Evaluación y Prueba del Algoritmo FastSLAM de Construcción de Mapas para un Robot Móvil

Julio Delgado Mangas
ULPGC

Noviembre 2007

Universidad de Las Palmas de Gran Canaria
Facultad de informática

PROYECTO: Evaluación y Prueba del Algoritmo FastSLAM de Construcción de Mapas para un Robot Móvil

ALUMNO: Julio Delgado Mangas

TUTOR: Dr. D. Antonio Carlos Domínguez Brito

COTUTOR: Dr. D. Jorge Cabrera Gámez

Agradecimientos

El Proyecto Fin de Carrera supone la culminación de un largo camino, sembrado de dificultades y esfuerzo, pero también de alegrías y recompensas. En este sentido, la primera persona a la que estoy agradecido, esperando que el resto no se sienta ofendido, es a quién hace ya algunos años simplemente me dijo: *"¿Y por qué no?"*. Quizá sin aquella escueta respuesta a mis lamentos, nunca hubiera iniciado de nuevo el camino universitario que en aquél tiempo se me antojaba inalcanzable.

A mis padres, mis hermanos, y mi familia en general, que desde siempre me inculcaron las bondades del camino del saber, y de la búsqueda de la excelencia a través del esfuerzo. Estoy especialmente agradecido a mis padres por soportar el día a día de un trabajador/estudiante, que no es siempre sencillo.

A Pili porque siempre ha estado a mi lado, apoyándome en esta aventura, y siempre acude rauda a mis llamadas. Espero que continúe siendo así en el futuro.

Un agradecimiento especial Antonio, mi tutor, por las muchas horas que ha pasado leyendo y releyendo esta memoria, y por el valor incalculable de sus sugerencias. La sangre nueva que circula por la Universidad tiene mucho que decir, y estoy seguro que tú vas a ser uno de sus líderes. La pasión con la que afrontas tu trabajo es muy motivadora.

A Jorge, mi *otro* tutor, sus preguntas y sugerencias siempre provocan la reflexión y el avance.

A mis profesores, a todos.

Por último, pero no por ello menos importante, agradezco a todos mis compañeros la ayuda prestada, los conocimientos intercambiados, las charlas intrascendentes y las trascendentales, en fin, el haber podido compartir con ellos estos años de sueños, miedos, inquietudes, agobios, etc. Gracias Laura, Samuel, David, Nauzet, Victor, Pedro, Sergio, Álvaro, $\cdots$.

Hay otras muchas personas a las que me gustaría mostrar mi agradecimiento, pero la extensión de los agradecimientos no debe exceder la del resto de la memoria, así que simple-

mente gracias a todos los que habéis formado parte de mi vida.

Índice general

Índice de figuras

Índice de tablas

Abstract

This work presents the test and assessment of the FastSLAM method, an algorithm to solve the SLAM problem. The SLAM (which stands for Simultaneous Localization and Mapping) problem is that of building a map of an environment where a mobile robot is evolving, and at the same time localizing the robot within the map being built.

The FastSLAM [40] is one of the lastest developments in SLAM research and shows several advantages over classical methods based on Kalman filters. Using a clever factorization of the SLAM problem, the complexity of the FastSLAM method is linear with the number of features in the environment, where the complexity of Kalman based filters is exponential. Furthermore the method is able to recover from wrong data association decisions, a problem that cause Kalman SLAM methods to diverge.

In order to achieve the goals of this project, a testbed to test and evaluate the FastSLAM algorithm has been defined. The main element of this testbed is a simulation software we have developed. Using this software we've been able to make tests with different configurations of the key elements involved in the SLAM process.

We've drawn some conclusions out of these tests that increase our knowlege of the SLAM problem in general, and the FastSLAM method in particular. Finally we present a reference configuration for the parameters of the algorithm. This configuration must be used carefully as a begining point in the way to a finer tunning.

Resumen

En el presente trabajo se prueba y evalúa el método FastSLAM, una familia de algoritmos para resolver el problema del SLAM. El problema del SLAM (siglas del inglés Simultaneous Localization and Mapping) consiste en construir un mapa de un entorno en el que se desenvuelve un robot, localizando al mismo tiempo al robot dentro del mapa que se está construyendo.

El FastSLAM es uno de los últimos desarrollos en la investigación del SLAM. Haciendo uso de ciertas propiedades del problema, se consigue factorizarlo de modo que la complejidad del método es lineal con respecto al número de hitos referenciales detectados en el entorno, frente a la complejidad exponencial de los métodos basados en filtros de Kalman. Además el método es capaz de recuperarse frente a asociaciones de datos incorrectas, un hecho que provoca la divergencia de los métodos de SLAM basados en Kalman.

Para alcanzar los objetivos del proyecto se ha definido en entorno para la prueba y evaluación del algoritmo FastSLAM. El elementos más importante del entorno es un software de simulación, que hemos desarrollado a medida. Mediante este software se han realizado pruebas con diferentes configuraciones de los elementos clave del algoritmo de SLAM.

A lo largo del proyecto se han obtenido una serie de conclusiones que aportan conocimiento tanto del problema del SLAM en general, como del algoritmo FastSLAM en particular. Por último se propone una configuración media de los parámetros más importantes del método, que debe ser usada como punto de partida para un ajuste fino de los mismos.

Capítulo 1

Introducción

La construcción automática de mapas mediante el uso de robots móviles es una tarea compleja y difícil. En el presente documento se prueba y evalúa en profundidad uno de los últimos métodos desarrollados para afrontar el problema; la familia de algoritmos FastSLAM. En este capítulo se va a describir someramente el proceso de construcción automática de mapas mediante los algoritmos FastSLAM, identificando los subproblemas que plantea y aportando soluciones a los mismos. Además se van a describir las condiciones en las que se ha llevado a cabo el estudio, ya que al ser éste un problema con alto grado de generalidad, se han aplicado ciertas restricciones para hacerlo tratable. Por último se expone la estructura de todo el documento.

1.1. Introducción

La capacidad de construir automáticamente mapas es una condición necesaria para conseguir desarrollar sistemas robóticos realmente autónomos. La gran capacidad de los seres humanos para desenvolverse en entornos desconocidos se debe, en parte, a la capacidad para crear representaciones mentales de dichos entornos, empleando fundamentalmente el sentido de la vista. Gracias a estas representaciones mentales el ser humano es capaz de reconocer lugares por los que previamente ha pasado, de modo que se sitúa a sí mismo dentro de esta representación, y conociendo dónde se encuentra es capaz de tomar decisiones para alcanzar unos objetivos prefijados

Los algoritmos de construcción automática de mapas persiguen dos objetivos, por un lado la obtención de una representación fiable del entorno, y por otro la localización del propio robot dentro de esa representación. Precisamente una de las características que complica en gran medida la construcción automática de mapas, es la necesidad de resolver ambas cuestiones simultáneamente. Esta simultaneidad ha provocado que se conozca de modo genérico a los métodos de construcción automática de mapas como métodos de SLAM (del inglés Simultaneous Localization and Mapping) o CML (del inglés Concurrent Localization and Mapping).

El principal factor que limita el proceso de SLAM son los ruidos de los sensores que proporcionan información (lo que para el ser humano serían principalmente los ojos y el

equilibrio) tanto del entorno como del propio robot. Con el discurrir de los años se ha comprobado que la mejor herramienta para manejar dichos errores es modelarlos empleando técnicas probabilísticas. Puede decirse que la mayoría de los métodos actuales que afrontan el problema son métodos probabilísticos.

Actualmente existe un gran esfuerzo investigador en esta área de la robótica, y no es de extrañar, dado que la consecución de sistemas robóticos auténticamente autónomos puede abrir las puertas a una infinidad de aplicaciones, y a la definitiva expansión de los robots a todas las facetas de la vida cotidiana. Además, al margen del mundo de la robótica, la capacidad para obtener mapas de un modo automático ofrece innumerables aplicaciones en sectores como la minería, la arquitectura, los servicios de emergencia ante desastres naturales, la exploración de otros planetas, etc.

1.2. El problema

La construcción automática de mapas pretende obtener un mapa o representación de un entorno, y determinar al mismo tiempo la posición en ese mapa del robot. El proceso debe llevarse a cabo a partir de información recibida por el robot mediante sensores. Típicamente se emplean codificadores (generalmente conocidos como *encoders*) que miden la cantidad de giro de cada una de las ruedas, para determinar el movimiento del robot (esta información se denomina *odometría*) , y medidores de rango láser, que recorren una serie de ángulos y devuelven para cada uno la distancia del objeto más cercano al robot, como medidas del entorno. En base a esta información debe proporcionarse un mapa y una posición del robot. Todos los sensores introducen cierto ruido en las medidas que debe ser considerado de alguna manera para construir adecuadamente mapas basados en dichas medidas ruidosas. Las fuentes típicas de error en la odometría son deformidades en las ruedas, deslizamiento o derrape de las mismas, etc. Los medidores de rango láser suelen ser muy precisos, pero presentan problemas frente a superficies reflectantes, como espejos, o superficies transparentes, como cristales.

Si pensamos en la forma en que los seres humanos obtenemos representaciones del entorno, observaremos que lo hacemos empleando conceptos de alto nivel como *pared*, *puerta*, *pasillo*, etc. Por desgracia transformar la información sensorial que recibe un robot es ente tipo de conceptos es aún un problema abierto, sujeto a una intensa investigación. Sin embargo si es posible transformar la información sensorial en elementos geométricos sencillos (generalmente conocidos como hitos referenciales o características) como rectas o esquinas, de hecho existen innumerables métodos para alcanzar estos propósitos. Así a partir de la información sensorial se obtendrían una serie de elementos geométricos sencillos que pasarían a formar parte del mapa, que sería en sí mismo una colección de éstos. Sin embargo estos elementos geométricos sencillos no pueden describir cualquier tipo de entorno. Si bien es posible describir de un modo adecuado un entorno típico de oficina mediante rectas y esquinas, de nada sirven estos hitos para describir un bosque o la superficie de Marte. El tipo de hitos a emplear para la construcción de mapas es otro tema que actualmente recibe mucha atención entre la comunidad investigadora.

Una vez que se obtiene un conjunto de hitos, fruto de una observación, hay que comparalos con el mapa que se está construyendo, para poder identificar si nos encontramos en una zona previamente visitada, o estamos explorando territorio desconocido. Este problema se conoce como *asociación de datos*. Este es sin duda el mayor problema al que se enfrentan todos los algoritmos de construcción automática de mapas. Los hitos detectados contienen información geométrica con respecto a la posición del robot, mientras que los hitos en el mapa se encuentran referidos a un sistema de coordenadas global, de modo que para poder compararlos habrá que emplear la información incierta sobre la posición del robot para realizar un cambio de coordenadas. Generalmente se transforman los hitos del mapa de coordenadas globales a coordenadas relativas al robot. Esta transformación de coordenadas añade incertidumbre al proceso y es posible que se produzcan errores, considerando que un hito observado se corresponde con un determinado hito del mapa, cuando en realidad se corresponde con otro, o no ha sido previamente observado. Esta información de emparejamiento es empleada a su vez para corregir la supuesta posición del robot, de modo que es posible entrar en una espiral en la que errores en la posición inducen errores en la asociación de datos y así sucesivamente.

Existen diversos acercamientos al problema de la construcción de mapas. En nuestro caso nos vamos a centrar en los algoritmos denominados FastSLAM [40] . Los algoritmos de la familia FastSLAM emplean un mecanismo que divide, hasta cierto punto, la determinación de la localización del robot, y la asociación de datos. El algoritmo mantiene una representación interna de múltiples robots, cada uno con su posición y su mapa. Cada una da las representaciones de un robot, con su posición y su mapa, es lo que se conoce como una partícula. Cada vez que el robot recibe información odométrica (por ejemplo: he avanzado 3 m en x y 2 m en y), toma cada partícula y actualiza su posición, teniendo en cuenta la información recibida, y añadiendo una cierta cantidad de ruido, de modo que cada partícula tendrá su propia posición. La posición así determinada se considera cierta al 100 %. Seguidamente se toman los datos de una observación y, partícula a partícula, se comparan con el mapa, es decir, se realiza la asociación de datos. De esta comparación es posible obtener un valor que mide lo mucho o poco que se parece lo observado al mapa de cada partícula. De este modo, empleando este valor de *similitud* se eliminarán del conjunto de partículas aquellas que tengan los valores más bajos, y se clonarán las que tengas los valores más altos, para mantener constante el número de partículas. Este proceso, en el que se eliminan unas partículas y se clonan otras, se denomina remuestreo, y es otro punto fundamental del manejo de las partículas, ya que la selección de partículas *buenas* y *malas* debe asegurar que se mantenga cierta variedad con el objeto de manejar correctamente la incertidumbre en la posición del robot, y de que el algoritmo se autocorrija cuando hay una mala asociación de datos. Así se consigue mantener siempre una población de partículas que explican razonablemente el entorno y que por lo tanto contienen una posición y un mapa próximos a la realidad. Con el tiempo las posiciones de todas las partículas tienden a concentrarse, esto es un indicativo de que el proceso se está llevando a cabo correctamente. Existen dos versiones del algoritmo FastSLAM, la 1.0, objeto de estudio del presente trabajo, y la 2.0 que aporta soluciones a algunos de los problemas conocidos de la versión anterior.

El tratamiento sistemático de los errores e incertidumbres que se encuentran a lo largo de

todo el proceso de la construcción automática de mapas se lleva a cabo empleando técnicas probabilísticas. Por lo general estas técnicas no suelen ser triviales, y requieren de un estudio profundo y sosegado. Por ejemplo el empleo de filtros de partículas en construcción de mapas se inició alrededor del año 2000, sin embargo la técnica en sí se originó en el contexto del proyecto Manhattan, allá por los años 40, para el desarrollo de la primera bomba nuclear [61]. Las aproximaciones probabilísticas a la construcción de mapas más relevantes se basan en el *teorema de Bayes*, enunciado el siglo XVIII por Thomas Bayes. Esto nos puede dar una idea de la amplitud de conocimientos que maneja la denominada robótica probabilística, rama dentro de la cual se encuadran las técnicas de construcción automática de mapas que se analizan en este trabajo.

Por último una cuestión que viene a complicar todo el proceso de construcción de mapas es el hecho del dinamismo de los entornos que se está cartografiando. Generalmente cuando se habla de dinamismo se hace referencia a dos tipos. Por un lado el dinamismo de elementos no estructurales como pueden ser personas u objetos tales como cajas, sillas, etc, que se desplazan o permanecen temporalmente en el entorno. Por otro, el dinamismo de los elementos estructurales, como puertas, armarios, y en general todo el mobiliario susceptible de algún cambio de posición. En general la construcción y navegación en entornos dinámicos es otro de los campos abiertos que actualmente recibe mucho interés por parte de la comunidad investigadora.

1.3. Marco de trabajo

El problema de la construcción automática de mapas es un problema muy general en todos sus aspectos. Empezando por el tipo de plataforma robótica empleada y pasando por los sensores manejados o la tipología de los entornos, es posible encontrar infinidad de configuraciones, cada una de ellas con sus particularidades y problemas propios. Por ello hemos definido unas condiciones de trabajo y unas restricciones que nos han permitido estudiar y comprender algunas de las características de los algoritmos FastSLAM:

- Plataforma robótica: Diferencial . Es un tipo de plataforma robótica muy empleado en investigación. Se caracteriza fundamentalmente por disponer de dos ruedas motrices que es posible controlar independientemente. En concreto empleamos la plataforma Pioneer 3-DX de MobileRobots (empresa anteriormente conocida como ActiveMedia), ya que es una de las más difundidas, y de la que se dispone en el laboratorio.

- Información Odométrica: Velocidades de traslación y rotación. Es posible emplear el ángulo de giro de cada rueda, pero mediante el uso de las velocidades se simplifican los cálculos. La plataforma Pioneer 3-DX proporciona este tipo de información.

- Sensor externo: Medidor de rango láser. Este tipo de sensores están muy difundidos en entornos de investigación además de ser el recomendado por los creadores del método FastSLAM [39, 67]. En concreto se emplea el modelo SICK LMS-200, por ser el más extendido y disponer de él en el laboratorio.

- Entornos: Los entornos en los que va a estudiarse el proceso son entornos de interiores, en los que es posible emplear hitos geométricamente sencillos como rectas y esquinas. Aparte de ser entornos con cierto grado de homogeneidad, existen infinidad de aplicaciones para robots móviles autónomos en este tipo de entornos. Además se considera que los entornos son estáticos.

- Hitos detectables: Rectas infinitas y esquinas. Ya que el tipo de entornos que va a estudiarse se caracteriza por una elevada presencia de rectas y esquinas, es adecuado detectar este tipo de hitos. El empleo de rectas infinitas se debe a su simplicidad de implementación y a ser uno de los más referenciados en la literatura.

- Fuentes de datos: Recorridos reales y simulaciones. Hoy en día existen herramientas que permiten simular el comportamiento de robots en entornos diseñados sintéticamente, de modo que pueden simularse complejos experimentos cuya realización práctica seria costosa. No obstante la ejecución del método en recorridos reales supone una prueba inequívoca de su funcionamiento.

1.4. Estructura del documento

Este documento se organiza en 11 capítulos y un apéndice:

- **Capítulo 1:** Esta introducción.

- **Capítulo 2:** En este capítulo se describe el problema de la construcción de mapas y se revisa la evolución histórica de las soluciones propuestas, para conocer el estado actual. Se describen los principales algoritmos empleados, haciendo énfasis en la familia de algoritmos FastSLAM, y se presenta una revisión bibliográfica sobre la construcción automática de mapas en general, y sobre FastSLAM en particular.

- **Capítulo 3:** Este capítulo muestra en detalle el algoritmo FastSLAM 1.0, que es el que realmente se ha implementado, tanto desde un punto de vista conceptual como matemático. Además se muestran los detalles de la implementación.

- **Capítulo 4:** Dentro de este capítulo se analiza la importancia de la extracción de características o hitos referenciales, además de exponer los algoritmos empleados en el proyecto para tal fin. Por último se exponen los detalles de su implementación.

- **Capítulo 5:** En este capítulo se estudia la importancia del proceso de asociación de datos, así como las aproximaciones más importantes, tanto desde un punto de vista conceptual como matemático. Además se muestran detalles de la implementación.

- **Capítulo 6:** Este capítulo expone un marco matemático y conceptual común a todos los filtros de partículas. Además se muestra como FastSLAM particulariza este marco general. El proceso de remuestreo es también objeto de un estudio extenso. Finalmente se muestran detalles de la implementación.

- **Capítulo 7:** En este capítulo se estudian en profundidad los filtros de Kalman, y se muestran detalles de su implementación, que incluyen el modelado de la plataforma robótica y los sensores.
- **Capítulo 8:** Dado que el proyecto ha sido elaborado atendiendo a los principios de la ingeniería del software, en este capítulo se exponen los resultados de la fase de análisis. Se muestran los requisitos, las herramientas a emplear, las fuentes de datos, y el tipo de resultados que se esperan.
- **Capítulo 9:** En este capítulo se muestran los resultados de la fase de diseño del proyecto, según los principios de la ingeniería del software y el diseño orientado a objetos. Se hace especial énfasis, dentro de todo el entorno de pruebas y evaluación, al diseño del software que ejecuta el algoritmo. Además se comenta la planificación y evolución temporal del desarrollo del proyecto.
- **Capítulo 10:** Este es el capítulo principal del proyecto. En él se estudian uno a uno todos los parámetros identificados en nuestra implementación del algoritmo FastSLAM. Finalmente se propone una configuración media inicial.
- **Capítulo 11:** Este último capítulo es una recopilación de los resultados y conclusiones que se han alcanzado en el presente proyecto. Además se plantean una serie de líneas de trabajo futuro.

Capítulo 2

Estado de la construcción automática de mapas

2.1. Introducción

La construcción de mapas viene siendo un campo de investigación muy activo en el ámbito de la robótica móvil durante las dos últimas décadas. La construcción de mapas afronta el problema de adquirir modelos espaciales de entornos físicos mediante el uso de robots móviles [66], como los que se puede observar en la figura 2.1. El problema de la construcción de mapas se considera uno de los hitos más importantes en el camino hacia robots que sean auténticamente autónomos. Un ejemplo de mapa obtenido automáticamente se muestra en la figura 2.2. A pesar de haberse realizado hasta la fecha avances significativos en el área, aún existen muchos retos por resolver. Hoy en día se dispone de métodos robustos para la construcción de mapas en entornos estáticos, estructurados y de un tamaño limitado, sin embargo, construir mapas de entornos no estructurados, dinámicos y de grandes dimensiones continúa siendo un problema abierto a la investigación.

Prácticamente todos los algoritmos actuales para la construcción automática de mapas mediante robots móviles son probabilísticos [66]. Algunos algoritmos son incrementales y pueden ser ejecutados en tiempo real, mientras que otros requieren múltiples pasadas sobre los datos. Algunos algoritmos requieren información exacta sobre la pose del robot, mientras que otros únicamente requieren información odométrica, muy poco precisa. Algunos algoritmos son capaces de tratar la correspondencia entre los datos referenciados en diferentes momentos, mientras que otros requieren que los hitos o características del entorno (elementos relevantes del entorno, referidos como *landmarks* en la literatura inglesa) contengan una firma que los haga identificables unívocamente.

El presente capítulo presenta una visión general del problema de la construcción automática de mapas, analizando los problemas que plantea y las distintas soluciones desarrolladas a lo largo de la historia, enfatizando las más actuales y exitosas.

La sección 2.2 describe en términos generales el problema de la construcción de mapas, mientras que la sección 2.3 describe detalladamente el problema de la construcción automática de mapas, analizando los factores que dificultan el proceso. La sección 2.4 pre-

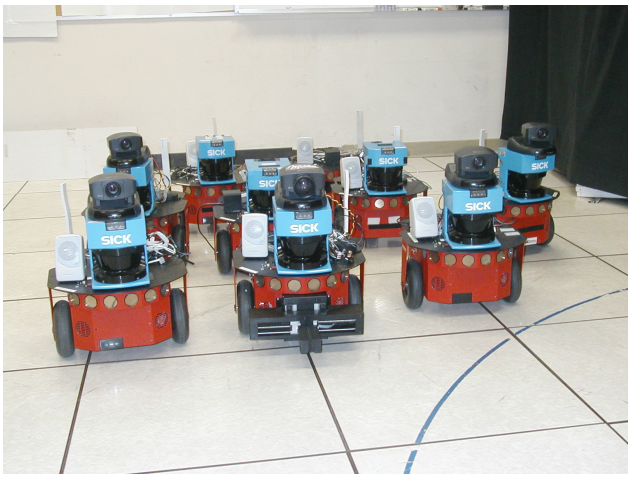


Figura 2.1: Diversas configuraciones de robots de la familia Pioneer, equipados con sensores de rango Láser SICK

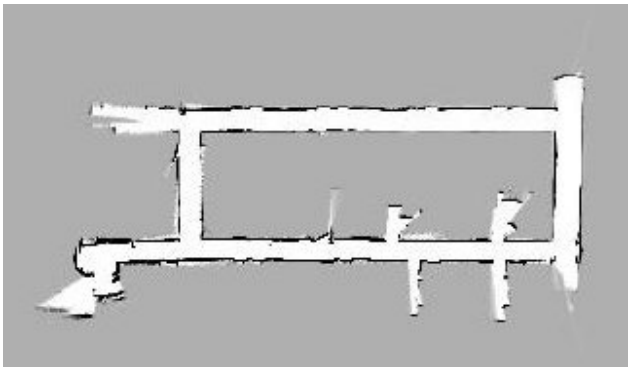


Figura 2.2: Mapa de rejilla construido automáticamente.

senta la evolución histórica de las distintas aproximaciones en la búsqueda de soluciones. Por último, la sección 2.5.1 analiza las soluciones basadas en filtros de Kalman.

2.2. El problema de la construcción de mapas

La cartografía [25, 26] o construcción de mapas, es el estudio y la práctica de la elaboración de representaciones de la tierra sobre una superficie plana. La cartografía combina ciencia, estética y habilidad técnica para crear representaciones equilibradas y útiles, capaces de transmitir información de un modo rápido y eficiente. El proceso cartográfico se basa en la premisa de que el mundo es medible, y de que es posible realizar representaciones fiables, o modelos de esa realidad. La función principal de un mapa es definir, explicar y permitir navegar en un determinado entorno.

