de comunicación con el dispositivo positión al que se encuentre conectado, según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema cerrar el canal de comunicación con el dispositivo *position*.
	2	El sistema cancela la suscripción al dispositivo *position* e imposibilita el envío de comandos a los motores del robot (simulado o real).
Excepciones	**Paso**	**Acción**
	2	Si el dispositivo *position* no está conectado debidamente, el sistema informará al usuario de esto y cancelará la operación.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno	

Tabla 3.20: *RF - 17*

RF - 18	**Mover robot real**	
Descripción	El sistema deberá permitir al usuario en cualquier momento decidir si el robot que va a controlar, enviando comandos a sus motores, va a ser un robot real (no simulado por Stage) según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema seleccionar como dispositivo *position* el robot físico real.
	2	El sistema selecciona como dispositivo *position* el robot real, y establece los ajustes necesarios para controlar los motores del robot real.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno	

Tabla 3.21: *RF - 18*

RF - 19	**Mover robot simulado**	
Descripción	El sistema deberá permitir al usuario en cualquier momento decidir si el robot que va a controlar, enviando comandos a sus motores, va a ser un robot simulado por Stage según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema seleccionar como dispositivo *position* uno simulado por el simulador Stage.
	2	El sistema selecciona como dispositivo *position* el simulado por stage, y establece los ajustes necesarios para controlar los motores del dispositivo simulado.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno	

Tabla 3.22: *RF - 19*

RF - 20	**Modificar velocidad**	
Descripción	El sistema deberá permitir al usuario, siempre que este suscrito a un dispositivo *position*, seleccionar un porcentaje de la velocidad máxima que se le va a enviar a los motores del dispositivo según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema modificar el porcentaje de la velocidad máxima que se le puede enviar a los motores del robot.
	2	El sistema solicita el nuevo porcentaje.
	3	El usuario introduce el dato pedido.
	4	El sistema envia un comando a los motores del robot indicando la nueva velocidad.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	La velocidad máxima que se puede enviar al robot real o simulado se mantendrá fija. Para variar la velocidad del robot será necesario utilizar un porcentaje de esta velocidad máxima.	

Tabla 3.23: *RF - 20*

RF - 21	**Controlar sentido**	
Descripción	El sistema deberá permitir al usuario, siempre que este suscrito a un dispositivo *position*, controlar el sentido de la marcha del robot, enviando a los motores del dispositivo los comandos adecuados según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema modificar el sentido de la marcha.
	2	El sistema solicita el nuevo sentido.
	3	El usuario introduce el dato pedido.
	4	El sistema envia un comando a los motores del robot de tal manera que el sentido de la marcha del robot sea el indicado por el usuario.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Se deberá poder permitir que el robot avance hacia delante y hacia atrás).	

Tabla 3.24: *RF - 21*

RF - 22	**Controlar giro**	
Descripción	El sistema deberá permitir al usuario, siempre que este suscrito a un dispositivo *position*, controlar el giro que debe realizar el robot, enviando a los motores del dispositivo los comandos adecuados según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema modificar el giro que está realizando el robot.
	2	El sistema solicita el nuevo valor para el giro.
	3	El usuario introduce el dato pedido.
	4	El sistema envia un comando a los motores del robot de tal manera que robot realice el giro indicado por el usuario.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno.	

Tabla 3.25: *RF - 22*

RF - 23	**Leer archivo con scans simulados**	
Descripción	El sistema deberá permitir al usuario en cualquier momento leer un archivo que tenga almacenados una serie de datos correspondientes a scans simulados en un entorno del que se dispone un mapa según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema comenzar el proceso de apertura de un archivo que contenga scans simulados.
	2	El sistema solicita al usuario la ubicación y el nombre del archivo.
	3	El usuario introduce los datos pedidos.
	4	El sistema lee los datos del archivo y los almacena en memoria principal.
Excepciones	**Paso**	**Acción**
	3	En el caso de que el archivo no tenga el formato apropiado se comunicará la situación al usuario y se cancelará la operación.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Estos datos servirán para realizar el análisis PCA.	

Tabla 3.26: *RF - 23*

RF - 24	**Calcular PCA**	
Descripción	El sistema deberá permitir al usuario, siempre que se haya leido un archivo con un conjunto de datos de scans simulados, realizar un análisis de componentes principales a dichos datos según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema comenzar el proceso para realizar un análisis de componentes principales al conjunto de datos de scans que se encuentra en memoria principal.
	2	El sistema realiza el análisis de componentes principales y lo almacena en memoria principal.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno.	

Tabla 3.27: *RF - 24*

RF - 25	**Guardar archivo con análisis PCA**	
Descripción	El sistema deberá permitir al usuario, siempre que se haya realizado un análisis de componentes principales, almacenar los datos resultantes en un archivo según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema comenzar el proceso para almacenar en disco el análisis PCA que se encuentre en memoria principal.
	2	El sistema solicita al usuario la ubicación y el nombre del archivo donde desea almacenar los datos.
	3	El usuario introduce los datos pedidos.
	4	El sistema almacena los datos en un archivo con la ubicación y el nombre indicados.
Importancia	importante	
Urgencia	hay presión	
Comentarios	Ninguno.	

Tabla 3.28: *RF - 25*

RF - 26	**Leer archivo con análisis PCA**	
Descripción	El sistema deberá permitir al usuario en cualquier momento leer un archivo que tenga almacenado un análisis de componentes principales de un archivo de scans simulados según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema comenzar el proceso de apertura de un archivo que contenga un análisis de componentes principales.
	2	El sistema solicita al usuario la ubicación y el nombre del archivo.
	3	El usuario introduce los datos pedidos.
	4	El sistema lee los datos del archivo y los almacena en memoria principal.
Excepciones	**Paso**	**Acción**
	3	En el caso de que el archivo no tenga el formato apropiado se comunicará la situación al usuario y se cancelará la operación.
Importancia	importante	
Urgencia	hay presión	
Comentarios	Ninguno.	

Tabla 3.29: *RF - 26*

RF - 27	**Localizar robot en el mapa**	
Descripción	El sistema deberá permitir al usuario, siempre que esté conectado al dispositivo láser y siempre que exista en memoria principal un análisis de componentes principales, utilizar el scan actual para decidir en que lugar del entorno se encuentra situado el láser, utilizando el análisis PCA, según se describe en el siguiente caso de uso:	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita al sistema comenzar el proceso para deducir la ubicación actual del dispositivo láser.
	2	El utiliza los datos que el dispositivo láser está percibiendo del entorno, calcula la posición más probable, y ofrece los resultados al usuario.
Importancia	vital	
Urgencia	inmediatamente	
Comentarios	Ninguno.	

Tabla 3.30: *RF - 27*

3.3. Requisitos no funcionales

RNF - 01	**Entorno de trabajo**
Descripción	El sistema deberá funcionar en entornos GNU/Linux que dispongan de la versión 2.x de GNOME. Se aconseja el uso de una máquina potente que permita trabajar con el entorno GNOME 2.x de forma fluida. La velocidad del procesador así como la memoria principal de que se disponga será bastante importante puesto que el análisis de componentes principales requiere bastante tiempo de procesador y bastante memoria RAM, dependiendo del tamaño de los datos que se procesen.
Importancia	vital
Urgencia	inmediatamente
Comentarios	Ninguno.

Tabla 3.31: *RNF - 01*

RNF - 02	**Implementación**
Descripción	El sistema deberá estar implementado en lenguaje C, utilizando todas las bibliotecas de funciones que linux, el proyecto GNOME, y otras librerías matemáticas como BLAS estén a disposición del desarrollador.
Importancia	vital
Urgencia	inmediatamente
Comentarios	Ninguno.

Tabla 3.32: *RNF - 02*

RNF - 03	**Usabilidad**
Descripción	El sistema deberá poseer una interfaz amigable y de fácil manejo y de fácil manejo dentro de lo posible.
Importancia	vital
Urgencia	inmediatamente
Comentarios	Para utilizar la interfaz de la aplicación no serán necesarios conocimientos avanzados de informática, pero sí se necesitará comprender bien los ciertos aspectos teóricos del desarrollo, ya que esta aplicación está desarrollada para el uso interno del departamento, no para el público en general.

Tabla 3.33: *RNF - 03*

Anexo 2B

DOCUMENTO DE ANÁLISIS DE REQUISITOS

Proyecto LasMap

Documento De Análisis De Requisitos

Versión 1.0

Julio 2005

Realizado Por	Tutores
Carlos Fernández Caramés	*Vidal Moreno Rodilla* *Belén Curto Diego*

Para

Departamento de Informática y Automática

Universidad de Salamanca

Índice general

Índice de figuras

1 Introducción

Una vez expuesto el documento de requisitos del sistema, cuyo cometido consiste en ser un elemento contractual entre la parte cliente y la parte de ingenieros del software, se pasa a realizar un pequeño análisis más exhaustivo de lo desarrollado en dicho documento y que se plasma en este anexo.

La función principal de este anexo consiste en realizar un modelo del sistema, basándose para ello en los requisitos descritos en el anexo anterior y utilizando una serie de métodos gráficos y textuales que se expondrán a continuación. Para realizar el modelado funcional del sistema, esto es, el modo en que el sistema procesa la información que recibe, se usarán los diagramas de flujo de datos (DFD) ya que es una herramienta de modelado que permite observar el sistema como una red de procesos funcionales interconectados unos con otros por medio de flujos y almacenes de datos. El diccionario de datos (DD) y las especificaciones de proceso, incrementarán de modo textual

el grado de detalle ofrecido por los diagramas de flujo de datos.

Puesto que el sistema que estamos tratando en parte es un sistema de control, ya que se comunica en tiempo real con un escáner láser leyendo datos de manera continua, será necesario utilizar la notación ampliada de *Ward y Mellor* [6], para sistemas de tiempo real. Esto nos permitirá observar el comportamiento del sistema bajo una perspectiva de control.

Para afrontar la tarea de modelar el sistema, se han seguido las directrices del método de *Yourdon* [5], ya que permite realizar un enfoque de modelado medio, distinto del enfoque clásico, ni totalmente descendente, ni totalmente ascendente. Básicamente consiste en realizar un modelo esencial del sistema, describiendo lo que el sistema debe hacer para satisfacer los requisitos del usuario diciendo lo mínimo posible (preferiblemente nada) acerca de cómo se implementará. El modelo esencial se divide a su vez en dos componentes principales: el modelo ambiental y el modelo de comportamiento, detallados en los siguientes apartados.

2

Modelo ambiental

El modelo ambiental sirve para definir las interfaces entre el sistema y el mundo exterior (denominado ambiente), y para identificar los eventos que ocurren en el exterior y a los cuales el sistema debe responder. Está compuesto por la declaración de propósitos, el diagrama de contexto, y la lista de acontecimientos, cada uno de los cuales se explica a continuación.

2.1. Declaración de propósitos

El propósito del sistema LASMAP es permitir contruir un mapa en dos dimensiones de un determinado entorno mediante la utilización de un escáner láser medidor de distancias. La lectura de los datos de láser se hará a través del servidor Player. Una vez construido el mapa, el sistema permitirá realizar un análisis de componentes principales sobre un conjunto de datos de láser

simulados en el entorno representado en el mapa. Finalmente, utilizando el análisis PCA y el entorno que el escáner láser está percibiendo en un determinado momento, el sistema ofrecerá la posibilidad de ofrecer la posición y orientación más probable en la que se encuentra el escáner láser dentro del mapa construido.

2.2. Diagrama de contexto

El diagrama de contexto es un caso especial de los diagramas de flujo de datos (DFD) en el cual un sólo proceso representa a todo el sistema.

2.3. Lista de acontecimientos

Se trata de una lista narrativa de los estímulos que ocurren en el exterior y a los cuales el sistema debe responder. En el caso de este proyecto, son los que siguen:

1. Modificar el zoom del área de dibujo.
2. Modificar la rejilla del área de dibujo.
3. Conectarse a un dispositivo.
4. Desconectarse de un dispositivo.
5. Construir un mapa, alineando scans.
6. Transformar un mapa en imagen en formato PNM.
7. Realizar un análisis de componentes principales.
8. Solicitar localizar el láser en el entorno.

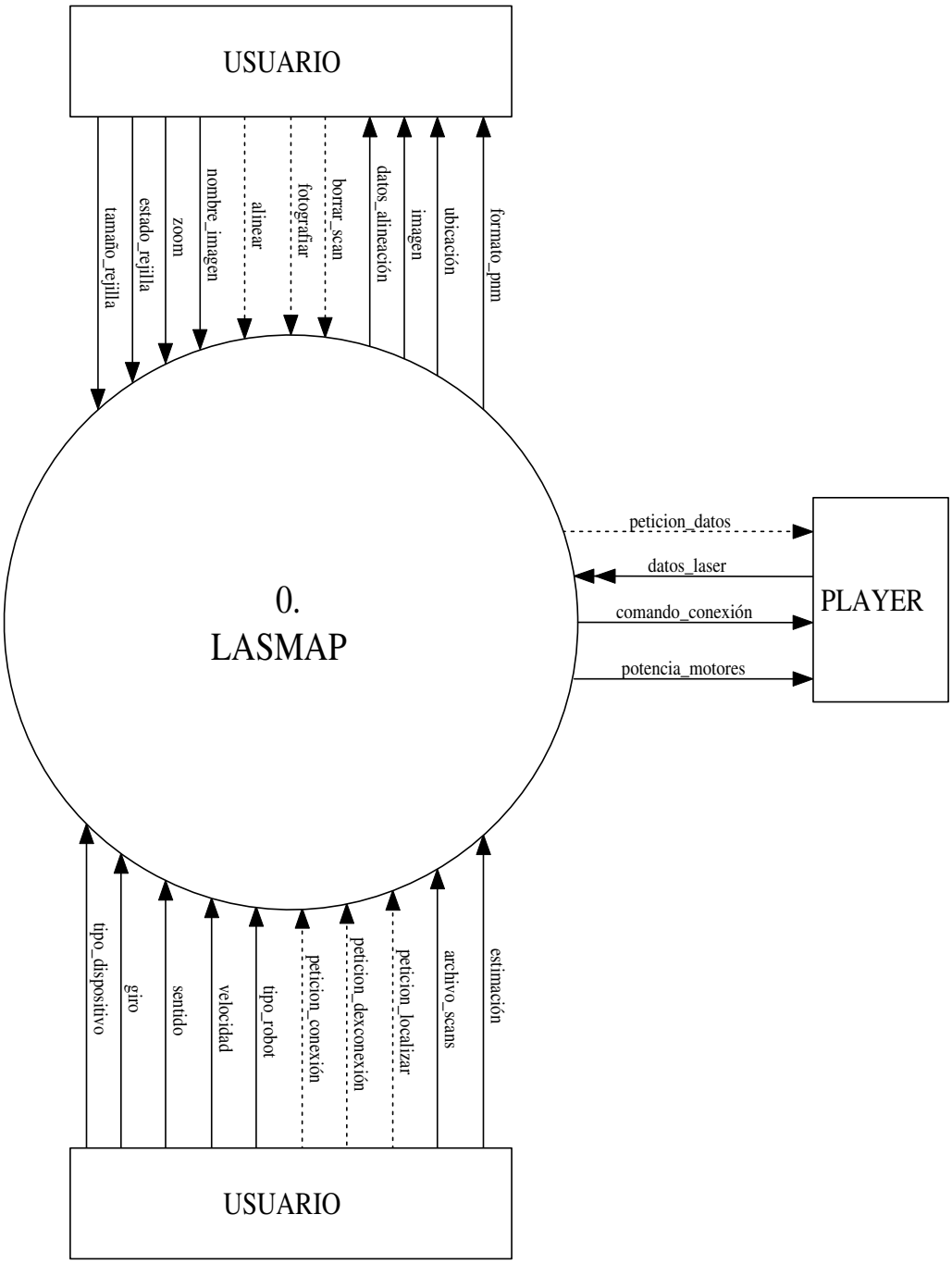

Figura 2.1: *Diagrama de contexto*

3

Modelo de comportamiento

El modelo de comportamiento describe el comportamiento que debe tener el sistema para interactuar satisfactoriamente con el ambiente que lo rodea. Para representar este comportamiento, se presenta a continuación el DFD de sistema, junto con una nivelación descendiente de los procesos que lo componen para lograr un mayor nivel de detalle.

3.1. DFD de sistema

El DFD de sistema es una descomposición del DFD de contexto, que sirve para mostrar el funcionamiento general del sisetma en cuestión, ofreciendo una visión general de las funciones del sistema, así como las relaciones entre ellas.