En realidad los primeros mapas de los que se tiene constancia son del cielo y no de la tierra. En las cuevas de Lascaux se han encontrado dibujados mapas celestes fechados 16.500 años a.C., en los que aparecen, por ejemplo, las estrellas más brillantes: Vega, Deneb y Altair. Sin embargo, los mapas terrestres más antiguos que existen fueron realizados por los babilonios hacia el 2300 a.C. [15]. Estos mapas estaban tallados en tablillas de arcilla y consistían en su mayor parte en mediciones de tierras realizadas con el fin de cobrar los impuestos. También se han encontrado en China mapas regionales más extensos, trazados en seda, fechados en el siglo II a.C. Los mapas eran raros en el antiguo Egipto. Sin embargo, aquellos que se han conservado muestran un énfasis en la geometría y las técnicas de registro de datos, posiblemente como fruto de la necesidad de volver a establecer los límites exactos de las propiedades tras las inundaciones anuales del Nilo.

Los mapas actuales se basan en la geografía matemática que se inició en la Grecia clásica. Un ejemplo de esta época lo encontramos en el mapa mostrado en la figura 2.3. Aunque los avances cartográficos conseguidos por los griegos llegaron a niveles de perfección que no volvieron a ser igualados hasta el siglo XV, la idea general del mundo de la que partían no era muy distinta de la de los babilonios. Fueron los sabios cosmógrafos, astrónomos y matemáticos los que establecieron las primeras directrices para la representación científica de la superficie terrestre. Destaca la figura de Eratóstenes (siglo III a.C.), quien dividió la Tierra en meridianos y paralelos aunque únicamente trazados sobre lugares bien conocidos y a intervalos irregulares (y no regulares como realizaría posteriormente, en el siglo II a.C. Hiparco de Nicea).

Figura 2.3: El mundo según Herodoto (siglo 5 B.C).

Durante el estancamiento geográfico medieval europeo, los navegantes árabes realizaron y utilizaron cartas geográficas de gran exactitud. Después de un largo periodo de silencio, se inicia un movimiento de recuperación de los clásicos griegos por obra de los árabes en los siglos VIII y IX. A partir de esta última fecha, el mundo islámico produce su propia cartografía, convirtiéndose en el continuador del desarrollo científico antiguo. Estos avances cartográficos llegan principalmente hasta Europa gracias a los intercambios de carácter comercial que se mantienen con los árabes, relaciones que se hicieron más fluidas durante el siglo XIII, provocando un mayor conocimiento por parte de los occidentales del mundo oriental. La gran figura será Al-Idrisi que usó como principal fuente el trabajo de Tolomeo y realizó un mapa del mundo en 1154.

Los avances de la cartografía en Europa fueron posteriores ya que los europeos no comenzaron a buscar nuevas vías de comercio hasta que no vieron cerrarse las rutas con Oriente, produciéndose en ese momento un florecimiento de la elaboración de mapas. El interés que despertó en los grandes reinos cristianos (España y Portugal) hizo que se financiaran grandes empresas marítimas abandonando el punto de vista del teólogo (el más importante durante el medioevo) y tomando en cuenta el del navegante. Surgen así los portulanos, término con el que se designan las cartas náuticas que tuvieron su apogeo desde el siglo XIII al XVI e incluso el XVII. En su origen esta palabra designaba los cuadernos de instrucciones en que los navegantes anotaban los rumbos y las distancias entre los puertos. Entre estos navegantes mediterráneos destacaban los mallorquines.

Figura 2.4: Mapa medieval de Europa.

Los portulanos están relacionados directamente con los modernos mapas. Estos libros de ruta trazaban, generalmente sin meridianos o paralelos, los rumbos principales de acuerdo a los 8 vientos más importantes, estos siempre de color negro. El procedimiento seguido era el de la "Raxon de Marteloio": líneas rectas de rumbo unían los puntos de salida con los de arribo. Estas cartas tenían dos características: sólo las costas se trazaban con cierta exactitud, y las cartas se hallaban siempre entrecruzadas por una red de líneas. Debido a su complejidad gráfica, estas cartas fueron constituyéndose en regalos para reyes y príncipes, hechas por importantes cartógrafos y artistas de la época.

A partir de la introducción del uso de la brújula en el Mediterráneo (finales del s. XIII) y del desarrollo del astrolabio, estas notas adquirieron una precisión cada vez mayor y comenzaron a redactarse libros de derrota en los que se detallaban los rumbos y las distancias. Trasponiendo los datos de estos libros a pergaminos y uniendo los distintos puntos entre sí, se trazaron las primeras cartas náuticas con ciertas garantías, a las que se denominó "cartas portulanas" o "portulanos", como el mostrado en la figura 2.4. No tenían coordenadas pero se trazaban a escala, de tipo lineal, que permitía indicar las distancias entre los distintos puertos en leguas marinas. En el siglo XV un nuevo hecho viene a marcar un avance importante, es el redescubrimiento de Tolomeo, momento a partir del cual la cartografía comenzó a adoptar técnicas más innovadoras que permiten levantar nuevos mapas en la época de los

grandes viajes de exploración. Los europeos cultos volvieron a pensar en una Tierra esférica y combinando las enseñanzas ptolemaicas con las aportadas por los portulanos, se creó el armazón del desarrollo cartográfico renacentista hasta la época de Mercator y Ortelius, quienes pusieron fin al imperio cartográfico de Tolomeo a mediados del siglo XVI.

En 1570, Abraham Ortelius, un cartógrafo flamenco, publicó el primer atlas moderno. Gerhardus Mercator (1512-1594) se hizo internacionalmente famoso en 1554 por un gran mapa de Europa. En un mapamundi del año 1569 utilizó el sistema de proyección de mapas que más tarde se bautizó con su nombre. Mercator sigue considerándose como uno de los mayores cartógrafos de la época de los descubrimientos; la proyección que concibió para su mapa del mundo resultó de un valor incalculable para todos los navegantes. La precisión de los mapas posteriores aumentó mucho debido a las determinaciones más precisas sobre latitud y longitud y a los cálculos sobre el tamaño y forma de la Tierra. Uno de estos mapas se muestra en la figura 2.5.

Figura 2.5: Mapa de Tenerife (siglo XVIII d.C.).

En el siglo XX, la cartografía ha experimentado una serie de importantes innovaciones técnicas. La fotografía área se desarrolló durante la I Guerra Mundial y se utilizó, de forma más generalizada, en la elaboración de mapas durante la II Guerra Mundial. Los Estados Unidos, que lanzaron en 1966 el satélite Pageos y continuaron en la década de 1970 con los tres satélites Landsat, están realizando estudios geodésicos completos de la superficie terrestre por medio de equipos fotográficos de alta resolución colocados en esos satélites. A pesar de los grandes avances técnicos y de los conocimientos cartográficos, quedan por realizar estudios y levantamientos topográficos y fotométricos de grandes áreas de la superficie terrestre que

no se han estudiado en detalle. Como ejemplo de este periodo se muestra en la figura 2.6 un mapa actual de Europa.

Figura 2.6: Mapa actual de Europa.

La tecnología ha sido un factor determinante para el avance en la cartografía. Además de avances en tecnología auxiliares como la aparición de la imprenta, los cambios más importantes se han debido a la aparición y mejora de instrumentos de medida, como la brújula, el astrolabio, el sextante, el láser, etc.

Todos los aspectos comentados nos dan una idea de la complejidad de la cartografía, y de los múltiples conocimientos que se deben integrar para conseguir un mapa de calidad. Además se pone de relieve la utilidad de los mapas como medio de trasmisión y conservación de conocimiento, o como instrumento para la navegación. Para un robot móvil autónomo es imprescindible la capacidad de cartografiar su entorno, de modo que sea capaz de abordar la navegación y la realización de tareas en entornos previamente no explorados. Por último hay que destacar la necesidad de obtener medidas fiables, tanto del entorno, como de la posición del elemento cartografiador, para poder ser capaces de construir mapas útiles.

2.3. El problema de la construcción automática de mapas

La construcción automática de mapas intenta construir de manera automática un modelo espacial del entorno en el que se encuentra un robot, a partir de información sensorial. Para

construir este mapa los robots deben estar equipados con sensores que les permitan percibir el mundo exterior. Sensores de uso habitual son cámaras, medidores de rango láser, como el que se muestra en la figura 2.7, sensores sónar, tecnología infrarroja, radares, sensores táctiles, brújulas y GPS. Sin embargo, todos los sensores están sometidos a cierto error, comúnmente denominado ruido de medición. Aun más importante es el hecho de la limitación de rango de los sensores. Por ejemplo, la luz o el sonido no pueden atravesar paredes. Estas limitaciones obligan al robot a navegar por su entorno para poder construir un mapa del mismo. Los comandos de movimiento (denominados controles) ejecutados durante la exploración del entorno, contienen información importante para la construcción de mapas, ya que están relacionados con los lugares desde los que fueron tomadas diferentes medidas con los sensores a bordo. El movimiento del robot también está sometido a errores, y por ellos los controles son por sí mismos insuficientes para determinar la pose (localización y orientación) de éste, relativa a su entorno.

Figura 2.7: Sensor de rango láser SICK LMS-200.

Uno de los retos más importantes en la construcción automática de mapas surge de la naturaleza del ruido de medición. Los errores de medición son estadísticamente dependientes. Esto se debe a la acumulación de los errores de los controles a lo largo del tiempo y afecta al modo en que se interpretan las medidas futuras. Tratar adecuadamente estos errores sistemáticos es clave para construir mapas correctamente, y es -además- un factor que complica enormemente el problema. Muchos de los algoritmos existentes son, en consecuencia, sorprendentemente complejos, tanto desde un punto de vista matemático, como desde un punto de vista de implementación práctica.

Un segundo factor que viene a complicar el proceso es la alta dimensionalidad del problema. La dimensión hace referencia al número de *entidades* que se representan en el mapa, ya sean puntos, rectas, puertas, ventanas, etc. Para ilustrar la idea, un mapa consistente en una descripción en un plano 2D, los más utilizados en robótica, tiene una dimensión del orden de los millares.

Probablemente el problema más complejo sea la *asociación de datos* o *correspondencia*. Este problema consiste en determinar si medidas tomadas en distintos instantes de tiempo corresponden al mismo objeto físico en el mundo. Este problema esta muy influenciado por el error acumulado en la pose del robot. La mayor parte de los avances científicos en esta materia se han producido en los últimos 10 años [66].

Otro problema a reseñar es el de que raramente un robot se encuentra en un entorno estático. Los entornos dinámicos cambian a lo largo del tiempo, a veces de un modo lento, y otras de un modo extremadamente rápido. En la figura 2.8 se muestra un robot guía de un museo. Los museos son entornos típicamente dinámicos, tanto por la movilidad de los visitantes, como por la posibilidad de cambio en la ubicación de elementos estructurales como puertas o vitrinas de exposición. Desafortunadamente no existen prácticamente algoritmos que permitan aprender mapas en entornos dinámicos. Un modo de abordar el dinamismo consiste en tratar el entorno como si se tratase de uno estático, en ventanas de tiempo muy pequeñas, de modo que la asunción estática sea correcta, y tratar los elementos dinámicos como ruido.

Figura 2.8: Robot guía de un museo, un típico entorno dinámico.

El último reto del problema es la elección de la trayectoria del robot mientras construye un mapa. Esta tarea se denomina *exploración robótica*. Los robots exploradores deben tratar con modelos parciales o incompletos del entorno, de modo que una estrategia viable de exploración debe estar preparada para afrontar contingencias y sorpresas que puedan aparecer durante la adquisición del mapa. El problema de la exploración es un problema de planificación [66] que se afronta habitualmente empleando heurísticas sencillas. A la hora de elegir el lugar al que moverse se deben considerar la ganancia de información esperada, el gasto de energía y tiempo, la posible pérdida de información sobre la pose en el camino, etc. A todo ello hay que añadir que el proceso de construcción de mapas debe llevarse a cabo en tiempo real.

2.4. Desarrollo histórico

La construcción de mapas mediante robots móviles tiene una larga historia. En los años 80 y principios de los 90 el problema estaba dividido en dos aproximaciones, la métrica y la topológica. Los mapas métricos capturan las características geométricas del entorno, mientras que los topológicos describen la conectividad de diferentes sitios. Uno de los primeros métodos métricos es el *algoritmo de rejilla de ocupación* de Elfes y Moravec [12, 13, 41]. Éste representa los mapas mediante rejillas finamente divididas, que modelan el espacio libre y ocupado del entorno, como el mapa de la figura 2.9. Un ejemplo de mapa topológico lo encontramos en el trabajo de Mataríc [35], Kuipers [30]. Los mapas topológicos representan los entornos como una lista de lugares significativos conectados mediante arcos. Los arcos son generalmente anotados con información sobre la manera de navegar de un sitio a otro. Sin embargo, la distinción entre mapas topológicos y métricos ha sido siempre difusa, ya que todos los mapas topológicos se apoyan en información geométrica. En la práctica, los mapas métricos son mapas con un grano más fino que los topológicos. Esta mayor resolución tiene un coste computacional, pero ayuda a solucionar algunos problemas realmente difíciles, como el problema de la correspondencia o asociación de datos mencionado en la sección 2.3.

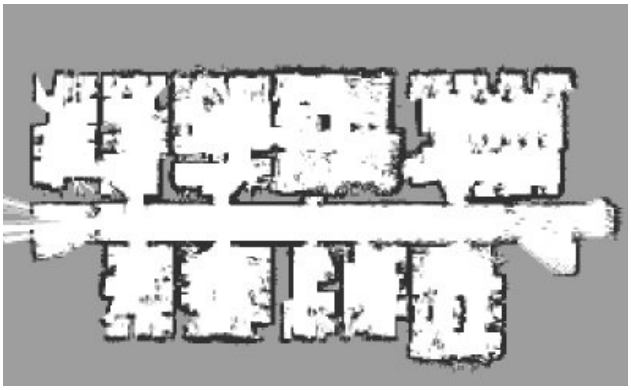


Figura 2.9: Mapa de rejilla. Las zonas blancas indican espacio libre, las zonas negras espacio ocupado y las zonas grises espacio desconocido.

Una segunda taxonomía separa entre *mapas centrados en el mundo* y *mapas centrados en el robot*. Los mapas centrados en el mundo se representan en un sistema de coordenadas global. Las entidades presentes en el mapa no contienen información sobre la medida sensorial que condujo a su descubrimiento. Los mapas centrados en el robot, al contrario, se describen en el espacio de las medidas. En concreto, describen las medidas sensoriales que un robot percibiría en diferentes sitios. Aunque existen aplicaciones específicas para ambos tipos de mapas, hoy en día los dominantes son los mapas centrados en el mundo.

Desde los años 90 la construcción automática de mapas ha sido dominada por técnicas probabilísticas. Una serie de artículos (*papers* en la literatura inglesa) de Smith, Self y Cheeseman [59, 58, 60], introdujeron un marco probabilístico para la resolución simultánea de la construcción de mapas y el problema inducido de localizar el robot de un modo relativo

al mapa que se está construyendo. Desde entonces se conoce a la construcción automática de mapas como *SLAM*(Simultaneous Localization and Mapping) o *CML* (Concurrent Mapping and Localization).

De los algoritmos empleados hoy en día, una familia de algoritmos emplea filtros de Kalman [70, 31] para estimar el mapa y la localización del robot. El mapa obtenido generalmente describe la localización de una serie de hitos referenciales (también denominados marcas o características), o elementos significativos del entorno. Una familia alternativa de algoritmos se basa en el algoritmo de *maximización de la expectación* (*expectation maximization* en la literatura inglesa, generalmente referenciado por la siglas EM) de Dempster [10]. Estos algoritmos afrontan específicamente el problema de la asociación de datos o correspondencia, que consiste en determinar si una medida sensorial tomada en distintos momentos, corresponde a la misma entidad física real. Una tercera familia de técnicas probabilísticas intenta identificar objetos en el entorno, como puertas, paredes, o muebles u otros objetos que pueden cambiar de sitio. Por último existen una serie de algoritmos híbridos que integra elementos de las distintas familias de algoritmos. Una familia exitosa y reciente de algoritmos híbridos la constituyen los denominados algoritmos FastSLAM. Éstos emplean elementos como filtros de Kalman, filtros de partículas y estimaciones de máxima similitud (también denominada máximum likelihood o ML) para abordar el problema del SLAM.

2.5. Algoritmos

2.5.1. Algoritmos basados en filtros de Kalman

Una aproximación clásica para la generación de mapas se basa en filtros de Kalman [70, 31]. Esta aproximación surge a partir de la publicación de una serie de artículos de Smith, Self, y Cheeseman [59, 58, 60], que desde 1985 a 1990 propusieron una formulación matemática para el problema, aún profusamente utilizada hoy en día.

2.5.2. Algoritmos de *Maximización de la Expectación* (EM)

Se trata de una alternativa reciente a los filtros de Kalman [70, 31]. EM es un algoritmo estadístico desarrollado por Dempster, Laird y Rubin [10]. Actualmente es una de las mejores soluciones al problema de la correspondencia en la construcción de mapas.

Los algoritmos EM generan mapas consistentes de entornos grandes y cíclicos, incluso si los hitos presentes son todos parecidos, y no pueden ser distinguidos en base a información sensorial. Sin embargo no retienen una noción completa de incertidumbre, sino que emplean un proceso de búsqueda heurística tipo *ascenso a la colina* (*hill climbing* en la literatura inglesa) o descenso del gradiente, en el espacio de todos los mapas, en un intento por encontrar el más parecido. Para hacer eso necesitan procesar los datos múltiples veces, y por lo tanto no generan mapas de un modo incremental, y tampoco pueden ser ejecutados en tiempo real.

2.5.3. Algoritmos híbridos

Existe una gran cantidad de algoritmos híbridos para la construcción automática de mapas. Éstos integran cálculos probabilísticos, con estimaciones de máxima similitud, más eficientes desde un punto de vista computacional. Destacan en esta categoría los algoritmos FastSLAM [40, 39], que integran la utilización de filtros de Kalman, filtros de partículas y estimaciones de máxima similitud, para conseguir resultados satisfactorios en tiempo real y entornos de gran tamaño.

2.5.4. Mapas de rejilla de ocupación

Todos los algoritmos mencionados anteriormente afrontan el problema desde el desconocimiento de la pose del robot (SLAM). La construcción de mapas cuando se conoce la pose del robot es un problema que también ha sido muy estudiado por la comunidad investigadora. El algoritmo de mapa de rejilla de Elfes y Moravec [12, 13, 41], aparecido a mediados de los 80 ha recibido por ello mucha atención. Un típico mapa de rejilla de ocupación se muestra en la figura 2.10.

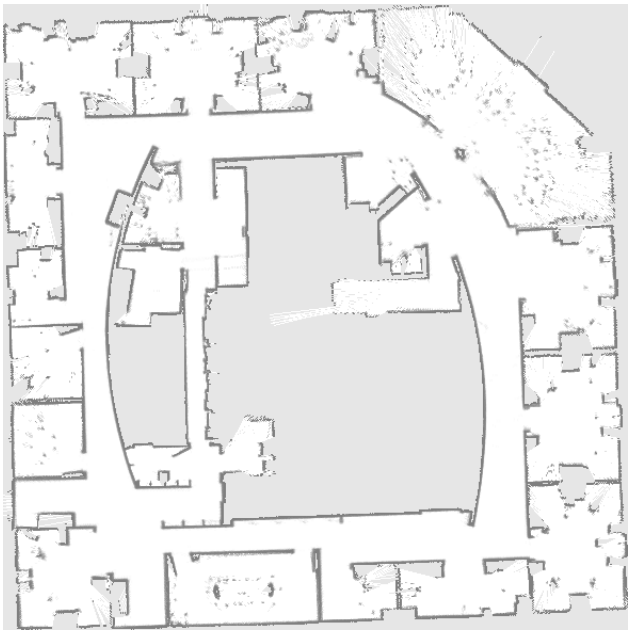


Figura 2.10: Mapa de rejilla. Las zonas blancas indican espacio libre, las zonas negras espacio ocupado y las zonas grises espacio desconocido.

El principal problema que intenta afrontar el algoritmo de mapa de rejilla es el de generar mapas métricos consistentes, a partir de información ruidosa o incompleta. Incluso conociendo la pose del robot, a veces es difícil decidir si un lugar en el entorno está o no ocupado, debido a ambigüedades en los datos del sensor. Las aplicaciones de los mapas de rejilla que

han dado mejores resultados requieren que los robots dispongan de medidores de rango, ya sean láser o sónar. Los mapas de rejilla son mapas probabilísticos.

2.5.5. Mapas de objetos

Otra familia de algoritmos para la construcción automática de mapas aborda el problema de la construcción de mapas compuestos por formas geométricas u objetos, como líneas, paredes, etc. Un ejemplo de este tipo de mapas se muestra en la figura 2.11. Una primera formulación de la idea fue propuesta por Chatila y Laumond [9] que propusieron representar mapas 2D mediante una colección de líneas en vez de mediante rejillas. Sin embargo no propusieron una solución algorítmica práctica. A partir de entonces han aparecido unos pocos algoritmos que usan información sobre objetos para la construcción de mapas.

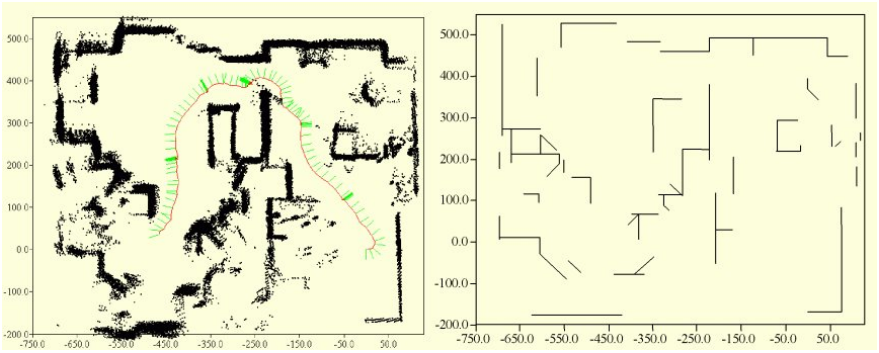


Figura 2.11: A la izquierda puede verse la información percibida por el sensor de rango láser, integrada con respecto al movimiento. A la derecha el mapa geométrico, compuesto por rectas, obtenido a partir de los barridos láser.

Este tipo de mapas presentan cuatro ventajas frente a los mapas de rejilla:

1. Los mapas son más compactos, especialmente si el entorno es estructurado.
2. Pueden ser más precisos.
3. Pueden representar entornos dinámicos.
4. Al encontrase conceptualmente más cercanos a la percepción humana, pueden facilitar la interacción hombre-máquina.

Sin embargo presentan una gran desventaja. Generalmente sólo son aplicables a entornos que pueden expresarse mediante formas geométricas sencillas y objetos.

2.5.6. Construcción de mapas de entornos dinámicos

Los entornos físicos cambian a lo largo del tiempo. La mayoría de los algoritmos para la construcción automática de mapas asumen que el entorno es estático, de modo que son

incapaces de manejar entornos dinámicos. Algunos de los algoritmos pueden ser modificados para manejar ciertos tipos de cambios en el entorno. Por ejemplo, los filtros de Kalman pueden ser adaptados para manejar hitos que se mueven lentamente, tratando dicho movimiento como ruido añadido a la localización de los hitos. De un modo similar los mapas de rejilla pueden soportar cierto tipo de cambio, como el cambio de estado en puertas, que pueden abrirse o cerrarse. De cualquier manera la construcción de mapas en entornos dinámicos ha sido un campo muy pobremente explorado hasta la fecha.

2.6. FastSLAM

La familia de algoritmos FastSLAM aplica filtros de partículas al problema del SLAM. Para manejar la incertidumbre del movimiento del robot, el espacio de los caminos posibles se aproxima mediante una serie de muestras. Cada camino muestreado obtiene su propio mapa del entorno, en el que los distintos hitos se estiman mediante filtros de Kalman extendidos. En este sentido se encuadra dentro de los algoritmos híbridos. El algoritmo resultante es computacionalmente eficiente, y proporciona nuevas aproximaciones al problema de la asociación de datos, absolutamente fundamental para el proceso de SLAM. Este tipo de algoritmos ha permitido la obtención de mapas de un tamaño y precisión sin precedentes, en diferentes dominios de aplicación.

Mediante los algoritmos FastSLAM se solucionan dos de los problemas más graves de los algoritmos clásicos, basados puramente en filtros de Kalman. Uno de los problemas es su elevado coste computacional. El orden $O(n^2)$, siendo n el número de hitos detectados en el entorno, de los algoritmos clásicos, pasa a ser $O(Log(n))$ en FastSLAM. Otro problema de los algoritmos clásicos es la extrema sensibilidad a la asociación de datos, que resulta en la divergencia del filtro en caso de realizar asociaciones erróneas, se sustituye por la posibilidad de mantener múltiples hipótesis. Mediante la utilización de filtros de partículas, los algoritmos FastSLAM permiten que las hipótesis erróneas sean eliminadas del filtro a lo largo del tiempo.