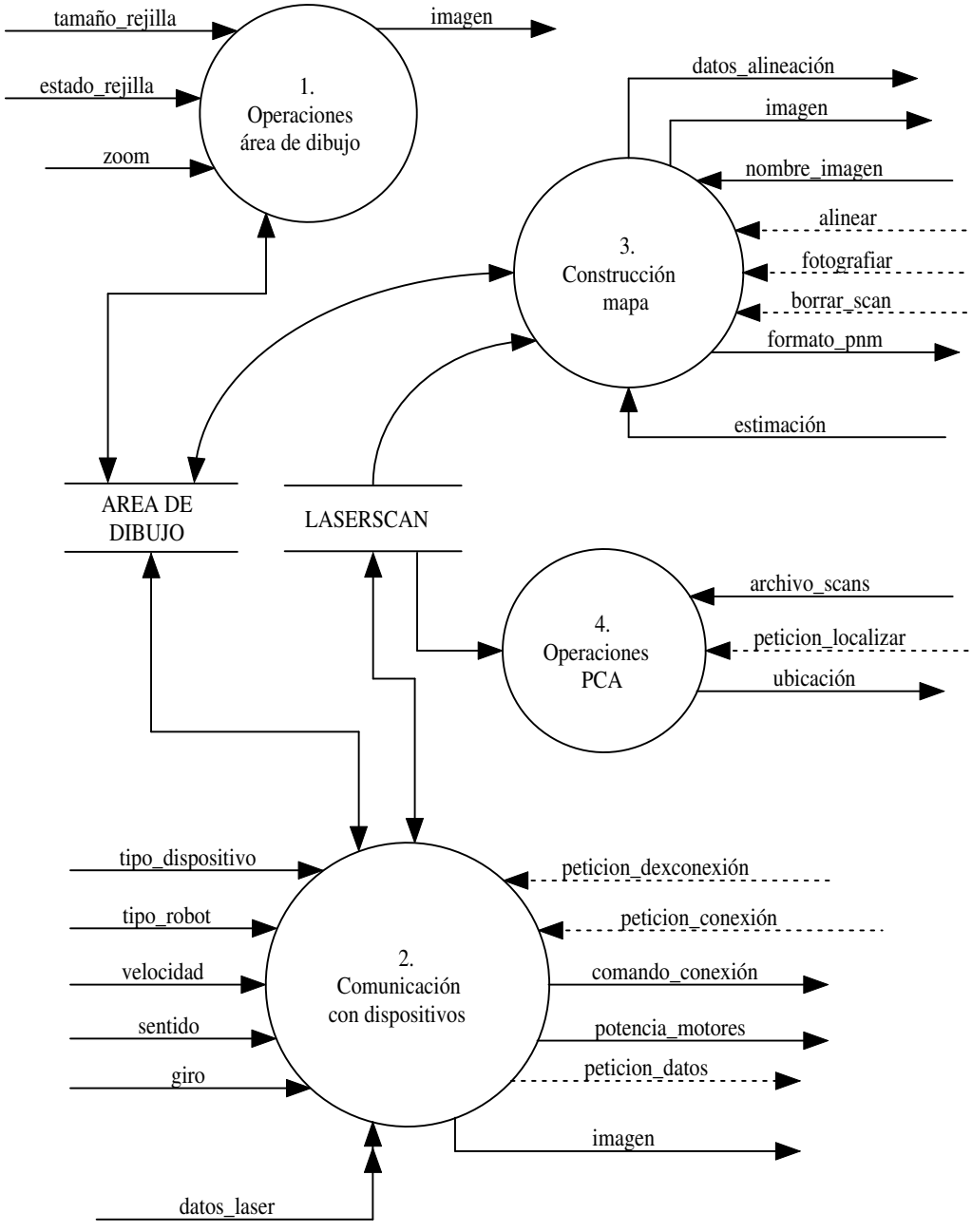

Figura 3.1: *Diagrama de sistema*

3.2. Nivelación descendente del DFD de sistema

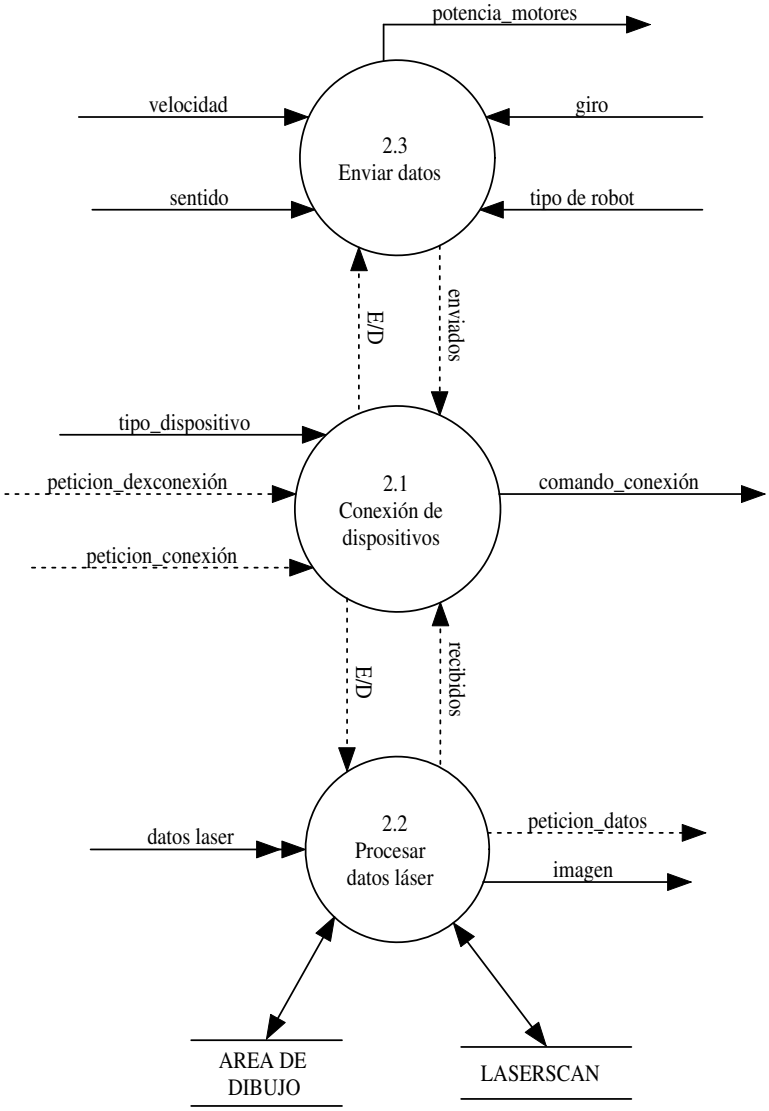

Figura 3.2: *Explosión del proceso 2. Comunicación con dispositivos.*

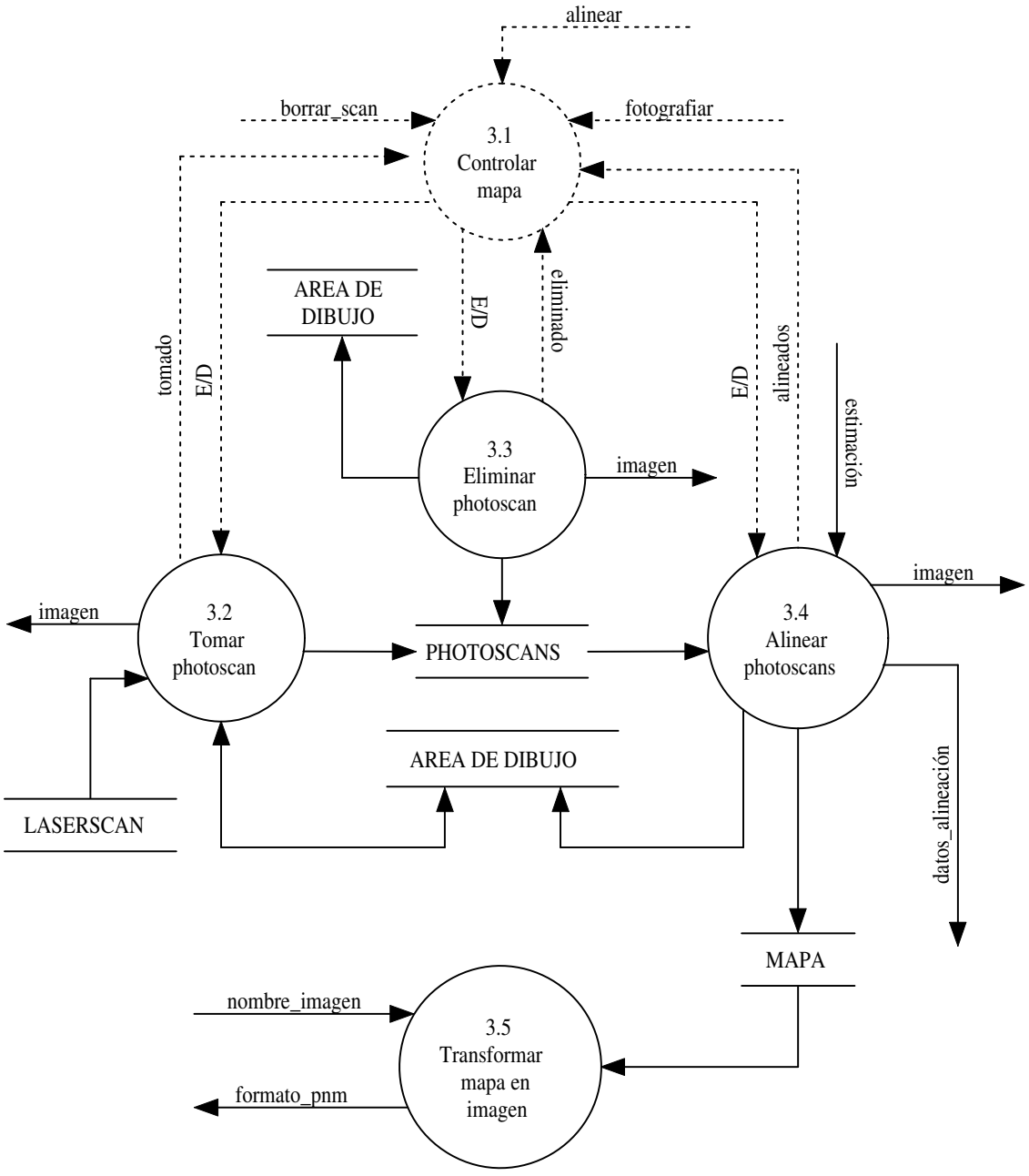

Figura 3.3: *Explosión del proceso 3. Construcción mapa.*

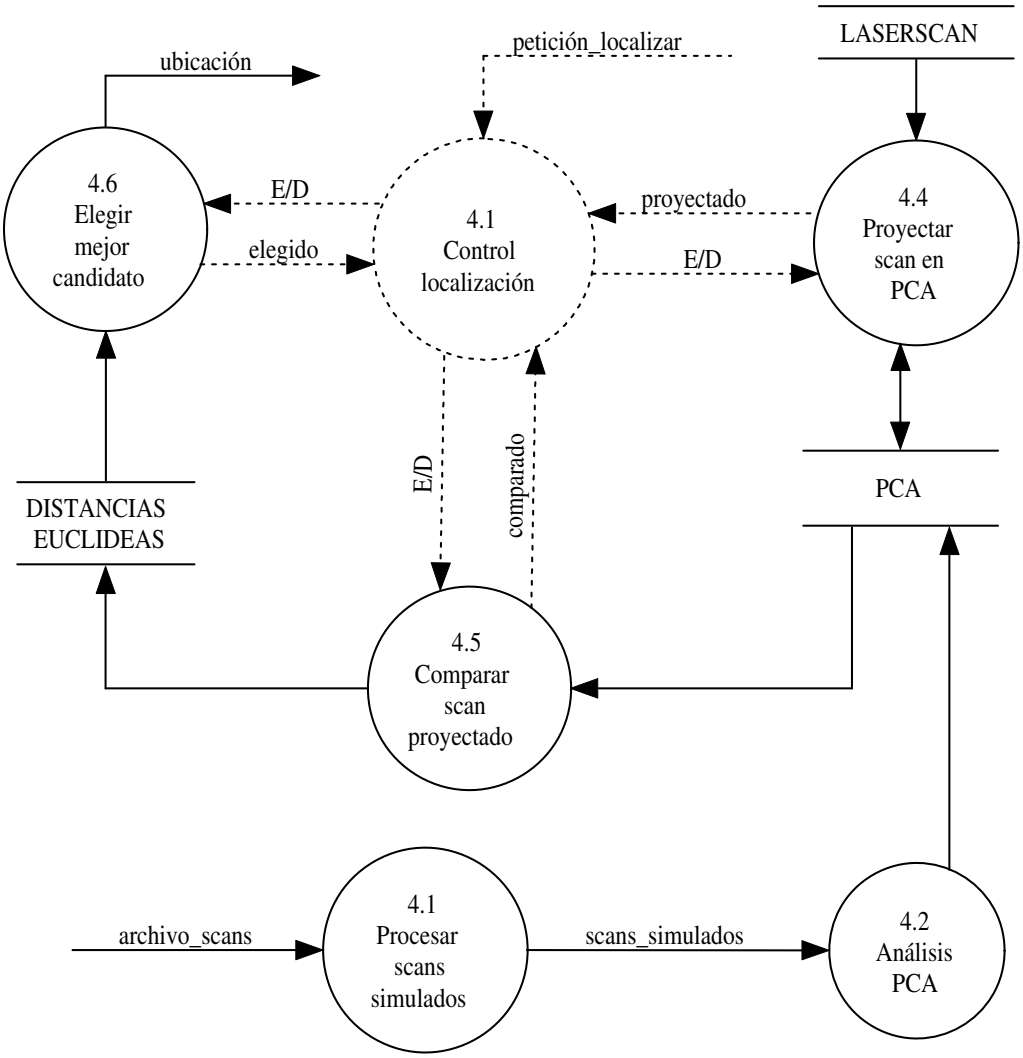

Figura 3.4: *Explosión del proceso 4. Operaciones PCA.*

3.3. Refinamiento final del DFD

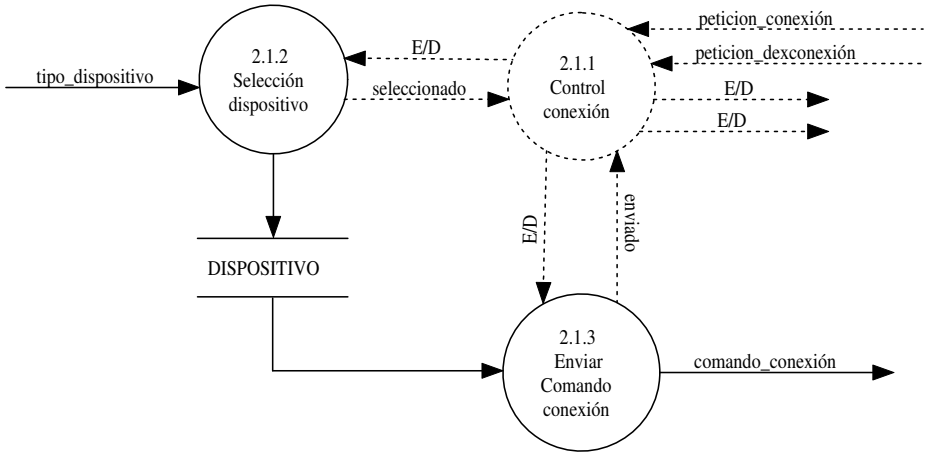

Figura 3.5: *Explosión del proceso 2.1. Conexión de dispositivos.*

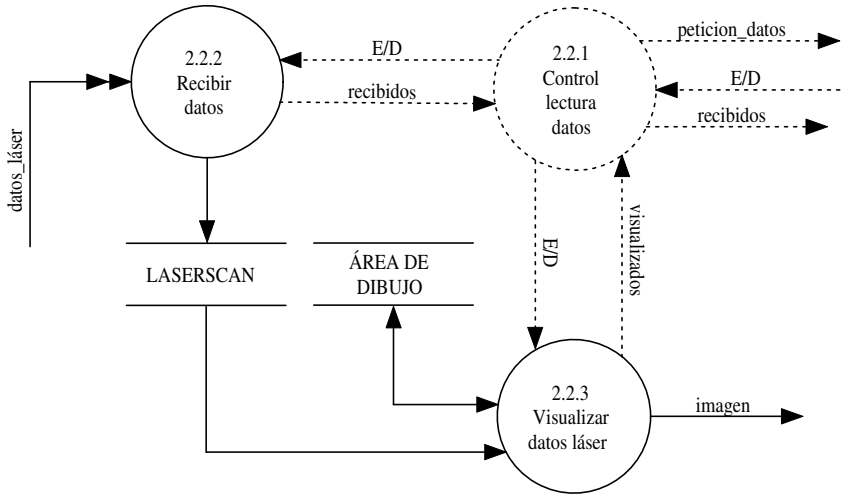

Figura 3.6: *Explosión del proceso 2.2. Recibir datos.*

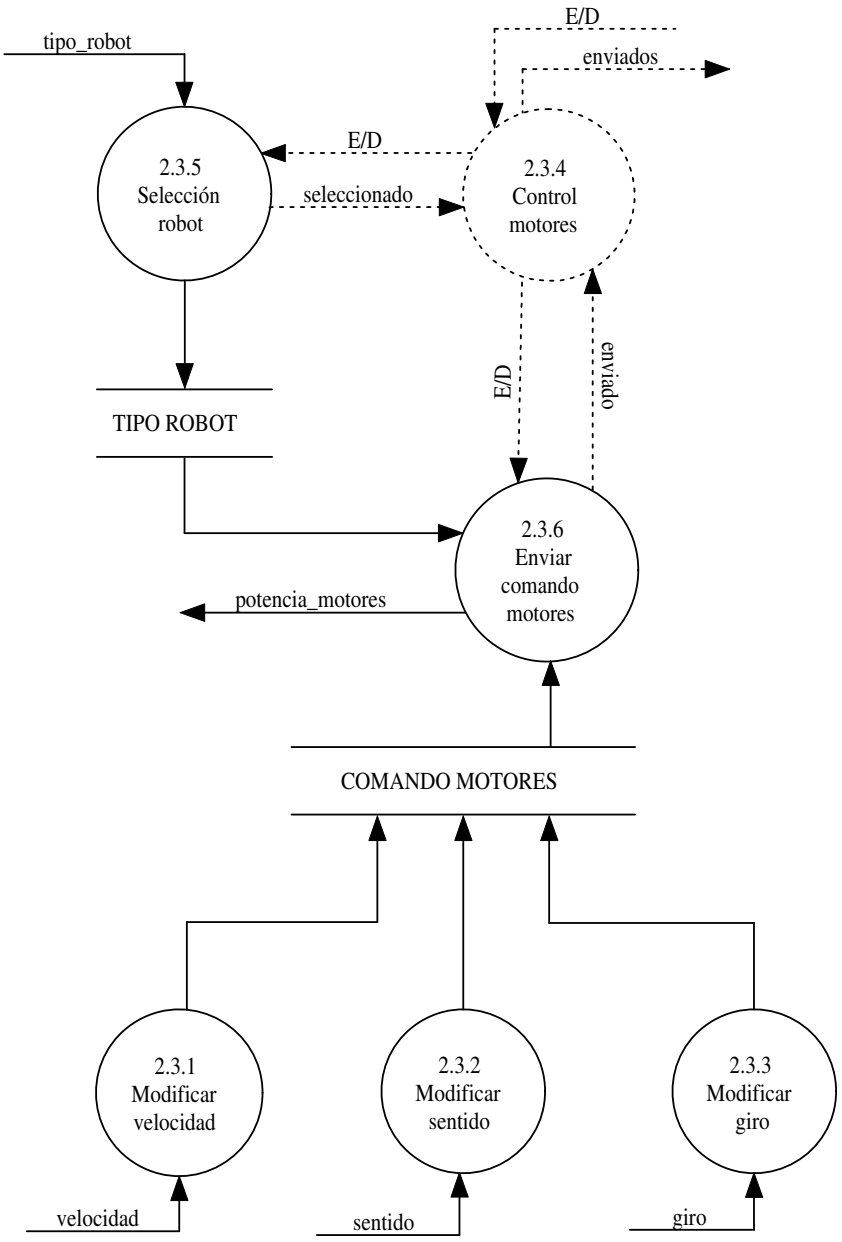

Figura 3.7: *Explosión del proceso 2.3. Enviar datos.*

3.4. Diccionario de datos

En este apartado se presenta una lista organizada de todos los datos utilizados por el sistema con definiciones precisas y rigurosas de los almacenes, flujos y elementos de datos. Utilizando un diccionario de datos se presenta una notación más concisa y compacta que definiendo los datos de forma narrativa.

3.4.1. Almacenes

ÁREA DE DIBUJO = * buffer en el que se almacena la imagen que está siendo mostrada en la pantalla en un momento dado *
COMANDO MOTORES = * petición de movimiento de los motores que será enviada a *Player* * = 2{ velocidad }2
DISPOSITIVO = * tipo de dispositivo seleccionado * = [láser | position]
DISTANCIAS EUCLÍDEAS = * distancias euclídeas entre un scan y sus posibles coincidencias * = { distancia }
LASERSCAN = * datos del escáner láser * = { PUNTO }
MAPA = { LASERSCAN }
PCA = * análisis de componentes principales * = pca_dim + num_eigens + num_muestras + vector_media + vector_caracteristico + datos_proyectados
PHOTOSCANS = 2{ PHOTOSCAN }2
PHOTOSCAN = @num_photoscan + { PUNTO }
PUNTO = @num_punto + coord_x + coord_y
TIPO ROBOT = [real | simulado]

3.4.2. Flujos de datos

alinear = * flujo discreto de control que indica que el usuario desea alinear dos photoscans *
alineados = * flujo discreto de control que indica que dos photoscans han sido alineados *

archivo_scans = * ruta absoluta de un archivo en memoria secundaria que contiene los scans simulados *
borrar_scan = * flujo discreto de control que indica que el usuario desea borrar un photoscan *
comando_conexión = * comando que se enviará a *Player* indicando el tipo de dispositivo a conectar o desconectar *
datos_alineación = coord_x + coord_y + orientacion
datos_láser = * paquete de datos recibido de *Player*, con los datos que el láser ha leído * **E/D** = * flujo discreto de control que indica que un proceso de control ha activado o desactivado a otro proceso *(Enable / Disable)**
eliminado = * flujo discreto de control que indica que un photoscan ha sido elminado *
enviado = * flujo discreto de control que indica que se han enviado las velocidades a los motores *
estado_rejilla = { visible | oculta }
estimación = coord_x + coord_y + orientación
formato_pnm = * conjunto de puntos que forman el mapa transformados en el formato de imagen PNM *
fotografiar = * flujo discreto de control que indica que el usuario desea tomar una instantánea de los datos del láser *
giro = * giro del robot * = { carácter_numérico }
imagen = * visualización del área de dibujo en la pantalla *
nombre_imagen = * ruta absoluta de la imagen que se va a guardar en memoria secundaria *
petición_conexión = * flujo discreto de control que indica que el usuario desea conectar un dispositivo *
peticion_datos = * flujo discreto de control para pedir datos a *Player* *
petición_desconexión = * flujo discreto de control que indica que el usuario desea desconectar un dispositivo *
petición_localizar = * flujo discreto de control que indica que el usuario desea conocer la posición actual del escáner láser en el entorno *

potencia_motores = 2{ velocidad }2
proyectado = * flujo discreto de control que indica que el scan actual ha sido proyectado en el espacio vectorial *
recibido = * flujo discreto de control que indica que se han recibido los datos láser *
seleccionado = * flujo discreto de control que indica que el usuario ha realizado una selección *
sentido = * sentido de la marcha del robot * = [adelante | atrás]
scans_simulados = * conjunto de scans tomados por el láser, en formato polar * = { scan_polar }
tamaño_rejilla = { carácter_numérico }
tomado = * flujo discreto de control que indica que se ha tomado un photoscan *
tipo_dispositivo = * tipo de dispositivo que el usuario ha seleccionado * = [láser | position]
tipo_robot = [real | simulado]
ubicación = coord_x + coord_y + orientacion
velocidad = * velocidad del robot * = { carácter_numérico }
visualizados = * flujo discreto de control que indica que se han visualizado los datos láser *
zoom = * porcentaje de zoom de la imagen * = { carácter_numérico }

3.4.3. Elementos de datos

carácter_numérico = [0-9]
elemento_matriz = { carácter_numérico }
coord_x = { carácter_numérico }
coord_y = { carácter_numérico }
datos_proyectados = { elemento_matriz }
distancia = { carácter_numérico }
num_eigens = * número de vectores propios que han sido utilizados * = { carácter_numérico }

num_muestras = * número de scans simulados * = { carácter_numérico }
num_photoscan = { carácter_numérico }
num_photoscan = { carácter_numérico }
num_punto = { carácter_numérico }
orientación = * angulo en el que se encuentra orientado el escáner láser * = { carácter_numérico }
pca_dim = * dimension de la matriz PCA * = { carácter_numérico }
scan_polar = * vector en el que cada uno de los elementos representa el valor del radio en coordenadas polares * = { elemento_matriz }
velocidad = * velocidad de un motor * = { carácter_numérico }
vector_media = { elemento_matriz }
vector_característico = { elemento_matriz }

Anexo 3

ESPECIFICACIÓN DE DISEÑO

Proyecto LasMap

Especificación De Diseño

Versión 1.0

Julio 2005

Realizado Por	Tutores
Carlos Fernández Caramés	*Vidal Moreno Rodilla* *Belén Curto Diego*

Para

Departamento de Informática y Automática

Universidad de Salamanca

Índice general

Índice de figuras

Índice de tablas

1 Introducción

En este anexo se realizará el diseño del sistema *LasMap*, siguiendo lo obtenido en la anterior fase de análisis y recogido en la *Especificación de Requisitos del Software*.

En el siguiente apartado, se desarrollará la estructura modular de la aplicación, utilizando la técnica gráfica conocida como *Diagramas de Estructuras*, junto con una leyenda de cada uno de los parámetros mostrados en los diagramas y las tablas de interfaz, en las que se muestra el uso de los parámetros que recibe cada módulo.

En el apartado 3, *Diseño de datos*, se hará un breve comentario sobre los datos que se deben almacenar en memoria secundaria para su uso posterior.

Finalmente, en el apartado 4, *Diseño de la interfaz*, se muestran todos los elementos empleados en el desarrollo de la interfaz del sistema con el usuario que se han utilizado en el proyecto.

2

Diseño arquitectónico

El objetivo del diseño arquitectónico es desarrollar la estructura modular del programa, representando las relaciones de control entre los módulos y definiendo las interfaces que facilitan el flujo de datos a lo largo del programa.

Para realizar esta tarea se van a emplear los Diagramas de Estructuras. Esta herramienta de diseño estructurado toma como base los Diagramas de Flujo de Datos (DFD) obtenidos en la fase de análisis previa.

2.1. Diagramas de estructuras

A continuación se presenta el diagrama de estructuras general de la aplicación, seguido de los diagrámas de cada módulo.

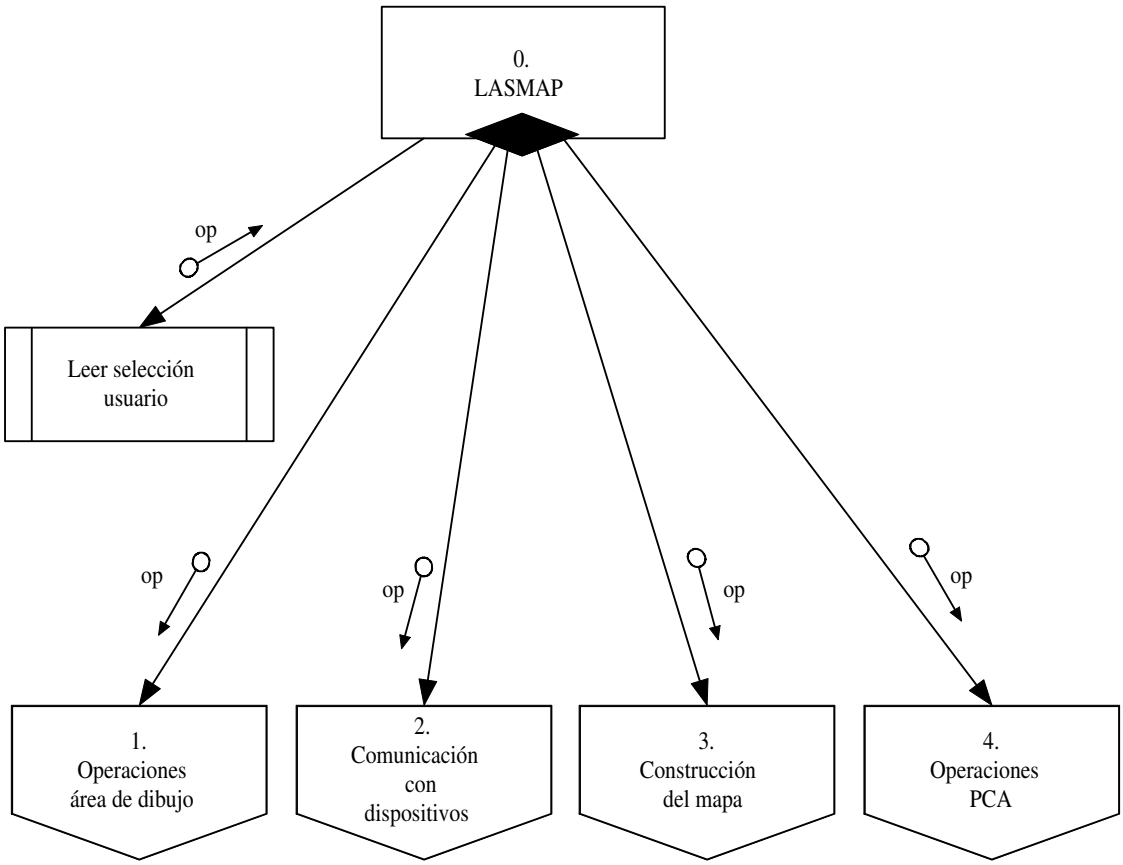

Figura 2.1: *Diagrama de estructuras: módulo inicial*

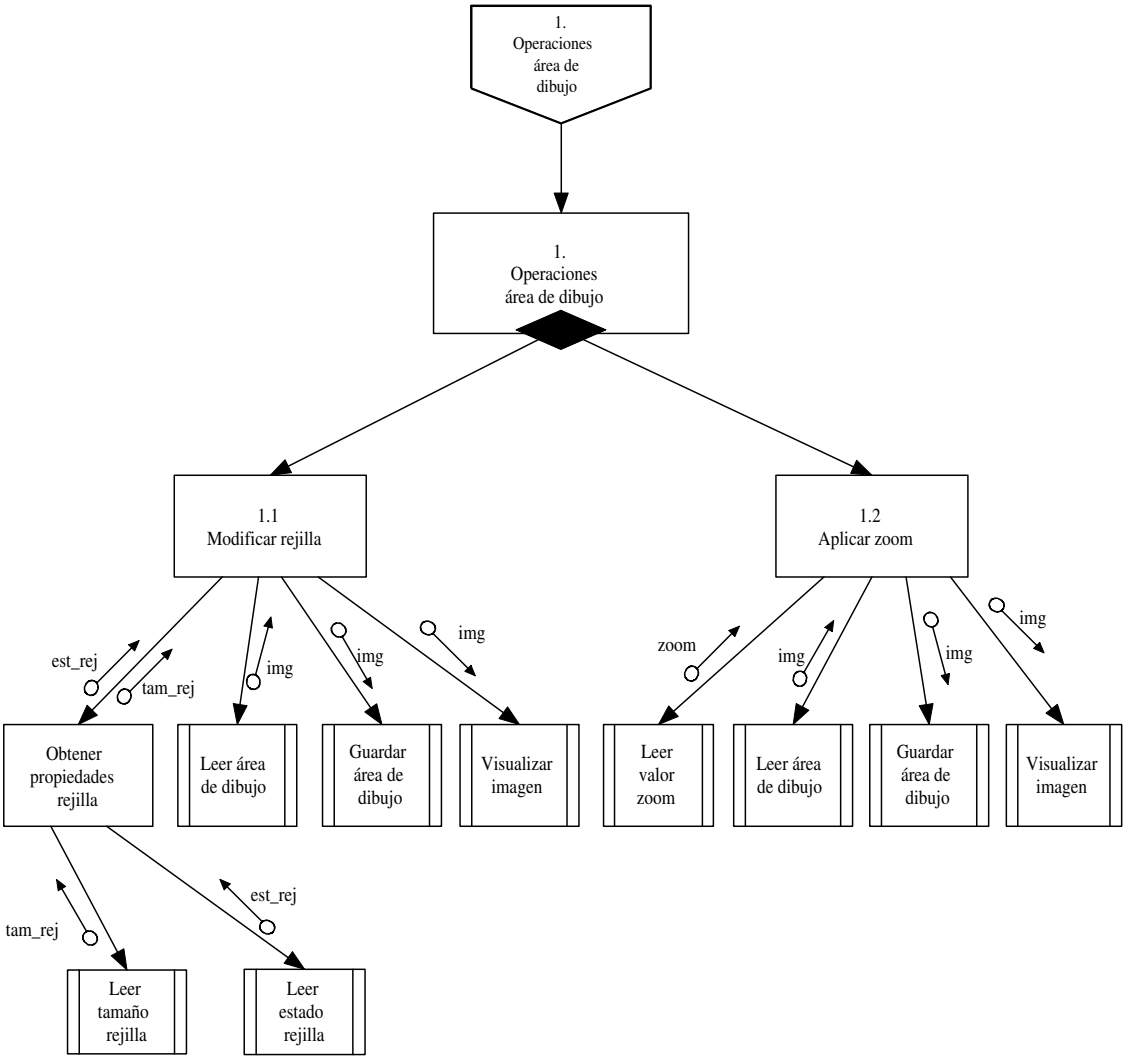

Figura 2.2: *Diagrama de estructuras del módulo "Operaciones área de dibujo".*

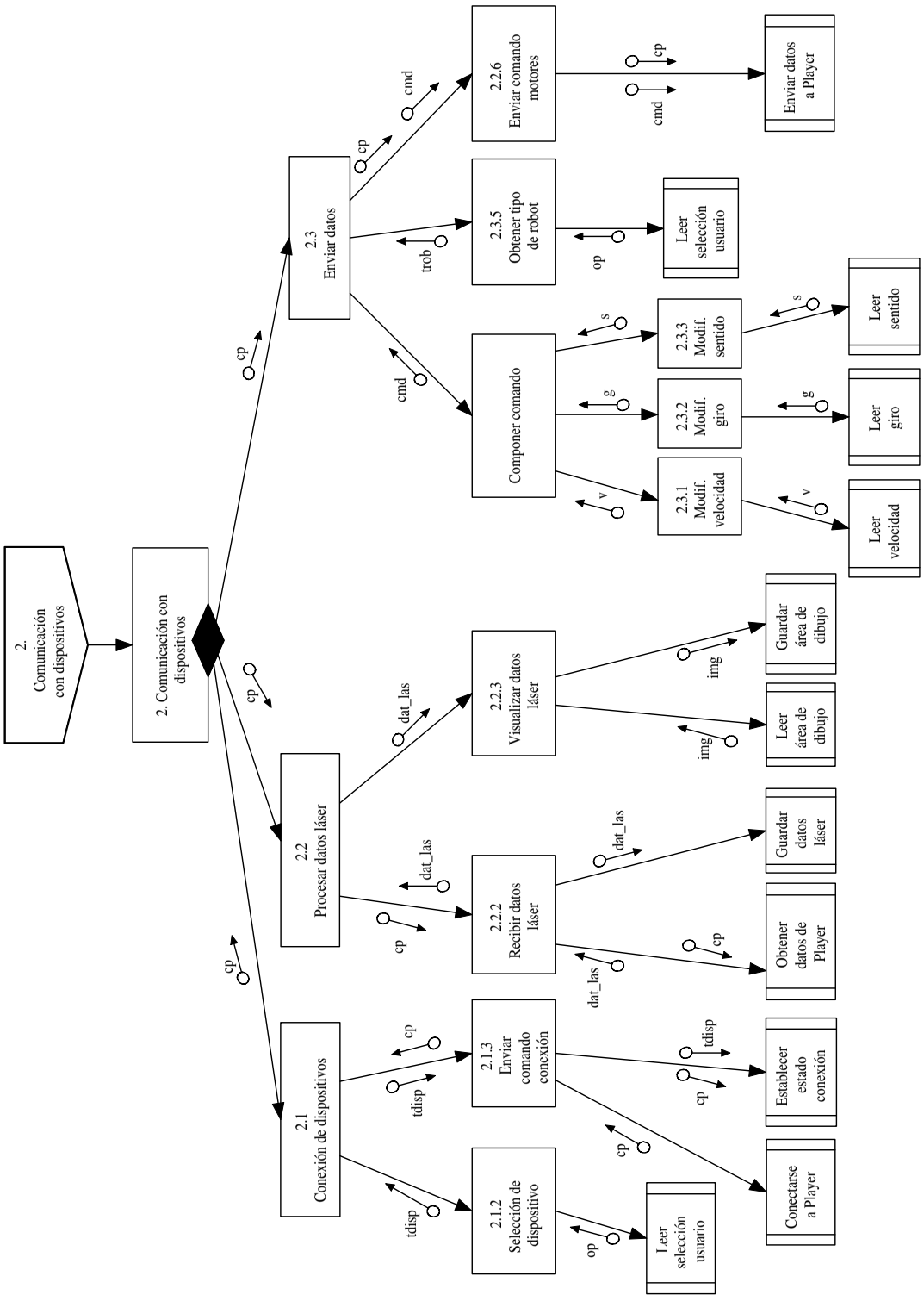

Figura 2.3: *Diagrama de estructuras del módulo "Comunicación con dispositivos".*

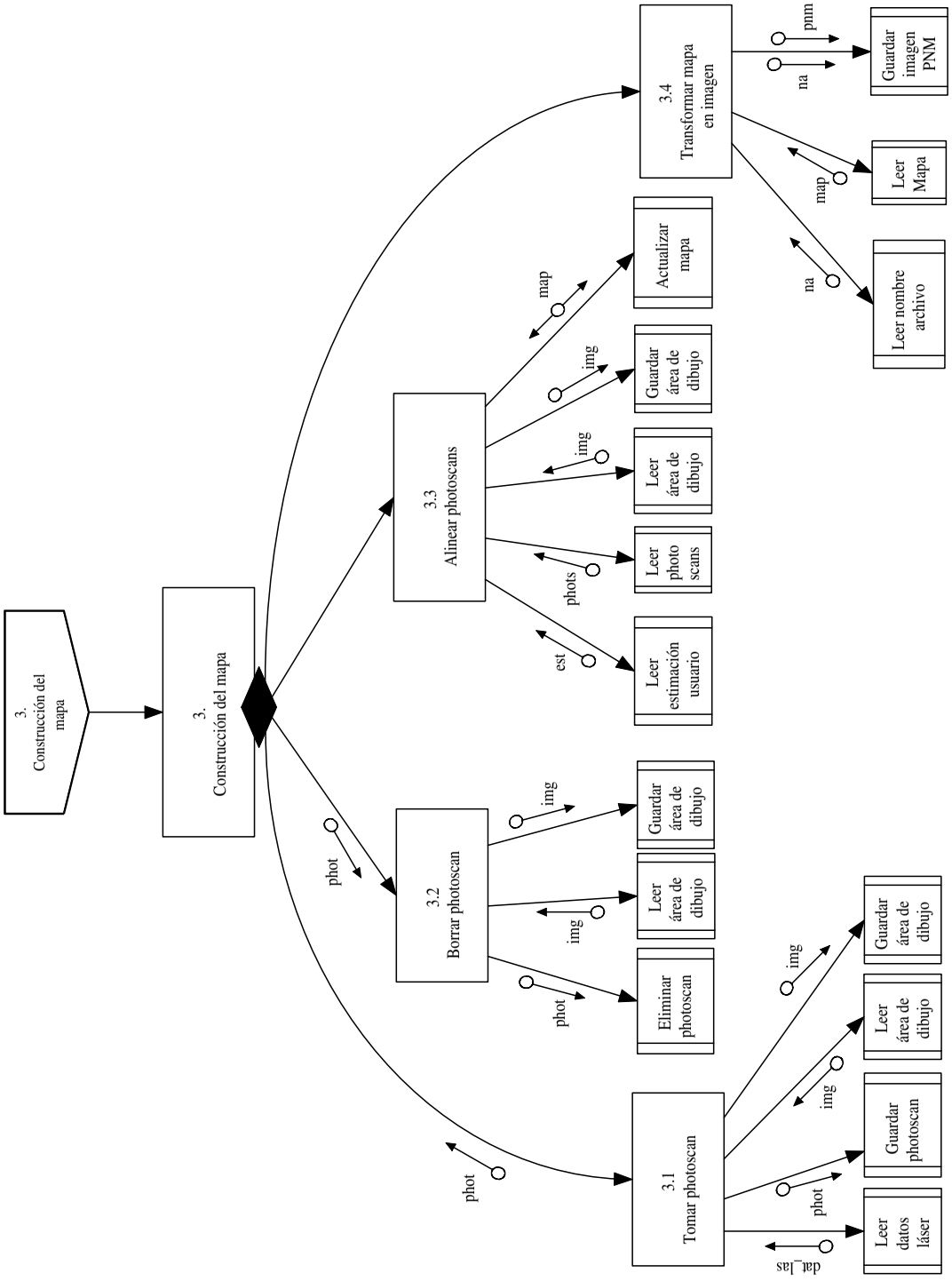

Figura 2.4: *Diagrama de estructuras del módulo "Construcción del mapa".*

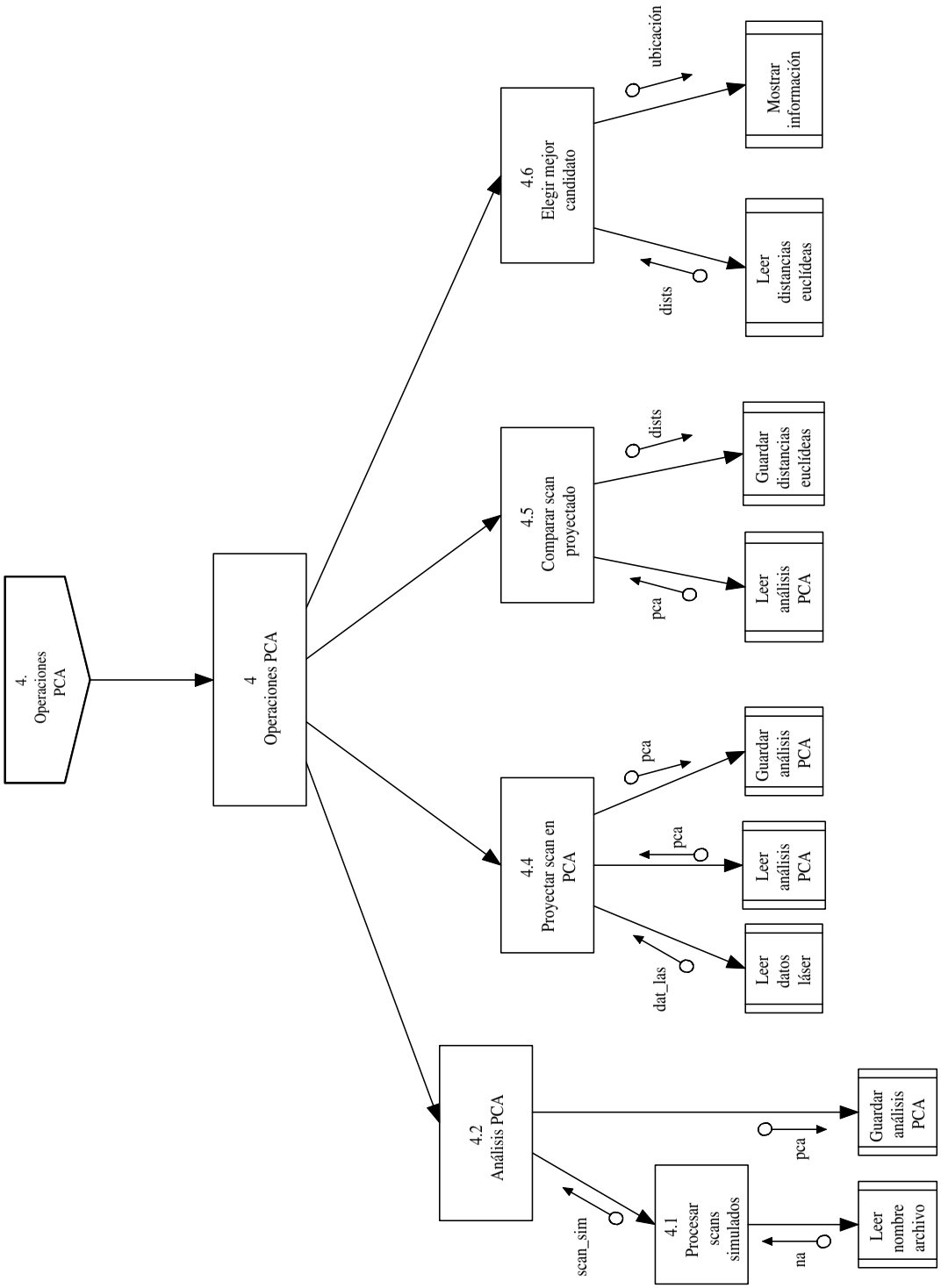

Figura 2.5: *Diagrama de estructuras del módulo "Operaciones PCA".*

2.2. Leyenda de parámetros

- **est_rej** = estado de la rejilla (visible u oculta)
- **cmd** = comando que se va a enviar a los motores
- **cp** = conexión con el servidor Player
- **dat_las** = datos del escáner láser
- **dists** = conjunto de distancias euclídeas entre un scan y los scans simulados
- **est** = estimación de desplazamiento y giro
- **g** = valor del giro
- **map** = mapa del entorno
- **na** = nombre del archivo
- **op** = opción seleccionada por el usuario
- **pca** = análisis de componentes principales
- **phot** = photoscan
- **phots** = conjunto de photoscans
- **pnm** = imagen en formato PNM correspondiente al mapa del entorno
- **img** = imagen que se muestra por pantalla
- **tam_rej** = tamaño de la rejilla
- **tdisp** = tipo de dispositivo
- **trob** = tipo de robot
- **s** = sentido de la marcha
- **scan_sim** = conjunto de scans simulados

- **ubicación** = información sobre la posible posición y giro del escáner láser
- **v** = valor de la velocidad
- **zoom** = valor del zoom para aplicar a la imagen

2.3. Tablas de interfaz

Las tablas de interfaz representan los parámetros que se pasan los diferentes módulos entre sí.

Las distintas opciones de la columna **"uso"** son:

- **P:** el parámetro es procesado.
- **M:** el parámetro es modificado.
- **T:** el parámetro es transferido por el módulo llamado a otor módulo que éste llama, sin modificar su valor.
- **C:** el parámetro es como una variable de control, quizás para actuar como índice conmutador, como un valro de un flag o para la especificación de una función que es usada por el módulo llamado.
- **I:** el parámetro es transferido a otro módulo y es modificado en este segundo módulo.

Módulo	**Parám. formal**	**Entr.**	**Sal.**	**Uso**	**Significado parámetro**
LasMap	-	-	-	-	-
Leer opción usuario (op)	op	Sí	No	I	Opción seleccionada por el usuario
Operaciones área de dibujo (op)	op	Sí	No	C	Opción seleccionada por el usuario
Modificar rejilla	-	-	-	-	-
Obtener propiedades rejilla (est_rej, tam_rej)	est_rej	No	Sí	I	Estado rejilla
	tam_rej	No	Sí	I	Tamaño rejilla
Leer tamaño rejilla (tam_rej)	tam_rej	No	Sí	I	Tamaño rejilla
Leer estado rejilla (est_rej)	est_rej	No	Sí	I	Estado rejilla
Leer área dibujo (img)	img	No	Sí	I	imagen
Guardar área dibujo (img)	img	Sí	No	P	imagen
Visualizar imagen (img)	img	Sí	No	P	imagen
Aplicar zoom	-	-	-	-	-
Leer valor zoom (zoom)	img	No	Sí	I	zoom

Tabla 2.1: *Tabla de interfaz del módulo 1.*

Módulo	Parám. formal	Entr.	Sal.	Uso	Significado parámetro
Comunic. dispositivos (op)	op	Sí	No	C	Opción seleccionada por el usuario
Conexión de dispositivos (cp)	cp	No	Sí	I	conexión a Player
Selección de dispositivo (tdisp)	tdisp	No	Sí	I	tipo de dispositivo
Leer selección usuario (op)	op	No	Sí	I	opción usuario
Enviar comando conexión (tdisp, cp)	tdisp	Sí	No	P	tipo de dispositivo
	cp	No	Sí	I	conexión a Player
Conectarse a Player (cp)	cp	No	Sí	I	conexión a Player
Establecer estado conexión (cp, tdisp)	cp	Sí	No	P	conexión a Player
	tdisp	Sí	No	P	tipo de dispositivo
Procesar datos láser (cp)	cp	Sí	No	T	conexión a Player
Recibir datos láser (cp, dat_las)	cp	Sí	No	T	conexión a Player
	dat_las	No	Sí	I	datos láser
Obtener datos de player (cp, dat_las)	cp	Sí	No	P	conexión a Player
	dat_las	No	Sí	I	datos láser
Guardar datos láser (dat_las)	dat_las	Sí	No	P	datos láser
Visualizar datos láser (dat_las)	dat_las	Sí	No	P	datos láser
Enviar datos (cp)	cp	Sí	No	T	conexión a Player
Componer comando (cmd)	cmd	No	Sí	P	comando motores
Modificar velocidad (v)	v	No	Sí	I	velocidad
Modificar giro (g)	g	No	Sí	I	giro
Modificar sentido (s)	s	No	Sí	I	sentido

continúa ...

Módulo	Parám. formal	Entr.	Sal.	Uso	Significado parámetro
Leer velocidad (v)	v	No	Sí	I	velocidad
Leer giro (g)	g	No	Sí	I	giro
Leer sentido (s)	s	No	Sí	I	sentido
Obtener tipo robot (trob)	trob	No	Sí	I	tipo de robot
Enviar comando motores (cmd, cp)	cp	Sí	No	T	conexión a Player
	cmd	Sí	No	I	comando motores
Enviar datos a Player (cmd, cp)	cp	Sí	No	P	conexión a Player
	cmd	Sí	No	P	comando motores

Tabla 2.2: *Tabla de interfaz del módulo 2.*

Módulo	Parám. formal	Entr.	Sal.	Uso	Significado parámetro
Construcción del mapa (op)	op	Sí	No	C	Opción seleccionada por el usuario
Tomar photoscan (phot)	phot	No	Sí	I	photoscan
Leer datos láser (dat_las)	dat_las	No	Sí	P	datos láser
Guardar photoscan (phot)	phot	Sí	No	P	photoscan
Borrar photoscan (phot)	phot	Sí	No	T	photoscan
Eliminar photoscan (phot)	phot	Sí	No	P	photoscan
Alinear photoscans	-	-	-	-	-
Leer estimación usuario (est)	est	No	Sí	I	estimación desplazamiento y giro
Leer photoscans (phots)	phots	No	Sí	I	conjunto photoscans
Actualizar mapa (map)	map	Sí	Sí	M	mapa del entorno
Transformar mapa en imagen	-	-	-	-	-
Leer nombre archivo (na)	na	No	Sí	I	nombre archivo
Leer mapa (map)	map	No	Sí	I	mapa del entorno
Guardar imagen pnm (na, pnm)	na	Sí	No	P	nombre archivo
	pnm	Sí	No	P	imagen en formato PNM

Tabla 2.3: *Tabla de interfaz del módulo 3.*

Módulo	Parám. formal	Entr.	Sal.	Uso	Significado parámetro
Operaciones PCA (op)	op	Sí	No	C	Opción seleccionada por el usuario
Análisis PCA	-	-	-	-	-
Procesar scans simulados (scan_sim)	scan_sim	No	Sí	I	scans simulados
Guardar análisis PCA (pca)	pca	Sí	No	P	análisis PCA
Proyectar scan en PCA)	-	-	-	-	-
Leer análisis PCA (pca)	pca	No	Sí	I	análisis PCA
Comparar scan proyectado	-	-	-	-	-
Guardar distancias euclideas (dists)	dists	Sí	No	P	distancias euclídeas
Elegir mejor candidato	-	-	-	-	-
Leer distancias euclideas (dists)	dists	No	Sí	I	distancias euclídeas
Mostrar información (ubicación)	ubicación	Sí	No	P	posición y giro del láser

Tabla 2.4: *Tabla de interfaz del módulo 4.*

3

Diseño de datos

El sistema necesita almacenar información sobre dos conceptos: el análisis de componentes principales sobre un conjunto de scans simulados, y el conjunto de puntos que forman un mapa construido por el usuario a base de alinear scans, pero transformados en una imagen en formato PNM.

Además, el sistema deberá ser capaz de leer los datos correspondientes a un conjunto de scans simulados en un determinado entorno, junto con las posiciones y orientaciones del lugar en el que se localizaba el escáner láser en el momento de obtener los datos. Estos datos no los almacenará el sistema, sino que los deberá crear el usuario programando un cliente del simulador *Stage*, para un entorno determinado. Puesto que éste es un proceso complejo para un computador, se ha optado por dejarlo en manos del usuario de momento. No obstante la creación de estos datos simulados se tratará de automatizar en versiones posteriores del proyecto.

3.1. Análisis de componentes principales

El análisis de componentes principales puede requerir un tiempo de procesamiento bastante elevado si los datos sobre los que se ha de realizar son de gran tamaño. Por lo tanto es conveniente almacenar un análisis PCA en memoria puesto que evita tener que esperar de nuevo el largo tiempo de procesamiento de los datos.

Será necesario almacenar los siguientes datos:

- La dimensión de la matriz de componentes principales (es una matriz simétrica).
- El número de vectores propios que se van a utilizar. Si la matriz de componentes principales tiene una dimensión igual a 361, por ejemplo, y el número de vectores propios que se van a utilizar es 10, significa que únicamente se deberán almacenar las 10 primeras filas de la matriz PCA.
- Número de scans simulados que se han procesado.
- Datos del vector media. Este vector representa la media de la matriz de datos que contiene el conjunto de scans simulados.
- Datos del vector característico. El vector característico es un subconjunto de la matriz de componentes principales, que contiene tantas filas como el número de vectores propios que se hayan decidido utilizar.
- Matriz de datos proyectada en el espacio vectorial. Es el resultado de proyectar la matriz de datos que contiene el conjunto de scans simulados en el espacio vectorial, utilizando para ello el vector característico.

3.2. Mapa en formato PNM

El simulador Stage (en su versión 1.3.4) para cargar un mapa del entorno necesita que este mapa sea una imagen en formato PGM tipo 5 (datos binarios) que cumpla las restricciones siguientes:

- Los pixels de color negro son interpretados como el vacío.

- Los pixels de color blanco son interpretados como objetos.

Por lo tanto la imagen correspondiente al mapa de un entorno determinado, debe tener el fondo completamente negro, y cada uno de los puntos que forman el mapa deben ser dibujados como pixels de color blanco.

3.3. Datos de scans simulados

Los datos que se leen del escáner láser pueden estar en formato polar o en formato cartesiano. Puesto que en el análisis de componentes principales se necesitan los datos en formato polar, se deberán almacenar todos y cada uno de los scans simulados en formato polar. Por otra parte, el haz de luz que el escáner lanza, puede emitirse con una diferencia angular de un grado o de medio grado, lo cual produce un conjunto de datos de 361 elementos (1 grado) o 181 elementos (0.5 grados). Por último es necesario saber cual era la posición en coordenadas cartesianas así como la orientación en la cual estaba situado el escáner láser a la hora de tomar los scans simulados; esto sera útil a la hora de intentar localizar el escáner láser en el entorno.

Por lo tanto, los datos necesarios a almacenar por parte del usuario, y que debe ser capaz de interpretar el sistema son:

- Número de scans simulados.
- Número de elementos por cada scan (361 ó 181).
- Datos correspondientes a los scans simulados en formato polar.
- Datos correspondientes a las posiciones y orientaciones en que fue situado el escáner láser a la hora de simular los scans.

4

Diseño de la interfaz

Se define el concepto interfaz de usuario como el conjunto de elementos a través de los cuales un usuario interactúa con un objeto para realizar una tarea determinada. Otro concepto muy importante es el de interfaz de usuario de un programa definido como el conjunto de elementos software y hardware de un ordenador que presentan información al usuario y le permiten interactuar con el sistema.

Estos conceptos no conviene confundirlos con el de interfaz gráfica de usuario o GUI que se refiere a la representación gráfica en la pantalla de un ordenador de los programas, datos y objetos, así como de la interacción con ellos. Este es el concepto que nos interesa y es lo que comúnmente se denomina como interfaz.

Existen tres perspectivas diferentes de la interfaz de usuario:

- **Modelo del usuario**. El usuario tiene su visión personal del sistema y por tanto, espera que éste se comporte de una forma determinada. Una interfaz siempre debe facilitar al usuario el proceso de crear un modelo mental efectivo. Para lograr este fin son de gran utilidad las metáforas ya que asocian un

dominio nuevo con uno ya conocido y familiar al usuario como puede ser la papelera de reciclaje dentro del escritorio del sistema.

- **Modelo del programador**. Es el más fácil de visualizar. Está constituido por los objetos que manipula éste y que deben estar siempre escondidos del usuario.

- **Modelo del diseñador**. Se encarga de describir los objetos que utiliza el usuario así como su presentación al mismo y las técnicas de interacción para su manipulación. Se trata de un intermediario entre el programador y el usuario. Consta de tres partes:

  - La presentación. Es lo primero que capta la atención del usuario, pero posteriormente pasará a un segundo plano. Conviene no abusar de ella introduciendo elementos que desvíen la atención de los puntos clave.
  - La interacción. Define como se comunicará el usuario a través de los diferentes dispositivos.
  - Las relaciones entre objetos. En esta parte será donde el diseñador determine la metáfora adecuada que encajará dentro del marco mental de usuario. Conviene que el modelo comience por esta parte y vaya subiendo hacia arriba.

Existen diferentes tipos de metáforas, donde las más importantes son las siguientes:

- Metáforas de presentación que a su vez se organizan en:

  - Tablas.
  - Pantallas de diálogo.
  - Objetos de tipo texto.
  - Gráficos y dibujos.

- Metáforas de interacción de mandatos:

  - Intérpretes de mandatos. Son aquellas en las que el usuario escribe órdenes utilizando un lenguaje con sintaxis propia. En este tipo de metáforas la información proporcionada por parte del sistema es escasa.
  - Menús de pantalla completa. Muestran una lista de opciones en la pantalla y permiten establecer una estructura jerárquica de menús y navegar a través de ella.
  - Menús desplegables. Son aquellos menús que se despliegan a partir de una barra de menú situada en la parte superior de la pantalla, cabe la posibilidad de menús en cascada o submenús.
  - Pantallas de diálogo. Permiten introducir y mostrar información en casillas etiquetadas, tienen la apariencia de un formulario.