En cuanto a las debilidades del algoritmo hay que mencionar fundamentalmente una que emana directamente de una de sus fortalezas. La posibilidad de eliminar partículas a lo largo del tiempo, de modo que es posible mantener diversas hipótesis de asociación de datos y caminos, provoca una subestimación de la incertidumbre que puede inducir nuevos problemas a la hora de realizar la asociación de datos, y el cierre de grandes bucles.

Aunque existen algunas extensiones para manejar entornos dinámicos mediante algoritmos FastSLAM, aún hay espacio para la investigación en este dominio. Otra vía de desarrollo es la aplicación de los principios de los algoritmos FastSLAM a aproximaciones no basadas en hitos. Un ejemplo de esta última forma de proceder lo encontramos en los denominados algoritmos Grid-Based FastSLAM [40], que emplean un mapa de rejilla de ocupación, en vez de una lista de hitos, para representar el entorno.

2.6.1. Revisión bibliográfica

En primer lugar, para todos aquellos que se acerquen por primera vez al fascinante e inmenso mundo de la robótica móvil, es aconsejable una lectura superficial de *"Introduction to Autonomous Mobile Robots"* [56]. En este libro encontramos una recopilación de tópicos generales relacionados con la robótica móvil y robots autónomos, esto es, tipos de locomoción, cinemática, percepción, localización, construcción de mapas, planificación, y navegación.

Una vez revisados los conceptos fundamentales de la robótica móvil conviene acercase a los métodos probabilísticos aplicados a la misma. Para este propósito, un libro que recopila prácticamente todo el conocimiento actual en este campo es *"Probabilistic Robotics"* [67]. Además de una introducción, algo incompleta, a los conceptos probabilísticos subyacentes, encontramos el desarrollo de modelos probabilísticos de movimiento y medida, un estudio de técnicas probabilistas de localización, construcción de mapas, localización y construcción de mapas simultáneos, estudio de los mapas de rejilla de ocupación, etc. La lectura de Probabilistic Robotics [67] debe ser acompañada de la lectura simultánea de *"Probability, Random Variables, and Stochastics Processes"* [48], libro en el que encontramos una introducción a los procesos estocásticos, base de todo el planteamiento de la robótica probabilística. El libro ofrece una breve introducción a la teoría de la probabilidad matemática y las variables estocásticas, pasando a exponer el concepto de proceso estocástico, y estudiando algunos procesos concretos como las cadenas de Markov en el capítulo 12. Especialmente interesante resulta el capítulo 13 en el que se exponen, entre otras cosas, los filtros de Kalman, como estimadores de procesos estocásticos.

Antes de estudiar específicamente los algoritmos FastSLAM, conviene tener una idea detallada de las implicaciones del proceso de SLAM y de las distintas formas de abordarlo existentes. Para ello el artículo *"Robotic Mapping: A survey"* [66] constituye una fuente ideal, que aborda el problema primero de un modo intuitivo, y luego de un modo matemático, además de mostrar la infinidad de métodos disponibles y ofrecer una ingente cantidad de bibliografía.

Para una descripción matemática más detallada del problema del SLAM desde la perspectiva de los filtros de Kalman es aconsejable consultar la Tesis de Michael Csorba titulada *"Simultaneous Localization and Map Building"* [31], que ofrece en su capítulo 2 un estudio sobre los filtros de Kalman y en su capítulo 3 sobre los filtros de Kalman aplicados al problema del SLAM. Para el lector curioso es aconsejable la lectura del artículo original en el que Kalman describe por primera vez su filtro [28]. Aunque el artículo emplea matemática de muy alto nivel, siempre resulta interesante la lectura del origen de tantos y tantos desarrollos, no sólo en el ámbito de la robótica móvil, sino en el control de infinidad de procesos, que van desde el seguimiento de objetos, hasta el sistema de navegación de las misiones Apolo a la Luna.

Con los conocimientos adquiridos mediante el uso de las fuentes comentadas hasta ahora, el lector se encuentra en situación de afrontar con éxito el estudio y comprensión de la familia de algoritmos FastSLAM. Esta tarea puede abordarse exclusivamente con la lectura de la publicación *"FastSLAM, A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics"* [40]. Este libro es el fruto de la fusión de diversos artículos

de los autores del método [39, 38, 45, 38]. En los capítulo 2 encontramos una extensa exposición del problema del SLAM, con un análisis somero de diversos métodos aplicables para su resolución. En los capítulos 3 y 4 se exponen las versiones 1.0 y 2.0 del algoritmo, respectivamente, destacando para la versión 2.0 los problemas de la versión anterior que resuelve. Ambas versiones se ilustran desde un punto de vista matemático y algorítmico, además de demostrar su funcionamiento mediante la exposición de resultados experimentales. También en el capítulo 4 hallamos extensiones al algoritmo y trabajo futuro.

Dado que es común el empleo de rectas como hitos detectables para la realización del SLAM, es recomendable la lectura de *"A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics"* [44]. Como el título indica, en este artículo encontramos un estudio comparativo de distintos métodos de detección de rectas, además de la descripción algorítmica de cada uno de ellos.

Capítulo 3

FastSLAM

3.1. Introducción

Los algoritmos FastSLAM son una propuesta para abordar el problema del SLAM en tiempo real, desde una perspectiva Bayesiana [45]. Los autores proponen el uso de un filtro de partículas para estimar la pose del vehículo y filtros extendidos de Kalman para estimar la localización de hitos referenciales o características existentes en el entorno.

Históricamente FastSLAM 1.0 fue la primera versión de la familia de algoritmos FastSLAM. Posteriormente se desarrolló la versión 2.0 que introduce una serie de cambios que permiten solucionar algunas de las carencias de la versión anterior. En el presente proyecto se va a desarrollar el algoritmo FastSLAM 1.0 por ser más sencillo de implementar prácticamente.

Una explicación informal del algoritmo es sencilla. Se sabe que es relativamente simple localizar las características (elementos relevantes, como esquinas o paredes) de un mapa, una vez conocido el camino que ha seguido el robot cartografiador [39]. Por este motivo el algoritmo propone evaluar la distribución probabilística del camino seguido por el robot mediante un filtro de partículas, es decir realizando apuestas sobre la pose (x, y, θ). Cada partícula considera su pose apostada como verdadera y realiza un proceso de localización de características, mediante filtros de Kalman extendidos (Extended Kalman Filters o EKF), empleando un filtro por cada característica. Una vez evaluada la bondad de la apuesta, se selecciona un conjunto de partículas "buenas", a partir de las cuales se vuelve a apostar la pose.

En el presente capítulo vamos a analizar la familia de algoritmos FastSLAM. Para ello se estudia desde un punto de vista matemático el proceso de SLAM y el modo en que los problemas que plantea son solucionados por dichos algoritmos, profundizando en su variante 1.0, seleccionada en el presente proyecto por su menor complejidad a la hora de ser implementada prácticamente. Además se esboza un algoritmo para su implementación, cuyos detalles serán desarrollados en capítulos posteriores.

En la siguiente sección se presenta el problema del SLAM desde un punto de vista probabilístico. En la sección 3.3 se desarrolla matemáticamente el algoritmo FastSLAM 1.0. Por último en la sección 3.4 se analizan detalles de la implementación.

3.2. Formulación probabilística del SLAM

Para describir el proceso de SLAM más formalmente nos referiremos al mapa usando la letra Θ. El mapa consiste en una colección de características, cada una de ellas denominada θ_n. Denominamos N al número total de características estacionarias presentes en un mapa. Se define como s_t a la pose del robot, donde t es un índice temporal discreto. La pose de un robot en un sistema de coordenadas cartesianas de dos dimensiones, consiste en su posición (x, y) y en su orientación angular θ. La secuencia $s^t = s_1, s_2, ..., s_t$ denota el camino que ha seguido el robot desde un tiempo inicial hasta un tiempo t.

Para adquirir un mapa el robot percibe el entorno. Esta percepción se lleva a cabo mediante sensores que le permiten percibir el entorno en forma de medidas de rango, distancia, apariencia, etc, de características cercanas. Sin pérdida de generalidad, asumimos que el robot observa únicamente un *hito referencial o característica* (landmark en inglés) en cada momento. La medida en el tiempo t se denota z_t, y puede ser la distancia y ángulo a una determinada característica, relativos al robot. Para cada medida z_t, n_t especifica la identidad de la característica observada, es decir, establece una correspondencia entre la característica observada y uno de los hitos presentes en el mapa. El rango de la variable de correspondencia n_t se encuentra dentro del conjunto finito $\{1, \cdots, N\}$.

En el núcleo del SLAM se encuentra un modelo generativo del sensor, esto es, una ley probabilística que especifica el proceso por el que se adquiere una medida. Llamamos a este modelo "*modelo de medida u observación*" y presenta la siguiente forma:

$$p(z_t|s_t, \theta_{n_t}, n_t) = g(\theta_{n_t}, s_t) + \varepsilon_t \tag{3.1}$$

Como puede observarse, el modelo de medida está condicionado por la pose del robot s_t, la identidad del hito detectado n_t, y la característica concreta que está siendo observada θ_{n_t}. El modelo es gobernado por una función determinista g, distorsionada por ruido. El ruido en el momento t se modela con la variable aleatoria ε_t, que se asume cumple una *distribución normal* [67, 39] de media nula y covarianza R_t. El asumir ruido Gaussiano es una aproximación habitual que funciona bien para un rango amplio de sensores [39, 66]. La función de medida g es usualmente no lineal. Por ejemplo el rango y ángulo de una característica son fácilmente calculables a través de funciones trigonométricas simples, que son no lineales con respecto las coordenadas del robot y de la característica percibida.

Una segunda fuente de información para resolver el problema del SLAM son los controles ejecutados por el vehículo. Los controles se denotan por u_t, y hacen referencia a los comandos ejecutados por los motores en el intervalo $[t-1, t)$. La ley probabilística que gobierna la evolución de la pose del robot se denomina habitualmente "*modelo cinemático o de movimiento*", y presenta la siguiente forma:

$$p(s_t|u_t, s_{t-1}) = h(u_t, s_{t-1}) + \delta_t \tag{3.2}$$

Como se desprende de esta expresión, la pose en tiempo t es una función h de la pose

en un momento anterior s_{t-1} y los controles ejecutados por el robot u_t en el intervalo de tiempo considerado, todo ello distorsionado por ruido Gaussiano. Este ruido se modela con la variable aleatoria δ_t, de media nula y covarianza P_t. Como ocurre para el caso del modelo de medida, la función h es habitualmente no lineal.

El objetivo del SLAM es recuperar el mapa a partir de medidas sensoriales z^t y controles u^t (el superíndice t denota la secuencia desde un tiempo inicial hasta un tiempo t). La mayoría de los algoritmos de SLAM son instancias de *filtros Bayesiano*, y como tales proporcionan para cada instante de tiempo, una distribución de probabilidad sobre el mapa Θ y la pose del robot s_t:

$$p(s_t, \Theta | z^t, u^t, n^t) \tag{3.3}$$

Si esta probabilidad se calcula recursivamente a partir de probabilidades del mismo tipo anteriores en el tiempo, el algoritmo de estimación es un filtro. La mayoría de los algoritmos de SLAM son instancias de filtros Bayesianos que calculan esta distribución a partir de una previamente calculada en el paso anterior:

$$p(s_t, \Theta | z^t, u^t, n^t) = \eta p(z_t | s_t, \Theta, n_t) \int p(s_t | s_{t-1}, u_t) p(s_{t-1}, \Theta | z^{t-1}, u^{t-1}, n^{t-1}) ds_{t-1} \tag{3.4}$$

Aquí η es una variable de normalización (equivalente a $p(z_t | z^{t-1}, u_t, n_t)$ en esta ecuación). El factor de normalización η no depende de las variables sobre las cuales se está calculando la distribución.

El filtro Bayesiano de la ecuación 3.4 se encuentra en el núcleo de la mayoría de los algoritmos de SLAM contemporáneos. En el caso de que g y h sean lineales, equivale al conocido *filtro de Kalman*. Los *filtros de Kalman extendidos* admiten que las funciones g y h sean no lineales, pero las aproximan usando una función lineal, obtenida mediante una expansión de Taylor. Los algoritmos FastSLAM realizan una factorización del filtro, de modo que es posible calcularlo mediante el producto de términos más sencillos.

3.3. FastSLAM 1.0. Desarrollo matemático

La primera impresión es que la distribución mostrada en la ecuación 3.3 captura toda la información relevante para realizar el proceso de SLAM. Sin embargo, existen otras distribuciones más elaboradas que pueden ser estimadas usando SLAM. El algoritmo FastSLAM estima la distribución sobre caminos seguidos por el robot s^t, en vez de hacerlo sobre poses:

$$p(s^t, \Theta | z^t, u^t, n^t) \tag{3.5}$$

A primera vista, estimar la distribución de todo el camino puede parecer una elección cuestionable. A medida que el camino crece, también lo hace el espacio sobre el que se define la distribución. Sin embargo, existen filtros que calculan tan eficazmente la distribución sobre caminos, como sobre poses en un determinado momento [39]. No obstante, la verdadera

motivación para emplear la ecuación 3.5 es que puede ser descompuesta en un producto de términos más pequeños, tal como se muestra en la ecuación 3.6. Esta factorización fue expuesta en primer lugar por Murphy y Russel en 1999 [42]:

$$p(s^t, \Theta | z^t, u^t, n^t) = p(s^t | n^t, z^t, u^t) \prod_{N=1}^{n} p(\theta_n | s^t, n^t, z^t) \tag{3.6}$$

Esta factorización establece que la distribución sobre mapas y caminos puede descomponerse en $N+1$ estimadores recursivos; un estimador para caminos del robot, $p(s^t|n^t, z^t, u^t)$, y N estimadores, uno por cada característica θ_n, $p(\theta_n|s^t, n^t, z^t)$, con $n = 1 \cdots N$, condicionados en la estimación del camino. El producto de estas probabilidades representa la distribución deseada, de un modo factorizado. La derivación matemática de la factorización puede consultarse en [40].

3.3.1. FastSLAM con asociación de datos conocida

El hecho de suponer conocida la asociación de datos significa que en todo momento se conoce la correspondencia unívoca entre los hitos referenciales observados en el entorno y los presentes en el mapa que se está construyendo. De este modo se podrá determinar la observación de un nuevo hito o bien la *reobservación* de uno ya observado anteriormente. Este supuesto rara vez se presenta en situaciones reales [40], sin embargo permite estudiar la estructura del algoritmo de un modo más claro. En la sección 3.3.2 se estudia el algoritmo para el caso en el que no se conoce a priori la asociación de datos.

FastSLAM calcula la distribución de los caminos del robot $p(s^t|n^t, z^t, u^t)$ empleando un filtro de partículas . El filtro de partículas tiene la propiedad de que la cantidad de computación requerida para cada actualización incremental permanece constante, sea cual sea la longitud del camino. Además permite manejar muy adecuadamente modelos no lineales de movimiento de robots. Las restantes N distribuciones, correspondientes cada una a una característica, se calculan mediante filtros de Kalman extendidos (conocidos habitualmente como filtros EKF). Cada filtro EKF estima una única característica, y por lo tanto su dimensión es baja. Cada filtro EKF está condicionado con la pose del robot, de modo que cada partícula posee su conjunto propio de filtros de Kalman. En total existen $(N \cdot M)$ EKFs, siendo M el número de partículas. Cada partícula presenta matemáticamente la siguiente forma:

$$S_t^{[m]} = \langle s^{t,[m]}, \mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]}, ..., \mu_{N,t}^{[m]} \Sigma_{N,t}^{[m]} \rangle \tag{3.7}$$

La notación entre corchetes $[m]$ indica el índice de la partícula; $s^{t,[m]}$ es la estimación del camino, y $\mu_{n,t}^{[m]}$ y $\Sigma_{n,t}^{[m]}$ son la media y la varianza que representan la localización de la n-ésima característica. Todas estas cantidades reunidas constituyen la m-ésima partícula $S_t^{[m]}$ de un total de M en la distribución de FastSLAM.

Filtrar, esto es, calcular la distribución en tiempo t basándose en la distribución en tiempo $t-1$, requiere generar un nuevo conjunto de partículas S_t a partir del conjunto S_{t-1}. Estas nuevas partículas incorporan los nuevos controles u_t y una medida z_t (con la correspondencia asociada n_t). Esta actualización se lleva a cabo en los siguientes pasos:

1. **Extender la distribución del camino muestreando nuevas poses**. FastSLAM toma la pose de una partícula y genera una nueva aplicando el modelo de movimiento:

$$s_t^{[m]} \sim p(s_t|u_t, s_{t-1}^{[m]}) \tag{3.8}$$

   Aquí $s_{t-1}^{[m]}$ es la estimación de la distribución de la pose del robot en tiempo $t-1$, correspondiente a la partícula m-ésima. Aplicando esto a cada partícula se obtiene un nuevo conjunto temporal de partículas.

2. **Actualizar la estimación de las características observadas**. El siguiente paso consiste en actualizar la estimación de la distribución de cada característica, expresada por su media $\mu_{n,t-1}^{[m]}$ y su varianza $\Sigma_{n,t-1}^{[m]}$. Los valores actualizados se añaden al conjunto temporal de partículas, además de la nueva pose.

   La actualización depende de si una característica n ha sido vista o no en tiempo t. Si una característica no ha sido vista, sus parámetros permanecen inalterados. Por el contrario si una característica ha sido vista se actualiza su posición mediante un EKF, que representa al filtro Bayesiano mostrado en la ecuación 3.9.

$$p(\theta_{n_t}|s^t, n^t, z^t) = \mu p(z_t|s_t, \theta_{n_t}, n_t) p(\theta_{n_t}|s^{t-1}, n^{t-1}, z^{t-1}) \tag{3.9}$$

   Si se determina que la observación no se corresponden con ninguna de las características presentes en el mapa, se añade al mismo.

3. **Remuestreo**. Como último paso se remuestrea el conjunto de partículas. A partir de un conjunto de N partículas se seleccionan $M \leq N$ partículas en función del denominado *factor de importancia*, que se explica posteriormente. En caso de ser $M < N$, se clonan algunas de las partículas seleccionadas hasta alcanzar N partículas en el conjunto final. Este tipo de remuestreo se denomina remuestreo con reemplazamiento. La necesidad de remuestrear emana del hecho de que las partículas en el conjunto temporal no están distribuidas de acuerdo con la distribución deseada, ya que el paso 1 genera las nuevas poses s_t de acuerdo únicamente a los controles más recientes, sin prestar atención a la medida z_t. El remuestrear es una técnica común en los filtros de partículas, como mecanismo para corregir el desajuste entre la distribución del filtro y la distribución deseada, y evitar así la degeneración [4, 11].

   Para realizar este proceso se asocia un peso, denominado *factor de importancia*, a cada partícula y se remuestrea de acuerdo a dicho peso. Para determinar el *factor de importancia* de cada partícula, es necesario calcular en primer lugar la distribución de caminos que presentan en realidad las partículas actuales, en la cual no se ha tenido en cuenta la última observación:

$$p(s^{t,[m]}|z^{t-1}, u^t, n^{t-1}) \tag{3.10}$$

   Por otro lado se calcula la distribución objetivo, que sí tiene en cuenta la última

observación:

$$p(s^{t,[m]}|z^t, u^t, n^t) \tag{3.11}$$

El factor de importancia se obtiene del cociente de la distribución objetivo, y la distribución que existe en las partículas:

$$w_t^{[m]} = \frac{p(s^{t,[m]}|z^t, u^t, n^t)}{p(s^{t,[m]}|z^{t-1}, u^t, n^{t-1})} \tag{3.12}$$

Esta expresión, sometida a diversas consideraciones y transformaciones probabilísticas nos conduce a una fórmula realmente computable para el *factor de importancia*:

$$w_t^{[m]} \approx \eta|2\pi Q_t^{[m]}|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - \hat{z}_t^{[m]})^T Q_t^{[m]-1}(z_t - \hat{z}_t^{[m]})\} \tag{3.13}$$

$$Q_t^{[m]} = G_t^{[m]T} \Sigma_{n,t-1}^{[m]} G_t^{[m]} + R_t \tag{3.14}$$

Donde z_t es la observación real, $\hat{z}^{[m]}$ la observación predicha, y $G_t^{[m]}$ es el *jacobiano* del modelo de observación. Los pesos resultantes se emplean para extraer M partículas con reemplazamiento (las más probables reemplazarán a las menos probables) del conjunto temporal. Mediante este proceso de remuestreo las partículas sobreviven en proporción a su *probabilidad de medida*.

Las técnicas de remuestreo (resampling) no son triviales, por ello se estudian más profundamente en la sección 6.3.

Estos tres pasos constituyen la regla de actualización del algoritmo FastSLAM 1.0 para SLAM con asociación de datos conocida. En realidad no es necesario mantener todo el camino de cada partícula, sino únicamente la última pose $s_{t-1}^{[m]}$ que es la que se usa para generar una nueva partícula en tiempo t. Como consecuencia, ni los requerimientos de tiempo, ni de memoria dependen de t.

3.3.2. FastSLAM con asociación de datos desconocida

El algoritmo FastSLAM no cambia mucho cuando la asociación de datos no es conocida a priori, con respecto al algoritmo cuando si se conoce la asociación de datos de antemano. En concreto los pasos del algoritmo son los siguientes:

1. **Extender la distribución del camino muestreando nuevas poses**.
2. **Calcular la asociación de datos**.
3. **Actualizar la estimación de los hitos referenciales observados**.
4. **Remuestrear** el conjunto de partículas.

Existen diversas aproximaciones al problema de la asociación de datos. La más simple de todas se conoce como *correspondencia de máxima similitud (maximum likelihood correspondence, o ML)* [66]. Este método consiste en determinar el valor más probable de la variable de correspondencia, y asumir este valor como verdadero.

Las técnicas de máxima similitud son frágiles si hay muchas hipótesis similares para la variable de correspondencia. Sin embargo, es posible diseñar el sistema para evitar que esto ocurra. Existen básicamente dos aproximaciones para reducir el peligro de realizar una asociación falsa. Primera, seleccionar aquellos hitos que son suficientemente distinguibles de manera que se reduzca significativamente el riesgo de confundirlos. Segunda, asegurarse de que la incertidumbre en la pose de robot se mantiene pequeña.

A pesar de todo lo dicho, lo cierto es que las técnicas de máxima similitud tienen una gran importancia práctica. Dada una observación z_t, un control u_t, una pose del robot s_t y un mapa m constituido por N características representadas por su media y su matriz de covarianza $(\mu_{n,t-1}, \Sigma_{n,t-1})$, el algoritmo para el cálculo de la asociación más probable se muestra en la tabla 3.1. Este algoritmo requiere definir a priori la probabilidad que se va a asignar a aquellos hitos referenciales que se observan por primera vez, P_0.

1:	Algorithm **MaxLikelihood**(u_t, z_t, m)	
2:	for $n = 1$ to N	
3:	$\hat{z}_n = g(\mu_{n,t-1}, s_t)$	// Predicción de medida según el modelo de observación
4:	$G_n = g'(\mu_{n,t-1}, s_t)$	// Calcula Jacobiano
5:	$Q_n = G_n^T \Sigma_{n,t-1} G_n + R_t$	// Covarianza de la medida
6:	$w_n = \lvert 2\pi Q_n \rvert^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - \hat{z}_n)^T Q_n^{-1}(z_t - \hat{z}_n)\}$	// Probabilidad de la correspondencia
7:	endfor	
8:	$w_{N+1} = P_0$	// Probabilidad asociada a un nuevo hito
9:	$\hat{n} = \underset{n=1,\ldots,N+1}{\arg\max}\ w_n$	// Correspondencia más probable
10:	return $\hat{n}$	
11:	endAlgorithm	

Tabla 3.1: Cálculo de la asociación de datos más probable.

La derivación matemática del algoritmo puede encontrarse en diversas publicaciones de Thrun [39, 67].

3.4. Detalles de implementación

En la figura 3.2 se muestra un resumen del algoritmo FastSLAM 1.0 con asociación de datos desconocida. Sin pérdida de generalidad se asume que se observa únicamente una característica en cada iteración [40]. Fundamentalmente se trata de la gestión de un filtro de partículas. Para cada partícula se realizan una serie de pasos. En primer lugar se calcula la asociación de datos (consultar sección 5) más probable. Se actualiza el mapa de la partícula teniendo en cuenta si la asociación de datos ha determinado que el hito observado es nuevo,

o un hito ya presente en el mapa, y se añade la partícula a un conjunto temporal. Una vez actualizadas todas las partículas, se obtiene el conjunto definitivo de partículas, mediante remuestreo del conjunto temporal. Este último paso no tiene por qué ejecutarse siempre, ya que es posible condicionar su ejecución a un umbral del número de partículas efectivas N_{eff} (esto se explica detalladamente en la sección 6.3).