Una vez vistas las posibles metáforas que pueden ser utilizadas conviene centrarse en las *interfaces gráficas de usuario (GUI)* y sus distintos componentes para poder localizarlos y situarlos con facilidad dentro del proyecto *LasMap*. Los principales componentes de una GUI son los siguientes:

- Componentes gráficos
  - Ventanas.
  - Menús desplegables.
  - Barras de herramientas.
  - Barra de estado.
  - Botones.
  - Iconos.
  - Cuadros de diálogo.
  - Cajas de comprobación (*check boxes*).
  - Deslizadores (sliders).
  - Botones de selección (*radio buttons*).
  - Cajas combinadas (Combo boxes)

  - Botones spin.
  - Listas emergentes (popup).
  - Entradas de texto.
  - Áreas de texto.

- Eventos. Son secuencias que desencadena el usuario y provocan la ejecución del software.

Una vez comentados los principales elementos que forman una GUI se observará como quedan integrados dichos elementos dentro de la aplicación *LasMap* para formar la *Interfaz Gráfica de Usuario*.

4.1. Menús

Dentro de la aplicación *LasMap* existirá una barra de menú principal (ver fig. 4.1) situada en la parte superior de la ventana.

Figura 4.1: *Menú principal*

El menú principal es una metáfora utilizada para dotar a la aplicación de funcionalidad. Es un *widget* que se deriva de la clase `GtkMenuBar`, y cada uno de los submenús que contiene son ítems del menú sensibles a los siguientes eventos del ratón:

- **superposición:** el ítem responde cuando el cursor del ratón se le superpone, mostrando un mensaje informativo.
- **click:** el ítem se activa con la generación del mensaje `button_raise`.

Menú archivo

El menú Archivo (ver fig. 4.2) es una metáfora de interacción de mandatos que permite al usuario guardar el mapa como una imagen (RF - 12) y salir de la aplicación.

Figura 4.2: *Menú archivo*

Menú Rejilla

El menú Rejilla (ver fig. 4.3) es una metáfora de interacción de mandatos que permite al decidir si mostrar o no una rejilla en el área de dibujo (RF - 06 y RF - 07) y decidir la distancia de separación entre cada línea de la rejilla (RF - 08).

Figura 4.3: *Menú rejilla*

Menú Position

El menú *position* (ver fig. 4.4) es una metáfora de interacción de mandatos destinada a que el usuario pueda conectarse y desconectarse de un dispositivo *position* (RF - 16 y RF - 17) y para que elija si el dispositivo *position* que está utilizando se trata de un dispositivo físico real (RF - 18) o simulado por *Stage* (RF - 19).

Figura 4.4: *Menú position*

Menú Láser

El menú Láser (ver fig. 4.5) es una metáfora de interacción de mandatos que le da la posibilidad al usuario de que se conecte (RF - 14) y desconecte (RF - 15) de

un dispositivo láser y para que elija si los datos que se reciben en tiempo real del láser se muestren gráficamente en el área de dibujo (RF - 01).

Figura 4.5: *Menú Láser*

Menú PCA

El menú PCA (ver fig. 4.6) es una metáfora de interacción de mandatos que permite al usuario realizar operaciones relacionadas con el análisis de componentes principales de un conjunto de scans simulados y la localización del escáner láser en un entorno utilizando el análisis PCA. Al elegir la opción "Calcular PCA", en primer lugar se lee un archivo con scans simulados (RF - 23), a continuación se realiza el análisis de componentes principales (RF - 24) y por último se le pregunta al usuario por una ubicación en memoria secundaria en la cual almacenar dicho análisis (RF - 25). La opción "Leer archivo con PCA" permite cargar en memoria principal un análisis PCA previamente almacenado (RF - 26). Finalmente la opción "Localizar" ofrecerá al usuario la posición y el giro más probables en las que se encuentra el escáner láser (RF - 27).

Figura 4.6: *Menú PCA*

Menú Ayuda

El menú Ayuda (ver fig. 4.7) es una metáfora de interacción de mandatos que reúne el conjunto de opciones para que el usuario pueda recibir ayuda sobre el manejo de la aplicación (RNF - 3).

Figura 4.7: *Menú Ayuda*

4.2. Cuadros de diálogo

En esta sección se describirán los cuadros de diálogo que el usuario podrá utilizar al trabajar con la aplicación *LasMap*.

Cuadro de diálogo de selección de archivos

Para la selección de archivos (tanto para abrir como para guardar) se han utilizado cuadros de diálogo como los de la fig. 4.8 es un cuadro estándar en el entorno de trabajo GNOME. En él se encuentan integrados un conjunto de componentes que se van a describir a continuación:

- Un cuadro de navegación de directorios.
- Un cuadro de navegación de archivos.
- Dos botones de navegación de directorios para ir de forma directa al escritorio o al directorio personal.
- Tres botones para la edición de las propiedades de un archivo.
- Un cuadro de entrada de texto mediante el cual el usuario puede introducir el nombre del archivo que se desea seleccionar.
- Dos botones de confirmación de acción para llevar a cabo diversas acciones.

Esta metáfora de presentación será utilizada en la mayoría de los casos de uso que tengan que ver con el almacenamiento o recuperación de la información, como serán: RF - 12 (Guardar imagen como mapa), RF - 23 (Leer archivo con scans simulados), RF - 25 (Guardar análisis PCA), y RF - 26 (Leer análisis PCA).

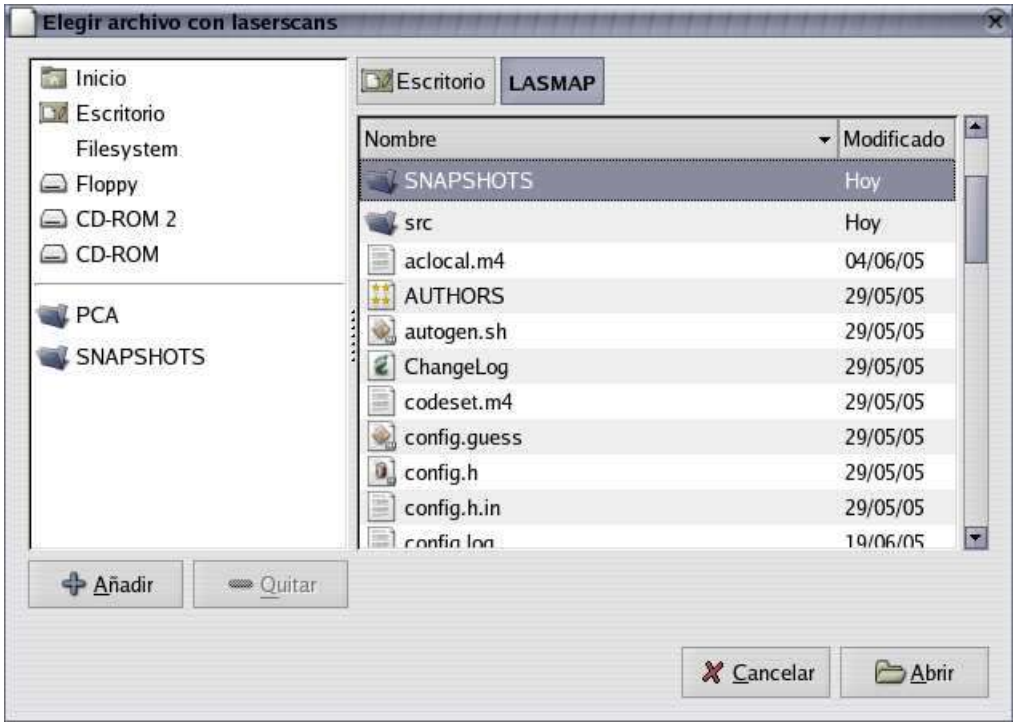

Figura 4.8: *Selección de archivos*

Cuadro de diálogo de entrada de datos

Para la petición de datos de tipo numérico se ha utilizado el cuadro de diálogo de la fig. 4.9. Permite al usuario mediante cuadros de entrada de texto, introducir los datos necesarios para estimar el desplazamiento y giro que ha sufrido el escáner láser durante la alineación de dos photoscans, siendo una parte del requisito RF - 11.

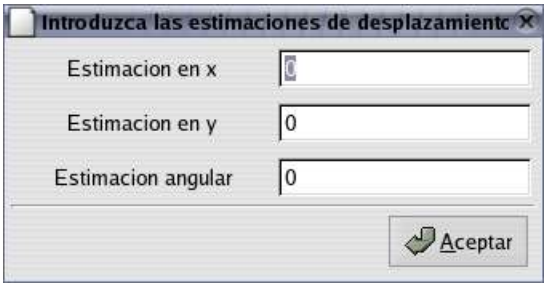

Figura 4.9: *Petición de datos*

Cuadro de diálogo de información

Siempre que se realice una operación (leer datos, guardar datos, alinear scans) con éxito, se mostrará un cuadro de diálogo similar al de la fig. 4.10, que contienen un mensaje informando que la operación fué realizada con éxito, o para ofrecer cualquier otro tipo de información (que no sea de error) al usuario. Estas metáforas se corresponden con el requito RNF - 3.

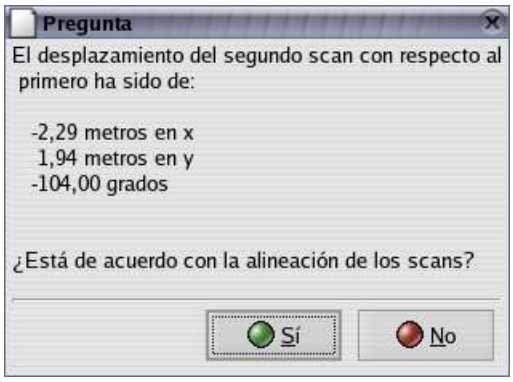

Figura 4.10: *Cuadro de diálogo de información*

Cuadro de diálogo de error

Un cuadro de diálogo de error (ver fig. 4.11 indica que ha ocurrido un error que imposibilita continuar con la realización de la tarea emprendida, como sería por ejemplo el no poder conectarse al servidor *Player*, o el intentar leer un archivo con un formato que es comprendido por la aplicación. Se corresponde con el requisito RNF - 3.

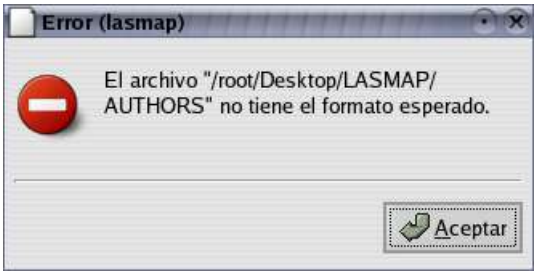

Figura 4.11: *Cuadro de diálogo de error*

4.3. Barras de herramientas

Son componentes de la interfaz gráfica de usuario creadas para satisfacer el requisito RNF - 3 para facilitar el trabajo con la aplicación. Cada barra de herramientas es un widget de la clase `GtkToolBar`, y dependiendo de cada barra, podrán estar en posición vertical, horizontal o ambas. Además para facilitar la adaptación del usuario a la aplicación se ha optado por convertir cada una de estás barras en un componente `BonoboDock`, lo cual permite que floten sobre la ventana principal de la aplicación o fuera de ella, , es decir, las barras de herramientas no necesitan estar acopladas a la ventana principal.

Barra de herramientas principal

En la barra de herramientas principal (ver fig. 4.12) se pueden apreciar un conjunto de cinco botones correspondientes a opciones relacionadas con el área de dibujo que recibirán un mayor uso, y por lo tanto serán más fácilmente ejecutables en una barra de herramientas que en la barra de menú. Esta barra puede situarse verticalmente, horizontalmente y flotando sobre la aplicación. La función de cada botón es la siguiente:

- Herramienta de zoom: consiste en un botón de tipo *spin* (el de más a la izquierda de la barra) para establecer el porcentaje de zoom con el que se verá el área de dibujo. Cumple con el requisito funcional RF - 04.
- Zoom 100 %: hace que el área de dibujo se muestre con un factor zoom que no aumente ni disminuya el área de dibujo, esto es, que se muestre con su tamaño original. Cumple con el requisito RF - 04.
- Zoom ajustado: hace que el área de dibujo, que puede ser más grande que la ventana en la que se muestra, tenga un factor zoom que le permita visualizarse por completo en la ventana. Cumple con el requisito RF - 04.
- Centrar: permite mostrar el origen de los ejes de coordenadas en el centro de la aplicación. (RF - 03)
- Salir: sale de la aplicación.

Figura 4.12: *Barra de herramientas principal*

Barra de herramientas para el manejo del robot

La barra de herramientas para el manejo del robot (ver fig. 4.13) permite al usuario controlar el robot móvil, tanto un robot físico como uno simulado. Mediante la barra de control de velocidad se puede controlar, como porcentaje de su velocidad máxima, la velocidad que alcanzará el robot en un momento dado (RF - 20). Con los botones con forma de flecha, el usuario puede controlar el sentido de la marcha del robot (RF - 21) y el giro que va a realizar (RF - 22).

Figura 4.13: *Barra de herramientas para el control del robot*

Barra de herramientas para la construcción del mapa

La barra de herramientas para la construcción de mapas (ver fig. 4.14) permite al usuario controlar el proceso de elaboración de un mapa tomando como unidades básicas de construcción los datos que proporciona el escáner láser. Las funciones que ofrece son las siguientes:

- Fotografiar los datos que percibe el láser en un momento dado (RF - 09)

- Borrar el último scan fotografiado (RF - 10)
- Hacer coincidir (RF - 11) los dos scans fotografiados y añadirlos al mapa global (RF - 05)
- Eliminar el mapa en proceso de construcción (RF - 13)

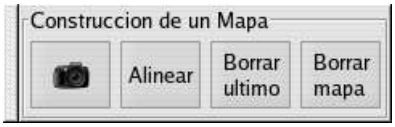

Figura 4.14: *Barra de herramientas para construir mapas*

4.4. Área de dibujo

El área de dibujo es un elemento de suma importancia en esta aplicación gráfica. Sirve para permitir llevar a cabo el requisito RNF - 3, y además interviene en parte de otros requisitos (RF - 01 hasta RF - 11). El área de dibujo es un *widget* del tipo `GnomeCanvas`, que está incluido en la biblioteca `Libgnomecanvas`. En la figura 4.15 se pueden apreciar los ejes de coordenadas, representados en el centro del área de dibujo, la rejilla, con una distancia de separación de un metro, y dos photoscans.

El área de dibujo se ha diseñado para que respondiera de manera interactiva a la detección del movimiento del ratón en su interior. Al producirse un evento de movimiento, automáticamente se indica en la barra de estado (en la parte inferior de la aplicación) las coordenadas (x, y) en metros del punto de la imagen en la que está situado el puntero.

Además incluye dos barras de desplazamiento (vertical y horizontal) para facilitar la visualización de una determinada parte del canvas, puesto que este normalmente ocupa más tamaño que la ventana que lo contiene.

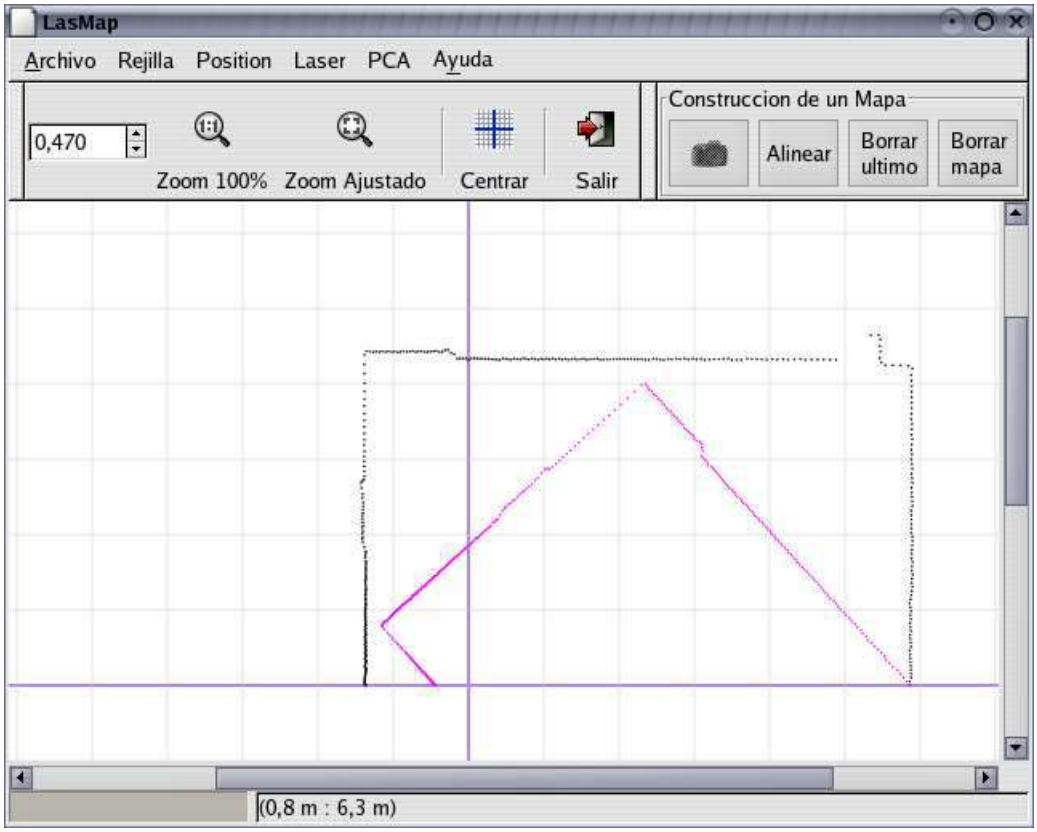

Figura 4.15: *Área de dibujo de la aplicación*

Anexo 4

DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

Proyecto LasMap

Manual del Programador

Versión 1.0

Julio 2005

Realizado Por	Tutores
Carlos Fernández Caramés	*Vidal Moreno Rodilla* *Belén Curto Diego*

Para

Departamento de Informática y Automática

Universidad de Salamanca

Índice general

1 Introducción

En este documento se va a tratar sobre la implementación, que ha sido el resultado de la aplicación del diseño a un lenguaje de programación para un ordenador personal tipo compatible IBM.

Debido al gran auge que están adquiriendo los sistemas operativos de libre distribución, liderados por Linux, se ha tomado esta plataforma como base a este proyecto que pretende ofrecer unos servicios de construcción de mapas y autolocalización de robots móviles autónomos. El proyecto ha sido diseñado para ser desarrollado con programación estructurada, ya que para su implementación se eligió un lenguaje no orientado a objetos como es C.

En la siguiente sección *Documentación de las bibliotecas* se hará una breve referencia al conjunto de bibliotecas que se han utilizado a lo largo de la implementación del proyecto.

En la secciones *Documentación de estructuras* y *Documentación de archivos* se recogerá una descripción del propósito de las estructuras y las funciones creadas durante el desarrollo de la aplicación.

El código fuente se proporciona en formato magnético en un CD-ROM aparte en el

subdirectorio *Fuentes*. Dentro de cada archivo `.c` donde se encuentran implementadas las funciones, y cada una de ellas está adecuadamente comentada utilizando una cabecera siguiendo el estilo de comentarios *javadoc*.

2

Documentación de las bibliotecas

El sistema desarrollado ha necesitado principalmente tres tipos de bibliotecas:

- Bibliotecas del entorno GNOME, para desarrollar la interfaz gráfica de usuario.
- Bibliotecas cliente de *Player*, para poder interactuar con este servidor.
- Bibliotecas de cálculo científico, para realizar cálculos con matrices y vectores y valores propios.

2.1. Bibliotecas de GNOME

La base de la arquitectura de GNOME la componen dos bibliotecas que no fueron originalmente creadas para el proyecto GNOME. Estas bibliotecas son `Glib` y `GTK+`, originalmente creadas por los desarrolladores de *The GIMP*, uno de los programas más famosos del mundo del software libre. A estas bibliotecas se unen otras como `libGnomeui` y `Libgnomecanvas`.

2.1.1. Glib

`Glib` es una biblioteca que contiene multitud de funcionalidad que es necesaria prácticamente en todos los programas, a la vez que incluye algunas funcionalidades de más alto nivel, como es, por ejemplo, el sistema de objetos. Se podría definir glib como una biblioteca de ayuda para la programación en C. Ofrece entre otras cosas:

- Tipos de datos portables que permiten al programador abstraerse de las diferencias entre distintos S.O.
- Gestión de memoria mejorada con respecto a `libc`.
- Tipos de datos abstractos como listas enlazadas, tablas *hash*, *arrays* dinámicos...

2.1.2. GTK+

`GTK+` ofrece todo lo necesario para el desarrollo de interfaces gráficas, pasando por los *widgets* más básicos (botones, cajas de texto, menús, ventanas, etc.) hasta otros mucho más complejos y elaborados que serán de gran ayuda a la hora de programar aplicaciones gráficas. `GTK+` está a su vez basado en `GDK`, que implementa el nivel más bajo de la arquitectura, es decir, las primitivas gráficas. Es una biblioteca que forma una capa sobre la implementación gráfica real (X Window, MS Windows, Mac OS X), y es por tanto la única parte de GTK+ que tiene que ser reescrita para soportar otra plataforma/sistema operativo. Es por esta razón por la que GTK+ ya ha sido portada a varios entornos (X Window, MS Windows, QNX, BeOS...)

2.1.3. libGnomeui

`libGnomeui` es una colección de elementos desarrollados en *GNOME* para el desarrollo de interfaces gráficas de usuario. Consiste principalmente en *widgets* diseñados para mejorar y extender los que ofrece `GTK+`. Algunos elementos interesantes de `libGnomeui` son:

- El *widget* `Gnomeapp` que facilita la tarea de crear la ventana principal de la aplicación. Se puede usar en conjunto con componentes `BonoboDock`, que permiten acoplar y desacoplar barras de herramientas.
- Funciones para la creación de cuadros de diálogo.

- Iconos *standard* de GNOME (para abrir, guardar, salir, y otras operaciones).

A partir de la versión 2.0 de `GTK+` se están portando las funciones incluidas en la biblioteca `libGnomeui` hacia `GTK+`.

2.1.4. Libgnomecanvas

La biblioteca `Libgnomecanvas` contiene uno de los *widgets* más importante desarrollados en el entorno *GNOME*, el `GnomeCanvas`. Tanto es así, que antes formaba parte de la biblioteca `libGnomeui`, pero a partir de la versión 2.0 de GNOME se independizó formando su propia biblioteca.

El canvas de *GNOME* es un *widget* que ofrece un método fácil y potente para presentar gráficamente los datos de una aplicación. Este objeto es un área de dubujo en blanco, donde se pueden insertar objectos `GnomeCanvasItem`, los cuales representan los elementos mostrados en el `GnomeCanvas`. De este modo es posible trabajar con gráfico en términos de objetos.

El `GnomeCanvas` puede trabajar en dos modos: el **modo RGB**, de mayor calidad gráfica (*antialiasing*)y el **modo GDK**, que ofrece una menor calidad. Además permite construir grupos de objetos para manejar varias entidades como una sola, y manejar el *zoom* del canvas de una manera realmente sencilla.

2.2. Bibliotecas clientes de Player

Player es un servidor de dispositivos de robots. Ofrece un control completo sobre los sensores y actuadores del robot. Cuando se ejecuta *Player* en un robot móvil, se puede crear un programa cliente utilizando *sockets* TCP estándar, y la comunicación se logra por medio del envío de un conjunto de mensajes muy simples.

Player está diseñado para ser independiente del lenguaje y plataforma utilizados. El programa cliente se puede ejecutar en cualquier tipo de máquina que tenga conexión a la red del robot móvil, y se puede escribir en cualquier lenguaje que pueda abrir y controlar un *socket* TCP. A pesar de que, según dicen los autores, es sencillo escribir programas clientes utilizando *sockets*, es más conveniente utilizar alguna de las bibliotecas clientes que se incluyen en la distribución. El paquete estándar de *Player/Stage* incluye bibliotecas cliente en los lenguajes C, C++, Python y TCL.

2.2.1. libplayerc

`libplayerc` es una biblioteca cliente para el servidor de dispositivos robóticos *Player*. Está escrita en C para maximizar la portabilidad, y con la esperanza de que otros usuario escriban envoltorios para esta biblioteca en otros lenguajes.

`libplayerc` está basada en un modelo de dispositivos *proxy*. Cada cliente mantiene un *proxy* local para cada uno de los dispositivos en el servidor remoto. Así, por ejemplo, se puede crear un *proxy* local para un dispositivo *position* o para un dispositivo *laser*. Existe también un *proxy* client especial que se utiliza para controlar el servidor *Player*.

2.2.2. C++ client library

La biblioteca cliente C++ es generalmente la más completa, puesto que es usada para probar nuevas caracterísiticas a medida que se van incluyendo en el servidor. Además es la biblioteca cliente más ampliamente utilizada por la comunidad de usuario y por lo tanto es la que está mejor depurada.

La biblioteca C++ está basada en un modelo de "servicios *proxy*", en el cual el cliente mantiene objetos locales que sirven de *proxy* de los servicios remotos. Hay dos clases de *proxy*: el proxy del servidor `PlayerClient` y otros proxy específicos de cada dispositivo. Para utilizar esta biblioteca se debe crear en primer lugar un *proxy* `PlayerClient` con el que se establecerá una conexión con el servidor *Player*, para a continuación utilizar los *proxy* de cada dispositivo para comunicarse con los sensores y actuadores.

2.3. Bibliotecas de cálculo científico

La aplicación desarrollada necesita realizar la técnica estadística conocida como **análisis de componentes principales** (PCA), y puesto que es un proceso que puede llegar a requerir mucho tiempo de procesamiento, es importante que esté lo más optimizado posible para que el tiempo dé cálculo sea el menor posible. Puesto que el análisis PCA necesita realizar cálculos complicados con matrices de grandes dimensiones, se puede optar por intentar optimizar personalmente multiplicaciones de matrices y otras complejas operaciones (en otras palabras, reinventar la rueda) o utilizar bibliotecas de cálculo científico como `BLAS` y `LAPACK`, muy optimizadas y depuradas por una amplia comunidad de expertos en la materia.

2.3.1. BLAS

La biblioteca `BLAS` (*Basic Linear Algebra Subroutines*, es un conjunto de funciones que implementan operaciones fundamentales del álgebra lineal tales como productos escalares entre vectores, multiplicaciones matriz-vector y operaciones matriz-matriz; todas ellas están altamente optimizadas. Estas funciones son consideradas como bloques de construcción de alta calidad y sirven como base para construir otros paquetes software de álgebra lineal de más alta calidad como por ejemplo LINPACK o LAPACK.

2.3.2. LAPACK

LA biblioteca `LAPACK` (*Linear Algebra PACKage*) implementa una serie de cálculos algebráicos más avanzados que BLAS, diseñados para resolver complejas operaciones matemáticas como por ejemplo diversos tipos de factorizaciones (LU, LL ...), cálculo de vectores propios, ecuaciones lineales, ...

3

Documentación de estructuras

3.1. Referencia de la Estructura estimation_t

`#include <correlation.h>`

Atributos públicos

- double **angle**
- double **x**
- double **y**

3.1.1. Descripción detallada

Estructura para almacenar una estimación de desplazamiento y giro de un laserscan con respecto a otro. La estimación hay que darla SIEMPRE con respecto al sistema de referencia del scan, no con respecto al mapa global o a las paredes de la habitación.

3.1.2. Documentación de los datos miembro

double estimation_t::angle

Angulo en grados. Negativo es counterclockwise. Positivo es clockwise.

double estimation_t::x

Desplazamiento en el eje x. Positivo es que estimamos que nos hemos movido a la derecha. Negativo a la izquierda.

double estimation_t::y

Desplazamiento en el eje y. Positivo es hacia adelante. Negativo hacia atrás.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- **correlation.h**

3.2. Referencia de la Estructura histogram_t

`#include <histogram.h>`

Atributos públicos

- double * **bin**
- size_t **n**
- double * **range**

3.2.1. Descripción detallada

Estructura en la que se almacenarán los datos correspondientes a un histograma.

3.2.2. Documentación de los datos miembro

double* histogram_t::bin

El valor de cada barra se almacena en un array de n elementos apuntado por "bin".

size_t histogram_t::n

Número de barras (bins) del histograma.

double* histogram_t::range

Los rangos de cada barra se almacenan en un array de n+2 elementos apuntado por "range".

La documentación para esta estructura fué generada a partir del siguiente archivo:

- **histogram.h**

3.3. Referencia de la Estructura map_dim_t

`#include <lecturalaser.h>`

Atributos públicos

- double **xmax**
- double **xmin**
- double **ymax**
- double **ymin**

3.3.1. Documentación de los datos miembro

double map_dim_t::xmax

Punto más a la derecha del mapa global (máximo en el eje x).

double map_dim_t::xmin

Punto más a la izquierda del mapa global (minimo en el eje x).

double map_dim_t::ymax

Punto más arriba del mapa global. (máximo en el eje y).

double map_dim_t::ymin

Punto más abajo del mapa global. (mínimo en el eje y).

La documentación para esta estructura fué generada a partir del siguiente archivo:

- **lecturalaser.h**

3.4. Referencia de la Estructura point_t

`#include <correlation.h>`

Atributos públicos

- double **x**
- double **y**

3.4.1. Descripción detallada

Estructura para almacenar un punto de un laserscan.

3.4.2. Documentación de los datos miembro

double point_t::x

Coordenada x del punto.

double point_t::y

Coordenada y del punto.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- **correlation.h**

3.5. Referencia de la Estructura scan_t

`#include <correlation.h>`

Atributos públicos

- double **angle** [LASER_SAMPLES]
- int **angle_count**
- **histogram_t** ∗ **hist**
- **point_t point** [LASER_SAMPLES]
- int **point_count**
- **histogram_t** ∗ **x_hist**
- **histogram_t** ∗ **y_hist**

3.5.1. Descripción detallada

Estructura para almacenar los datos necesarios para un laserscan.

3.5.2. Documentación de los datos miembro

double scan_t::angle[LASER_SAMPLES]

Array de angulos necesarios para el cálculo de histogramas.

int scan_t::angle_count

Cuantos ángulos hay

histogram_t∗ scan_t::hist

Histograma angular del laserscan.

point_t scan_t::point[LASER_SAMPLES]

Array de puntos que forman un laserscan.

int scan_t::point_count

Cuantos puntos tiene el laserscan.

histogram_t∗ scan_t::x_hist

Histograma-X del laserscan

histogram_t∗ scan_t::y_hist

Histograma-Y del laserscan

La documentación para esta estructura fué generada a partir del siguiente archivo:

- **correlation.h**

4

Documentación de archivos

4.1. Referencia del Archivo bars.c

```
#include <gnome.h>
#include "main_window.h"
#include "bars.h"
#include "canvas.h"
#include "motors.h"
#include "lecturalaser.h"
#include "pca.h"
```

Funciones

- void **about_cb** (GtkWidget *widget, gpointer data)
- GtkWidget * **create_laserbar_widget** (void)
- GtkWidget * **create_navigation_widget** (void)

- void **install_menu_bar** (GtkWidget *__app__)
- GtkWidget * **install_status_bar** (GtkWidget *__app__)
- GtkWidget * **install_tool_bar** (GtkWidget *__app__)
- void **set_align_button_enabled** (gboolean sensitive)
- void **set_delete_photoscan_button_enabled** (gboolean sensitive)
- void **set_laserbar_widget_sensitivity** (gboolean sensitive)
- void **set_nav_widget_sensitivity** (gboolean sensitive)
- void **set_photoscan_button_enabled** (gboolean sensitive)
- void **set_spin_value** (double valor)

Variables

- GnomeUIInfo **file_menu** []
- GnomeUIInfo **grid_menu** []
- GnomeUIInfo **help_menu** []
- GnomeUIInfo **laser_menu** []
- GtkWidget * **laserbar_widget**
- GnomeUIInfo **main_menu** []
- GtkWidget * **nav_widget**
- GnomeUIInfo **pca_menu** []
- GnomeUIInfo **position_menu** []
- GnomeUIInfo **position_rlm** []
- GnomeUIInfo **radio_list_menu** []
- GtkWidget * **spinbutton**
- GnomeUIInfo **tool_bar_struct** []

4.1.1. Documentación de las funciones

void about_cb (GtkWidget * *widget*, gpointer *data*)

Muestra el típico cuadro de diálogo mostrando la información acerca de la aplicación y de su autor.

GtkWidget * create_laserbar_widget (void)

Crea un widget que contiene una serie de botones para controlar el dispositivo "laser" todo dentro de una GtkTable. Permite tomar photoscans, alinear dos photoscans y borrar el último photoscan tomado en caso de haberlo tomado por error.

Devuelve:

El widget que contiene todo lo anterior.

GtkWidget * create_navigation_widget (void)

Crea un widget que contiene una serie de botones para controlar el robot móvil, todo dentro de un GtkFrame. Permite moverse hacia adelante, hacia atrás, girar sobre si mismo hacia la derecha y la izquierda, y tomar curvas.

Devuelve:

El widget que contiene todo lo anterior.

void install_menu_bar (GtkWidget * *app*)

Instala la barra de menús en la aplicación principal.

Parámetros:

app Ventana principal de la aplicación.

GtkWidget* install_status_bar (GtkWidget * *app*)

Instala la barra de estado en la aplicación principal.

Parámetros:

app Ventana principal de la aplicación.

GtkWidget* install_tool_bar (GtkWidget * *app*)

Instala la barra de herramientas principal de la aplicación , la barra de herramientas para manejar el dispositivo "position" y la barra de herramientas para manejar el dispositivo "laser". Todos se instalan como componentes bonobo dock para que puedan cambiarse de sitio e incluso "flotar" en cualquier parte.

Parámetros:

app Ventana principal de la aplicación.

void set_align_button_enabled (gboolean *sensitive*)

Establece si el botón para alinear photoscans se muestra o no activo.

Parámetros:

sensitive TRUE para activar, FALSE para desactivar.

void set_delete_photoscan_button_enabled (gboolean *sensitive*)

Establece si el botón para borrar el último photoscan se muestra o no activo.

Parámetros:

sensitive TRUE para activar, FALSE para desactivar.

void set_laserbar_widget_sensitivity (gboolean *sensitive*)

Establece si el widget de control del láser se muestra o no activo.

Parámetros:

sensitive TRUE para activar, FALSE para desactivar.

void set_nav_widget_sensitivity (gboolean *sensitive*)

Establece si el widget de navegación se muestra o no activo.

Parámetros:

sensitive TRUE para activar, FALSE para desactivar.

void set_photoscan_button_enabled (gboolean *sensitive*)

Establece si el botón para tomar photoscans se muestra o no activo.

Parámetros:

sensitive TRUE para activar, FALSE para desactivar.

void set_spin_value (double *valor*)

Modifica el valor del botón "spin" que muestra el valor del zoom del canvas. Es útil porque mediante los botones de la barra de herramientas se puede cambiar el zoom sin que se modifique el valor del botón "spin", y gracias a esta función se puede actualizar.

Parámetros:

valor Nuevo valor que se pondrá en el botón "spin".

4.1.2. Documentación de las variables

GnomeUIInfo file_menu[]

Valor inicial:

```
{
        GNOMEUIINFO_ITEM_STOCK(N_("Guardar Mapa"),
                N_("Guarda el mapa en formato EPS y PNM"),
                save_map_cb, GNOME_STOCK_MENU_SAVE),
        GNOMEUIINFO_SEPARATOR,
        GNOMEUIINFO_MENU_EXIT_ITEM(main_window_delete_event_cb, NULL),

        GNOMEUIINFO_END
}
```

Menu archivo. Contiene los elementos "Guardar Mapa" y "Salir".

GnomeUIInfo grid_menu[]

Valor inicial:

```
{
        GNOMEUIINFO_TOGGLEITEM(N_("Mostrar"), N_("Muestra la rejilla"),
                                                        set_grid_state_cb, NULL),

        GNOMEUIINFO_SEPARATOR,

        GNOMEUIINFO_RADIOLIST(radio_list_menu),

        GNOMEUIINFO_END
}
```

Menu Rejilla. Permite mostrar u ocultar la rejilla y elegir una opción de anchura de entre los elementos de una lista.

GnomeUIInfo help_menu[]

Valor inicial:

```
{
        GNOMEUIINFO_ITEM_NONE(N_("Ayuda"),
                N_("Inicia la ayuda de la aplicación"),
                NULL),

        GNOMEUIINFO_SEPARATOR,

        GNOMEUIINFO_MENU_ABOUT_ITEM(about_cb, NULL),

        GNOMEUIINFO_END
}
```

Menú ayuda. Muestra una ventana con la información "Acerca de ..."

GnomeUIInfo laser_menu[]

Valor inicial:

```
{
        GNOMEUIINFO_TOGGLEITEM(N_("Subscribirse"),
                N_("Subscribirse al dispositivo LASER del robot"),
                laser_subscribe_cb, NULL),

        GNOMEUIINFO_TOGGLEITEM(N_("Mostrar"),
                N_("Dibuja el laserscan en tiempo real"),
                set_show_laser_state_cb, NULL),

        GNOMEUIINFO_END
}
```

Menu Laser. Permite suscribirse o cancelar la suscripción a un dispositivo "laser" conectado a un servidor Player. Permite mostrar u ocultar el scan en tiempo real recibido del laser.

GtkWidget* laserbar_widget

Widget que contiene botónes para manejar el láser.

GnomeUIInfo main_menu[]

Valor inicial:

```
{
        GNOMEUIINFO_MENU_FILE_TREE(file_menu),

        GNOMEUIINFO_SUBTREE("Rejilla", grid_menu),
        GNOMEUIINFO_SUBTREE("Position", position_menu),
        GNOMEUIINFO_SUBTREE("Laser", laser_menu),
        GNOMEUIINFO_SUBTREE("PCA", pca_menu),
        GNOMEUIINFO_MENU_HELP_TREE(help_menu),
        GNOMEUIINFO_END
}
```

Menú principal de la aplicación. Contiene los submenús "Archivo", "Rejilla", "Position", "Laser", "PCA" y "Ayuda".

GtkWidget* nav_widget

Widget que contiene todos los botones de navegación para controlar un robot móvil.

GnomeUIInfo pca_menu[]

Valor inicial:

```
{
        GNOMEUIINFO_ITEM_NONE(N_("Calcular PCA"),
                N_("Inicia el analisis de componentes principales"),
                menu_pca_cb),
        GNOMEUIINFO_ITEM_STOCK(N_("Leer archivo con PCA"),
                N_("Abre un archivo que contiene un analisis PCA"),
                menu_read_pca_cb, GNOME_STOCK_MENU_OPEN),
        GNOMEUIINFO_ITEM_NONE(N_("Localizar"),
                N_("Realiza una estimacion de la posicion utilizando datos PCA ya calculados"),
                pca_localize_cb),
        GNOMEUIINFO_END
}
```

Menú PCA. Permite realizar el cálculo de los componentes principales y guardar los cálculos hechos. Permite abrir un archivo con un cálculo previamente realizado. Permite utilizar el scan en tiempo real para estimar la posición más probable en la que se localiza

GnomeUIInfo position_menu[]

Valor inicial:

```
{
        GNOMEUIINFO_TOGGLEITEM(N_("Subscribirse"),
                N_("Subscribirse al dispositivo POSITION del robot"),
                position_subscribe_cb, NULL),

        GNOMEUIINFO_SEPARATOR,

        GNOMEUIINFO_RADIOLIST(position_rlm),

        GNOMEUIINFO_END
}
```

Menu Position. Permite suscribirse o cancelar la suscripción a un dispositivo "position" conectado a un servidor Player. Permite elegir si se está trabajando con el simulador stage o con el robot real (los comandos a enviar serán distintos).

GnomeUIInfo position_rlm[]

Valor inicial:

```
{
        GNOMEUIINFO_RADIOITEM_DATA(N_("Simulador Stage"),
                N_("Envia los comandos de velocidad a un dispositivo position simulado por Stage"),
                set_position_simulated_state_cb, (gpointer)1, NULL),

        GNOMEUIINFO_RADIOITEM_DATA(N_("Robot Real"),
                N_("Envia los comandos de velocidad a un dispositivo position en el robot real"),
                set_position_simulated_state_cb, (gpointer)2, NULL),

        GNOMEUIINFO_END
}
```

Lista de botones de tipo "radio_button" para elegir si se está trabajando con el simulador stage o con el robot real.

GnomeUIInfo radio_list_menu[]

Valor inicial:

```
{
        GNOMEUIINFO_RADIOITEM_DATA(N_("1 metro"),
                N_("Muestra lineas de rejilla cada 1 metro"),
                set_grid_width_cb, (gpointer)1, NULL),

        GNOMEUIINFO_RADIOITEM_DATA(N_("2 metros"),
                N_("Muestra lineas de rejilla cada 2 metros"),
                set_grid_width_cb, (gpointer)2, NULL),

        GNOMEUIINFO_RADIOITEM_DATA(N_("5 metros"),
                N_("Muestra lineas de rejilla cada 5 metros"),
                set_grid_width_cb, (gpointer)5, NULL),

        GNOMEUIINFO_RADIOITEM_DATA(N_("10 metros"),
                N_("Muestra lineas de rejilla cada 10 metros"),
                set_grid_width_cb, (gpointer)10, NULL),

        GNOMEUIINFO_END
}
```

Lista de botones de tipo "radio_button". Contiene varios tipos de anchura de rejilla para elegir uno de entre todos.

GtkWidget* spinbutton

Widget en el que se visualiza el factor zoom que tiene el canvas en todo momento.

GnomeUIInfo tool_bar_struct[]

Valor inicial:

```
{
        GNOMEUIINFO_ITEM_STOCK(N_("Zoom 100%"), N_("Muestra la imagen sin factor zoom"),
                                                zoom_100_cb, GTK_STOCK_ZOOM_100),

        GNOMEUIINFO_ITEM_STOCK(N_("Zoom Ajustado"),
                N_("Establece el zoom necesario para que se vea la imagen por completo"),
                zoom_fit_cb, GTK_STOCK_ZOOM_FIT),

        GNOMEUIINFO_SEPARATOR,

        {GNOME_APP_UI_ITEM, "Centrar",
                "Muestra el origen de coordenadas en el centro de la ventana",
```

```
                (gpointer)center_canvas_scroll_cb, NULL, NULL,
                GNOME_APP_PIXMAP_FILENAME, "center.png", 0, (GdkModifierType) 0, NULL},


        GNOMEUIINFO_SEPARATOR,

        GNOMEUIINFO_ITEM_STOCK(N_("Salir"), N_("Salir del programa"),
                main_window_delete_event_cb, GNOME_STOCK_PIXMAP_EXIT),
        GNOMEUIINFO_END
}
```

Estructura que contiene los botones "Zoom 100 %", "Zoom Ajustado", "Centrar" y "Salir", de la barra de herramientas principal de la aplicación.

4.2. Referencia del Archivo canvas.c

```
#include <gnome.h>
#include <libgnomecanvas/libgnomecanvas.h>
#include <gdk-pixbuf/gdk-pixbuf.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <glib.h>
#include "main_window.h"
#include "canvas.h"
#include "bars.h"
#include "correlation.h"
```

Funciones

- int **add_scan_to_map** (**scan_t** s)
- void **center_canvas_scroll_cb** (GtkWidget *widget, gpointer data)
- void **draw_bg_rect** (void)
- int **draw_first_photoscan** (**scan_t** s)
- void **draw_grid** (void)
- int **draw_realtime_scan** (**scan_t** s, gboolean show)
- int **draw_scan_symbol** (double x, double y, double alpha)
- int **draw_second_photoscan** (**scan_t** s, gboolean borrar)
- GtkWidget * **get_present_canvas** (void)
- void **install_scrolled_canvas** (GtkWidget *app)
- gint **item_events** (GnomeCanvasItem *item, GdkEvent *event, gpointer data)
- void **resize_bg_rect_and_canvas** (int x, int y)
- void **set_grid_state_cb** (GtkWidget *widget, gpointer data)
- void **set_grid_width_cb** (GtkWidget *widget, gpointer data)
- void **zoom_100_cb** (GtkWidget *widget, gpointer data)
- void **zoom_changed_cb** (GtkAdjustment *adj, gpointer data)
- void **zoom_fit_cb** (GtkWidget *widget, gpointer data)

Variables

- GnomeCanvasItem * **bg_rect**
- GtkWidget * **canvas**
- double **canvas_height** = 768.0
- double **canvas_width** = 1024.0
- GnomeCanvasGroup * **grid_group**
- double **grid_width** = 100.0
- gboolean **show_grid** = TRUE
- GnomeCanvasItem * **x_axis_item**
- GnomeCanvasItem * **y_axis_item**

4.2.1. Documentación de las funciones

int add_scan_to_map (scan_t *s*)

Dibuja en el canvas, centrado en el origen de coordenadas, un scan en color azul, para que sea identificado como una parte del mapa global. Cada vez que se llama a esta función se dibuja un nuevo scan en el canvas, no se borra cada vez.

Parámetros:

s Scan que se va a dibujar en el canvas.

Devuelve:

-1 si el scan no tiene puntos. Devuelve 0 si todo fue bien.

void center_canvas_scroll_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que es llamada desde el botón "centrar" de la aplicación principal. Sirve para que el origen de coordenadas se muestre en el centro de la ventana, evitando así tener que buscar el origen centrando el canvas mediante las barras de scroll.

void draw_bg_rect (void)

Crea un rectángulo blanco que ocupa todo el canvas, y dibuja los ejes de coordenadas, en los que serán centrados los dibujos.

int draw_first_photoscan (scan_t *s*)

Dibuja en el canvas, centrado en el origen de coordenadas, un scan en color negro, para que sea identificado como primer photoscan. Cada vez que se llama esta función el scan es borrado del canvas y vuelto a dibujar.

Parámetros:

s Scan que va a ser dibujado.

Devuelve:

-1 si el número de puntos del scan es menor que 1. Devuelve 0 si acaba bien.

void draw_grid (void)

Dibuja una rejilla partiendo del centro de los ejes de coordenadas. Cada pixel de la pantalla se considera como un centímetro. Por ejemplo, 100 pixels son un metro. La anchura de la rejilla se establece en la variable global grid_width.

int draw_realtime_scan (scan_t *s*, gboolean *show*)

Función para dibujar el scan leido del láser en tiempo real. Se puede optar por mostrarlo u ocultarlo.

Parámetros:

s Scan en tiempo real.

show TRUE si se quiere visualizar, FALSE en caso contrario.

Devuelve:

0 si todo funciona bien, mayor que cero en caso contrario.

int draw_scan_symbol (double *x*, double *y*, double *alpha*)

Dibuja un simbolo consistente en un circulo y una linea que indique la orientación en la posición exacta en la que se encuentre el escaner laser con respecto al mapa global.

Parámetros:

x coordenada x del scan.

y coordenada y del scan.

alpha Angulo en grados que representa la orientación del láser.

int draw_second_photoscan (scan_t *s*, gboolean *borrar*)

Dibuja en el canvas, centrado en el origen de coordenadas, un scan en color magenta, para que sea identificado como segundo photoscan. Cada vez que se llama esta función el scan es borrado del canvas y vuelto a dibujar.

Parámetros:

s Scan que va a ser dibujado.

borrar Si se desea borrar del canvas, debe valer TRUE, y entonces no se tiene encuenta el primer parámetro. Si se desea dibujar, debe valer FALSE.

Devuelve:

-1 si el número de puntos del scan es menor que 1. Devuelve -2 si se borró del canvas. Devuelve 0 si acaba bien.

GtkWidget* get_present_canvas (void)

Devuelve un puntero al canvas de dibujo de la aplicación

void install_scrolled_canvas (GtkWidget * *app*)

Crea una ventana con scrolling automático, y dentro de ella un widget canvas antialiased. Dibuja los ejes de coordenadas y un grid de un metro de ancho.

Parámetros:

app Ventana principal de la aplicación.

gint item_events (GnomeCanvasItem * *item*, GdkEvent * *event*, gpointer *data*) `[static]`

Función callback. Es llamada cada vez que ocurre un evento de ratón en el canvas. Su función es imprimir en la barra de estado las coordenadas en metros del lugar que se encuentre apuntando el ratón.

void resize_bg_rect_and_canvas (int *x*, int *y*)

Modifica el tamaño del canvas. También modifica el tamaño del rectángulo blanco que sirve de fondo y de los ejes de coordenadas.

Parámetros:

x Nueva anchura del canvas.

y Nueva altura del canvas.

void set_grid_state_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama desde el menú. Sirve para mostrar u ocultar la rejilla que se muestra encima del canvas.

void set_grid_width_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama desde el menú de la aplicación principal. Dependiendo de la opción elegida en el menú se establecerá una cierta separación entre cada linea de la rejilla.

Parámetros:

widget GtkWidget desde el que se produjo el evento de menú.

data Distancia en metros entre cada linea de la rejilla.

void zoom_100_cb (GtkWidget * *widget*, gpointer *data*)

Establece el factor zoom del canvas sin aumentar ni disminuir, es decir al cien por cien de su tamaño real.

void zoom_changed_cb (GtkAdjustment * *adj*, gpointer *data*)

Establece el factor zoom del canvas, especificando el numero de pixels que corresponden a una unidad del canvas.

void zoom_fit_cb (GtkWidget * *widget*, gpointer *data*)

Establece el factor zoom del canvas al valor necesario para que todo el canvas al completo se pueda visualizar en la ventana de la aplicación sin necesidad de utilizar las barras de scrolling.

4.2.2. Documentación de las variables

GnomeCanvasItem* bg_rect

Rectángulo blanco de fondo, para pintar encima.

GtkWidget∗ canvas

Lienzo de dibujo sobre el que se mostrarán todos los resultados gráficos.

double canvas_height = 768.0

Ancho del canvas.

double canvas_width = 1024.0

Alto del canvas.

GnomeCanvasGroup∗ grid_group

Grupo de líneas que forman la rejilla.

double grid_width = 100.0

Ancho de rejilla (grid) en centimetros.

gboolean show_grid = TRUE

TRUE si se debe mostrar el grid, FALSE en caso contrario.