Algorithm FastSLAM 1.0(z_t, u_t, S_{t-1})

$S_t = S_{aux} = 0$

for $m = 1$ to M // Recorrer todas las partículas

tomar $\langle s_{t-1}^{[m]}, N_{t-1}^{[m]}, \langle \mu_{1,t-1}^{[m]}, \Sigma_{1,t-1}^{[m]} \rangle, \cdots, \langle \mu_{N_{t-1}^{[m]},t-1}^{[m]}, \Sigma_{N_{t-1}^{[m]},t-1}^{[m]} \rangle \rangle$ de S_{t-1}

$s_t^{[m]} \sim p(s_t | s_{t-1}^{[m]}, u_t)$ // Muestrear una nueva pose

for $n = 1$ to $N_{t-1}^{[m]}$ // Similitud de la observación

$\hat{z}_n = g(\mu_{n,t-1}^{[m]}, s_t^{[m]})$ // Predicción de la medida

$G_n = g'(\mu_{n,t-1}^{[m]}, s_t^{[m]})$ // Calcula Jacobiano

$Q_n = G_n^T \Sigma_{n,t-1}^{[m]} G_n + R_t$ // Covarianza de la medida

$w_n = |2\pi Q_n|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - \hat{z}_n)^T Q_n^{-1}(z_t - \hat{z}_n)\}$ // Similitud de la asociación

endfor

$w_{N_{t-1}^{[m]}+1} = P_0$ // Similitud de una nueva observación.

$\hat{n} = \underset{n=1,\ldots,N_{t-1}^{[m]}+1}{\arg\max} \; w_n$ //Correspondencia de máxima similitud

$N_t^{[m]} = max\{N_{t-1}^{[m]}, \hat{n}\}$ // Nuevo número de hitos en el mapa

for $n = 0$ to $N_t^{[m]}$ // Actualizar filtros de Kalman

if $n = N_{t-1}^{[m]} + 1$ // ¿Es un nuevo hito?

$\mu_{n,t}^{[m]} = g^{-1}(z_t, s_t^{[m]})$ //

$\Sigma_{n,t}^{[m]} = G_n^{-1} R_t (G_n^{-1})^T$ // Covarianza inicial

else if $n = \hat{n}$ // ¿Hito observado de nuevo?

$K = \Sigma_{n,t-1}^{[m]} G_{\hat{n}} Q_{\hat{n}}^{-1}$ // Ganancia de Kalman

$\mu_{n,t}^{[m]} = \mu_{n,t-1}^{[m]} + K(z_t - \hat{z}_t)^T$ // Actualización de la media

$\Sigma_{n,t}^{[m]} = (I - K G_{\hat{n}}^T) \Sigma_{n,t-1}^{[m]}$ // Actualización de la covarianza

else // Hitos no observados

$\mu_{n,t}^{[m]} = \mu_{n,t-1}^{[m]}$ // Se copia la media

$\Sigma_{n,t}^{[m]} = \Sigma_{n,t-1}^{[m]}$ // Se copia la covarianza

endif

endfor

$w_t^{[m]} = w_{\hat{n}}$ // Se añade la partícula al conjunto auxiliar S_{aux}.

añadir $\langle s_t^{[m]}, N_t^{[m]}, \langle \mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]} \rangle, \cdots, \langle \mu_{N_t^{[m]},t}^{[m]}, \Sigma_{N_t^{[m]},t}^{[m]}, w_t^{[m]} \rangle \rangle$ a S_{aux}
endfor
$S_t = 0$ // Nuevo conjunto de partículas
for $m = 1$ to M // Remuestrear M partículas
seleccionar de S_{aux} índice m con probabilidad $\propto w_t^{[m]}$
añadir $\langle s_t^{[m]}, N_t^{[m]}, \langle \mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]} \rangle, \cdots, \langle \mu_{N_t^{[m]},t}^{[m]}, \Sigma_{N_t^{[m]},t}^{[m]} \rangle \rangle$ de S_{aux} a S_t
endfor
return S_t
endAlgorithm

Tabla 3.2: Resumen del algoritmo FastSLAM 1.0 con asociación de datos desconocida. Se asume la observación de una única característica en cada iteración.

En el algoritmo anterior, una partícula en el momento previo a la ejecución del algoritmo se simboliza por $\langle s_{t-1}^{[m]}, N_{t-1}^{[m]}, \langle \mu_{1,t-1}^{[m]}, \Sigma_{1,t-1}^{[m]} \rangle, \cdots, \langle \mu_{N_{t-1}^{[m]},t-1}^{[m]}, \Sigma_{N_{t-1}^{[m]},t-1}^{[m]} \rangle \rangle$, donde $s_{t-1}^{[m]}$ es la pose, $N_{t-1}^{[m]}$ el número de hitos del mapa, y $\langle \mu_{i,t-1}^{[m]}, \Sigma_{i,t-1}^{[m]} \rangle$, la representación de un hito, donde $\mu_{i,t-1}^{[m]}$ es la media del estado, y $\Sigma_{i,t-1}^{[m]}$ la covarianza, para $i = \{1, \cdots, N_{t-1}^{[m]}\}$.

Capítulo 4

Extracción de características

4.1. Introducción

Para un robot móvil es importante conocer la posición que ocupa, ya sea en un entorno conocido, del que se dispone de un mapa, o desconocido. Una estimación precisa de la posición se encuentra en el corazón de cualquier sistema de navegación, tal como la localización, la construcción dinámica de mapas, o la planificación de trayectorias. La odometría no es suficiente para una estimación precisa de la posición, debido al error ilimitado que induce [67]. El problema se afronta haciendo uso de distintos sensores extereoceptivos [44], como láser, sónar, etc. El sensor más empleado actualmente es el medidor de rango láser 2D. Su aplicaciones varían desde la localización, hasta la construcción dinámica de mapas o la evitación de colisiones con obstáculos [44]. El medidor de rango láser presenta muchas ventajas con respecto a otros sensores: produce medidas de rango más densas y precisas, presenta una frecuencia de muestreo muy elevada y una alta resolución angular [44]. Por estos motivos y por disponer de este tipo de sensores en el laboratorio, el medidor de rango láser va a ser el sensor empleado en el presente proyecto.

La principal tarea de la localización consiste en hacer corresponder los datos percibidos con la información contenida en un mapa a priori, o registrada hasta el momento presente. Existen dos técnicas comúnmente empleadas en el emparejamiento: emparejamiento basado en puntos y emparejamiento basado en hitos referenciales o características.

En vez de trabajar directamente con los puntos del *barrido láser* (comúnmente referido en la bibliografía como *scan láser*, o simplemente *scan*), técnica denominada emparejamiento basado en puntos, el emparejamiento basado en características primero transforma los datos directamente obtenidos del sensor en características geométricas. En la figura 4.1 puede observarse un barrido láser y las rectas detectadas en él. Estos hitos o características extraídas son las que se emplean en el emparejamiento o correspondencia. Este método es más compacto, requiere menos capacidad de almacenamiento y menos poder computacional (al menos durante el emparejamiento) y es capaz de proporcionar información rica y precisa [54, 44, 7, 3, 8].

De entre los distintos tipos de primitivas geométricas que es posible considerar, la más sencilla es la recta infinita [44]. Es posible describir de un modo adecuado muchos entornos de

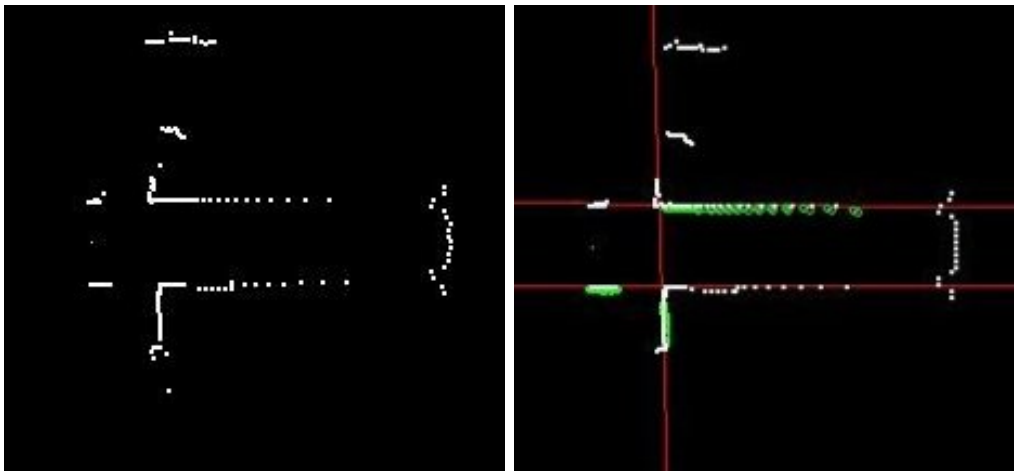


Figura 4.1: A la izquierda un barrido láser, y a la derecha las rectas detectadas superpuestas al barrido.

interior simplemente empleando líneas rectas. Además es posible detectar esquinas a partir de las líneas rectas detectadas, lo que enriquece la información con un coste computacional mínimo.

Existen numerosos algoritmos para la extracción de segmentos rectilíneos a partir de información obtenida mediante un sensor de rango láser. Una comparación de los más habituales puede encontrarse en [44]. En base a los datos presentados en el mencionado artículo hemos seleccionado el método de extracción de líneas rectas denominado *Split & Merge* [54, 44]. Este algoritmo es simple conceptualmente y a nivel de implementación, y se acomoda a el tipo de datos densos y precisos proporcionados por un medidor de rango láser. Para la detección de esquinas simplemente se buscan las intersecciones *reales* entre las líneas detectadas.

En el presente capítulo vamos a presentar el algoritmo para la detección de segmentos rectilíneos *Split & Merge*, y un algoritmo para la extracción de esquinas a partir de segmentos rectilíneos. Además se comentarán detalles de la implementación de ambos.

En la sección siguiente se explica en detalle el método de extracción de líneas rectas *Split and Merge*. A continuación, en la sección 4.3 se explica el método para la extracción de esquinas y, finalmente, en la sección 4.4 se analizan aspectos específicos de la implementación de ambos.

4.2. Extracción de líneas rectas. Split & Merge

Se trata probablemente del algoritmo de extracción de rectas más popular [44], originado en el contexto de la visión por computador [49]. Tal como se muestra en la tabla 4.1 el algoritmo comienza inicializando una pila (o una lista) con un conjunto S_0 de puntos $\langle P_1, \cdots, P_n \rangle$. Mientras la pila no esté vacía, se extrae un conjunto de puntos S de la misma.

A continuación se realiza el ajuste de la recta R que pasa por los puntos del conjunto S, y se determina el punto P_{max} más alejado de la recta. Si la distancia de P_{max} a R es menor que un cierto umbral D_{max}, se introduce R en una lista de rectas potenciales $ListaRectas$ y se continúa iterando. Si la distancia supera el umbral D_{max} se divide el conjunto S en S_1 y S_2 sobre el punto P_{max}, se introducen los conjuntos resultantes en la pila, y se continúa iterando. Una vez que la pila está vacía, se toma la lista de rectas potenciales $ListaRectas$, y se unen aquellas que resulten ser colineales. Ésta constituye la versión más básica del algoritmo.

Algorithm Split and Merge($\langle P_1, \cdots, P_n \rangle$) return $ListaRectas$
- $S_0 = \langle P_1, \cdots, P_n \rangle$ // Conjunto de puntos inicial
- push(S_1,$PilaConjuntosPuntos$) // Pila de conjuntos de puntos
- $ListaRectas = \langle\rangle$ // Lista de rectas
- while $PilaConjuntosPuntos \neq$ vacía
  - $S = pop(PilaConjuntosPuntos)$ // Se toma un conjunto de la pila
  - $R = AjustarRecta(S)$ // Se ajusta una recta al conjunto de puntos
  - $P_{max} = \arg\max_{P \in R} Distancia(P, R)$ // Punto más lejano a la recta
  - if $Distancia(P_{max}, R) < D_{max}$ // Si no supera el umbral
    - add(R,ListaRectas) // Añadir a la lista de rectas
    - continue
  - else // Si supera el umbral
    - Split S into S_1 and S_2 on P_{max} // Dividir el segmento en dos
    - push(S_1,$PilaConjuntosPuntos$) // Añadir a la pila
    - push(S_2,$PilaConjuntosPuntos$) // Añadir a la pila
  - endif
- endwhile
- Unir segmentos colineales de $ListaRectas$
- return $ListaRectas$

endAlgorithm

Tabla 4.1: Resumen del algoritmo Split &Merge.

Existen varias posibilidades para enriquecer el funcionamiento del algoritmo, de ellas las más destacables son el establecimiento de umbrales de tamaño mínimo y de número mínimo de puntos de un segmento válido.

Los detalles concretos del algoritmo empleado en el presente proyecto se comentan en la sección 4.4. Aquellos aspectos que permiten particularizar una implementación son:

- El algoritmo de *ajuste* de una recta a partir de una serie de puntos.
- El algoritmo de detección de segmentos colineales.
- Umbral de número mínimo de puntos de un segmento (P_{min}).
- Umbral de longitud mínima de un segmento (L_{min}).
- Umbral de distancia máxima de un punto a una recta, para considerarlo integrante de ella (D_{max}).

- Las condiciones de colinealidad de segmentos.

Existen diversas variantes del algoritmo. Por ejemplo, si en vez de ajustar la recta con todos los puntos del segmento, se hace únicamente con el primero y el último, el método se conoce en la literatura como *Iterative-End-Point-Fit* [44].

4.3. Extracción de esquinas

Para la detección de esquinas se ha empleado un método sencillo, pero muy eficaz. Una vez detectadas las líneas presentes en un barrido láser, se calculan todas las posibles intersecciones (puntos 1 y 2 del algoritmo mostrado en la tabla 4.2), y se toman como verdaderas esquinas aquellas intersecciones que se encuentran dentro de un determinado rango de distancia con respecto a los dos segmentos cuya intersección se está analizando. La distancia entre una intersección y una recta se calcula como la distancia mínima entre la intersección y alguno de los puntos integrantes de la recta, que a nivel de barrido láser aún se encuentran disponibles. Un ejemplo del concepto de esquinas verdaderas se muestra en la figura 4.2. En esta figura se muestran dos esquinas verdaderas, correspondientes a las dos intersecciones que se encuentran situadas sobre las dos rectas cuya intersección las ha generado. También se muestra una esquina ficticia, correspondiente con la intersección que no se encuentra sobre ninguna de las rectas cuya intersección la ha generado. Es necesario definir un umbral máximo de distancia de una intersección a las rectas que la han generado, a partir del cual se considera que dicha intersección no es una esquina. Con todo esto, el algoritmo se muestra en la tabla 4.2.

1: Inicializar: Crear una lista con todas las rectas detectadas
2: Para cada pareja de rectas (r_1, r_2) calcular su intersección
3: Calcular distancia mínima d_1 de la intersección a r_1
4: Calcular distancia mínima d_2 de la intersección a r_2
5: si d_1 y $d_2 <= umbral$ incluir intersección en la lista de esquinas

---

Tabla 4.2: Algoritmo básico de extracción de esquinas

4.4. Detalles de implementación

Nuestra implementación del algoritmo *Split & Merge* se particulariza en los siguientes aspectos:

- Se realiza el ajuste de la recta con los puntos extremo del conjunto. Esta variante del algoritmo se conoce como *Iterative-End-Point-Fit*, y tiene como principal ventaja el coste computacional.

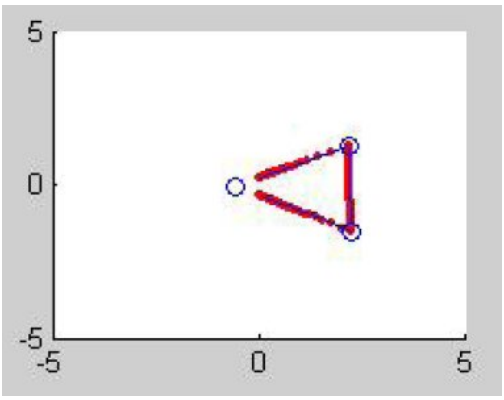


Figura 4.2: Dos esquinas reales y una ficticia.

- Se establece un umbral de distancia máxima D_{max} de un punto a la recta de la que forma parte. Valores típicos de D_{max} se encuentran en el intervalo (2 - 3 cm) (ver Sección 10.4.1 para obtener detalles sobre su determinación).
- Se establece un umbral de longitud mínima de un segmento válido L_{min}. Valores típicos de L_{min} se encuentran en el intervalo (0.5 - 2 m) (ver Sección 10.4.2 para obtener detalles sobre su determinación).
- Se establece un umbral de número mínimo de puntos de un segmento válido P_{min}. Valores típicos de P_{min} se encuentran en torno a 15 puntos (ver Sección 10.4.3 para obtener detalles sobre su determinación).

Modificando estos umbrales se está modificando la tolerancia del algoritmo frente a rectas ruidosas. En general cuando se emplean sensores de rango láser los datos obtenidos son poco ruidosos, así que se puede ser riguroso en el establecimiento de estos parámetros.

En la tabla 4.3 puede observarse el esquema del algoritmo *Split & Merge* implementado en nuestro sistema. Este algoritmo añade al algoritmo básico de la sección 4.2 los mencionados umbrales mínimos de longitud y puntos. Además, en vez de un esquema de ejecución iterativo, se ha optado por un esquema recursivo, que facilita la implementación y la comprensión del método. Allí dónde se emplean listas de rectas, es igualmente posible emplear vectores, siempre y cuando se entienda que el operador "+" aplicado a vectores genera un nuevo vector que contiene todos los elementos presentes en los dos vectores originarios, sin tener en cuenta posibles repeticiones (en el caso de existir elementos comunes en los vectores que se están sumando, el vector suma contendrá elementos repetidos).

Una vez realizada la detección de segmentos se debe realizar la unión de segmentos colineales, para ello se emplea el algoritmo mostrado en la tabla 4.4. El algoritmo busca todas las rectas colineales en una lista, las extrae de la lista y las fusiona, añadiendo el resultado a una lista de salida, así hasta que la lista de entrada queda vacía. La operación de fusión o *merge* consiste en la unión de los puntos que integran las dos rectas que estań siendo fusionadas, y el cálculo de los parámetros de la recta resultante.

Algorithm Split and Merge($\langle P_1, \cdots, P_n \rangle$) return *ListaRectas*
- $S = \langle P_1, \cdots, P_n \rangle$ // Conjunto de puntos inicial
- if S = vacío
  - return null
- if $NumeroPuntos(S) < P_{min}$ // Comprobación del umbral P_{min}
  - return null
- $ListaRectas = \langle\rangle$ // Lista de rectas
- $ListaRectas1 = \langle\rangle$ // Lista de rectas
- $ListaRectas2 = \langle\rangle$ // Lista de rectas
- $R = AjustarRecta(S)$ // Se ajusta una recta al conjunto de puntos
- if $Longitud(R) < L_{min}$ // Comprobación del umbral L_{min}
  - return null
- add(R,ListaRectas)
- $P_{max} = \arg\max_{P \in R} Distancia(P, R)$ // Punto más lejano a la recta
- if $Distancia(P_{max}, R) < D_{max}$ // Si no supera el umbral
  - devolver ListaRectas
- else // Si supera el umbral
  - Split S into S_1 and S_2 on P_{max} // Dividir el segmento en dos
  - ListaRectas1 = **Split and Merge**(S_1)
  - ListaRectas2 = **Split and Merge**(S_2)
  - return ListaRectas1 + ListaRectas2 // Devolver la unión de las Listas de rectas
- endif

endAlgorithm

Tabla 4.3: Resumen del algoritmo Split &Merge.

Algorithm Colineales($\langle R_1, \cdots, R_n \rangle$) return ListaRectas
- L = $\langle R_1, \cdots, r_n \rangle$ // Lista de rectas a procesar
- ListaSalida = $\langle\rangle$ // Lista de rectas de salida
- for each R_i in L
  - delete(R_i,L)
  - for each R_j in L
    - if $abs(R_i.\rho - R_j.\rho) < \rho_{max}$ AND
      - $abs(R_i.\theta - R_j.\theta) < \theta_{max}$
      - $R_i = merge(R_i, R_j)$
      - delete(R_j,L)
    - endif
  - endfor
  - add(R_i,ListaSalida)
- endfor
- return ListaSalida

endAlgoritmo

Tabla 4.4: Unión de segmentos colineales.

Un punto clave del algoritmo es la condición de colinealidad, que se determina mediante umbrales mínimos para la diferencia de las coordenadas polares (ρ en metros, θ, en radianes)

de los segmentos. Se ha determinado (ver sección 10.4.4) que unos umbrales adecuados son:

- $abs(\rho_1 - \rho_2) < 0{,}1$ m
- $abs(\theta_1 - \theta_2) < 0{,}07$ radianes

En cuanto a la detección de esquinas se ha determinado que el umbral de distancia máxima de una intersección a los segmentos que la originan debe encontrarse en torno a los 10 cm. La obtención de este valor se muestra en detalle en la sección 10.4.5.

Capítulo 5

Asociación de datos

5.1. Introducción

La utilización de un sensor montado en un vehículo para construir y actualizar un mapa del entorno en el que el vehículo está navegando, plantea el complejo problema de la asociación de datos [43], uno de los más difíciles que involucra el proceso de SLAM [45]. Este problema consiste en relacionar las observaciones del sensor con los elementos incluidos en el mapa, y está muy influenciado por el error acumulado en la pose del robot, y el error en las medidas. Obtener una solución correcta es crucial, ya que un error en la asociación provoca que métodos de localización, como el filtro de Kalman extendido (EKF), diverjan [43, 38]. La mayor parte de los avances científicos en esta materia se han producido en los últimos 10 años [66].

Cuando se afronta el problema del *SLAM* en situaciones reales, es muy raro conocer la correspondencia entre las observaciones y los elementos del mapa. El número total de elementos que integran el entorno también es desconocido [38]. Cada vez que el robot realiza una observación, esa lectura debe asociarse a un elemento del mapa, o bien considerar que proviene de un elemento no visto anteriormente.

Dos factores contribuyen a la incertidumbre a la hora de afrontar el proceso de SLAM desde un punto de vista probabilístico (mediante un filtro Bayesiano): *el ruido de medición u observación, y el ruido de movimiento*. Cada uno de ellos conduce a un tipo diferente de ambigüedad en la asociación de datos. El ruido en el modelo de observación conduce a una mayor incertidumbre en la posición de los elementos del entorno, y ésta conduce a ambigüedad en la medida, o confusión de elementos cercanos (ver Figura 5.1). Un error debido a ambigüedad en la medida tendrá un efecto relativamente bajo sobre la estimación, ya que la observación podría haber provenido plausiblemente de cualquiera de los elementos del entorno [38].

La ambigüedad debida al ruido de movimiento puede tener consecuencias mucho más serias. Cantidades mayores de ruido conducen a una mayor incertidumbre en la *pose* del robot después de ejecutar una acción de control del movimiento. Si esta incertidumbre es suficientemente grande, las diferentes poses plausibles del robot conducirán a asociaciones de datos drásticamente diferentes. La ambigüedad en el movimiento es inducida fácilmente

si existe una incertidumbre angular significativa en la estimación de la pose del robot (ver Figura 5.2).

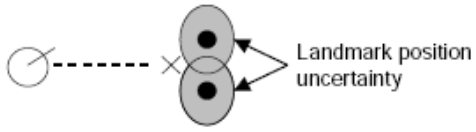


Figura 5.1: Ambigüedad en la medida. Dos elementos (mostrados como círculos negros) son tan cercanos que la observación (mostrada como X) puede pertenecer plausiblemente a cualquiera de los dos.

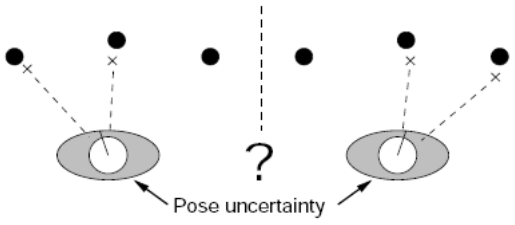


Figura 5.2: Ambigüedad en el movimiento: Las observaciones pueden ser asociadas con elementos completamente distintos, si la orientación del robot cambia ligeramente.