GnomeCanvasItem∗ x_axis_item

Canvas item para el eje x.

GnomeCanvasItem∗ y_axis_item

Canvas item para el eje y.

4.3. Referencia del Archivo correlation.c

```
#include <math.h>
#include "histogram.h"
#include "correlation.h"
#include "player.h"
#include "canvas.h"
```

Funciones

- void **align_to_x_axis** (**scan_t** *s1, **scan_t** *s2, double crossco_angle, double *main_dir)
- void **angle_computation** (**scan_t** *s)
- int **append_scan_to_file** (char *filename, **scan_t** scan)
- double **crosscorrelation** (**scan_t** s1, **scan_t** s2)
- void **get_scan_x_ends** (**scan_t** s1, **scan_t** s2, double *min, double *max)
- void **get_scan_y_ends** (**scan_t** s1, **scan_t** s2, double *min, double *max)
- void **make_angular_histogram** (**scan_t** *s1, **scan_t** *s2, int interval)
- void **projection_filter** (**scan_t** *s1, **scan_t** *s2, **estimation_t** est)
- int **read_scan_from_file** (char *filename, **scan_t** *scan)
- void **read_scan_from_playerc_laser_t** (playerc_laser_t *p, **scan_t** *s)
- **scan_t rotate_scan** (**scan_t** s, double degrees)
- void **shifts_calculation** (**scan_t** s1_orig, **scan_t** s2_orig, **estimation_t** est, double *angle_shift, double *x_shift, double *y_shift, double *main_dir)
- **scan_t translate_scan** (**scan_t** s, double x, double y)
- **scan_t trim_scan** (**scan_t** s1, **scan_t** s2)
- int **write_scan_to_file** (char *filename, **scan_t** scan)
- double **x_crosscorrelation** (**scan_t** s1, **scan_t** s2, const int shift)
- double **y_crosscorrelation** (**scan_t** s1, **scan_t** s2, const int shift)

4.3.1. Documentación de las funciones

void align_to_x_axis (scan_t * *s1*, scan_t * *s2*, double *crossco_angle*, double * *main_dir*)

Dados dos scans, calcula la dirección principal del primer scan y gira el primer scan de modo que la dirección principal sea paralela al eje de las x. El segundo scan se alinea

tambien con el eje de las x utilizando el angulo de crosscorrelacion con respecto al primer scan y la dirección principal del primer scan.

Parámetros:

s1 Primer scan.

s2 Segungo scan.

crossco_angle Angulo de crosscorrelacióentre los dos scans.

main_dir Dirección principal del primer scan.

void angle_computation (scan_t * *s*)

Realiza un cálculo de angulos del scan que se le pása como parámetro. Cada ángulo se mide entre un punto y el que está SMOOTH posiciónes más adelante, para de este modo suavizar posteriormente los picos del histograma. (en lugar de coger un punto y su inmediato siguiente).

Parámetros:

s Scan que contiene los puntos, y en el que se almacenarán los ángulos calculados.

int append_scan_to_file (char * *filename*, scan_t *scan*)

AÃ177ade un scan a un archivo ya existente, en el que cada línea es un par de coordenadas (x,y). Si el archivo no existe se crea.

Parámetros:

filename Nombre del archivo de destino.

scan Nombre del scan origen.

Devuelve:

0 si todo fue bien, -1 en caso de error.

double crosscorrelation (scan_t *s1*, scan_t *s2*)

Calcula el angulo de crosscorrelacion entre dos scans, esto es, el ángulo que es necesario girar el segundo scan para que ambos estén alineados en la misma dirección. Trata de encontrar el máximo girando el segundo scan (en realidad trasladando el histograma) entre -60 y +60 grados para ver en qué momento la superposicion de ambos scans es máxima.

Parámetros:

s1 Primer scan.

s2 Segundo scan.

Devuelve:

El ángulo de crosscorrelación en grados.

void get_scan_x_ends (scan_t *s1*, scan_t *s2*, double * *min*, double * *max*)

Dados un par de scans, calcula el punto más a la izquierda y más a la derecha de ambos scans en común, esto es, el mínimo y máximo en el eje x.

Parámetros:

s1 Primer scan.

s2 Segundo scan.

min Valor mínimo encontrado a lo largo del eje x.

max Valor máximo encontrado a lo largo del eje x.

void get_scan_y_ends (scan_t *s1*, scan_t *s2*, double * *min*, double * *max*)

Dados un par de scans, calcula el punto más a arriba y más abajo de ambos scans en común, esto es, el mínimo y máximo en el eje y

Parámetros:

s1 Primer scan.

s2 Segundo scan.

min Valor mínimo encontrado a lo largo del eje y.

max Valor máximo encontrado a lo largo del eje y.

void make_angular_histogram (scan_t * *s1*, scan_t * *s2*, int *interval*)

Reserva memoria para los histogramas angulares de los scans que se les pasa como parámetro, y acumula los ángulos en dichos histogramas.

Parámetros:

s1 Primer scan.

s2 Segundo scan.

interval Ancho en grados de cada barra del histograma.

void projection_filter (scan_t * *s1*, scan_t * *s2*, estimation_t *est*)

Gira y mueve s2 hasta donde está s1, según la estimación, y después se eliminan los puntos de s2 que no se pueden ver desde s1. Luego se giran los dos scans el angulo inverso al estimado y se recorta el primer scan con respecto al segundo.

Parámetros:

s1 Primer scan.

s2 Segundo scan.

est Estimacion de desplazamiento y giro del segundo scan con respecto al primero. La estimación x e y es en metros (est.y = 2 es 2 metros, no 2 cm).

Devuelve:

Los dos scans recortados en la posición inicial que estaban

int read_scan_from_file (char * *filename*, scan_t * *scan*)

Lee un scan de un archivo, en el que cada línea es un par de coordenadas (x,y).

Parámetros:

filename Nombre del archivo que contiene el scan.

scan Variable en la que se almacenará el scan leido del archivo.

Devuelve:

0 si todo fue bien, -1 en caso de error.

void read_scan_from_playerc_laser_t (playerc_laser_t * *p*, scan_t * *s*)

Transforma un scan contenido en una estructura de tipo playerc_laser_t en una estructura de tipo **scan_t**(p. 307).

Parámetros:

p Scan de origen.

s Scan de destino.

scan_t rotate_scan (scan_t *s*, double *degrees*)

Rota un scan un cierto ángulo.

Parámetros:

s Scan que va a ser rotado.

degrees Número de grados a rotar. Un número negativo indica que el scan se gira en sentido antihorario (en realidad lo que hace es rotar los ejes de coordenadas en sentido horario, con lo cual el efecto de rotación sobre el scan es un giro antihorario).

Devuelve:

El nuevo scan rotado.

void shifts_calculation (scan_t *s1_orig*, scan_t *s2_orig*, estimation_t *est*, double * *angle_shift*, double * *x_shift*, double * *y_shift*, double * *main_dir*)

Calcula el desplazamiento horizontal y el giro necesario para hacer coincidir el segundo scan con el primero, y que haya las mínimas diferencias posibles. Para superponer el segundo scan encima del primero, es necesario primero rotar y despues trasladar (que no es lo mismo que a la inversa).

Parámetros:

s1_orig Primer scan.

s2_orig Segundo scan.

est Estimación del desplazamiento y giro del segundo scan con respecto al primero.

angle_shift Giro en grados calculado por el algoritmo.

x_shift Desplazamiento en metros en el eje x calculado por el algoritmo.

y_shift Desplazamiento en metros en el eje y calculado por el algoritmo.

main_dir Dirección principal del primer scan.

scan_t translate_scan (scan_t *s*, double *x*, double *y*)

Traslada un scan en los ejes "x" e "y" un cierto valor.

Parámetros:

s Scan que va a ser rotado.

x Número de centímetros a trasladar en el eje x. Positivo es hacia la derecha.

y Número de centímetros a trasladar en el eje y. Positivo es hacia arriba.

Devuelve:
El nuevo scan trasladado.

scan_t trim_scan (scan_t *s1*, scan_t *s2*)

Función que filtra los puntos del scan s2 que no se pueden ver desde el scan s1. Para ello, traza una recta entre el primer y el último punto del scan s1, y elimina los puntos de s2 que estén por debajo de esa recta. Es conveniente que los scans se encuentren mirando hacia el norte, de lo contrario esta función podría no tener efecto.

Parámetros:

s1 Scan con respecto al cual se va a filtrar.

s2 Scan que va a ser filtrado.

Devuelve:
El nuevo scan con los puntos filtrados

int write_scan_to_file (char * *filename*, scan_t *scan*)

Escribe un scan a un archivo, en el que cada línea es un par de coordenadas (x,y). El contenido del archivo es truncado a cero cada vez que se llama a esta función.

Parámetros:

filename Nombre del archivo de destino.

scan Nombre del scan origen.

Devuelve:
0 si todo fue bien, -1 en caso de error.

double x_crosscorrelation (scan_t *s1*, scan_t *s2*, const int *shift*)

Calcula el desplazamiento de crosscorrelacion entre dos scans en el eje x, esto es, la distancia que es necesario trasladar el segundo scan en el eje x para hacerlos coincidir. Trata de encontrar el máximo trasladando el segundo scan (en realidad trasladando el histograma) entre -shift y +shift centímetros para ver en qué momento la superposicion de ambos scans es máxima.

Parámetros:

s1 Primer scan.

s2 Segundo scan.

shift Número de centímetros que se va a trasladar el segundo scan para intentar superponerlo con el primero.

Devuelve:

El desplazamiento en el eje x en centímetros.

double y_crosscorrelation (scan_t *s1*, scan_t *s2*, const int *shift*)

Calcula el desplazamiento de crosscorrelacion entre dos scans en el eje y, esto es, la distancia que es necesario trasladar el segundo scan en el eje y para hacerlos coincidir. Trata de encontrar el máximo trasladando el segundo scan (en realidad trasladando el histograma) entre -shift y +shift centímetros para ver en qué momento la superposicion de ambos scans es máxima.

Parámetros:

s1 Primer scan.

s2 Segundo scan.

shift Número de centímetros que se va a trasladar el segundo scan para intentar superponerlo con el primero.

Devuelve:

El desplazamiento en el eje y en centímetros.

4.4. Referencia del Archivo correlation.h

```
#include "histogram.h"
#include "player.h"
#include "playerc.h"
```

Clases

- struct **estimation_t**
- struct **point_t**
- struct **scan_t**

Definiciones

- #define **DEG2RAD**(x) (x∗PI/180)
- #define **LASER_SAMPLES** 361
- #define **PI** 3.1415926535
- #define **RAD2DEG**(x) (x∗180/PI)
- #define **SMOOTH**

4.4.1. Documentación de las definiciones

#define DEG2RAD(x) (x∗PI/180) /∗∗< Función para transformar de grados a radianes ∗/

Función para transformar de grados a radianes

#define LASER_SAMPLES 361 /∗∗< Máximo numero de valores que devuelve el escáner láser ∗/

Máximo numero de valores que devuelve el escáner láser

#define PI 3.1415926535 /∗∗< Definición de PI ∗/

Definición de PI

#define RAD2DEG(x) (x∗180/PI) /∗∗< Función para transformar de radianes a grados ∗/

Función para transformar de radianes a grados

#define SMOOTH

Valor:

```
10
```

Numero de puntos de distancia que vamos a dejar para calcular el ángulo entre un punto y su siguiente

4.5. Referencia del Archivo eps.c

```
#include <stdio.h>
#include <ghostscript/errors.h>
#include <ghostscript/iapi.h>
#include <gnome.h>
#include "correlation.h"
```

Funciones

- int **eps2pnm** (char *filename, int dimx, int dimy)
- int **write_eps_file_header** (char *filename, int llx, int lly, int urx, int ury)
- int **write_scan_to_eps_file** (char *filename, **scan_t** scan, double xmin, double ymin)

4.5.1. Documentación de las funciones

int eps2pnm (char * *filename*, int *dimx*, int *dimy*)

Transforma un archivo en formato "eps" (postscript encapsulado) en un archivo "pnm" (Portable Anymap File). Concretamente se transforma en "pgm", de tipo 5 (datos en formato binario. El nombre de salida será el mismo que el de entrada, más la extensión ".pnm".

Parámetros:

filename Nombre del fichero "eps" que se va a transformar.

dimx Ancho de la imagen.

dimy Alto de la imagen.

Devuelve:

0 si todo fue correctamente, -1 en caso de error.

int write_eps_file_header (char * *filename*, int *llx*, int *lly*, int *urx*, int *ury*)

Escribe la cabecera de una imagen en formato de postscript encapsulado con las dimensiones indicadas y rellena la imagen de negro. Esto es necesario porque el simulador stage interpreta como obstaculos lo que es blanco sobre fondo negro

Parámetros:

filename nombre del archivo en el que se guardará la imagen

llx lower left x (esquina inferior izquierda, coordenada x)

lly lower left y (esquina inferior izquierda, coordenada y)

urx upper right x (esquina superior derecha, coordenada x)

ury upper right y (esquina superior derecha, coordenada y)

Devuelve:

0 si todo funcionó correctamente, -1 en caso de error.

int write_scan_to_eps_file (char * *filename*, scan_t *scan*, double *xmin*, double *ymin*)

Escribe los puntos correspondientes a un laserscan en un fichero de formato postscript encapsulado. Es necesario indicar el punto más a la izquierda de todos los scans que se van a escribir, y el punto más abajo, para desplazar hacia la derecha y hacia arriba el scan y que no se salga fuera de la imagen.

Parámetros:

filename nombre del archivo en el que se va a almacenar el laserscan.

scan laserscan que se va a almacenar.

xmin coordenada x más pequeÃ177a de todos los scans que se van a escribir. Debe ser un valor negativo.

ymin coordenada y más pequeÃ177a de todos los scans que se van a escribir. Debe ser un valor negativo.

Devuelve:

0 si todo funcionó correctamente, -1 en caso de error.

4.6. Referencia del Archivo histogram.c

```
#include "histogram.h"
#include <stdlib.h>
#include <math.h>
```

Funciones

- int **histogram_accumulate** (**histogram_t** *h, double x)
- int **histogram_alloc** (**histogram_t** **h, size_t n)
- int **histogram_max_bin** (const **histogram_t** *h)
- double **histogram_mean** (const **histogram_t** *h)
- int **histogram_set_ranges_circular** (**histogram_t** *h, double xmin, double xmax)
- void **make_circular** (double range[], size_t n, double xmin, double xmax)

4.6.1. Documentación de las funciones

int histogram_accumulate (histogram_t * *h*, double *x*)

Función que suma una unidad al número de elementos correspondientes a la barra en cuyo rango recae el valor "x".

Parámetros:

h Histograma.

x Valor que va a ser acumulado en el histograma.

Devuelve:

0 si todo fue bien, 1 si "x" esta por encima del máximo rango del histograma, o -1 si "x" está por debajo del mínimo rango del histograma.

int histogram_alloc (histogram_t ** *h*, size_t *n*)

Esta función reserva memoria para un histograma.

Parámetros:

h Histograma para el cual se va a reservar memoria.

n Número de barras del histograma.

Devuelve:

0 si todo fue bien, menor que 0 en caso de error.

int histogram_max_bin (const histogram_t * *h*)

Función que calcula el indice de la barra del histograma que contiene el valor máximo.

Parámetros:

h Histograma.

Devuelve:

Ã141ndice de la barra cuyo valor es el máximo del histograma.

double histogram_mean (const histogram_t * *h*)

Función que devuelve el valor medio del histograma.

Parámetros:

h Histograma.

Devuelve:

El valor medio del histograma.

int histogram_set_ranges_circular (histogram_t * *h*, double *xmin*, double *xmax*)

Esta función establece los rangos de un histograma de tal modo que los rangos asignados a cada barra sean uniformes, entre "xmin" y "xmax". Además pone a cero el contenido de todas las barras.

Parámetros:

h Histograma en el que se van a establecer los rangos.

xmin Valor del mínimo rango.

xmax Valor del rango máximo.

Devuelve:

0 si todo fue bien, -1 en caso de error.

void make_circular (double *range*[], size_t *n*, double *xmin*, double *xmax*) `[static]`

Función que hace que cada rango de cada barra del histograma tenga el mismo tamaÃ177o, sabiendo que hay "n" elementos, el rango mínimo es "xmin" y el máximo "xmax". Además hace que el histograma sea circular: la primera y la última barra son la misma.

Parámetros:

range Array de rangos del histograma.

n Número de elementos del histograma.

xmin Valor del rango mínimo del histograma.

xmax Valor del rango máximo del histograma.

4.7. Referencia del Archivo lasmap.c

```
#include <gnome.h>
#include "main_window.h"
```

Funciones

- int **main** (int argc, char *argv[])

4.7.1. Documentación de las funciones

int main (int *argc*, char * *argv*[])

Función principal de la aplicación. Inicia las librerías del entorno GNOME y crea la ventana principal del programa. Además arranca el bucle principal de eventos, pasando a un estado de espera hasta que el usuario interaccione con la aplicación.

Parámetros:

argc Número de argumentos que se le pasan al programa.

argv Vector que contiene cada uno de los argumentos pasados al programa.

Devuelve:

Esta función devuelve siempre 0.

4.8. Referencia del Archivo lecturalaser.cc

```
#include <stdio.h>
#include <playerclient.h>
#include <gnome.h>
#include <math.h>
#include "lecturalaser.h"
#include "correlation.h"
#include "canvas.h"
#include "bars.h"
#include "eps.h"
#include "main_window.h"
#include "pca.h"
```

Funciones

- void **add_scan_to_list** (**scan_t** s)
- void **align_scans_cb** (GtkWidget *widget, gpointer data)
- int **conectar_a_laser** (char *hostname)
- void **delete_map_list_cb** (GtkWidget *widget, gpointer data)
- void **delete_photoscan_cb** (GtkWidget *widget, gpointer data)
- void **laser_proxy2scan_t** (**scan_t** *s)
- void **laser_subscribe_cb** (GtkWidget *widget, gpointer data)
- void **pca_localize_cb** (GtkWidget *widget, gpointer data)
- gboolean **poll_laser_data** (gpointer data)
- void **real_time_map** (void)
- void **remove_timer** ()
- void **save_map_cb** (GtkWidget *widget, gpointer data)
- void **set_show_laser_state_cb** (GtkWidget *widget, gpointer data)
- void **take_photoscan_cb** (GtkWidget *widget, gpointer data)

Variables

- PlayerClient * **client**
- **scan_t last_scan**

- LaserProxy * **lp**
- **scan_t new_photoscan**
- int **number_of_photoscan** = 0
- **scan_t old_photoscan**
- GList * **scan_list** = NULL
- gboolean **show_realtime_scan** = TRUE
- gint **timer**

4.8.1. Documentación de las funciones

void add_scan_to_list (scan_t *s*)

AÃ177ade un scan a la lista enlazada de scans. Actualiza los valores máximos y mínimos del primer elemento de la lista, esto es, del mapa.

Parámetros:

s Scan que se pretende aÃ177adir a la lista enlazada

void align_scans_cb (GtkWidget * *widget*, gpointer *data*)

Funcion que sirve para alinear dos photoscans. Solicita al usuario una estimación del desplazamiento lateral y angular que se ha realizado al tomar el segundo photoscan con respecto al primero. Redibuja el segundo photoscan en la posición que el algoritmo calcula que coincidirá con el primer photoscan. Se el usuario acepta los valores calculados por el algoritmo, se aÃ177ade el scan al mapa, en caso contrario se debe tomar un nuevo photoscan e intentar nuevamente alinear los photoscans.

Calculamos las acumulaciones de traslación

int conectar_a_laser (char * *hostname*)

Crea un cliente de Player en el servidor que se le indique, y crea un láser proxy por el cual se recibirá la información del láser. Establece el modo de comunicación con Player de modo que sólo se envíen nuevos datos bajo demanda. Finalmente establece un temporizador para leer datos del láser cada 250 ms.

Parámetros:

hostname Nombre del host al cual conectarse.

void delete_map_list_cb (GtkWidget * *widget*, gpointer *data*)

void delete_photoscan_cb (GtkWidget * *widget*, gpointer *data*)

Función de rellamada que se llama cuando se pulsa el botón "borrar último scan". Sólo permite eliminar el segundo scan mostrado en la pantalla.

void laser_proxy2scan_t (scan_t * *s*)

Transforma los últimos datos del láser leidos por Player y los almacena en una estructura **scan_t**(p. 307).

Parámetros:

s Estructura en la que se almacenarán los datos del láser.

void laser_subscribe_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que es llamada desde el menú de la aplicación, para conectarse al escáner láser que tendremos conectado en un puerto serie. Activa y desactiva los elementos del menú convenientemente para que el usuario pueda o no interactuar con ellos.

Parámetros:

∗***widget*** Widget en el que se produjo el evento de menú

data Datos que recibe esta función callback

void pca_localize_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama desde el menú de la aplicación principal. Obtiene el scan en tiempo real en coordendas polares y llama a una función para localizar dicho scan en el mapa, mediante un análisis de componentes principales.

gboolean poll_laser_data (gpointer *data*)

Función que es llamada cada cierto tiempo establecido mediante un temporizador. Lee datos del laser conectado a Player y si son nuevos los datos dibuja un scan en tiempo real en el canvas.

void real_time_map (void)

Función para dibujar un mapa en tiempo real, sin necesitar dar estimaciones

void remove_timer (void)

Elimina el timer para leer datos del láser.

Ver también:
poll_laser_data(p. 342)

void save_map_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que es llamada desde el menú de la aplicación, y se encarga de leer la lista enlazada de scans que forman el mapa, así como sus dimensiones y guardarla en disco como un archivo de postscript encapsulado. Le suma un pequeÃ177o margen al mapa para que quede sin rozar los márgenes de la imagen.

void set_show_laser_state_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama desde el menú. Sirve para mostrar u ocultar el scan en tiempo real.

void take_photoscan_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando se pulsa el botón "fotografiar scan". Cuando se toma el primer photoscan, se dibuja en negro y se aÃ177ade al mapa final. Cuando se toma el segundo, se dibuja en morado y se activan los botones "alinear" y "borrar ultimo scan".

4.8.2. Documentación de las variables

PlayerClient* client

Cliente de Player que se utilizará para la comunicación con el láser.

scan_t last_scan

Ã154ltimo scan disponible: scan en tiempo real.

LaserProxy* lp

LaserProxy del que se leerán los datos que nos proporcione el láser.

scan_t new_photoscan

Photoscan más nuevo.

int number_of_photoscan = 0

Número de photoscan que estamos tomando. 0 es el primero, 1 el segundo

scan_t old_photoscan

Photoscan más antiguo.

GList* scan_list = NULL

lista (doble enlazada) de scans almacenados. El primer elemento contendrá una estructura **map_dim_t**(p. 305), con las dimensiones del mapa, el resto de elementos serán estructuras **scan_t**(p. 307).

gboolean show_realtime_scan = TRUE

TRUE si se debe dibujar el scan en tiempo real. FALSE en caso contrario.

gint timer

Temporizador que se utilizará para sondear si el láser tiene nuevos datos.

4.9. Referencia del Archivo main_window.c

```
#include <gnome.h>
#include "bars.h"
#include "canvas.h"
#include "main_window.h"
```

Funciones

- gboolean **align_check_dialog** (gchar *message)
- gboolean **ask_estimation_dialog** (double *angle, double *x, double *y)
- void **error_message** (const gchar *message)
- gdouble **get_number_from_float_text_entry** (const gchar *string)
- void **info_message** (const gchar *message)
- GtkWidget * **install_main_window** (void)
- gint **main_window_delete_event_cb** (GtkWidget *widget, GdkEvent *Event, gpointer data)
- void **set_status_bar_progress** (gfloat percent)
- void **set_status_bar_text** (const gchar *text)
- void **warning_message** (const gchar *message)

Variables

- GtkWidget * **app**
- GtkWidget * **statusbar**

4.9.1. Documentación de las funciones

gboolean align_check_dialog (gchar * *message*)

Funcion que pregunta al usuario si está de acuerdo con la alineación de los scans decidida por el algoritmo.

Parámetros:

message Mensaje a mostrar en el cuadro de diálogo.

Devuelve:

Devuelve TRUE si se hizo click en OK, FALSE si se pulsó cancel, o se cerró.

gboolean ask_estimation_dialog (double * *angle*, double * *x*, double * *y*)

Muestra un cuadro de dialogo en el que se pide al usuario que introduzca una estimación del desplazamiento y el giro que diferencian al segundo scan del primero. Si se dejan los campos vacíos se toma 0 como valor.

Parámetros:

angle Ángulo en grados introducido por el usuario.

x Desplazamiento en metros en el eje x introducido por el usuario, considerando el desplazamiento antes de producirse el giro estimado.

y Desplazamiento en metros en el eje y introducido por el usuario, considerando el desplazamiento antes de producirse el giro estimado.

void error_message (const gchar * *message*)

Muestra un mensaje de error (error importante). La ventana en la que se muestra es modal. Hasta que no se interactue con el cuadro de diálogo, no se continua con la ejecución del programa.

Parámetros:

message Cadena de texto que contiene el mensaje.

gdouble get_number_from_float_text_entry (const gchar * *string*)

Comprueba si una cadena de texto es un numero en formato de coma flotante.