Existen múltiples y muy distintas soluciones al problema de la asociación de datos. En el presente proyecto hemos decidido analizar dos aproximaciones. Por un lado el algoritmo del *vecino más cercano, con umbral* (conocido comúnmente como *Gated Nearest Neigbor*, NN o GNN), por ser un método clásico muy referenciado en la bibliografía [43, 22, 38, 45], y por otro el algoritmo de *máxima similitud* (conocido comúnmente como *maximum likelihood* o ML) por ser el recomendado por los creadores del algoritmo FastSLAM [38, 40, 39, 67]. Aunque no afectan a nuestro proyecto, cabe mencionar la existencia de métodos de asociación de datos mediante *emparejamiento de barridos* (comúnmente referido en la literatura, tanto en lengua inglesa como española, como *scan matching*), que son prometedores para entornos en los que existen bucles muy grandes [40]. Estos métodos calculan la pose que permite encajar del mejor modo posible un barrido con uno o varios barridos anteriores, permitiendo realizar correcciones hacia atrás en el tiempo en el momento en que se cierra un bucle.

En el presente capítulo vamos a analizar desde un punto de vista matemático el problema de la asociación de datos, así como dos de las soluciones más empleadas, los algoritmos GNN y ML. Además se mostrarán detalles de la implementación del método ML, seleccionado para ser implementado, por ser el recomendado por los creadores del algoritmo FastSLAM [38, 40, 39, 67].

En la sección siguiente se analizan brevemente distintos acercamientos empleados habitualmente para la asociación de datos en SLAM. En la sección 5.3 se desarrolla el algoritmo

Gated Nearest Neighbor desde un punto de vista matemático. Del mismo modo se desarrolla en la sección 5.4 el algoritmo *maximum likelihood*, además de analizar ciertos detalles de su implementación.

5.2. Asociación de datos en SLAM

Generalmente un algoritmo de asociación de datos se compone de dos elementos: un *test* para determinar la compatibilidad entre una observación y un elemento de un mapa, dada una estimación de la pose del robot, y un *criterio de selección*, para elegir los mejores emparejamientos entre el conjunto de emparejamientos compatibles [43].

Un algoritmo clásico, muy empleado en problemas de seguimiento, es el *Gated Nearest Neighbor* (GNN). El *test chi cuadrado* para la *innovación normalizada* [43] se usa para determinar la compatibilidad, y la regla *nearest neighbor* [45, 43] (el vecino más cercano, según la *distancia de Mahalanobis*) se emplea para seleccionar los mejores emparejamientos.

La gran ventaja de esta solución, además de su simplicidad conceptual, es su complejidad computacional $O(m * n)$, donde m es el número de hitos observador y n el número de hitos presentes en el mapa. El algoritmo *GNN* es un método seguro para elementos como rectas detectadas a partir de sensores láser, donde la densidad de elementos es baja y la precisión del sensor es alta, mientras el error del vehículo se mantenga moderado [43]. Sin embargo, su rendimiento decrece rápidamente cuando aumenta la incertidumbre de la posición relativa de los elementos del entorno con respecto al robot, como ocurre cuando se revisita una región previamente cartografiada, después de recorrer un bucle grande. El rendimiento también cae cuando se emplean sensores menos precisos como sónar, o visión monocular basada en bordes.

Michael Montemerlo y Sebastian Thrun proponen el uso de una asociación de datos basada en un criterio de máxima similitud (*maximum likelihood*) [38, 40, 39, 67]. En concreto proponen tomar aquellas asociaciones n_t que maximizan la probabilidad de obtener la medida del sensor z_t, a partir de todos los datos disponibles (histórico de poses s^t, histórico de asociaciones $\hat{n}^{t-1}$, histórico de observaciones z^{t-1}, e histórico de controles u^t).

$$\hat{n} = \underset{n_t}{argmax}\, p(z_t|n_t, \hat{n}^{t-1}, s^t, z^{t-1}, u^t) \tag{5.1}$$

El término $p(z_t|n_t, \hat{n}^{t-1}, s^t, z^{t-1}, u^t)$ es referido como *similitud* (*likelihood*), y este acercamiento es un ejemplo de un estimador ML (maximum likelihood) [40]. La asociación de datos ML también se denomina Nearest Neighbor, interpretando el logaritmo negativo de la similitud como una distancia. Para distribuciones gaussianas, el logaritmo negativo de la similitud es la distancia de Mahalanobis, y el estimador selecciona asociaciones mediante la minimización de ésta.

Mientras que la asociación ML afronta la ambigüedad en el movimiento, no hace lo mismo con la ambigüedad en la medida. Cada observación se empareja con el elemento que más probablemente la ha generado, sin embargo si el error de medida es grande, puede haber varias asociaciones plausibles por cada observación.

Si se produce más de una observación en cada paso, es posible aplicar métodos que imponen restricciones a las posibles asociaciones, teniendo en cuenta consideraciones geométricas o probabilísticas [43]. Dentro de este grupo encontramos métodos como *asociación de datos probabilística conjunta* más conocida como *joint probabilistic data association* [22] o *ramificación y poda por compatibilidad conjunta* denominado habitualmente *joint compatibility branch and bound* [47, 43]. Otra técnica interesante, aplicable en el caso de observaciones múltiples, es la denominada *multiple hypothesis tracking*, o *seguimiento de múltiples hipótesis* [52].

5.3. Gated Nearest Neighbor

Como ya se ha mencionado, un algoritmo de asociación de datos se compone de dos elementos: un *test* para determinar la compatibilidad entre una observación y un elemento de un mapa, a partir de una estimación de la localización del robot, y un *criterio de selección* para elegir los mejores emparejamientos entre el conjunto de emparejamientos compatibles. GNN emplea el *test de chi cuadrado* para la *innovación normalizada* [43] para determinar la compatibilidad, y la regla *nearest neighbor* (el vecino más cercano, según la *distancia de Mahalanobis*) para seleccionar los mejores emparejamientos. Las principales ventajas del algoritmo son su simplicidad conceptual y computacional ($O(mn)$). Sin embargo el algoritmo pierde efectividad en entornos muy densos, o cuando existe alta incertidumbre en la posición del vehículo.

5.3.1. Desarrollo matemático

En cartografía estocástica (o construcción probabilística de mapas) el estado de un vehículo R y de un conjunto de n características $\{F_1, ..., F_n\}$ del entorno en el que el vehículo está navegando, se representa mediante un vector x. Sea $\hat{x}$ la estimación de la localización del vehículo y de las características, y P la covarianza del error de estimación:

$$\hat{x} = \begin{pmatrix} \hat{X}_R \\ \hat{X}_{F_1} \\ \vdots \\ \hat{X}_{F_n} \end{pmatrix}; \quad P = \begin{pmatrix} P_R & P_{RF_1} & \cdots & P_{RF_n} \\ P_{RF_1}^T & P_{F_1} & \cdots & P_{F_1F_n} \\ \cdots & \cdots & \cdots & \cdots \\ P_{RF_n}^T & P_{F_1F_n}^T & \cdots & P_{F_n} \end{pmatrix}$$

De un modo similar sea $\hat{y}$ un conjunto de m medidas $\{E_1, \cdots, E_m\}$ de características del entorno, obtenidas mediante un sensor montado en un vehículo, afectadas por ruido Gaussaino:

$$\hat{y} = y + u; \qquad u \sim N(0, S)$$
$$\hat{y} = \begin{pmatrix} \hat{y}_{E_1} \\ \vdots \\ \hat{y}_{E_m} \end{pmatrix}; \quad S = \begin{pmatrix} S_{E_1} & \cdots & S_{E_1 E_m} \\ \cdots & \cdots & \cdots \\ S_{E_1 E_m}^T & \cdots & S_{E_m} \end{pmatrix}$$

donde y es el valor teórico de las observaciones. Una observación E_i y su correspondiente característica F_j están relacionadas por una función de la forma

$$f_{ij}(x, y) = 0 \tag{5.2}$$

que establece que la posición relativa entre una medida y su correspondiente característica, debe ser cero.

El propósito de la asociación de datos es generar una hipótesis $H_m = \{j_1, \cdots, j_m\}$ que empareja cada medida E_i con una característica del mapa F_j. Los algoritmos de asociación de datos deben elegir de alguna manera entre todas las posibles hipótesis, llevando a cabo validaciones que determinen la compatibilidad entre las medidas y las características presentes en el mapa. El espacio de soluciones generado es exponencial, y puede ser interpretado como un árbol de m niveles [43], al que pueden aplicarse técnicas de inteligencia artificial, para encontrar una solución.

Cuando se considera únicamente una observación, el algoritmo GNN se denomina *algoritmo de compatibilidad individual del vecino más cercano* (*Individual Compatibility Nearest Neighbor* (ICNN) en inglés) [43]. Éste simplemente empareja cada medida con la característica considerada más compatible de acuerdo a la ecuación 5.2. Dado que habitualmente la función de medida no es lineal, es necesario linealizarla alrededor de la estimación actual

$$f_{ij_i} \simeq h_{ij_i} + H_{ij_i}(x - \hat{x}) + G_{ij_i}(y - \hat{y}) \tag{5.3}$$

donde

$$h_{ij_i} = f_{ij_i}(\hat{x}, \hat{y}); \quad H_{ij_i} = \frac{\partial f_{ij_i}}{\partial x}|_{(\hat{x},\hat{y})}$$

$$G_{ij_i} = \frac{\partial f_{ij_i}}{\partial y}|_{(\hat{x},\hat{y})}$$

El vector h_{ij_i} representa la innovación del emparejamiento entre E_i y F_i. De 5.2 y 5.3 puede obtenerse su covarianza como

$$\begin{aligned} C_{ij_i} &= H_{if_i} Cov(x - \hat{x}) H_{ij_i}^T + G_{if_i} Cov(y - \hat{y}) G_{if_i}^T \\ &= H_{ij_i} P H_{ij_i}^T + G_{ij_i} S G_{ij_i}^T \end{aligned} \tag{5.4}$$

La compatibilidad individual entre E_i y F_i puede determinarse mediante un test de

innovación que mide la *distancia de Mahalanobis* del siguiente modo

$$D_{ij}^2 = h_{ij_i}^T C_{ij_i}^{-1} h_{ij_i} < \chi_{d,\alpha}^2 \tag{5.5}$$

donde $d = dim(f_{ij_i})$, y α es el nivel de confianza deseado. Este test, aplicado al estado predicho, determina el subconjunto de características del mapa compatibles con una medida E_i.

El criterio de selección *nearest neighbor* (vecino más cercano) para una medida dada, consiste en seleccionar entre todas las características compatibles aquella con la menor distancia de Mahalanobis.

5.4. Maximum likelihood

El acercamiento estándar a la asociación de datos empleado con los filtros de Kalman extendidos (EKF), consiste en asignar cada observación a una característica usando una regla de máxima similitud, es decir, se asigna cada observación a la característica que más probablemente la haya generado. Si la probabilidad de que una observación pertenezca a una característica existente es muy baja, puede considerarse su inclusión como una nueva característica [40].

La asociación de datos mediante el método de máxima similitud generalmente da buenos resultados cuando la asociación de datos correcta es significativamente más probable que las asociaciones incorrectas [40]. Sin embargo, si la incertidumbre en la posición de una característica es elevada, más de una asociación recibirá una alta probabilidad. Dependiendo del método empleado para la construcción del mapa, la elección de una asociación incorrecta puede tener unas consecuencias desastrosas en la precisión del mapa resultante [40, 45]. Este tipo de ambigüedad puede ser inducida con facilidad si los sensores del robot son muy ruidosos.

Ya que cada partícula en el algoritmo FastSLAM representa un recorrido específico del robot, no es necesario aplicar las mismas asociaciones de datos a todas ellas. Las asociaciones de datos en FastSLAM pueden realizarse individualmente para cada partícula [40]. Aquellas partículas que hayan recibido asociaciones de datos correctas recibirán un mayor peso y tendrán más posibilidades de ser remuestreadas en el futuro. Por el contrario aquellas partículas que hayan recibido asociaciones de datos incorrectas recibirán un peso bajo y serán eliminadas durante el remuestreo.

5.4.1. Desarrollo matemático

En el caso del filtro extendido de Kalman, la probabilidad de una observación puede escribirse como una función de la diferencia entre la observación realizada z_t y la observación esperada $\hat{z}_{n_t}$ para el hito n. Esta diferencia se conoce como *innovación*. Sea n_t el vector de asociaciones, que relaciona cada observación con un elemento del mapa en el instante t, s^t el camino seguido por el robot, u^t los controles ejecutados por el robot, y sea $\hat{n}^{t-1}$, el vector

de asociaciones para el instante $t-1$,

$$\hat{n} = \underset{n_t}{argmax}\ p(z_t|n_t, \hat{n}^{t-1}, s^t, z^{t-1}, u^t) \tag{5.6}$$

$$= \underset{n_t}{argmax}\ \frac{1}{\sqrt{|2\pi Z_t|}} e^{(-\frac{1}{2}(z_t-\hat{z}_{n_t})^T Z_t^{-1}(z_t-\hat{z}_{n_t}))} \tag{5.7}$$

Donde Z_t es la covarianza de la estimación, cuyo cálculo se muestra en la figura 5.3. Esta heurística para la asociación de datos se reformula habitualmente en términos logarítmicos

$$\hat{n}_t = \underset{n_t}{argmin}\ \ln|Z_t| + (z_t - \hat{z}_{n_t})^T Z_t^{-1}(z_t - \hat{z}_{n_t}) \tag{5.8}$$

El segundo término de la ecuación 5.8 es la *distancia de Mahalanobis*, una distancia métrica normalizada por las covarianzas de la observación y de la estimación.

En FastSLAM 1.0 se utiliza un EKF como estimador para cada una de las características presentes en el mapa. Para poder calcular la probabilidad de una asociación será necesario, pues, emplear las ecuaciones características del filtro tal como se muestra en la figura 5.3. Los filtros de Kalman extendidos se estudian en profundidad en la sección 7.3.

$$\begin{aligned}
\hat{z}_n &= g(\mu_{n,t-1}, s_t) \\
G_n &= g'(\mu_{n,t-1}, s_t) \\
Z_t &= G_n^T \Sigma_{n,t-1} G_n + R_t \\
w_n &= |2\pi Q_n|^{-\frac{1}{2}} \exp\{-\tfrac{1}{2}(z_t - \hat{z}_n)^T Q_n^{-1}(z_t - \hat{z}_n)\} \\
w_{N+1} &= P_0 \\
\hat{n} &= \underset{n=1,\ldots,N+1}{\arg\max}\ (w_n)
\end{aligned}$$

Figura 5.3: Empleo de las ecuaciones del EKF para el cálculo de la asociación de datos.

5.4.2. Detalles de implementación

Para la implementación de la asociación de datos mediante el método de máxima similitud (*maximum likelihood*) se ha empleado el algoritmo de la tabla 5.1. Los parámetros necesarios son la pose del robot s_t, una observación z_t, y un mapa m. El mapa es una colección de hitos referenciales representados por su media μ_n, y su covarianza Σ_n.

Como puede observarse en la línea 6 se almacena la probabilidad de cada una de las posibles asociaciones y en la línea 8 la probabilidad de que la observación corresponda a una nueva característica. Se selecciona en la línea 9 la asociación que tenga una mayor probabilidad. Este proceso se realiza para cada uno de los hitos observados en un momento determinado. Tras realizar la asociación de datos siguiendo el algoritmo de la tabla 5.1, se analizan todas las asociaciones decididas, y en caso de asociarse la misma observación a

1: Algorithm **MaxLikelihood**(u_t, z_t, m)
2: for $n = 1$ to N
3: $\hat{z}_n = g(\mu_{n,t-1}, s_t)$ // Predicción de medida según el modelo de observación.
4: $G_n = g'(\mu_{n,t-1}, s_t)$ // Calcula Jacobiano
5: $Q_n = G_n^T \Sigma_{n,t-1} G_n + R_t$ // Covarianza de la medida
6: $w_n = |2\pi Q_n|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - \hat{z}_n)^T Q_n^{-1}(z_t - \hat{z}_n)\}$ // Probabilidad de la correspondencia
7: endfor
8: $w_{N+1} = P_0$ // Probabilidad asociada a un nuevo hito
9: $\hat{n} = \underset{n=1,\ldots,N+1}{\arg\max}\ (w_n)$ // Correspondencia más probable
10: return $\hat{n}$
11: endAlgorithm

Tabla 5.1: Cálculo de la asociación de datos más probable.

distintas características del mapa, no se tiene en cuenta ninguna de estas asociaciones [38]. El algoritmo debe ejecutarse individualmente para cada una de las partículas del filtro.

De todos los datos que emplea el algoritmo dos de ellos deben ser *fijados manualmente*. Se trata de la matriz covarianza del ruido de medida R_t y de la probabilidad de observación de una nueva característica P_0. De las pruebas mostradas en la sección 10.6.2 se desprende que valores de P_0 por debajo de 0.01 dan buenos resultados. Siguiendo un criterio conservador aconsejamos el empleo de valores del orden de las milésimas.

El modelo del sensor asume que el ruido sigue una distribución Gaussiana de media 0 y covarianza R_t. En principio los datos concernientes al sensor deben ser proporcionados por el fabricante. No obstante, en la sección 10.5.2 se realiza un estudio de la influencia de distintos valores de R_t en el algoritmo FastSLAM. Como valores típicos para la covarianza del error del modelo de observación debemos considerar:

$$R_t = \begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix}; \begin{array}{l} \sigma_\rho^2 = [0{,}01 - 0{,}08] \\ \sigma_\theta^2 = [0{,}0008 - 0{,}003] \end{array}$$

Por su parte Juan Nieto y Tim Bailey [57] recomiendan unos valores iniciales de,

$$\begin{aligned} \sigma_\rho^2 &= (0{,}1)^2 = 0{,}01 \\ \sigma_\theta^2 &= (1{,}0 * \pi/180)^2 = 0{,}017 \end{aligned}$$

aunque estos valores dependen fundamentalmente de las características del sensor empleado.

Capítulo 6

Filtro de Partículas

6.1. Introducción

Los filtros de partículas, también conocidos como *métodos secuenciales de Monte Carlo*, son métodos de Monte Carlo [27] empleados para aproximar la distribución posterior del filtro Bayesiano, problema generalmente intratable desde un punto de vista analítico [67, 4, 11]. La idea principal tras los fitros de partículas es representar la función de distribución *posterior* del filtro Bayesiano mediante un conjunto de muestras aleatorias con pesos asociados [67, 4], y calcular estimaciones de acuerdo a dichas muestras y sus pesos. Cuando el número de partículas es muy grande, esta caracterización de Monte Carlo se convierte en una representación equivalente de la función de distribución de probabilidad *posterior*, y el filtro se aproxima a un estimador Bayesiano óptimo [4]. Se trata de una aproximación, pero el hecho de ser no paramétrica permite representar un mayor espacio de distribuciones, no únicamente Gaussianas [67].

En el presente capítulo se tratan en profundidad y desde un punto de vista matemático los filtros de partículas. A pesar de existir diversas variedades de filtros de partículas todos se basan en un núcleo común, que se analiza de modo detallado. Se destacan sus fortalezas y debilidades, lo que conduce al estudio del remuestreo como técnica para mejorar el rendimiento de los filtros de partículas. Además se muestran aspectos concretos de la implementación.

En la sección 6.2 se desarrolla matemáticamente un marco común a todos los filtros de partículas, conocido como *SIS* (Sequential Importance Sampling). En la sección 6.3 se aborda la técnica del *remuestreo* o *resampling* como medio para evitar la degradación de los filtros de partículas. Por último, en la sección 6.4 se analizan aspectos concretos de nuestra implementación de los filtros de partículas y el remuestreo.

6.2. Desarrollo matemático

El denominado *algoritmo de muestreo secuencial de importancia* (conocido en la literatura en lengua inglesa como Sequential Importance Sampling o SIS) es un método de Monte Carlo [27] que constituye la base de la mayor parte de los filtros secuenciales de Monte

Carlo [11] desarrollados hasta hoy [4]. Esta aproximación recibe varios nombres en la bibliografía como *filtro bootstrap*, *algoritmo de condensación*, *aproximación de partículas que interactúan*, y *supervivencia del mejor adaptado*. Se trata de una técnica para implementar un filtro Bayesiano recursivo por medio de simulaciones de Monte Carlo. La idea principal tras los fitros de partículas es representar la función de distribución posterior mediante un conjunto de muestras aleatorias con pesos asociados [67, 4], y calcular estimaciones de acuerdo a dichas muestras y sus pesos, tal como puede observarse en la figura 6.1. Cuando el número de partículas es muy grande, esta caracterización de Monte Carlo se convierte en una representación equivalente de la función de distribución de probabilidad posterior, y el filtro se aproxima a un estimador Bayesiano óptimo [4].

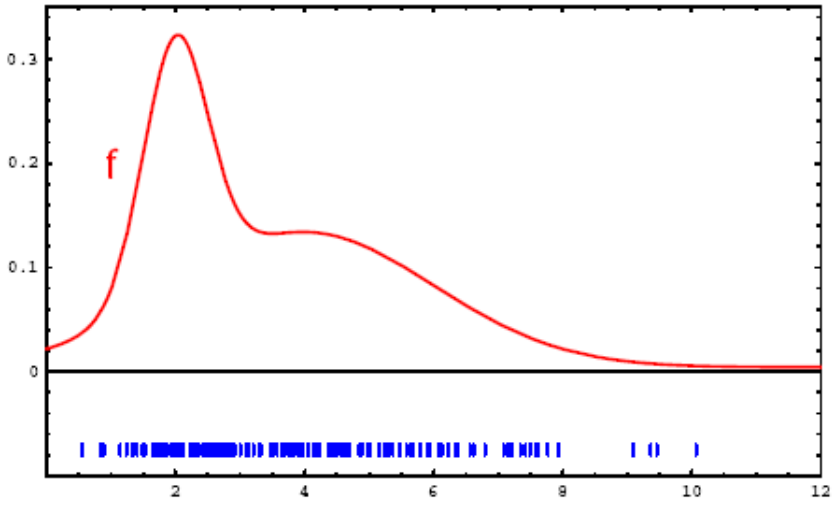


Figura 6.1: Filtro de partículas. La probabilidad se representa mediante la densidad de partículas, de modo que es posible representar cualquier tipo de distribución.

Sea x_k el vector de estado de un proceso estocástico cuya evolución está regida por una función del tipo:

$$x_k = f_k(x_{k-1}, v_{k-1}) \tag{6.1}$$

posiblemente no lineal, donde v_{k-1} es una secuencia de ruidos independientes e idénticamente distribuídos (comúnmente referido en estadística como i.i.d.). El proceso puede ser observado de acuerdo a la siguiente expresión:

$$z_k = h_k(x_k, n_k) \tag{6.2}$$

también posiblemente no lineal, donde n_k es una secuencia de ruidos i.i.d.

Nuestro objetivo frente a este sistema es estimar x_k en base a las observaciones $z_{1:k} = z_i, i = 1, \cdots, k$ hasta tiempo $t = k$. Desde un punto de vista Bayesiano se trata de determinar recursivamente cierto grado de creencia en el vector de estado x_k para $t = k$, a partir de la secuencia de observaciones z_k, es decir, se pretende construir la función de distribución de

probabilidad $p(x_k, z_{1:k})$.

Sea $\{\{x_{0:k}^1, w_k^1\}, \{x_{0:k}^2, w_k^2\}, \cdots, \{x_{0:k}^{N_s}, w_k^{N_s}\}\}$ un conjunto de partículas que caracterizan la función de distribución de probabilidad posterior $p(x_{0:k}|z_{1:k})$, donde $\{x_{0:k}^i, i = 0, \cdots, N_s\}$ es el estado histórico de cada partícula con pesos asociados $\{w_k^i, i = 0, \cdots, N_s\}$, $x_{0:k} = \{x_j, j = 0, \cdots, k\}$ es el conjunto de todos los estados hasta el tiempo $t = k$, y $z_{1:k} = \{z_l, l = 1, \cdots, k\}$ es el conjunto de todas las observaciones hasta el tiempo $t = k$. Los pesos están normalizados de modo que $\sum_i w_k^i = 1$. Entonces la densidad posterior para $t = k$ puede aproximarse por:

$$p(x_{0:k}|z_{1:k}) = \sum_{i=1}^{N_s} w_k^i \delta(x_{0:k} - x_{0:k}^i) \tag{6.3}$$

donde δ es la conocida "*delta de Dirac*". De este modo tenemos una aproximación discreta ponderada al verdadero posterior $p(x_{0:k}|z_{1:k})$. Los pesos se eligen utilizando el principio de *Importance Sampling (muestreo por importancia)*[11]. Este principio se basa en lo siguiente: Supongamos $p(x) \propto \pi(x)$ es una densidad de probabilidad difícil de muestrear, pero para la que $\pi(x)$ puede ser evaluado. Sean $\{x^i \sim q(x), i = 1, \cdots, N_s\}$ muestras fácilmente obtenidas de una función propuesta $q(.)$, denominada *densidad de importancia*. Entonces una aproximación ponderada a la densidad $p(.)$ vienen dada por:

$$p(x) \approx \sum_{i=1}^{N_s} w^i \delta(x - x^i) \tag{6.4}$$

donde

$$w^i \propto \frac{\pi(x^i)}{q(x^i)} \tag{6.5}$$

es el peso normalizado de la partícula i-ésima.

Si las muestras $x_{0:k}^i$ fueran obtenidas de una densidad de importancia $q(x_{0:k}|z_{1:k})$, los pesos quedan definidos como

$$w_k^i \propto \frac{p(x_{0:k}|z_{1:k})}{q(x_{0:k}|z_{1:k})} \tag{6.6}$$

Volviendo al caso sencuencial, cada iteración se puede disponer de un conjunto de muestras que forman una aproximación de $p(x_{0:k-1}|z_{1:k-1})$ y querer aproximar $p(x_{0:k}|z_{1:k})$, con un nuevo conjunto de partículas. Si la densidad de importancia se elije tal que es posible factorizarla de modo que

$$q(x_{0:k}|z_{1:k}) = q(x_{0:k}|x_{1:k-1}, z_{1:k}) q(x_{0:k-1}|z_{1:k-1}) \tag{6.7}$$

entonces es posible obtener muestras $x_{0:k}^i \sim q(x_{0:k}|z_{1:k})$ aumentando cada una de las muestras existentes $x_{0:k-1}^i \sim q(x_{0:k-1}|z_{1:k-1})$ con el nuevo estado $x_k^i \sim q(x_{0:k}|x_{0:k-1}, z_{1:k})$. Empleando consideraciones probabilísticas se llega a la siguiente expresión para el cálculo de los pesos [4]

$$w_k^i \approx w_{k-1}^i \frac{p(z_k|x_k^i) p(x_k^i|x_{k-1}^i)}{q(x_k|x_{0:k-1}, z_{1:k})} \tag{6.8}$$

Ademas, si $q(x_k|x_{0:k-1}, z_{1:k}) = q(x_k|x_{k-1}, z_k)$, entonces la densidad de importancia depende únicamente de x_{k-1} y z_k. Esto es particularmente interesante cuando lo que se requiere en cada paso es una estimación filtrada de $p(x_k|z_{1:k})$. En este tipo de situaciones sólo es necesario almacenar x_k^i de modo que es posible descartar el camino $x_{0:k-1}^i$, y la historia de observaciones $z_{1:k-1}$. El peso modificado entonces queda

$$w_k^i \approx w_{k-1}^i \frac{p(z_k|x_k^i)p(x_k^i|x_{k-1}^i)}{q(x_k|x_{k-1}, z_k)} \tag{6.9}$$

y la densidad filtrada posterior $p(x_k|z_{1:k})$ puede aproximarse por

$$p(x_k|z_{1:k}) = \sum_{i=1}^{N_s} w_k^i \delta(x_k - x_k^i) \tag{6.10}$$

donde los pesos están definidos según 6.9. Todo el desarrollo matemático expuesto se plasma en el algoritmo de la tabla 6.1.

1: Algorithm **SIS**$(\{x_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, z_k)$
2: **for** $i = 1$ to N_s
3: obtener muestras $x_k^i \sim q(x_k|x_{k-1}, z_k)$
4: calcuar w_k^i según 6.9
5: **endfor**
6: return $\{x_k^i, w_k^i\}_{i=1}^{N_s}$

Tabla 6.1: Algoritmo del filtro de partículas SIS

Una derivación interesante desde el punto de vista del SLAM del filtro SIS es la conocida como filtro SIR (Sampling Importance Resampling). Se trata de un método de Monte Carlo aplicable a problemas de filtrado Bayesiano recursivo. El filtro SIR establece unos requisitos para poder ser aplicado:

1. Las funciones $f_k(.,.)$ y $h_k(.,.)$ deben ser conocidas.

2. Debe ser posible muestrear el ruido del proceso v_{k-1} y la distribución de probabilidad anterior.

3. La función de similitud $p(z_k|x_k)$ debe poder ser evaluada, o al menos estimada mediante una función proporcional a ella.

El fitro SIR emplea la siguiente densidad de importancia:

$$q(x_k|x_{k-1,z_{1:k}}^i) = p(x_k|x_{k-1}^i) \tag{6.11}$$

además impone la realización del proceso de remuestreo en cada iteración. Con la mencionada densidad de importancia los pesos de cada partícula quedan:

$$w_k^i \propto w_{k-1}^i p(z_k|x_k^i) \tag{6.12}$$

FastSLAM se basa en estas consideraciones y en una cierta factorización de la distribución de probabilidad posterior que limita la dimensión del vector de estado (ver capítulo 3).

6.3. Remuestreo

El remuestreo (referido comunmente como *resampling* en la bibliografía, incluso en la escrita en lengua española) es una técnica comúnmente empleada junto con *filtros de partículas*, para evitar la degeneración que sufren éstos con el discurrir de las iteraciones [4, 11].

El problema de la degeneración es un problema habitual de los filtros de partículas SIS, consistente en que tras unas pocas iteraciones, todas las partículas menos una tienen pesos despreciables. Ha sido demostrado [11] que la varianza de los pesos sólo puede crecer a lo largo del tiempo, de modo que es imposible evitar el problema de la degeneración. Esta degeneración implica que se dedica un gran esfuerzo computacional a la actualización de partículas cuya contribución a la aproximación $p(x_k|z_{1:k})$ es prácticamente nula. Una medida adecuada para la degeneración es el número de partículas efectivas [4] N_{eff}, definido como

$$N_{eff} = \frac{N_s}{1 + Var(w_k^{*i})} \tag{6.13}$$

donde N_s representa el número de partículas en el filtro, y $w_k^{*i} = p(x_k^i|z_{1:k})/q(x_i|x_{k-1}^i, x_k)$ se denomina *peso verdadero*. Esta expresión no puede ser evaluada exáctamente, pero puede obtenerse una estimación $\widehat{N_{eff}}$ de N_{eff} mediante

$$\widehat{N_{eff}} = \frac{1}{\sum_{i=1}^{N_s}(w_k^i)^2} \tag{6.14}$$

donde w_k^i es el peso normalizado calculado mediante la ecuación 6.8. Es necesario destacar que $N_{eff} \leq N_s$, y que valores pequeños de N_{eff} indican una degeneración severa. Existen diversos métodos para reducir los efectos de la degeneración como pueden ser el aumento de N_s, indeseable desde un punto de vista computacional, la elección cuidadosa de la función de densidad de importancia, o el remuestreo [4]. Nos decantamos en el presente proyecto por el remuestreo, debido a que el método FastSLAM fija a priori una función de densidad de importancia y recomienda el uso de remuestreo [40, 39, 67, 38].

La idea básica del remuestreo es eliminar partículas que tienen pesos pequeños y concentrarse en partículas con pesos grandes. El proceso de remuestreo implica generar un nuevo conjunto de partículas $\{x_k^{i^*}\}_{i=1}^{N_S}$ remuestreando (con reemplazamiento) N_S veces una repre-

sentación discreta aproximada de $p(x_k|z_{1:k})$ dada por

$$p(x_k|z_{1:k}) \approx \sum_{i=1}^{N_s} w_k^i \delta(x_k - x_k^i) \tag{6.15}$$

de manera que la probabilidad de muestrear una partícula sea proporcional a su peso

$$Pr(x_k^{i^*} = x_k^j) = w_k^j \tag{6.16}$$

La muestra resultante es una muestra i.i.d. (independiente e idénticamente distribuída) de la densidad discreta de la ecuación 6.16, y por tanto los pesos de las nuevas partículas son $w_k^i = 1/N_S$.

En cuanto al remuestreo hay dos aspectos a considerar. El primero es cuándo realizarlo. En este sentido existen dos aproximaciones, una que remuestrea en cada iteración, y otra que lo hace cuando el N_{eff} es menor que un cierto umbral parametrizable N_T. El segundo aspecto es el tipo de remuestreo a aplicar. Entre los métodos de muestreo existente, tales como remuestreo estratificado, sistemático o residual, hemos seleccionado el remuestreo sistemático por ser el recomendado por uno de los autores del algoritmo FastSLAM [67]. El remuestreo sistemático se basa en consideraciones de probabilidad y estadística que van mas allá del ámbito de este documento, estudiadas y desarrolladas por los Madow en una serie de artículos [34, 32, 33].

6.3.1. Remuestreo secuencial

El algoritmo de *muestreo sistemático* propuesto por los Madow [34, 32, 33], también denominado, en el ámbito de los filtros de partículas, *muestreo o remuestreo de baja varianza* [67], es un método de muestreo probabilístico fácil de implementar y que produce buenos resultados. En realidad el algoritmo aplicado al filtro de partículas combina el muestreo sistemático con el muestreo basado en el peso de cada partícula. Se realiza un muestreo sistemático en el espacio de las probabilidades y se busca aquella partícula cuya probabilidad acumulada supere a la probabilidad muestreada. En la tabla 6.2 se muestra esquemáticamente el algoritmo. En cada paso por la instrucción 5 se va realizando el barrido secuencial del espacio de las probabilidades, mientras que en las instrucciones 6 a 9 se calcula la probabilidad acumulada (o peso acumulado) de las partículas recorridas, hasta que esta probabilidad acumulada es superior a la muestra actual del estado de las probabilidades, entonces la última partícula recorrida se incorpora al nuevo conjunto de partículas, tal como puede verse en la instrucción 10.

Tres son las principales ventajas de este algoritmo. En primer lugar cubre todo el espacio muestral de un modo más sistemático que un simple muestreo aleatorio, abierto a la realización de un muestreo sesgado. En segundo lugar, si todas las partículas tienen el mismo peso, el remuestreo no elimina ninguna de ellas, con lo cual no es necesario detener el remuestreo en caso de no haberse producido ninguna observación. Por último, para M partículas el algoritmo presenta una complejidad computacional $O(M)$. Por estas razones la mayoría de

```
     Algorithm muestreo_sistematico(Particula[] X, Peso[] W) return Particula[]
1:     r = random(0,(X.length)^-1)
2:     c = W[1]
3:     i = 1
4:     for m = 1 to M
5:       u = r + (m-1) * (X.length)^-1
6:       while u > c do
7:         i = i + 1
8:         c = c + W[i]
9:       endwhile
10:      añadir X[i] a Xsalida
11:    endif
12:    return Xsalida
13:  endAltorithm
```

Tabla 6.2: Algoritmo de muestreo secuencial o de baja varianza. Se considera que los vectores comienzan con el índice 1.

las implementaciones de filtros de partículas aplicados a problemas robóticos suelen emplear mecanismos similares [67]. Tras la aplicación del remuestreo los pesos de todas las partículas se deben establecer a un mismo valor, ya sea 1 o bien M^{-1}.

6.4. Detalles de implementación

La implementación del filtro de partículas se apoya en otras partes fundamentales del sistema, como la asociación de datos o los modelos de movimiento y observación, por lo que su estructura es sencilla. Como se muestra en la tabla 6.3 en primer lugar se anota cuál es la mejor partícula, y se actualiza la pose de cada partícula, en caso de haberse ejecutado un control. Se comprueba que se ha observado algún hito, en caso contrario se termina la ejecución del filtrado. Para cada partícula se calcula la asociación de datos, se incorporan o actualizan los hitos correspondientes y se calcula el peso. El peso de una partícula, tras la asociación de datos, se calcula como el producto del peso anterior y un peso calculado como la probabilidad de observación de cada uno de los hitos que han intervenido en el proceso de asociación de datos, es decir, los hitos integrantes de la observación actual. Estos hitos pueden considerarse nuevas observaciones, o bien asociarlos a un hito ya presente en el mapa de la partícula, lo que supone distintos modos de calcular la probabilidad de observación (esto se estudia detalladamente en la sección 5). Por último se vuelve a anotar la mejor partícula ya que se han recalculado los pesos, y se lanza el algoritmo de muestreo.

Disponemos de dos algoritmos de remuestreo en la implementación actual. Ambos emplean el algoritmo de remuestreo secuencial o de baja varianza de la tabla 6.2. Sin embargo, uno de ellos siempre remuestrea, mientras que el otro establece un umbral para el número de partículas efectivas o N_{eff} por debajo del cual remuestrea. El cálculo del número de

```
     Algorithm filtro_particulas()
 1:      if se ha ejecutado un control
 2:          actualizar pose de las partículas
 3:      endif
 4:      if no hay observación terminar
 5:      for each partícula P in filtro
 6:          Realizar asociación de datos
 7:          peso = 1
 8:          for each hito h
 9:              Si h es nuevo añadir
10:              Si h no es nuevo aplicar EKF
11:              peso = peso * probabilidad(h)
12:          endfor
13:          P.peso = P.peso * peso
14:      endfor
15:      Buscar partícula P = arg max_{P_i ∈ filtro} (Peso(P_i)) y almacenarla
16:      Aplicar algoritmo de remuestreo
17:  P.peso = 1/filtro.NumeroDeParticulas
18:  endAlgorithm
```

Tabla 6.3: Algoritmo que gobierna la evolución del filtro de partículas

partículas efectivas se muestra en la tabla 6.4.

```
1:  Calcular Wt, suma de los pesos de todas las partículas
2:  invNeff = Neff = 0
3:  for each partícula P
4:      P.peso = P.peso/Wt
5:      invNeff = invNeff + (P.peso)^2
6:  endfor
7:  N_eff = 1/invNeff
```

Tabla 6.4: Cálculo del número de partículas efectivas o N_{eff}

Capítulo 7

Filtros de Kalman

7.1. Introducción

En 1960, R.E. Kalman publicó su ahora famosísimo artículo en el que se describe una solución recursiva al problema del filtrado lineal de datos discretos [28]. Desde aquel momento, debido en gran medida a los avances en el campo de la computación digital, el *filtro de Kalman* ha sido objeto de infinidad de estudios y aplicaciones, particularmente en el campo de la navegación autónoma o asistida [70].

Los filtros de Kalman son *filtros Bayesianos* que representan la *distribución posterior* mediante distribuciones gaussianas. El filtro de Kalman se compone de un conjunto de ecuaciones que permiten estimar el estado de un proceso, minimizando la media del error al cuadrado. El filtro es muy potente en diversos aspectos: permite estimar el pasado, el presente, y el futuro, y es capaz de hacerlo incluso cuando el modelo del sistema no se conoce con precisión [70].

Los filtros de Kalman son la aproximación clásica para la generación de mapas [66], y en mayor o menor medida siguen formando parte de los métodos de construcción de mapas actualmente en uso. Por ello en el presente capítulo analizamos en primer lugar el filtro de Kalman clásico aplicado a la construcción de mapas. A continuación se estudia el denominado filtro de Kalman extendido, y los detalles de su implementación, ya que este es el filtro de Kalman que empleamos en el proyecto.

En la sección siguiente se desarrolla matemáticamente el *filtro de Kalman lineal* para su aplicación a la construcción de mapas mediante robots móviles. De igual modo en la sección 7.3 se desarrolla matemáticamente el *filtro de Kalman extendido* y se analizan aspectos concretos de su implementación.

7.2. Filtro de Kalman Lineal (Linear Kalman Filter o LKF)

El filtro de Kalman afronta el problema general de estimar el estado $x \in R^n$ de un proceso controlado de modo discreto en el tiempo, dirigido por una ecuación en diferencias, ***lineal***

estocástica de la forma

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{7.1}$$

donde x_t y x_{t-1} son vectores de estado, y u_t es el vector de controles (los comandos ejecutados por el sistema) para tiempo t. Consideramos, por cuestiones de notación, que estos vectores son vectores columna. A_t y B_t son matrices. A_t es una matriz cuadrada n x n, donde n es la dimensión del vector de estado x_t. B_t es de dimensión n x m, donde m es la dimensión del vector de controles u_t. Multiplicando los vectores de estado y control por las matrices A_t y B_t respectivamente, obtenemos la función de transición de estado, que es lineal. De este modo los filtros de Kalman asumen que la dinámica del sistema es lineal.

La variable aleatoria ϵ_t en la ecuación 7.1 es un vector aleatorio que sigue una distribución gaussiana, que modela la aleatoriedad en el proceso de transición de estado. Su dimensión es la del vector de estado, su media es 0 y su covarianza R_t. Una probabilidad de transición de estado de la forma 7.1 se denomina gaussiana lineal, para reflejar el hecho de que es lineal en sus argumentos, con ruido gaussiano añadido:

$$p(x_t|u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\} \tag{7.2}$$

Además el proceso debe ser observable, siendo el modelo de medida u observación lineal con ruido Gaussiano añadido:

$$z_t = C_t x_t + \delta_t \tag{7.3}$$

Aquí C_t es una matriz de dimensiones k x n, donde k es la dimensión del vector de medida z_t. El vector δ_t modela el ruido de medida. Se trata de un ruido Gaussiano de media nula y covarianza Q_t. La probabilidad de medida se obtiene de la siguiente distribución normal multivariable:

$$p(z_t|x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\} \tag{7.4}$$

El filtro de Kalman es un filtro Bayesiano, y formulado como tal presenta la siguiente forma

$$p(x_t|z^t, u^t) = \mu p(z_t|x_t) \int p(x_t|u_t, x_{t-1}) p(x_{t-1}|z^{t-1}, u^{t-1}) dx_{t-1} \tag{7.5}$$

el superíndice t se refiere a todos los datos hasta $t = t$, es decir,

$$\begin{aligned} z^t &= \{z_1, z_2, \cdots, z_t\} \\ u^t &= \{u_1, u_2, \cdots, u_t\} \end{aligned}$$

hay que tener en cuenta que las cantidades estimadas (estimación del vector de estado) vienen caracterizadas, por ser variables estocásticas, por su vector de medias μ_t, y la matriz de covarianza Σ_t.

El filtro de Kalman estima un proceso empleando una forma de control retroalimentado o *feedback control*. El filtro estima el estado del proceso en un momento determinado, y después obtiene retroalimentación en forma de medidas (con ruido). De este modo las ecuaciones

del filtro de Kalman se dividen en dos grupos: ecuaciones de predicción o actualización temporal (time update), y ecuaciones de corrección o actualización de medida (measurement update). Las ecuaciones de predicción son responsables de proyectar hacia adelante en el tiempo las estimaciones del estado μ y su covarianza Σ para obtener una estimación a priori para el siguiente paso en el tiempo. Las ecuaciones de corrección son responsables de la retroalimentación, es decir, incorporan una medida a la estimación a priori, para obtener una estimación a posteriori mejorada [28].

En la tabla 7.1 se muestran las ecuaciones de predicción del filtro de Kalman Lineal. Puede observarse como estas ecuaciones proyectan las estimaciones del estado y la covarianza desde tiempo t_1 hasta tiempo t.

$\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$	// Predicción del estado
$\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$	// Predicción de la covarianza

Tabla 7.1: Ecuaciones de predicción del filtro de Kalman Lineal.

En la tabla 7.2 se muestran las ecuaciones de corrección del filtro de Kalman Lineal. La primera tarea durante la etapa de corrección consiste en calcular la ganancia de Kalman K_t. Este parámetro modela la diferente *credibilidad* que se asigna a la predicción y a la medida, a la hora de calcular la estimación a posteriori. Los siguientes pasos consisten en calcular efectivamente las estimaciones a posteriori del estado y de la covarianza.

$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$	// Ganancia de Kalman
$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$	// Corrección del estado
$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t)$	

Tabla 7.2: Ecuaciones de corrección del filtro de Kalman Lineal.

Después de cada paso de *predicción-corrección* el proceso se repite, tomando la estimación a posteriori previa como base para proyectar la nueva estimación a priori. Esta naturaleza recursiva es uno de los grandes atractivos de los filtros de Kalman, ya que facilita enormemente su implementación, frente a otros filtros como los filtros de Wiener diseñados para operar sobre todos los datos directamente para generar cada estimación. En vez de esto, el filtro de Kalman condiciona recursivamente la presente estimación, en todas las medidas pasadas.

Un elemento crucial de los filtros Bayesianos es que el estado x_t contenga todas las cantidades desconocidas que pueden tener influencia sobre las medidas del sensor. En el contexto de la construcción robótica de mapas se consideran típicamente dos cantidades: el mapa y la pose del robot en el entorno. Ambas influencian las medidas del sensor a lo largo del tiempo [66]. De este modo el problema de la cartografía es en realidad el de la estimación

conjunta del mapa y la pose del robot. Si empleamos m para denotar el mapa y s para la pose del robot, obtenemos el siguiente filtro Bayesiano

$$p(s_t, m_t|z^t, u^t) = \mu p(z_t|s_t, m_t) \int\int p(s_t, m_t|u_t, s_{t-1}, m_{t-1}) p(s_{t-1}, m_{t-1}|z^{t-1}, u^{t-1}) ds_{t-1} dm_{t-1} \tag{7.6}$$

La mayoría de los algoritmos consideran que el mundo es estático, lo que implica que el índice temporal puede ser omitido del mapa. Además, se asume que el movimiento del robot es independiente del mapa. Esto conduce a la forma final del filtro Bayesiano para el problema de la cartografía robótica

$$p(s_t, m|z^t, u^t) = \mu p(z_t|s_t, m) \int p(s_t, m|u_t, s_{t-1}) p(s_{t-1}, m|z^{t-1}, u^{t-1}) ds_{t-1} \tag{7.7}$$

Para poner el estimador mostrado en la ecuación 7.7 en funcionamiento se deben especificar dos probabilidades generativas: $p(s_t, m|u_t, s_{t-1})$ y $p(z_t|s_t, m)$. Se asume habitualmente que estas distribuciones son invariantes, es decir, que no dependen del tiempo. Ambas constituyen los modelos generativos del robot y su entorno. La primera se conoce como *modelo de movimiento*, mientras que la segunda es el *modelo de percepción*. El modelo de movimiento determina el efecto de un control u sobre la pose del robot s. El modelo de percepción describe, en términos probabilísticos, cómo las medidas del sensor z son generadas para diferentes poses s y mapas m [66].

7.3. Filtro de Kalman Extendido (Extended Kalman Filter o EKF)

Como se ha analizado en la sección 7.2, el filtro de Kalman afronta el problema general de estimar el estado $x \in R^n$ de un proceso controlado de modo discreto en el tiempo, dirigido por una ecuación en diferencias, ***lineal*** estocástica [70]. Sin embargo, el asumir transiciones de estado y medidas lineales con ruido Gaussiano añadido, rara vez se corresponde con la realidad [67]. Por ejemplo, un robot que se mueve con velocidad de traslación y rotación constantes, típicamente se mueve en una trayectoria circular, que no puede ser descrita por transiciones de estado lineales. El filtro de Kalman Extendido no realiza la asunción de la linealidad. Ahora se asume que la probabilidad del nuevo estado y la de medida están gobernadas por funciones no lineales [31, 70, 67] g y h , respectivamente:

$$x_t = g(u_t, x_{t-1}) + \epsilon_t \tag{7.8}$$

$$z_t = h(x_t) + \delta_t \tag{7.9}$$

El EKF calcula una aproximación de la distribución de probabilidad del estado [67]. Esta aproximación se representa con una Gaussiana. En particular x_t para tiempo t se representa por una media μ_t y una convarianza Σ_t. Para poder llevar a cabo los cálculos, EKF aproxima

las funciones no lineales g y h mediante una expansión de Taylor de primer orden:

$$\begin{aligned} g'(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1}) && (7.10)\\ &= g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) && (7.11)\\ h(x_t) &\approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)(x_t - \bar{\mu}_t) && (7.12)\\ &= h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) && (7.13) \end{aligned}$$

G_t es una matriz de n x n, siendo n la dimensión del vector de estado. Se trata de la matriz *Jacobiana* del modelo de transición de estado. El valor del G_t depende de u_t y μ_{t-1}, de modo que varía para diferentes puntos en el tiempo. H_t es la matriz Jacobiana del modelo de observación. En este caso la expansión de Taylor se realiza alrededor de $\bar{\mu}_t$, el estado más probablemente ocupado por el robot en el momento de la linealización.

A partir de consideraciones probabilísticas se llega a la derivación del algoritmo del filtro de Kalman Extendido, mostrado en la figura 7.3. Como ya se ha mencionado en la sección 7.2 las ecuaciones de los filtros de Kalman se dividen en dos grupos: ecuaciones de predicción y ecuaciones de corrección. Las ecuaciones de predicción se corresponden con las líneas 2 y 3 algoritmo, y en ellas se proyecta hacia adelante en el tiempo la estimación del estado y de su covarianza, obteniendo la que se denomina estimación a priori. Las ecuaciones de corrección se corresponden con las líneas 4, 5 y 6 del algoritmo, y en ellas se calcula la ganancia de Kalman K_t y y se corrige la estimación a priori, incorporando una medida, para obtener la denominada estimación a posteriori. K_t modela la diferente *credibilidad* que se asigna a la predicción y a la medida, a la hora de calcular la estimación a posteriori.