Parámetros:

string Cadena que se va a comprobar

Devuelve:

El número contenido en la cadena de texto, o 1.0e25 en caso de error.

void info_message (const gchar * *message*)

Muestra un mensaje informativo. La ventana en la que se muestra es modal. Hasta que no se interactue con el cuadro de diálogo, no se continua con la ejecución del programa.

Parámetros:

message Cadena de texto que contiene el mensaje.

GtkWidget* install_main_window (void)

Crea la ventana principal del programa, una zona de visualización (canvas), las barras de estado, de menús y de herramientas.

Devuelve:
Puntero GtkWidget a la ventana principal del programa.

gint main_window_delete_event_cb (GtkWidget * *widget*, GdkEvent * *Event*, gpointer *data*)

Función callback que se llama desde cualquier elemento de la aplicación que se encargue de cerrarla. Muestra un cuadro de diálogo preguntando al usuario si está seguro de abandonar el programa.

Devuelve:
FALSE (si el usuario desea cerrar la aplicación), necesario para abandonar la aplicación. TRUE si el usuario decide no cerrar la aplicación.

void set_status_bar_progress (gfloat *percent*)

Establece el porcentaje de la barra de progreso de la aplicación.

Parámetros:
percent Valor entre 0 y 1, correspondiente al procentaje deseado.

void set_status_bar_text (const gchar * *text*)

Muestra un mensaje en la barra de estado de la aplicación.

Parámetros:
text Cadena de texto que se mostrará en la barra de estado.

void warning_message (const gchar * *message*)

Muestra un mensaje de aviso (error leve). La ventana en la que se muestra es modal. Hasta que no se interactue con el cuadro de diálogo, no se continua con la ejecución del programa.

Parámetros:
message Cadena de texto que contiene el mensaje.

4.9.2. Documentación de las variables

GtkWidget* app

ventana principal de la aplicación.

GtkWidget* statusbar

barra de estado de la aplicación.

4.10. Referencia del Archivo motors.cc

```
#include <playerclient.h>
#include <gnome.h>
#include <stdio.h>
#include "player.h"
#include "playerc.h"
#include "bars.h"
#include "motors.h"
```

Definiciones

- #define **CURVE_PERCENT**

Funciones

- int **conectar_a_position** (char *hostname)
- void **position_palante_cb** (GtkWidget *widget, gpointer data)
- void **position_patras_cb** (GtkWidget *widget, gpointer data)
- void **position_quieto_parao_cb** (GtkWidget *widget, gpointer data)
- void **position_set_speed_cb** (GtkWidget *widget, gpointer data)
- void **position_subscribe_cb** (GtkWidget *widget, gpointer data)
- void **position_turn_down_left_cb** (GtkWidget *widget, gpointer data)
- void **position_turn_down_right_cb** (GtkWidget *widget, gpointer data)
- void **position_turn_up_left_cb** (GtkWidget *widget, gpointer data)
- void **position_turn_up_right_cb** (GtkWidget *widget, gpointer data)
- void **position_turnover_left_cb** (GtkWidget *widget, gpointer data)
- void **position_turnover_right_cb** (GtkWidget *widget, gpointer data)
- void **position_write_speed_command** (void)
- void **set_position_simulated_state_cb** (GtkWidget *widget, gpointer data)

Variables

- double **angular_mod** = 0.0
- double **angular_speed** = 0.0
- double **linear_mod** = 0.0

- double **linear_speed** = 0.0
- const double **max_speed** = 2.0
- double **maxmotorspeed** = 128.0
- double **motor1** = 0.0
- double **motor2** = 0.0
- double **motorspeed** = 0.0
- gboolean **position_simulated** = TRUE
- PositionProxy * **pp**
- PlayerClient * **robot**

4.10.1. Documentación de las definiciones

#define CURVE_PERCENT

Valor:

```
0.50
```

Porcentaje de rotación de uno de los dos motores del robot con respecto al otro. El valor 0.50 indica, por ejemplo que un motor avanzará al 100 por 100 de la velocidad establecida, y el otro motor avanzará al 50 por ciento, produciéndose por tanto una curva en el movimiento del robot.

4.10.2. Documentación de las funciones

int conectar_a_position (char * *hostname*)

Crea un cliente de Player en el servidor que se le indique, y crea un position proxy por el cual se enviará la información de velocidad de los motores. Establece el modo

Parámetros:

hostname Nombre del host al cual conectarse.

void position_palante_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando el usuario desea que el robot se mueva trazando una linea recta hacia adelante, pulsando el botón adecuado en el widget de navegación para el control del robot móvil.

void position_patras_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando el usuario desea que el robot se mueva trazando una linea recta hacia atrás, pulsando el botón adecuado en el widget de navegación para el control del robot móvil.

void position_quieto_parao_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando el usuario desea que el robot se detenga, pulsando para ello el botón "detener" en el widget de navegación para el control del robot móvil.

void position_set_speed_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cada vez que el usuario desea cambiar la velocidad del robot móvil, modificando la barra de escala horizontal proporcionada en el widget de navegación para conseguir tal efecto.

void position_subscribe_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que es llamada desde el menú de la aplicación, para conectarse al dispositivo position del robot. Activa y desactiva los elementos del widget para que el usuario pueda o no interactuar con ellos, dependiendo de si estamos conectados al dispositivo position o no.

Parámetros:

***widget** Widget en el que se produjo el evento de menú

data Datos que recibe esta función callback

void position_turn_down_left_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando el usuario desea que el robot trace una curva hacia la izquierda, al mismo tiempo que se desplaza hacia atrás, pulsando para ello el botón adecuado en el widge de navegación para el control del robot móvil.

void position_turn_down_right_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando el usuario desea que el robot trace una curva hacia la derecha, al mismo tiempo que se desplaza hacia atrás, pulsando para ello el botón adecuado en el widge de navegación para el control del robot móvil.

void position_turn_up_left_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando el usuario desea que el robot trace una curva hacia la izquierda, al mismo tiempo que se desplaza hacia adelante, pulsando para ello el botón adecuado en el widge de navegación para el control del robot móvil.

void position_turn_up_right_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando el usuario desea que el robot trace una curva hacia la derecha, al mismo tiempo que se desplaza hacia adelante, pulsando para ello el botón adecuado en el widge de navegación para el control del robot móvil.

void position_turnover_left_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando el usuario desea que el robot gire sobre si mismo hacia la izquierda, pulsando para ello el botón adecuado en el widget de navegación para el control del robot móvil.

void position_turnover_right_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama cuando el usuario desea que el robot gire sobre si mismo hacia la derecha, pulsando para ello el botón adecuado en el widget de navegación para el control del robot móvil.

void position_write_speed_command (void)

Envia un comando al robot móvil (al simulador o al robot real) utilizando para ello la función SetSpeed del PositionProxy de Player.

void set_position_simulated_state_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama desde el menú de la aplicación principal. Dependiendo de la opción elegida en el menú se establecerá si estamos conectados a un dispositivo position simulado por Stage o a un dispositivo position en un robot real.

Parámetros:

widget GtkWidget desde el que se produjo el evento de menú.

data Vale 1 si la opción elegida es simulador. Vale 2 si la opción elegida es el robot real.

4.10.3. Documentación de las variables

double angular_mod = 0.0

double angular_speed = 0.0

double linear_mod = 0.0

double linear_speed = 0.0

const double max_speed = 2.0

Máxima velocidad que podrá tomar el dispositivo position en Stage

double maxmotorspeed = 128.0

Máxima velocidad que se puede enviar a los motores del robot real

double motor1 = 0.0

Velocidad que se enviará al motor IZQUIERDO del robot real

double motor2 = 0.0

Velocidad que se enviará al motor DERECHO del robot real

double motorspeed = 0.0

Porcentaje de la velocidad máxima que se pueden enviar a los motores del robot real

gboolean position_simulated = TRUE

TRUE si el dispositivo position está simulado por stage, FALSE si el dispositivo position está en un robot real.

PositionProxy* pp `[static]`

PositionProxy al que se enviarán los datos con el movimiento que deseamos que realizen los motores.

PlayerClient* robot [static]

Cliente de Player que se utilizará para la comunicación con el dispositivo position.

4.11. Referencia del Archivo pca.c

```
#include <stdio.h>
#include <cblas.h>
#include <gnome.h>
#include "main_window.h"
```

Funciones

- int **datafile2binaryfile** (char *ifilename, char *ofilename)
- int **localize_scan** (float scan_polar[], int n)
- void **menu_pca_cb** (GtkWidget *widget, gpointer data)
- void **menu_read_pca_cb** (GtkWidget *widget, gpointer data)
- void **normalize_data** (float **matrix, long rows, long cols)
- int **perform_pca** (char *fname)
- int **read_binary_datafile** (char *filename, float **matriz, long *lines, long *it_line)
- int **read_pca_file** (char *filename)
- int **save_matrix2file** (char *filename, float *mat, int n)
- int **ssyev** (char jobz, char uplo, int n, float *a, int lda, float *w, float *work, int lwork)
- void **ssyev_** (char *jobz, char *uplo, int *n, float *a, int *lda, float *w, float *work, int *lwork, int *info)
- int **write_pca2file** (char *filename)

Variables

- float * **feature_vector** = NULL
- float * **final_data** = NULL
- float * **mean** = NULL
- int **num_eigens** = 0
- long **num_muestras** = 0
- int **pca_mat_dim** = 0
- float * **pca_matrix** = NULL

4.11.1. Documentación de las funciones

int datafile2binaryfile (char * *ifilename*, char * *ofilename*)

Toma como entrada el fichero de texto "ifilename", que deberá contener en cada fila 361 lecturas correspondientes a un laserscan ,y lo escribe en el fichero "ofilename", que contendrá los mismos datos pero en formato binario, para que ocupe menos y se lea más rápido.

Parámetros:

ifilename Nombre del fichero de entrada.

ofilename Nombre del fichero de salida.

Devuelve:

0 si todo fue bien, -1 en caso de error.

int localize_scan (float *scan_polar*[ ], int *n*)

Función que localiza un scan dentro de una base de datos de scans para un entorno determinado. Para ello proyecta el scan que se le pasa como parámetro en el espacio vectorial calculado al realizar el análisis de componentes principales. Una vez proyectado, lo compara con todos los scans sintéticos de que disponemos en el entorno que estamos trabajando. El que tenga menor distancia euclidea será el que más se parezca y por tanto será la posición aproximada en la que se encuentra el scan.

Nesecita que estén calculados previamente feature_vector, final_data, el vector media de la matriz original (mean), el número de eigenvectors (num_eigens), el número de scans sintéticos (num_muestras), y la dimensión de la matriz PCA (pca_mat_dim). Todo esto se puede obtener bien habiendo realizado un análisis PCA o leyéndo de un archivo.

Parámetros:

scan_polar Scan en formato polar que se va a tratar de localizar en la base de datos de scans sintéticos.

n Número de elementos del scan en formato polar.

void menu_pca_cb (GtkWidget * *widget*, gpointer *data*)

perform_pca_cb

Función callback que pregunta por un archivo que contenga laserscans en formato binario. Si se cancela, no hará nada. Si se le proporciona un archivo válido realizará el análisis de componentes principales y mostrará un cuadro de diálogo para guardar el contenido del análisis PCA

void menu_read_pca_cb (GtkWidget * *widget*, gpointer *data*)

Función callback que se llama desde el menú de la aplicación principal para leer un archivo en el que se guarden todos los datos de un análisis PCA previo. Muestra un cuadro de diálogo para que el usuario elija un archivo.

void normalize_data (float ** *matrix*, long *rows*, long *cols*)

Toma como entrada matrix, de dimensiones rows*cols, y devuelve en matrix la matriz normalizada, esto es, con la media de cada columna restada a cada elemento de cada fila

Parámetros:

matrix matriz que va a ser normalizada

rows número de filas de matrix

cols número de columnas de matrix

int perform_pca (char * *fname*)

Realiza un análisis de componentes principales sobre un fichero que contenga una serie de scans sintéticos. Cada fila del fichero debe ser un scan en formato polar, pero en formato binario.

Parámetros:

fname Nombre del fichero que contiene los scans.

Devuelve:

0 si todo fue bien, -1 en caso de error.

int read_binary_datafile (char * *filename*, float ** *matriz*, long * *lines*, long * *it_line*)

Lee el archivo binario "filename" que contiene una serie de datos del escáner laser y los guarda en "matriz". Además devuelve en "lines" el número de filas de la matriz y en "it_line" el número de columnas (items por linea).

Parámetros:

filename nombre del archivo que se va a leer

matriz matriz que contiene los datos del fichero "filename"

lines número de filas de la matriz

it_line número de columnas de la matriz

Devuelve:
0 si todo fue bien, menor que cero en caso de error.

int read_pca_file (char * *filename*)

Lee un archivo de datos en el que está almacenado un análisis de componentes principales.

Parámetros:
filename Fichero en el que está almacenado el análisis PCA.

Devuelve:
0 si el fichero era correcto, -1 en caso de error.

int save_matrix2file (char * *filename*, float * *mat*, int *n*)

Toma como entrada la matriz cuadrada "mat", de dimensión n y la guarda en el fichero "filename".

Parámetros:
filename Nombre del fichero en el que se almacenará la matriz.

mat Matriz cuadrada que contiene los datos a almacenar.

n Dimensión de la matriz.

Devuelve:
0 si todo fue bien, -1 en caso de error.

int ssyev (char *jobz*, char *uplo*, int *n*, float * *a*, int *lda*, float * *w*, float * *work*, int *lwork*)

void ssyev_ (char * *jobz*, char * *uplo*, int * *n*, float * *a*, int * *lda*, float * *w*, float * *work*, int * *lwork*, int * *info*)

Interfaz para llamar a la función de lapack (en Fortran) ssyev.

int write_pca2file (char * *filename*)

Escribe en un fichero el contenido del actual análisis PCA realizado. Almacena feature_vector, final_data, el vector media de la matriz original (mean), el número de eigenvectors (num_eigens), el número de scans sintéticos (num_muestras), y la dimensión de la matriz PCA (pca_mat_dim). Todo esto se puede obtener bien habiendo realizado un análisis PCA o leyéndo de un archivo.

El formato del archivo es:

#pca_mat_dim int

#num_eigens int

#num_muestras long

#mean

datos binarios del vector media, terminados en nueva línea.

#feature_vector

datos binarios del vector característico, terminados en nueva línea.

#final_data

datos binarios final_data, terminados en nueva línea.

Parámetros:

filename Nombre del archivo que se va a escribir.

Devuelve:

-1 en caso de error, 0 si todo fue bien.

4.11.2. Documentación de las variables

float* feature_vector = NULL

Vector característico. Cada FILA es un eigenvector (solo hay num_eigens FILAS) el eigenvector más importante se encuentra en la primera fila. Tiene pca_mat_dim columnas.

float* final_data = NULL

Scans sintéticos proyectados en el espacio vectorial. Cada columna de final data es un scan sintetico, pero proyectado en el espacio vectorial. El valor más importante de cada columna es el primero. Las dimensiones de esta matriz son num_eigen filas x num_muestras columnas.

float* mean = NULL

Media de la matriz de datos de scan sintéticos

int num_eigens = 0

Número de eigenvectors necesarios para que se contenga la información de todo el entorno. Un número muy bajo probablemente no represente fielmente el entorno.

long num_muestras = 0

Numero de scans sintéticos de que disponemos.

int pca_mat_dim = 0

Dimensión de la matriz de componentes principales (es simétrica)

float* pca_matrix = NULL

Matriz de componentes principales. Cada FILA es un eigenvector, la última FILA contiene el más importante.

Anexo 5

MANUAL DEL USUARIO

Proyecto LasMap

Manual Del Usuario

Versión 1.0

Julio 2005

Realizado Por	Tutores
Carlos Fernández Caramés	*Vidal Moreno Rodilla* *Belén Curto Diego*

Para

Departamento de Informática y Automática

Universidad de Salamanca

Índice general

Índice de figuras

1 Introducción

El objetivo de este proyecto es presentar una solución práctica al problema del SLAM, utilizando para ello un escáner láser de medición de distancias que vaya montado en un robot móvil. Además de haber construido un robot para el transporte del láser se ha creado una aplicación que permite controlar el robot mediante una interfaz gráfica. Además permite obtener datos del láser y utilizar de manera sencilla las técnicas para la autolocalización absoluta en un entorno conocido, y la localización relativa junto con la construcción de mapas en entornos desconocidos. La aplicación está pensada para funcionar en el entorno Linux/GNOME. Para poder interactuar con el robot móvil, será necesario lanzar el servidor *Player* modificado en el mini-pc incluido en el robot, así como un servidor para controlar los motores en el módulo controlador DK-40. En el presente anexo se describe en primer lugar cómo realizar una instalación de la aplicación, el servidor *Player* modificado, y el servidor DK-40. A continuación se describe cómo utilizar la aplicación en conjunto con los dos servidores.

2 Instalación

2.1. Instalación de la aplicación

La aplicación debe instalarse en un PC de sobremesa. Si se desea instalar la aplicación LasMap será necesario tener instalada la versión 2.0 o superior de GNOME. Se recomienda trabajar con la distribución Fedora Core 2 u otra similar, pero habiendo instalado las herramientas de desarrollo, incluyendo GTK+. Como bibliotecas adicionales se necesitan los paquetes de desarrollo (con las ficheros de cabecera .h) siguientes:

- Biblioteca de cálculo científico BLAS y su interfaz de uso en C, CBLAS.
- Biblioteca de cálculo científico LAPACK (no se necesita CLAPACK).
- Intérprete *ghostscript* de *postscript*.

Para proceder a la instalación de la aplicación se tendrán las fuentes de la aplicación en el archivo comprimido `lasmap-1.0.tar.gz`.

El comando para descomprimirlo es el que se muestra a continuación:

```
$ tar -xzvf lasmap-1.0.tar.gz
```

Ahora ya se puede instalar en el sistema. Bastará poner las siguientes líneas:

```
$ cd lasmap-1.0
$ ./configure
$ ./make
$ ./make install
```

Después de esto se tendrá instalada la aplicación en el disco duro. Si se desea lanzar LasMap, bastará con teclear:

```
$ lasmap
```

2.2. Instalación de *Player/Stage*

El servidor Player debe instalarse en el mini-pc instalado en el robot móvil. Para que la aplicación pueda manejar correctamente el robot móvil a través de *Player*, no basta con instalar una distribución estándar de éste. Por el contrario, se debe instalar el servidor *Player* modificado en su versión 1.5, cuyas fuentes estarán comprimidas en el archivo `playermod-1.5.tar.gz`.

El comando para descomprimirlo es el siguiente:

```
$ tar -xzvf playermod-1.5.tar.gz
```

A continuación se deberá instalar en el sistema, no bastará con una instalación local, ya que la aplicación gráfica desarrollada en este proyecto necesita hacer uso de las bibliotecas cliente de Player, en su versión modificada. Para ello, como super-usuario, se deberán poner las siguientes líneas:

```
# cd playermod-1.5
# ./configure
# ./make
# ./make install
```

Después de esto se tendrá instalado el servidor *Player* modificado para funcionar correctamente con el robot, de modo inalámbrico.

Si se desea instalar *Stage* para comprobar las funciones de la aplicación en un en-

torno simulado, bastará con realizar los mismos pasos anteriores, pero con la versión 1.3.4 de *Stage*, cuyas fuentes comprimidas se pueden encontrar en `stage-1.3.4.tar.gz`. **Para utilizar *Stage* con interfaz gráfica**, antes que *Stage* se deberá descomprimir e instalar la biblioteca `librtk-2.3.0.tar.gz` en el sistema.

2.3. Instalación del servidor en el DK-40

Para poder enviar comandos a los motores, la aplicación gráfica se comunica con el servidor *Player* modificado, y éste a su vez se comunica con el módulo controlador DK-40. Por lo tanto, se necesita un servidor que instalar en el DK-40, y éste se puede encontrar en el CD que acompaña a esta memoria, con el nombre de "sabinasv.exe".

El DK-40 es un servidor que tiene una *shell* tipo MS-DOS. Para instalar "sabinasv.exe" en el DK-40 será necesario enviarlo por ftp. Para ello, desde el directorio en que se encuentre "sabinasv.exe" se debe teclear lo siguiente:

```
$ ftp castanyo.fis.usal.es
A:\ put sabinasv.exe
A:\ quit
```

Donde `castanyo.fis.usal.es` es el nombre de host asignado al DK-40 incluido en el robot.

3

Uso general del sistema

3.1. Lanzamiento de los servidores y la aplicación

El sistema software necesita que se ejecuten simultáneamente tres aplicaciones para poder aprovechar totalmente las funcionalidades implementadas. Esto se debe a que la aplicación gráfica desarrollada necesita comunicarse con el servidor *Player* modificado, y éste, a su vez con el servidor instalado en módulo controlador del DK-40.

3.1.1. DK-40

Por tanto, si se desea utilizar el robot móvil, primero se ha de lanzar el servidor `sabinasv.exe`, instalado en el DK-40. Para ello, se debe hacer lo siguiente:

```
$ telnet castanyo.fis.usal.es
A:\ sabinasv.exe
```

De este modo el servidor en el DK-40 quedará a la espera para recibir órdenes y actuar sobre los motores.

3.1.2. Servidor *Player*

A continuación se debe lanzar, en el mini-pc a bordo del robot móvil, el servidor *Player* modificado, indicándole el fichero de configuración que debe cargar. El fichero de configuración debe tener el siguiente contenido:

```
position:0 (driver ‘‘motor_generico’’ robot ‘‘castanio’’ )
laser:0 ( driver ‘‘sicklms200’’ port ‘‘/dev/ttyS0’’ range_res 10 resolution 50)
```

Con la primera línea se indica que el dispositivo *position* de *Player* va a hacer uso del driver *motor_genérico*, y que el nombre nombre del host del DK-40 es *castanio*. La IP con el nombre del host debe estar dada de alta en el fichero `/etc/hosts` del siguiente modo:

```
212.128.168.119 castanyo.fis.usal.es castanyo castanio
```

La segunda línea del fichero de configuración indica que el dispositivo láser utilizará el *driver* normal de *Player* para acceder al dispositivo láser (esto es, el *driver* sicklms200), y que éste se configurará por el puerto */dev/ttyS0* (com1), con una resolución de alcance de 10 mm (y alcance por tanto de 80 metros) y una resolución angular de 50 * 0.01 = 0.5 grados. Otras configuraciones posibles son `range_res 1` para 8 metros de alcance y 1mm de resolución, y `resolution 100` para 1 grado de resolución angular.

Por lo tanto, una vez creado este fichero de configuración, bastará con lanzar, en el mini-pc, el siguiente comando:

```
$ player nombre_fichero_configuracion.cfg
```

3.1.3. Aplicación gráfica

Una vez que se han lanzado el servidor del DK-40 y el servidor *Player* en el mini-pc, se puede lanzar la aplicación gráfica en el PC de sobremesa. Antes de nada, se recomienda establecer la variable de entorno `$LANG` con un valor distinto al

que por defecto viene en una instalación en español de Fedora Core 2 (su valor es `es_ES.UTF-8`). Se recomienda ejecutar la siguiente instrucción:
`$LANG=es_US.UTF-8`
Ésta instrucción hace que como idioma del equipo de utilice el español de Estados Unidos, de modo que el separador decimal se establezca como un punto, en lugar de una coma, como está por defecto. Por lo tanto a partir de ahora la aplicación entenderá los números en punto flotante como el siguiente: `143.322` y los escribirá en los ficheros también en este formato. Si se introduce un número como por ejemplo `143,322` no lo entenderá como un número en punto flotante, y los resultados pueden ser impredecibles. Otros valores que se ha comprobado que también funcionan son:
`$LANG=es_GT.UTF-8` — Español de Guatemala.
`$LANG=es_MX.UTF-8` — Español de Méjico.

Esto es importante a la hora de introducir las estimaciones de usuario para alinear dos scans, y a la hora de guardar un mapa como una imagen en formato PNM, ya que se utiliza el intérprete `ghostscript` y este solo entiende los números en punto flotante que utilizan como separador decimal un punto.

3.2. Manejo de la aplicación

La aplicación desarrollada no está pensada para ser utilizada por un usuario habitual, sino se ha desarrollado para uso interno del Grupo de Robótica de la Universidad de Salamanca. Por lo tanto se parte de que el usuario tendrá unos avanzados conocimientos de informática.

A continuación se describirá el manejo de la aplicación para realizar las tres tareas principales para las que se desarrolló:

- Control del robot móvil.
- Construcción de un mapa a base de alinear pares de scans.
- Autolocalización basada en análisis de componentes principales.

3.2.1. Control del robot móvil

Para poder manejar el robot móvil, en primer lugar se deberá hacer click en el menú *position* y a continuación seleccionar la opción *suscribirse*. Si todo funcionó correctamente, se habilitará la barra de herramientas de la fig. 3.1 para el control del robot móvil. Si se numeran los botones de la barra de navegación de izquierda a derecha y de arriba abajo, la funcionalidad de cada uno tiene el siguiente efecto sobre el robot.