1:	Algorithm **ExtendedKalmanFilter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2:	$\bar{\mu}_t = g(u_t, \mu_{t-1})$
3:	$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4:	$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5:	$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:	$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
7:	return μ_t, Σ_t
8:	endAlgorithm

Tabla 7.3: Algoritmo del Filtro de Kalman Extendido

7.4. Detalles de implementación

Antes de comentar aspectos específicos de la implementación es necesario definir los modelos de movimiento y observación que va a soportar nuestra aplicación. Como plataforma robótica, que determina el modelo de movimiento, se ha seleccionado la denominada *plataforma diferencial* [67], por ser el tipo de plataforma disponible en el laboratorio, y por ser muy empleada en entornos de investigación. En el laborario se dispone de la plataforma diferencial

Pioneer P3-DX [23] de la empresa MobileRobots (antes conocida como ActiveMedia). No obstante, el diseño de la aplicación permite la integración de otro tipo de plataformas de un modo sencillo. Como sensor se ha seleccionado un sensor de rango láser SICK LMS-200 [53], por ser el tipo de medidor disponible en el laboratorio que proporciona suficiente precisión como para poder ser empleado en la construcción de mapas.

En la presente sección vamos a analizar los modelos matemáticos que se derivan del empleo de una plataforma diferencial y un sensor de rango láser, como medios de movimiento y de captación de datos respectivamente. Estos modelos son el *modelo de movimiento* y el *modelo de observación*.

En la sección 7.4.1 obtendremos el modelo de movimiento, a partir de las propiedades de la plataforma diferencial, además de comentar aspectos concretos de su implementación. Análogamente en la sección 7.4.2 obtendremos el modelo de observación a partir de las características de un sensor de rango láser, y comentaremos aspectos concretos de su implementación. Por último, en la sección 7.4.3 se comentan una serie de parámetros de los modelos que deben ser seleccionados a priori.

7.4.1. Plataforma diferencial. Modelo de movimiento

Se describe a continuación el denominado *modelo de movimiento basado en velocidad* [67]. Este modelo supone que los controles que se aplican al robot son una velocidad de traslación v y una velocidad de rotación ω, durante un cierto intervalo de tiempo Δt. Lo cierto es que muchas plataformas robóticas proporcionan esta información, tras procesar la información de los codificadores o *encoders* de sus ruedas.

Sea $x_{t-1} = (x, y, \theta)^T$ la pose inicial del robot, y supongamos que mantenemos la velocidad constante a $(v\ \omega)^T$ en el intervalo Δt. El robot recorrerá un cierto arco correspondiente a un circulo cuyo centro se encuentra en [67]:

$$x_c = x - \frac{v}{\omega}\sin\theta \tag{7.14}$$

$$y_c = y + \frac{v}{\omega}\cos\theta \tag{7.15}$$

Tras un cierto intervalo de tiempo Δt el robot se encontrará en $x_t = (x', y', \theta')^T$ con:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x_c + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ y_c - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \theta + \omega\Delta t \end{pmatrix} \tag{7.16}$$

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ \frac{v}{\omega}\cos\theta - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \omega\Delta t \end{pmatrix} \tag{7.17}$$

Este constituye un modelo simple, pero muy empleado en robótica. Este modelo se es-

cuentra afectado por cierto ruido ε de modo que:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta+\omega\Delta t) \\ \frac{v}{\omega}\cos\theta - \frac{v}{\omega}\cos(\theta+\omega\Delta t) \\ \omega\Delta t \end{pmatrix} + \begin{pmatrix} \varepsilon_{\sigma_x^2} \\ \varepsilon_{\sigma_y^2} \\ \varepsilon_{\sigma_\theta^2} \end{pmatrix} \tag{7.18}$$

donde $\varepsilon_{\sigma_x^2}$, $\varepsilon_{\sigma_y^2}$ y $\varepsilon_{\sigma_\theta^2}$ son variables Gaussianas con media cero y varianzas σ_x^2, σ_y^2 y σ_θ^2, respectivamente.

7.4.2. Sensor de rango láser. Modelo de observación

En robótica móvil, es común denominar características, hitos, marcas del terreno o localizaciones a determinados objetos relevantes del entorno. Por ejemplo, en entornos *interiores* una característica puede ser una pata de una mesa, o una puerta; en entornos exteriores puede ser el tronco de un árbol, o la esquina de un edificio.

El modelo más común para procesar hitos asume que el sensor puede medir el rango (módulo del vector que une el centro de coordenadas y el hito) y el ángulo (ángulo del vector que une el centro de coordenadas y el objeto) del hito, relativos al sistema de coordenadas del robot.

Si denominamos r al rango, y ϕ al ángulo, el *vector de observaciones* es una colección de duplas:

$$f(z_t) = \{f_t^1, f_t^2, ...\} = \{\begin{pmatrix} r_t^1 \\ \phi_t^1 \end{pmatrix}, \begin{pmatrix} r_t^2 \\ \phi_t^2 \end{pmatrix}, ...\} \tag{7.19}$$

donde z_t son datos sensoriales obtenidos para tiempo t. El número de características identificadas en cada paso es variable. Además, muchos algoritmos de robótica probabilística asumen independencia condicional entre las distintas características, esto es,

$$p(f(z_t)|x_t, m) = \prod_i p(r_t^i, \phi_t^i|x_t, m) \tag{7.20}$$

donde x_t es la pose del robot para tiempo t, y m el mapa en la memoria del robot. La independencia condicional es válida si el ruido que afecta a cada medida $(r_t^i, \phi_t^i)^T$ es independiente del ruido que afecta a las demás.

Los modelos de medida basados en hitos referenciales se definen habitualmente solo para mapas basados en características. Estos mapas consisten en una lista de características, $m = \{m_1, m_2, ...\}$, cada una con unas coordenadas $(m_{i,x}, m_{i,y})$.

El vector de medida para un sensor de hitos libre de ruido se especifica fácilmente empleando relaciones geométricas. El ruido se modela por medio de ruido Gaussiano independiente para el rango ρ y el ángulo θ. El modelo de medida resultante se formula para el caso donde la característica i-ésima en tiempo t corresponde con el hito j-ésimo en el mapa. La pose del robot viene dada por $x_t = (x, y, \theta)^T$.

$$\begin{pmatrix} r_t^i \\ \phi_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x}-x)^2 + (m_{j,y}-y)^2} \\ atan2(m_{j,y}-y, m_{j,x}-x) - \theta \end{pmatrix} + \begin{pmatrix} \delta_{\sigma_r^2} \\ \delta_{\sigma_\theta^2} \end{pmatrix} \tag{7.21}$$

Aquí $\delta_{\sigma_r^2}$ y $\delta_{\sigma_\theta^2}$ son variables Gaussianas con media cero y varianzas σ_r^2 y σ_θ^2, respectivamente.

Para implementar este modelo de medida, es necesario definir una variable que establezca la correspondencia entre la característica f_t^i y el hito m_j del mapa. Esta variable se denota por c_t^i e indica unívocamente al identidad de la característica observada, de modo que $j = c_t^i$ significa que el hito observado f_t^i se corresponde con el hito del mapa m_j.

Dado que en nuestra situación de partida no se conoce esta correspondencia entre hitos en el mapa y características encontradas en una medida, habrá que desarrollar un método para construir el vector de correspondencias c_t. Este problema se conoce como *asociación de datos* (ver capítulo 5). Una vez conocido o calculado c_t puede procederse a calcular el *Jacobiano* del modelo de observación.

El *Jacobiano* del *modelo de observación*, necesario para poder emplear un EKF para establecer la localización de las características es:

$$H_t^i = \begin{pmatrix} \frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q_t}} & \frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q_t}} \\ \frac{\bar{\mu}_{t,y} - m_{j,y}}{q_t} & \frac{m_{j,x} - \bar{\mu}_{t,x}}{q_t} \end{pmatrix} \tag{7.22}$$

con $q_t = (m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2$, y $j = c_t^i$ si el hito corresponde con la medida z_t^i.

Otra relación importante es el *modelo inverso de observación*, que permite trasladar las coordenadas de una observación, desde un sistema de coordenadas centrado en el robot (m_{x_0}, m_{y_0}), hasta el sistema de coordenadas global (m_x, m_y). Este modelo se emplea a la hora de incorporar hitos a un mapa:

$$\begin{pmatrix} m_x \\ m_y \end{pmatrix} = \begin{pmatrix} m_{x_0}\cos(\theta) - m_{y_0}\sin(\theta) + x_t \\ m_{y_0}\sin(\theta) + m_{y_0}\cos(\theta) + y_t \end{pmatrix} \tag{7.23}$$

Estos modelos cambian si el tipo de hito que se está detectando es una línea recta. Es común emplear como hito para SLAM líneas rectas infinitas, representadas en su forma polar

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{7.24}$$

de este modo una recta *infinita* viene representada por el par (ρ, θ). El modelo de medida resultante se formula para el caso donde la característica i-ésima a tiempo t corresponde con el hito j-ésimo en el mapa. La pose del robot viene dada por $x_t = (x_r, y_r, \theta_r)^T$.

$$\begin{pmatrix} \rho_i' \\ \theta_i' \end{pmatrix} = \begin{pmatrix} \rho_j \\ \theta_j \end{pmatrix} - \begin{pmatrix} x_r\cos(\theta_j) + y_r\sin(\theta_j) \\ \theta_r \end{pmatrix} + \begin{pmatrix} \delta_{\sigma_\rho^2} \\ \delta_{\sigma_\theta^2} \end{pmatrix} \tag{7.25}$$

Aquí $\delta_{\sigma_\rho^2}$ y $\delta_{\sigma_\theta^2}$ son variables Gaussianas con media cero y varianzas σ_ρ^2 y σ_θ^2, respectivamente. El *Jacobiano* de este modelo, necesario para aplicar un *Filtro de Kalman Extendido (EKF)*, para la localización de los hitos, es:

$$H_t^i = \begin{pmatrix} 0 & x_r\sin(\theta_j) - y_r\cos(\theta_j) \\ 1 & 0 \end{pmatrix} \tag{7.26}$$

Otra relación importante es el *modelo inverso de observación*, que permite trasladar las

coordenadas de una observación, desde un sistema de coordenadas centrado en el robot, hasta el sistema de coordenadas global. Este modelo se emplea a la hora de incorporar hitos a un mapa:

$$\begin{pmatrix} \rho'_j \\ \theta'_j \end{pmatrix} = \begin{pmatrix} \rho_i \\ \theta_i \end{pmatrix} + \begin{pmatrix} x_r \cos(\theta_i - \theta_r) - y_r \sin(\theta_i - \theta_r) \\ \theta_r \end{pmatrix} \tag{7.27}$$

7.4.3. Parámetros

El los modelos vistos anteriormente hay una serie de parámetros que deben seleccionarse a mano, cuyos valores influencian de modo notable los resultados obtenidos. Estos parámetros son los siguientes, donde las cantidades entre corchetes indican un posible rango de variación de ese elemento de la covarianza:

- La covarianza del ruido del modelo de movimiento

$$R_t = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix} = \begin{pmatrix} [0{,}003 - 0{,}08] & 0 & 0 \\ 0 & [0{,}003 - 0{,}08] & 0 \\ 0 & 0 & [0{,}003 - 0{,}008] \end{pmatrix}$$

- La covarianza del ruido del modelo de medida, percepción u observación

$$Q_t = \begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix} = \begin{pmatrix} [0{,}01 - 0{,}08] & 0 \\ 0 & [0{,}0008 - 0{,}003] \end{pmatrix}$$

Las pruebas realizadas acerca de la influencia de estos parámetros se muestran en la sección 10.5.

Capítulo 8

Análisis

8.1. Introducción

Abordar un problema como la prueba y evaluación del algoritmo FastSLAM no es una tarea sencilla. Una forma de reducir significativamente los riesgos inherentes al desarrollo es aplicar los dictados de la ingeniería del software, que nos proporcionan unos principios metodológicos fundamentados en la experiencia y madurados a lo largo de los años.

La primera etapa en todo proyecto es la fase de análisis. Se trata de determinar exactamente qué es lo que queremos *construir*, qué servicios debe proporcionar y cuáles son las restricciones (funcionales, hardware, tolerancia a fallos, etc, $\cdots$) que debe cumplir. Además es necesario definir las herramientas de desarrollo que van a emplearse. En nuestro caso queremos definir un entorno de pruebas que debe integrar:

- Software desarrollado a medida, en concreto se trata de la aplicación que ejecuta el algoritmo FastSLAM.
- Software desarrollado por terceros.
  - Herramientas para la presentación de resultados, como procesador de textos, hoja de cálculo, tratamiento de imágenes, etc.
  - Herramientas de control y simulación de robots, para la obtención de datos que alimenten el algoritmo.
- Fuentes de datos externas, es decir, archivos de registro de recorridos robóticos.

Además, para llevar a cabo todo el desarrollo del proyecto serán necesarias:

- Herramientas de programación, es decir, compilador, depurador, diseñador de interfaces gráficas, etc, $\cdots$.
- Herramientas de diseño UML.
- Herramientas de planificación y control.

Todos estos aspectos se estudian en profundidad y se definen detalladamente en el presente capítulo. En la sección 8.2 se define el entorno de prueba y evaluación, y se profundiza en el análisis de la aplicación desarrollada a medida. En la sección 8.3 se definen todas y cada una de las herramientas que forman parte tanto del desarrollo, como del entorno final. Seguidamente, en la sección 8.4 se definen las fuentes de datos que van a emplearse, y por último en la sección 8.5 se estudia el tipo de resultados que debe proporcionar el entorno.

8.2. Requisitos

8.2.1. Visión

El objetivo último de prueba y evaluación del algoritmo FastSLAM, impone la necesidad de definir un entorno para llevar a cabo las mencionadas pruebas, obtener datos, y realizar la evaluación. En este sentido decidimos desarrollar una aplicación a medida para la prueba y obtención de datos, realizando el posterior análisis de los datos con el apoyo de herramientas ya existentes. Inicialmente nuestra aplicación se alimenta de archivos de registro (archivos de *log*) de recorridos robóticos, tanto simulados como reales, aunque debe ser flexible y estar preparada para la incorporación futura de otras fuentes de datos. Para la presentación de los resultados se van a emplear mapas de rejilla, muy ilustrativos a nivel visual, y gráficas de error del mapa generado, e hitos detectados, en función de la variación de los distintos parámetros que afectan al algoritmo.

Por lo tanto puede decirse que uno de los primeros objetivos del proyecto es el desarrollo de una aplicación que ejecute el algoritmo FastSLAM, para su integración en el entorno de prueba y evaluación (en este sentido, debe considerarse que todos los capítulos anteriores en los que se explica el algoritmo forman parte del análisis, ya que es imprescindible comprender el algoritmo para poder implementarlo). La mencionada aplicación debe permitir probar y extraer información para evaluar el algoritmo de localización y construcción simultánea de mapas FastSLAM 1.0. Por lo tanto deberá soportar diferentes configuraciones, tanto de las fuentes de datos, como de los parámetros intrínsecos del propio algoritmo (métodos de extracción de características, número de partículas, etc). Estas configuraciones deben poder definirse manualmente, cargarse y grabarse en un archivo, o establecerse como predeterminadas. También debe permitir la ejecución por lotes e interactiva. La aplicación debe proporcionar datos sobre la *calidad* del mapa obtenido, que serán tratados posteriormente por herramientas externas.

Otra característica importante de la aplicación es que sus usuarios van a ser personal investigador. Esto ha de tenerse en cuenta a la hora de definir las distintas interfaces de usuario, ya sean gráficas o de línea de comandos.

Entre las opciones de visualización que aportan información útil, y facilitan la comprensión del proceso encontramos:

- Visualización de la configuración actual de los parámetros más relevantes del algoritmo.
- Visualización del barrido láser actual en coordenadas del robot, con características identificadas, incluyendo la nube de partículas.

- Visualización del mapa de rejilla aproximado construido mediante la contribución en cada paso de la que en ese momento sea la mejor partícula.

Fuera de un contexto de ejecución propiamente dicho son interesantes las siguientes opciones:

- Visualización del mapa de rejilla obtenido por aplicación única de la mejor partícula en un momento dado.
- Visualización del mapa de rejilla obtenido atendiendo únicamente a la información odométrica.
- Volcado de mapas y barridos láser a un formato gráfico.
- Ejecución por lotes de simulaciones para el estudio de la influencia de la variación de un determinado parámetro.
- Carga y almacenamiento en disco de configuraciones, y establecimiento de una configuración como predeterminada.

8.2.2. Modelo de casos de uso

El análisis de los casos de uso constituye una herramienta fundamental para la captura de requisitos de una aplicación o un sistema. En la figura 8.1 se muestra un diagrama de casos de uso con los principales casos de uso del usuario *investigador*.

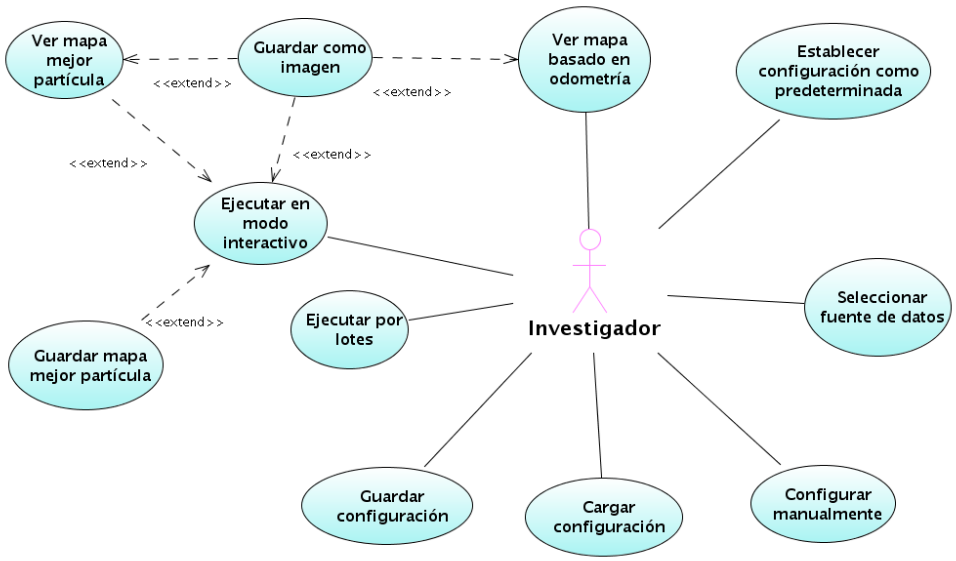


Figura 8.1: Casos de uso más relevantes del simulador.

A continuación se describe cada uno de los casos de uso. Encontramos dos tipos de casos, por un lado aquellos relacionados con la configuración y por otro los relacionados con la ejecución y obtención de resultados.

8.2.2.1. Seleccionar origen de datos

Identificador	SELECCIONAR ORIGEN DE DATOS
Versión/Fecha	V0.5- 23/08/2007
Actores	Investigador
Descripción	El usuario selecciona el tipo de archivo fuente de datos, y el propio archivo.
Flujo Normal	1. El usuario selecciona el tipo de archivo de un combo box con las diferentes opciones. 2. El usuario selecciona el archivo de datos. Debe existir concordancia entre el tipo y el archivo seleccionado. 3. Se notifica al usuario la correcta carga del archivo.
Flujo Alternativo	1. Si existe algún problema, como que el archivo no existe, o el formato seleccionado y el formato del archivo no coinciden, se informa al usuario y se vuelve a mostrar el formulario de selección de tipo y archivo.
Precondición	No tiene
Postcondición	Origen configurado
Rendimiento	No existen restricciones específicas.
Frecuencia	Habitual, es uno de los modos de cambiar el mapa que se está simulando.
Importancia	Alta, es uno de los puntos de entrada al programa.
Urgencia	La más alta, es uno de los puntos de entrada al programa.
Comentarios	En realidad la mayoría de las veces se cambiará de mapa cambiando el archivo de configuración.

8.2.2.2. Cargar un archivo de configuración

Identificador	CARGAR ARCHIVO DE CONFIGURACIÓN
Versión/Fecha	v0.2 – 20/06/2007
Actores	Investigador

Descripción	El usuario carga una configuración de parámetros del algoritmo desde un archivo de configuración. Como el formato del archivo es perfectamente conocido se garantiza que la carga sólo se realizará si el archivo es un verdadero archivo de configuración. La configuración incluye los parámetros del algoritmo, y el archivo fuente de los datos.
Flujo Normal	1. El usuario selecciona un archivo mediante un cuadro de diálogo de selección de archivo. 2. El sistema avisa de que se va a producir un cambio en la configuración y pide confirmación. 3. Si todo es correcto se vuelve a la ventana principal del simulador.
Flujo Alternativo	1. Si el archivo no existe o ocurre algún tipo de error durante el proceso de carga, se notifica al usuario y se vuelve a mostrar el cuadro de diálogo de selección de archivo.
Precondición	No tiene.
Postcondición	Origen configurado.
Rendimiento	No existen restricciones específicas.
Frecuencia	Se emplea habitualmente, ya que permite probar diversas situaciones cómodamente.
Importancia	Alta, debe ser cómodo ya que se usará mucho.
Urgencia	Secundario, es más importante poder configurar manualmente, al menos inicialmente.
Comentarios	Se espera que durante las primeras ejecuciones se emplee mucho la configuración manual, sin embargo con el tiempo el usuario habrá grabado la mayoría de las configuraciones con lo que el uso de la configuración manual será para afinar una configuración cargada.

8.2.2.3. Guardar configuración en un archivo

Identificador	GUARDAR CONFIGURACIÓN
Versión/Fecha	v0.3 – 23/08/2007
Actores	Investigador

Descripción	El usuario desea guardar la configuración actual del simulador en un archivo que posibilite su posterior recuperación. Los archivos de configuración tienen un formato preestablecido.
Flujo Normal	1. El usuario selecciona un archivo mediante un cuadro de diálogo de selección de archivo. 2. Si la configuración se ha guardado correctamente se avisa al usuario.
Flujo Alternativo	1. Si el archivo ya existe se pide confirmación para sobreescribirlo. 2. Ante cualquier otro fallo se informa y se vuelve a la ventana de selección de archivo.
Precondición	Origen configurado.
Postcondición	No tiene.
Rendimiento	No existen restricciones específicas.
Frecuencia	Se emplea habitualmente, ya que el objeto de la aplicación es probar el algoritmo en diferentes configuraciones.
Importancia	Alta, debe ser cómodo ya que se usará mucho.
Urgencia	Secundario, es más importante poder configurar manualmente.
Comentarios	

8.2.2.4. Configurar manualmente

Identificador	CONFIGURAR MANUALMENTE
Versión/Fecha	V 0.2 – 23/08/2007
Actores	Investigador

Descripción	Se trata de configurar todos aquellos parámetros que modifican el funcionamiento del algoritmo. Los parámetros modificables son: ▪ Número de partículas. ▪ Tipo de características (Punto , Recta , Ambas) ▪ Probabilidad de una nueva observación ▪ Para el algoritmo Slit & Merge es posible configurar: • Mínimo número de puntos en una recta • Máxima distancia a la recta • Mínima longitud de una recta ▪ Modelo del robot (Diferencial con odometría , Diferencial sin odometría) ▪ Covarianza del modelo de observación($\sigma_x^2, \sigma_y^2, \sigma_\theta^2$) ▪ Algoritmo de remuestreo: • Tipo de remuestreador (R. con umbral N_{eff}, R sin umbral) • Número de partículas efectivas (cuando sea aplicable)
Flujo Normal	1. El usuario introduce los parámetros de configuración en un único formulario. 2. El usuario confirma. 3. El sistema comprueba que los parámetros constituyen una configuración correcta. 4. En caso de configuración correcta, el sistema muestra un mensaje, pide confirmación, y actúa en consecuencia. 5. Si el usuario Cancela se vuelve a la ventana principal, sin modificar la configuración.

Flujo alternativo	1. En caso de producirse algún error se muestra información sobre el mismo y se vuelve al formulario de configuración.
Precondición	No tiene.
Postcondición	No tiene.
Rendimiento	No existen restricciones.
Frecuencia	Uso habitual.
Importancia	Fundamental.
Urgencia	Debe ser una de las primeras funcionalidades.
Comentarios	Dado que el sistema debe permitir evaluar diferentes configuraciones del algoritmo, este caso de uso se convierte en fundamental.