Figura 3.1: *Barra de herramientas para el control del robot*

1. Avanza hacia adelante y al mismo tiempo gira a la izquierda.
2. Avanza hacia adelante.
3. Avanza hacia adelante y al mismo tiempo gira a la derecha.
4. Sin funcionalidad en estos momentos.
5. Gira sobre si mismo en sentido antihorario.
6. Gira sobre si mismo en sentido horario.
7. Sin funcionalidad en estos momentos.
8. Avanza hacia atrás y al mismo tiempo gira a la izquierda.

9. Avanza hacia atrás.

10. Avanza hacia atrás y al mismo tiempo gira a la derecha.

La barra de desplazamiento colocada bajo la etiqueta "Velocidad del robot" permite establecer la velocidad a la que se moverán los motores del robot, siendo 100 la máxima velocidad de los motores. Si se pulsa el botón "Detener" el robot se detendrá a pesar de que la velocidad no esté puesta a cero. Si se pulsa cualquier otro botón, a continuación, el robot se moverá a la velocidad que estaba establecida.

3.2.2. Construcción de un mapa

Para poder construir un mapa se necesita estar obteniendo los datos del láser en tiempo real. Para ello se deberá hacer click en el menú *Laser* y a continuación seleccionar la opción *suscribirse*. Una vez conectados correctamente, se activará la barra de herramientas para la construcción del mapa de la fig. 3.2.

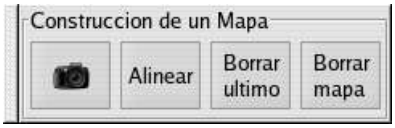

Figura 3.2: *Barra de herramientas para construir mapas*

El botón con el icono de la cámara de fotos permite tomar un photoscan, esto es, fijar de manera permanente los datos del escáner láser en el área de dibujo. El primer photoscan aparece dibujado en color negro. Si se toma un segundo photoscan (se dibuja en color morado) en una posición y orientación distinta, se activan el resto de botones de esta barra de herramientas.

El botón "Alinear" utiliza el algoritmo de correlación cruzada para averiguar la traslación y rotación que se ha efectuado. En caso de pulsar este botón, el sistema le pregunta al usuario por una estimación del movimiento del escáner, como se puede ver en la fig. 3.3. Un valor positivo en x indica que se estima que el láser se ha desplazado a la derecha, mientras que negativo indica que lo ha hecho hacia la izquierda. Un valor positivo en y indica que se estima que el láser se ha desplazado hacia adelante, mientras que negativo indica que lo ha hecho hacia atrás. Finalmente,

un valor positivo en la *Estimación angular* indica que se estima que el láser ha sufrido una rotación en sentido horario, mientras que un valor positivo indica rotación en sentido antihorario. Cabe destacar que la estimación en x e y se debe tener en cuenta antes de que se haya efectuado el giro.

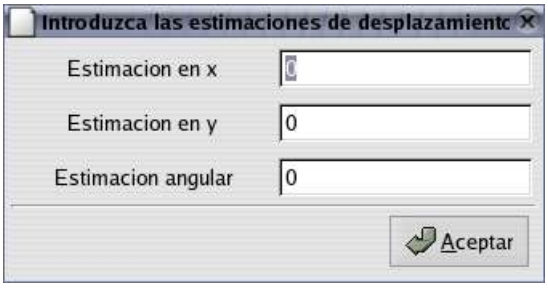

Figura 3.3: *Petición de datos*

Si no se introduce estimación, se podrá calcular un desplazamiento máximo de 1 metro a la izquierda, derecha, adelante o atrás, y una rotación de 60º máximo, en sentido horario o antihorario.

El sistema una vez calculada la alineación traslada el photoscan segundo (el morado) y lo intentará hacer coincidir con el primer photoscan (el negro). A continuación el usuario debe decidir si está conforme con la alineación de los photoscans realizada por el sistema. Para ello deberá observar si las partes comunes del photoscan segundo (morado) y el primero (negro) coinciden a la perfección. Si la alineación es correcta, se añadirá el segundo photoscan al mapa global. El mapa global se dibujará en azul, y además se añadirá un punto rojo con una flecha simulando la posición actual del láser dentro del mapa.

Los otros dos botones tienen la siguiente funcionalidad. El botón "Borrar último" elimina el **segundo** photoscan tomado (el dibujado en morado), en caso de haber cometido un error. El primer photoscan no puede eliminarse con este botón, pero sí se puede hacer mediante el botón "Borrar mapa", el cual, además elimina el mapa global.

En la fig. 3.4 se puede observar el estado de la aplicación en el momento de alinear dos photoscans.

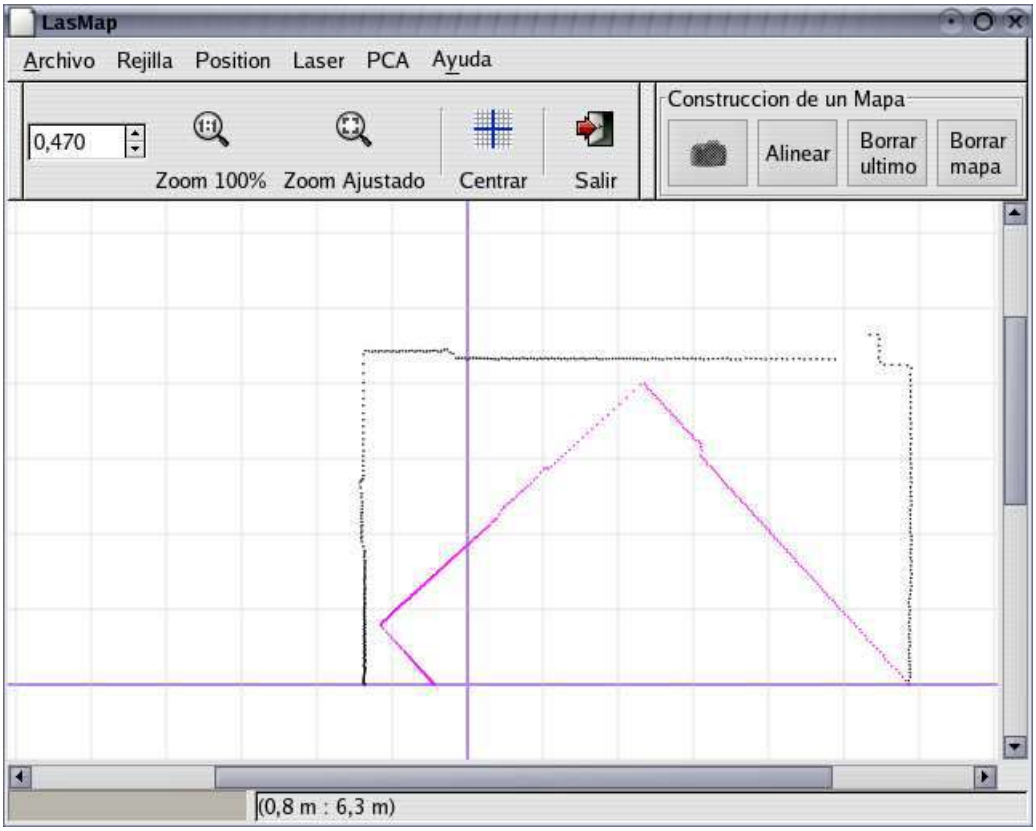

Figura 3.4: *Alineando dos photoscans*

Transformación del mapa en imagen PNM

Para llevar a cabo esta transformación será necesario en primer lugar tener por seguro que el separador decimal del sistema utilizado en los números de punto flotante es un punto y no una coma. Para ello se deberán seguir los pasos descritos en la sección 3.1.3. En segundo lugar se debe seleccionar la opción *Guardar Mapa* del menú archivo. El sistema pedirá un nombre de archivo en el que guardar el mapa, mediante un cuadro de diálogo, y guardará dos archivos, uno en formato EPS y otro en formato PNM. Se recomienda dar un nombre de archivo terminado en la extensión `.eps`. Por ejemplo, al dar al sistema el nombre `mapa.eps` para almacenar el mapa, se guardará un archivo en formato EPS denominado `mapa.eps` y otro en formato PNM denominado `mapa.eps.pnm`.

3.2.3. Uso del simulador *Stage* para construir una cuadrícula densa de *scans* simulados

A la hora de lanzar el simulador *Stage*, se necesita así mismo cambiar el valor de la variable de entorno `$LANG`, tal y como se describe en el apartado 3.1.3 de este anexo.

El usuario debe utilizar el mapa en formato PNM, que se explica como construir y transformar en la sección anterior, y cargarlo en el simulador *Stage*, de modo que le sirva como mapa del entorno en el que realizar las simulaciones. En el CD se proporciona un archivo de configuración de ejemplo para el simulador *Stage*, en el que se carga un mapa PNM, y un dispositivo *laser* y un dispositivo *truth*. Además se proporciona un programa cliente mediante el cual obtener una cuadrícula de *scans* simulados en el mapa cargado, haciendo uso para ello del dispositivo *laser* y del dispositivo *truth*. Este dispositivo permite colocar directamente en cualquier ubicación el escáner láser, sin necesidad de tener que hacer uso del dispositivo *position* para trasladarse de un sitio a otro.

Se recomienda ejecutar *Stage* con la opción -f, de modo que se ejecute tan rápido como pueda. Ésto acelerará en gran medida el tiempo necesario para realizar la simulación.

3.2.4. Autolocalización basada en PCA

En primer, para utilizar esta opción hace falta estar obteniendo los datos del láser en tiempo real. Para ello se deberá hacer click en el menú *Laser* y a continuación seleccionar la opción *suscribirse*.

A continuación se deberá hacer click en el menú *PCA* y elegir una de estas dos alternativas:

- Seleccionar la opción *Calcular PCA*. Entonces el sistema solicita un archivo que tenga una colección de *scans* simulados correspondientes a un determinado entorno. Acto seguido, se llevará a cabo el análisis PCA de dichos *scans*. Esto puede requerir bastante tiempo, dependiendo de la potencia del PC y del tamaño de los datos de entrada. Una vez realizado el análisis PCA, se

preguntará al usuario si desea almacenar los datos del análisis en un fichero, pudiendo aceptar o rechazar la invitación.

- Seleccionar la opción leer archivo con PCA. El sistema carga los datos correspondientes a un anterior análisis PCA, mostrando para ello un cuadro de diálogo al usuario para que seleccione qué archivo cargar.

En cualquiera de estos dos casos, se activará la opción *Localizar* del menú PCA. Si se hace click en esta opción, el sistema obtendrá los datos que está tomando del entorno el escáner láser en ese mismo instante de tiempo. A continuación realizará un cálculo de la posición basándose en el análisis PCA e indicará por pantalla las coordenadas más probables en las que puede estar ubicado el escáner láser.

Glosario

- **API:** ***A****pplication* ***P****rogramming* ***I****nterface* (Interfaz de Programación de Aplicaciones). Es un conjunto de especificaciones de comunicación entre componentes software. Representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software.

- **BLAS:** ***B****asic* ***L****inear* ***A****lgebra* ***S****ubroutines* (Funciones Básicas de Álgebra Lineal). Es una biblioteca de cálculo científico que implementa operaciones vector - vector, vector - matriz y matriz - matriz.

- **CAD:** ***C****omputer* ***A****ided* ***D****esign*. Diseño asistido por ordenador. Consiste en una amplia gama de herramientas informáticas que ayudan tanto a los ingenieros como a los arquitectos a realizar sus tareas de diseño.

- **Ciclo de vida:** Periodo de tiempo cuyo comienzo es el momento en que se decide desarrollar un producto software y su finalización coincide con el momento de pérdida de utilidad por parte del producto desarrollado. Comprende varias fases, las más genéricas son la de análisis, diseño, codificación, pruebas, mantenimiento e incluso retirada.

- **Crosstalk**: En Castellano, interferencia. Es un suceso que ocurre en determinados ambientes (normalmente abarrotados de objetos), cuando las ondas de sonido se reflejan en múltiples objetos, y pueden ser recibidas por otros sensores distintos del que las emitió.

- **Dead reckoning:** Cálculo a ojo. Es una expresión derivada del término naútico deduced reckoning (cálculo basado en inferencia), y era un procedimiento matemático simple para inferir la ubicación actual de un navío haciendo cálculos basados en el rumbo y la velocidad de navegación a lo largo de un período de tiempo.

- **DD:** **D**iccionario de **D**atos.

- **DFD: D**iagrama de **F**lujo de **D**atos.
- **Dispositivo:** Componente electrónico que puede interactuar con un ordenador. Por ejemplo un motor, un láser, un gps...
- **Driver:** Programa que permite que un sistema operativo u otro programa de ordenador interactue (envie y reciba información) con un dispositivo hardware.
- **Encoder**: En castellano se podría traducir como codificador óptico. Consiste en un emisor de luz infrarroja enfrentado a un receptor de infrarrojos. El haz de luz es periódicamente interrumpido por un disco codificado con un patrón de ranuras, que está unido en su centro al eje del motor. Teniendo en cuenta el patrón de codificación del disco y la velocidad de interrupciones del haz de luz, se puede obtener la velocidad a la que está girando el motor.
- **EPS:** ***E****ncapsulated* ***P****ost****S****cript* (PostScript Encapsulado). Formato de imagen basado en el estándar Postscript.
- **GDK:** ***G****raphics* ***D****rawing* ***K****it*. API de escritura de rutinas gráficas.
- **GIMP:** ***G****NU* ***I****mage* ***M****anipulation* ***P****rocess*. Programa de manipulación de gráficos que se distribuye bajo licencia GPL.
- **GNOME:** Acrónimo de ***G****NU* ***N****etwork* ***O****bject* ***M****odel* ***E****nvironment*. Entorno orientado a componentes visuales bajo licencia GPL.
- **GNU:** Acrónimo de ***G****NU is* ***N****ot* ***U****nix*. Conjunto de sistemas operativos clónicos de Unix.
- **GPL:** Acrónimo de ***G****eneral* ***P****ublic* ***L****icense*. Licencia de libre distribución del software.
- **Grados *Rankine*:** El grado *Rankine* tiene su punto de cero absoluto a -460 °F y los intervalos de grado son idénticos al intervalo de grado

Fahrenheit. La relación entre la temperatura en grados Rankine (°R) y la temperatura correspondiente en grados Fahrenheit (°F) es: $Tr = Tf + 460$.

- **GTK+:** Acrónimo de ***G****IMP* ***T****ool****K****it*. API de programación de entornos gráficos con licencia GPL.
- **GUI:** Acrónimo de ***G****raphic* ***U****ser* ***I****nterface*. Conjunto de elementos comúnmente conocidos como ventanas y botones que son los encargados de la comunicación entre el usuario y el sistema.
- **Interfaz:** Especificación que existe entre componentes software, que determina un modo concreto de interacción a través de propiedades de otros módulos software, los cuales abstraen y encapsulan sus datos.
- **IP:** ***I****nternet* ***P****rotocol* (Protocolo de Internet). Pertenece a la capa de red. Es un protocolo orientado a datos usado por los host origen y destino para comunicarse a través de una red de paquetes conmutados; proporciona un servicio de datagramas no confiable.
- **LAPACK:** ***L****inear* ***A****lgebra* ***PACK****age* (Paquete de Álgebra Lineal). Es una biblioteca de cálculo cientifico que implementa operaciones complejas como factorizaciones, cálculo de vectores y valores propios, problemas lineales de cuadrados mínimos, etc.
- **Láser:** ***L****ight* ***A****mplification by* ***S****timulated* ***E****mission of* ***R****adiation* (amplificación de luz mediante la emisión inducida de radiación).
- **Laserscan:** conjunto de datos (medidas de distancia) que proporciona un escáner láser al realizar un barrido del entorno.
- **Lidar:** ***Li****ght* ***D****etection* ***A****nd* ***R****anging.* (detección y medición de distancias mediante luz).
- **LMS:** ***L****aser* ***M****easurement* ***S****ystem*. Sistema de medición basado en láser.

- **Odometría:** Técnica de posicionamiento que emplea información de sensores para obtener una aproximación de la posición real a la que se encuentra un sistema móvil, en un determinado instante, respecto a un sistema de referencia inicial.

- **PCA:** ***P****rincipal* ***C****omponents* ***A****nalysis* (Análisis de Componentes Principales). Es una técnica estadística para descubrir patrones en los datos, de modo que se destaquen sus similitudes y diferencias.

- **Photoscan:** Este término se refiere a tomar una instantánea del entorno que el escáner láser esta percibiendo en un instante determinado y almacenarlo de forma permanente. Es similar a hacer una fotografía con una cámara de fotos: la cámara de fotos está continuamente percibiendo el entorno y cuando hacemos una foto, se almacena de forma permanente, para acto seguido volver a percibir el entorno en tiempo real.

- **Player:** servidor en red multihilo para el control de robots. Forma parte del proyecto de código abierto *Player/Stage* [31].

- **PNM:** formato de imagen que responde a las siglas ***P****ortable a****N****y* ***M****ap file*. Es una abstracción de el subconjunto de formatos que lo componen: *portable bitmap*, *portable greymap* y *portable pixmap*.

- **PGM:** ***P****ortable* ***G****rey* ***M****ap*. Representa una imagen gráfica en modo de escala de grises y tiene dos variantes, formato texto (P2) y formato binario (P5).

- **Radar:** ***Ra****dio* ***D****etection* ***A****nd* ***R****anging* (detección y medición de distancias mediante ondas radioeléctricas)

- **Socket:** Combinación de una dirección IP, un protocolo y un número de puerto. Un *socket* se suele usar en redes de ordenadores para crear un enlace de comunicación bidireccional entre dos programas. El sistema operativo BSD presentó por primera vez los *sockets* en 1983. Cada

socket se asocia a un puerto, el cual permite al protocolo de la capa de transporte (normalmente TCP o UDP) identificar a qué aplicación debe enviar los datos que recibe.

- **Stage:** simulador multi-robot en dos dimensiones. Forma parte del proyecto de código abierto *Player/Stage* [31].

- **SLAM:** ***S****imultaneous* ***L****ocalization* ***A****nd* ***M****apping*. Técnica usada por robots y vehículos autónomos para construir un mapa en un entorno desconocido, mientras al mismo tiempo se mantiene actualizada posición del robot dentro del mapa.

- **TCP:** ***T****ransmission* ***C****ontrol* ***P****rotocol* (Protocolo de control de transmisión). Pertenece a la capa de transporte. Es un protocolo confiable orientado a la conexión que permite que un flujo de bytes originado en una máquina se entregue sin errores (en el mismo orden y sin que falte ninguna parte) en cualquier otra máquina de la red.

- **TOF:** ***T****ime* ***O****f* ***F****light* (tiempo de vuelo). Es el tiempo que tarda un pulso de energía en ir y volver hasta el objeto más cercano con el que choque.

- **Transductor:** Dispositivo que transforma el efecto de una causa física, como la presión, la temperatura, la dilatación, la humedad, etc., en otro tipo de señal, normalmente eléctrica.

Bibliografía

[1] Harlow, Eric. *"Guía Avanzada de Desarrollo de aplicaciones Linux con GTK+ y GDK."* Prentice Hall, 1999.

[2] Lamport, Leslie. *"LaTeX, A Document Preparation System. User's guide and reference manual"*. Digital Equipment Corporation. November, 1994.

[3] Pennington, Havoc. *"GTK+/Gnome Application Developement."* New Riders Publishing, 1999.

[4] Sheets, John R. *"Writing GNOME Applications."* Addison-Wesley, 2001.

[5] Yourdon, Edward. *"Análisis estructurado moderno."* Prentice Hall, 1989.

[6] Ward, Paul T. and Mellor, Stephen J. *"Structures Developement for Real-Time Systems. Volume I: Introduction and Tools."* Yourdon Press/Prentice Hall, 1985.

[7] Borenstein, J., Everett, H.R. y Feng, L. *"Where am I? Sensors and Methods for Mobile Robot Positioning.* Universisty of Michingan, 1996.

[8] Röfer, Thomas. *Using Histogram Correlation to Create Consistent Laser Scan Maps.* Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2002.

[9] Weiss, G. y Puttkamer, E.v. *A map based on laserscans without geometric interpretation.* In Proceedings of Intelligent Autonomous Systems, Vol. 4 (IAS-4), pages 403-407. IOS Press, Karlsruhe, Germany, 1995.

[10] Weiss, G., Weztler, C., Puttkamer, E.v. *Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Range-Finder Scans.* In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1994.

[11] Crowley, J.L., Wallner, F. y Schiele, B. *Position estimation using principal components of range data.* Proceedings 1998 IEEE International Conference on Robotics and Automation, pages 3121–3128, 1998.

[12] Wallner, F. *Position Estimation for a Mobile Robot from Principal Components of Laseer Data.* PhD. thesis, INPG, Oct. 1997.

[13] Smith, Lindsay I. *A tutorial on Principal Components Analysis.* February 26, 2002.

[14] Lu, F. y Milios, E. *Robot pose estimation in unknown environments by matching 2d range scans.* Intelligent and Robotic Systems, 18:249–275, 1997.

[15] Zezhong, X., Jilin, L. y Zhiyu, X.. *Map building and Localization Using 2D Range scanner.* Proceedings 2003 International Symposium on Computational Intelligence in Robotics and Automation, pages 848–853. 2003.

[16] Cox, Ingemar J. *Blanche — An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle.* IEEE Transactions on Robotics and Automation, Vol 7, Nº2, April 1991.

[17] Kirby, M. y Sirovich, L. *Application of the Karhunen-Lokve Procedure for the Characterization of Human Faces.* IEEE Transactions On Pattern Analysis and Machine Intelligence. Vol. 12, Nº. I , January 1990.

[18] Turk, M. y Pentland, A. *Eigenfaces for recognition.* Journal of Cognitive Neuroscience, Vol. 3, Nº. 1, 1991.

[19] Morales Sánchez, Alejandro. *Integración de Robots Móviles en el Servidor Multi-Robot Player.* Trabajo de Grado, Universidad de Salamanca. Julio, 2004.

[20] Oetiker, Tobias, Partl, H., Hyna, I. y Schlegl, E. *"The Not So Short Introduction to LaTeX 2ε".* Versión 4.16, Mayo, 2005.

[21] BLAS - *Basic Linear Algebra Subroutines.*
Sitio web: <http://www.netlib.org/blas/>, 2005.

[22] BLAST - *BLAS Technical Forum.*
Sitio web: <http://www.netlib.org/blast-forum/>, 2005.

[23] LAPACK - *Linear Algebra PACKage*, versión 3.0.
Sitio web: <http://www.netlib.org/lapack/>. Mayo, 2000.

[24] CLAPACK - *f2c'ed version of LAPACK*, versión 3.0.
Sitio web: <http://www.netlib.org/clapack/>. Mayo, 2000.

[25] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorenson. *"LAPACK User's Guide"*. Society for Industrial and Applied Mathematics, third edition, August 1999.
Sitio web: <http://www.netlib.org/lapack/lug/index.html>.

[26] *Numerical Recipes books on-line.*
Sitio web: <http://library.lanl.gov/numerical/index.html>

[27] Grant, Michael C. *"Numerical Linear Algebra Software"*, October 2003.
Sitio web: <http://www.stanford.edu/class/ee392o/nlas.pdf>

[28] Jack Dongarra, Eric Grosse, Petter Bjorstad, Maggie Bowman, David Gay, Tim Hop- kins, and Cleve Moler. *Netlib.*
Sitio web: <http://www.netlib.org>, 2003.

[29] *"A quick reference guide for the BLAS"*. Universidad de Tenesee. Mayo 1997. Sitio web: <http://www.netlib.org/blas/blasqr.ps>

[30] *"Documentation for the C interface to the BLAS"*. *BLAST Forum*. Agosto, 2001. Sitio web: <http://www.netlib.org/blas/blast-forum/cinterface.pdf>

[31] Player/Stage - *The Player/Stage Project*. Sitio web: <http://playerstage.sourceforge.net/>. Junio, 2005.

[32] Durán, A., Bernárdez, B. *Metodología para la elicitación de Requisitos de Sistemas Software v 2.1*, 2000.

[33] Gnome - The Free Software Desktop Project. Sitio web: <http://www.gnome.org>, 2005.

[34] The GNU Operating System. Sitio web: <http://www.gnu.org>, 2005.

[35] GTK+ - The GIMP Toolkit. Sitio web: <http://www.gtk.org>, 2005.

[36] The MathWorks - Matlab & Simulink for Technical Computing Sitio web: <http://www.mathworks.com>, 2005.

[37] Autodesk España Sitio web: <http://www.autodesk.es>, 2005.

[38] Gnuplot Sitio web: <http://www.gnuplot.info>, 2005.

[39] Gnu Scientific Library Sitio web: <http://www.gnu.org/software/gsl/>, 2005.

[40] Ghostscript, Ghostview and Gsview Sitio web <http://www.cs.wisc.edu/~ghost/>, 2005.

[41] TEXnicCenter - LATEX IDE
Sitio web <http://www.toolscenter.org>, 2005.

[42] Controlador MD22 para dos motores de corriente continua.
Sitio web <http://www.superrobotica.com/S310107.htm>, 2005.

[43] Módulo controlador DK-40.
Sitio web <http://www.beck-ipc.com>, 2005.

[44] Wikipedia - The Free Encyclopedia. English version.
Sitio web: <http://en.wikipedia.org>

[45] Wikipedia - La Enciclopedia Libre. Versión en Español.
Sitio web: <http://en.wikipedia.org>