8.2.2.5. Establecer una configuración por defecto

Identificador	ESTABLECER CONFIGURACIÓN POR DEFECTO
Versión/Fecha	v0.2 – 23/08/2007
Actores	Investigador
Descripción	El usuario puede convertir la configuración actual del algoritmo en la configuración de arranque por defecto. Esto lo va a poder realizar desde el menú de configuración del algoritmo.
Flujo Normal	1. El usuario solicita convertir la configuración actual en configuración por defecto. 2. El sistema solicita confirmación de la operación. 3. Se sobreescribe el archivo de configuración por defecto (init.cfg).
Precondición	No tiene.
Postcondición	No tiene.
Rendimiento	No existen restricciones.
Frecuencia	Poco frecuente.
Importancia	No es crítico para el software, aunque añade comodidad de uso.
Urgencia	No es urgente.
Comentarios	Esta opción aumenta la sensación de comodidad al usar el software.

8.2.2.6. Ejecutar de modo interactivo

Identificador	EJECUTAR EN MODO INTERACTIVO
Versión/Fecha	v0.2 – 20/06/2007
Actores	Investigador
Descripción	Se ejecuta el algoritmo observando la evolución temporal del mismo. Es posible ejecutar hasta el fin, detener y reanudar la ejecución, o ejecutar únicamente un paso. La visualización abarca dos aspectos, por un lado el barrido láser actual, con la nube de puntos y los hitos detectados, y por otro el mapa de rejilla realizado por la contribución de la mejor partícula en cada paso.
Extendido por	Ver mapa de la mejor partícula (sección 8.2.2.8), Guardar el mapa de la mejor partícula (sección 8.2.2.9), Guardar como imagen (sección 8.2.2.10).
Flujo Normal	1. El usuario comienza la ejecución, ya sea indefinida o de un paso. 2. En cualquier momento el usuario puede detener la ejecución o finalizarla. Si se finaliza la ejecución, la siguiente vez que se comience a ejecutar, lo hará desde el principio. Si simplemente se detiene la ejecución, se continuará posteriormente en el punto en el que se detuvo. 3. Cuando se alcanza el final del archivo fuente de datos se desactivan los botones de ejecutar.
Precondición	Origen de datos configurado.
Postcondición	Ejecutando.
Rendimiento	Es crucial un rendimiento adecuado de la ejecución del algoritmo.
Frecuencia	Constantemente utilizado, es la razón de ser de la herramienta.
Importancia	Fundamental.
Urgencia	Elevada.
Comentarios	Esta parte va a ser la que realmente se perciba como aplicación y de la que los investigadores más información van a extraer, a pesar de que exista una interfaz de línea de comandos para extraer datos.

8.2.2.7. Ver mapa basado en la odometría

Identificador	VER MAPA ODOMETRÍA
Versión/Fecha	v0.2 – 23/08/2007

Actores	Investigador
Descripción	Se muestra la evolución del mapa de rejilla obtenido teniendo en cuenta únicamente la información odométrica.
Extendido por	Guardar como Imagen (sección 8.2.2.10)
Flujo Normal	1. El usuario solicita la visualización del mapa. 2. Se muestra en una ventana nueva la *evolución* del mapa. El usuario puede detener el proceso en cualquier momento.
Precondición	Origen configurado.
Postcondición	No tiene.
Rendimiento	No hay restricciones
Frecuencia	Esporádica
Importancia	Secundaria.
Urgencia	Poca.
Comentarios	A pesar de ser una funcionalidad simple, aporta mucha información, al permitir observar la mejoría del mapa obtenido aplicando el algoritmo.

8.2.2.8. Ver mapa de la mejor partícula

Identificador	VER MAPA DE LA MEJOR PARTÍCULA
Versión/Fecha	v0.2 – 23/08/2007
Actores	Investigador
Descripción	Se muestra el mapa de rejilla obtenido por la mejor partícula hasta el momento presente de la simulación. No se muestra su evolución, sino su estado final.
Extendido por	Guardar como Imagen (sección 8.2.2.10)
Extiende a	Ejecutar de modo interactivo (sección 8.2.2.6)
Flujo Normal	1. El usuario solicita la visualización del mapa. 2. Se muestra en una ventana el mapa resultado.
Precondición	Ejecutando.
Postcondición	No tiene.
Rendimiento	No hay restricciones
Frecuencia	Habitual
Importancia	Moderada.
Urgencia	Poca.

Comentarios	En el fondo este es uno de los productos de la ejecución del algoritmo.

8.2.2.9. Guardar el mapa de la mejor partícula

Identificador	GUARDAR MAPA DE LA MEJOR PARTÍCULA
Versión/Fecha	v0.2 – 23/08/2007
Actores	Investigador
Descripción	Se almacenan en un archivo los hitos referenciales o características que integran el mapa de la mejor partícula hasta el momento presente de la simulación.
Extendido por	Guardar como Imagen (sección 8.2.2.10)
Extiende a	Ejecutar interactivamente (sección 8.2.2.6)
Flujo Normal	1. El usuario solicita archivar el mapa de la mejor partícula. Se muestra un cuadro de diálogo de selección de archivo para que se especifique un archivo destino. 2. Si todo va bien se vuelve a la ventana principal. 3. En caso de error se notifica al usuario y se vuelve a mostrar el cuadro de diálogo de selección de archivo.
Precondición	Ejecutando.
Postcondición	No tiene.
Rendimiento	No hay restricciones
Frecuencia	Habitual
Importancia	Alta.
Urgencia	Alta.
Comentarios	En el fondo este es uno de los productos de la ejecución del algoritmo.

8.2.2.10. Guardar como imagen

Identificador	GUARDAR COMO IMAGEN
Versión/Fecha	v0.2 – 23/08/2007
Actores	Investigador
Descripción	Se almacena en formato jpg cualquiera de los mapas de rejilla o visualizaciones mostradas por el simulador.

Extiende a	Ejecutar interactivamente (sección 8.2.2.6), Ver mapa basado en la odometría (sección 8.2.2.7), Ver mapa de la mejor partícula (sección 8.2.2.8).
Flujo Normal	1. El usuario hace click con el botón derecho del ratón sobre el mapa o visualización que desea guardar como imagen. 2. Aparece un menú contextual con la opción de guardar como imagen. 3. Se muestra un cuadro de diálogo de selección de archivo. 4. Si el archivo destino ya existe se notifica y se pide confirmación.
Precondición	No tiene.
Postcondición	No tiene.
Rendimiento	No hay restricciones
Frecuencia	Habitual
Importancia	Alta.
Urgencia	Alta.
Comentarios	En el fondo este es uno de los productos de la ejecución del algoritmo.

8.2.2.11. Ejecutar por lotes

Identificador	EJECUTAR POR LOTES
Versión/Fecha	v0.2 – 23/08/2007
Actores	Investigador
Descripción	El usuario puede ejecutar una serie de simulaciones en las que desea variar uno de los parámetros de la simulación, además puede obtener una medida de la calidad del mapa obtenido en cada una de las simulaciones. Este tipo de funcionalidad se proporciona más adecuadamente desde una interfaz de línea de comandos.

Flujo Normal	1. El usuario especifica el archivo origen de los datos, el parámetro que desea variar y los distintos valores del mismo. Además debe especificar una raíz para el nombre de los archivos de salida que van a ser los mapas de la mejores partículas de cada simulación y un archivo con datos de calidad. También debe especificarse el archivo que contiene los datos de los hitos reales presentes en el mapa, para poder medir la calidad. 2. El sistema informa sobre la simulación en curso y genera los archivos de mapas y calidad. 3. El sistema informa del final de las simulaciones.
Flujo alternativo	1. Ante cualquier contingencia, como ficheros de datos erróneos, o parámetros incorrectos, el sistema aborta la simulación actual e intenta realizar la siguiente.
Precondición	No tiene.
Postcondición	No tiene.
Rendimiento	Debe ser lo más rápido posible.
Frecuencia	Habitual
Importancia	Alta.
Urgencia	Alta.
Comentarios	Fundamental para poder evaluar el algoritmo. Este uso está orientado a mapa sintéticos de los que se dispone datos acerca de los hitos que los integran, y que por tanto permiten medir la diferencia entre ellos y el mapa obtenido tras la simulación del algoritmo.

8.3. Herramientas

La motivación de realizar un software *reutilizable*, *flexible* y *portable*, junto con el deseo de emplear un lenguaje claro, en el que sea sencillo *descubrir* el algoritmo, moderno y completamente orientado a objetos, nos conducen a emplear Java [17, 55] como lenguaje de programación. La idea de aplicar los principios de la ingeniería del software y el diseño y la programación orientados a objetos a un proyecto complejo, han sido también factores de decisión. Además Java es ahora software de libre distribución, y existen numerosos entornos de programación y librerías disponibles para su uso.

Como entorno de programación se emplea NetBeans [24], desarrollado por la empresa Sun y de libre distribución. Este entorno tiene una integración magnífica con el lenguaje Java y facilita enormemente el desarrollo. Por último como librería matemática, fundamentalmente para cálculos matriciales, se emplea JAMA [46], una librería de libre distribución con un amplio predicamento entre la comunidad científica universitaria. Por último en las etapas de análisis y diseño se han empleado la herramientas ArgoUML [68] y NetBeans para la elaboración de todo tipo de diagramas UML [6]. Estas herramientas permiten, entre otras cosas, generar esqueletos de código Java, a partir de diagramas de clases.

Todas las herramientas pueden usarse indistintamente en sistemas operativos Windows o Unix/Linux. Esto también ha supuesto un elemento de peso a la hora de tomar una decisión.

Durante el proyecto ha sido necesario tanto simular recorridos de un robot en entornos sintéticos, como grabar datos de recorridos reales. Para estos cometidos se ha empleado la herramienta Player/Stage [51], que posee una amplia difusión en ambientes de investigación. Para la elaboración de los mapas sintéticos se han empleado las herramienta Qcad [16] y Gimp [65].

Por último, toda la documentación ha sido elaborada empleando LaTeX [29], Bibtex [29] para facilitar la elaboración de la bibliografía, y Openoffice Calc[36] para la elaboración de gráficas.

En cuanto al hardware se refiere, dado el carácter de procesamiento fuera de linea, no en tiempo real, de datos recopilados previamente por sistemas robóticos reales, o fruto de la simulación, no existen restricciones ni para el equipo empleado en el desarrollo, ni para la ejecución.

A lo largo de esta sección vamos a presentar las distintas herramientas empleadas durante el desarrollo del proyecto, analizando sus principales características, y su idoneidad para ser aplicadas al proyecto.

8.3.0.1. Java

Java es un lenguaje de programación de alto nivel de propósito general, multiplataforma, semi-interpretado, orientado a objetos y multihilo.

La tecnología Java está compuesta básicamente por 2 elementos: el lenguaje Java y su plataforma. La plataforma Java se sintetiza en la arquitectura de la figura 8.2. Esta arquitectura consta fundamentalmente de dos elementos, la máquina virtual Java (JVM), y la API [17] (interfaz de programación de aplicaciones, application program interface, en inglés). Hay disponibles diferentes máquinas virtuales, algunas de las cuales ofrecen velocidades de ejecución prácticamente idénticas a las de programas escritos en C++, por lo que en rendimiento no es un factor negativo a la hora de elegir Java, a pesar de ser un lenguaje semi-interpretado. La API de Java crece constantemente proporcionando servicios al desarrollador para gran cantidad de tareas tanto de uso común como específicas de ciertos ámbitos, como el tratamiento de imágenes, el acceso a redes, etc. Además Java es un lenguaje muy seguro que realiza todo tipo de comprobaciones, tanto en tiempo de compilación, como en tiempo de ejecución, con lo que los programas resultantes suelen tener menos cantidad de errores y evitan la ejecución de código malicioso.

Java es independiente de la máquina y del Sistema Operativo lo que permite ejecutar un programa hecho en Java en, por ejemplo, un ordenador con Linux o Windows, en una PDA o en un teléfono móvil.

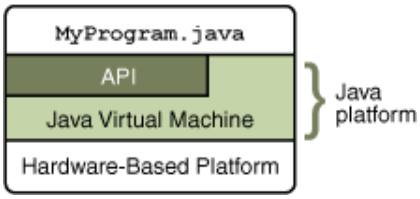


Figura 8.2: La API y la Java VM aislan el programa de la capa hardware.

El código fuente se compila a un lenguaje que no es específico de la plataforma, sino que es un código intermedio denominado "bytecode". Este código no es ejecutado directamente, ya que no es comprendido por la máquina física, sino que es interpretado por un programa denominado Máquina Virtual o más específicamente JVM (Máquina Virtual de Java). La JVM es específica de la plataforma. De esta forma para ejecutar un programa Java sólo necesitamos la máquina virtual de la plataforma que estemos usando. Así por ejemplo existen máquinas virtuales para Sistemas Operativos Windows, Linux, Apple y Solaris. La mayor parte de los navegadores web tienen una JVM instalada para poder visualizar *applets*, que son pequeñas aplicaciones Java que se incrustan en páginas web.

Java es un lenguaje de propósito general ya que con él se pueden programar desde simples aplicaciones de consola hasta grandes sistemas, pasando por applets [1], servlets [1] (programas cliente-servidor más potentes que los clásicos CGI [18]), páginas JSP [1] (Java Server Pages, similares a las páginas PHP [2] o ASP [37]), programas de acceso a bases de datos, etc.

8.3.0.2. NetBeans

NetBeans es un entorno de desarrollo integrado (IDE) de código abierto (*open source*) para desarrolladores de software [24]. El IDE se encuentra disponible para diversas plataformas, como Windows, Linux, Soraris y MacOs. Proporciona a los desarrolladores las herramientas necesarias para crear aplicaciones profesionales de escritorio, empresariales, web y móviles.

Entre todas las herramientas que incorpora, que son muchas, cabe destacar el sistema de control de versiones, la herramienta de análisis de rendimiento (*profiling*), el editor de código avanzado, el diseñador de formularios (ver figura 8.3) para diseñar interfaces gráficas de usuario, y el diseñador UML que permite tanto diseñar como obtener el diseño subyacente a un código fuente.

Las herramientas de análisis de rendimiento o *profiling* proporcionan información sobre el consumo de tiempo y memoria de una aplicación. De este modo es posible detectar puntos susceptibles de mejora y de este modo aumentar el rendimiento global de la aplicación.

El editor de código avanzado (ver figura 8.4) sangra, completa y resalta de acuerdo a la sintaxis, el código fuente. Además realiza análisis sintáctico del código mientras se escribe,

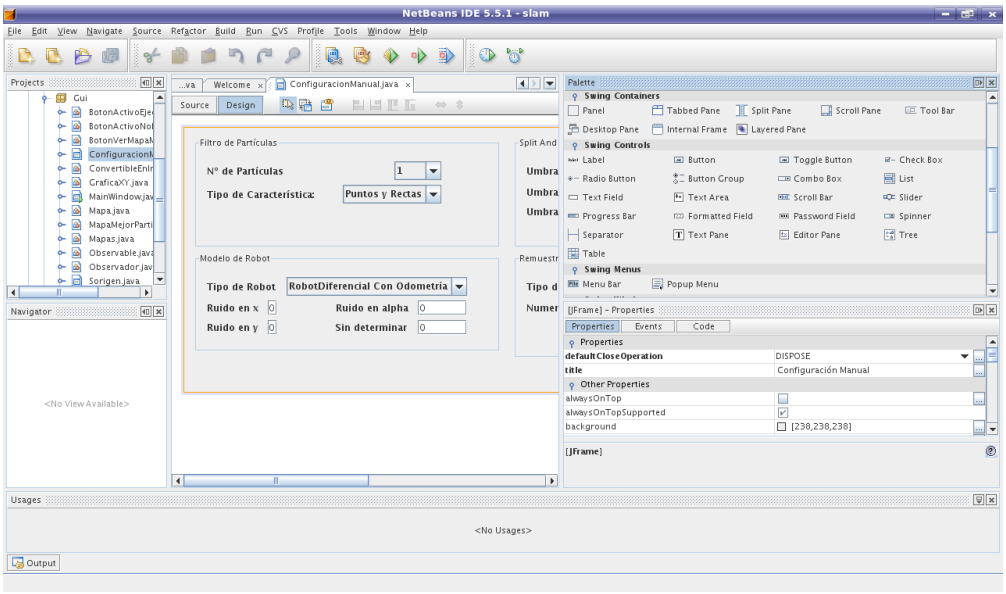


Figura 8.3: Diseñador de formularios de NetBeans

autocompleta palabras y paréntesis, resalta errores y muestra sugerencias, además de mostrar la propia documentación de la API de Java. El editor puede ser completamente modificado para ajustarlo a las necesidades del usuario, además de ofrecer integración completa con las herramientas de depuración, de refactorización y diseño.

En cuanto al diseñador UML, esta herramienta permite centrarse en el diseño y generar posteriormente un esqueleto de código, u obtener el diseño a partir de un código fuente. Es posible establecer una relación entre un diseño y un cierto código de modo que los cambios en uno se reflejen en el otro y viceversa. Entre otros, el diseñador permite crear diagramas de clase, de actividades, de colaboración, de componentes, de despliegue, etc. Además el diseñador proporciona una serie de plantillas para el empleo de patrones de diseño, como el patrón *builder*, *observer*, y otros.

Además NetBeans es una herramienta en constante desarrollo y para la que existen infinidad de complementos habitualmente descargables desde la web oficial `http://www.netbeans.org`.

8.3.0.3. JAMA

JAMA (JAva MAtrix) [46] es un paquete para Java de álgebra lineal básica. Proporciona clases a nivel de usuario para construir y manipular matrices. Su funcionalidad es suficiente para solucionar problemas rutinarios, y es fácil de entender para usuarios no expertos. De entre todas sus posibilidades a nosotros nos interesa la clase Matrix. Esta clase proporciona las operaciones fundamentales del álgebra lineal numérica. Las operaciones aritméticas básicas son suma, multiplicación y cálculo de determinantes. Además es posible calcular la matriz inversa, realizar transposiciones, resolver sistemas lineales, realizar ajustes por míni-

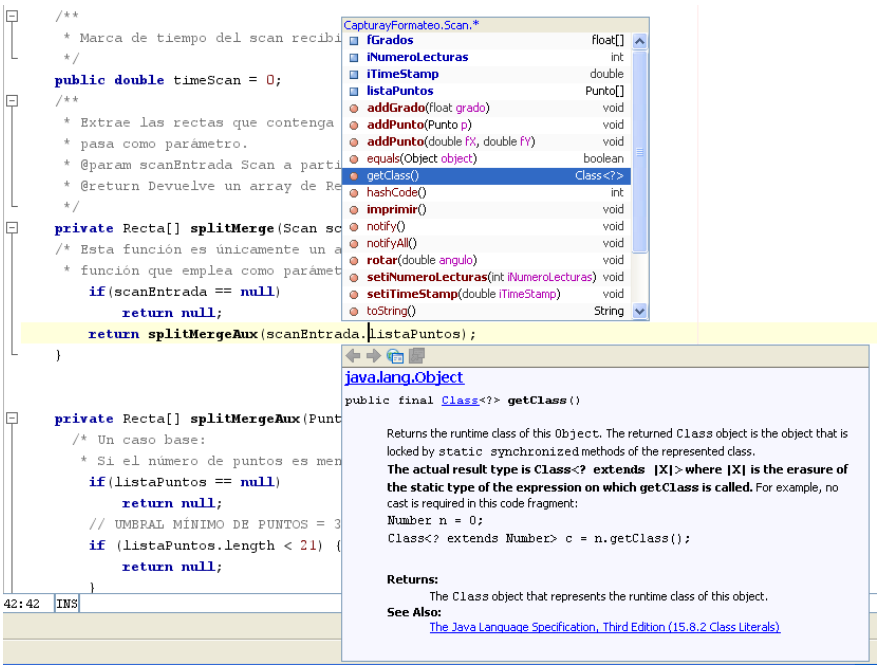


Figura 8.4: Editor de código avanzado de NetBeans.

mos cuadrados, etc.

En conjunto ofrece todas las capacidades de manipulación de matrices, y de álgebra lineal en general, que se necesitan en el presente proyecto, todo esto con un diseño puramente orientado a objetos.

8.3.0.4. ArgoUML

ArgoUML [68] es una herramienta de modelado UML de código abierto, que incluye soporte para el estándar UML 1.4 [19], que se ejecuta sobre cualquier plataforma Java. Entre sus muchas características cabe destacar el soporte de 9 tipos diferentes de diagramas UML, entre ellos los diagramas de clases, como el que puede observarse en la figura 8.5, de actividades, de casos de uso, de colaboración, de secuencia, etc. Proporciona, además, soporte para XMI [20], un estándar XML [69] desarrollado para herramientas de modelado UML. ArgoUML genera código para Java, C++, C#, PHP4 y PHP5, además permite extraer el diseño a partir de un determinado código fuente. Otra característica interesante es el aporte de críticas al diseño. En cualquier momento podemos consultar una lista de sugerencias, generadas dinámicamente, para mejorar el diseño.

El desarrollo de esta herramienta se encuentra en un momento muy activo, por lo que es probable esperar nuevas y mejores características en un futuro próximo.

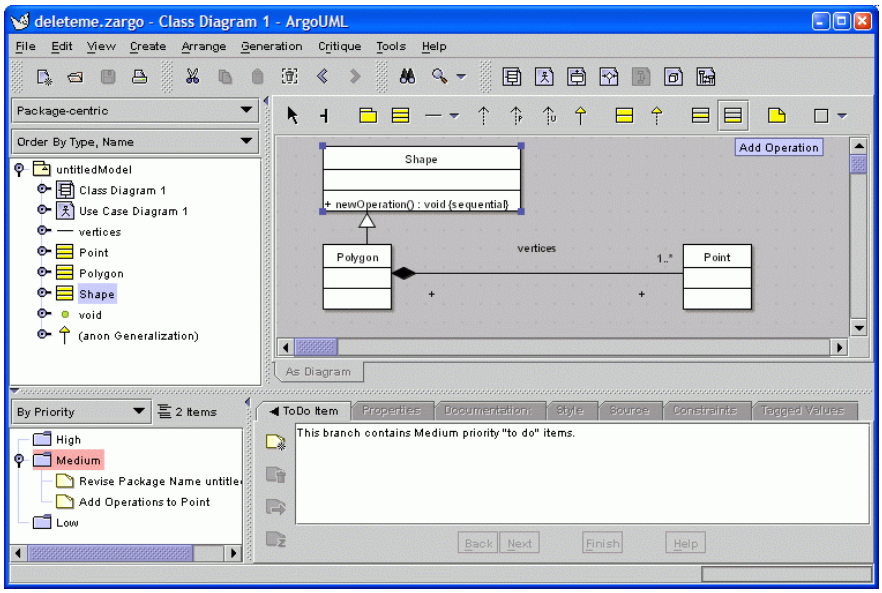


Figura 8.5: Herramienta de modelado UML, ArgoUML.

8.3.0.5. Latex

LaTeX [29] es un sistema de composición tipográfica de alta calidad. Además incluye características diseñadas para la producción de documentación técnica y científica. LaTeX es un estándar "*de facto*" para la comunicación y publicación de documentos científicos. No se trata de un procesador de textos típico, como por ejemplo Microsoft Word, de hecho se incide en no tener en cuenta la apariencia del documento, sino el contenido, y es el propio procesador LaTeX el que se encarga del aspecto final.

Entre otras, LaTeX contiene las siguientes características [29]:

- Permite generar artículos, informes técnicos, libros, presentaciones.
- Es capaz de controlar grandes documentos que contienen secciones, referencias cruzadas, tablas y figuras.
- Posibilita la composición tipográfica de fórmulas matemáticas complejas.
- Permite la generación automática de bibliografías e índices.

LaTeX se emplea integrado en una distribución TeX [21]. En concreto para Windows empleamos la distribución MikTeX, mientras que para Linux empleamos la distribución TeTeX. La razón para emplear estas distribuciones es que cada una es la más difundida para su respectivo sistema operativo.

Para realizar la edición de documentos LaTeX es conveniente emplear editores específicos que se integran fácilmente con las distribuciones TeX, y que facilitan enormemente la tarea. Tanto para Windows como para Linux empleamos Texmaker, un editor de código abierto.

8.3.0.6. Qcad

QCad [16] es un programa de diseño asistido por ordenador (CAD) de código libre para diseño 2D. Funciona en sistemas operativos Linux, Mac OS X, Unix y Microsoft Windows. Su desarrollo es llevado a cabo por la empresa Ribbonsoft. Gran parte de la interfaz (ver figura 8.6) y de los conceptos sobre su uso son iguales que los de AutoCAD. Sus características más relevantes son:

- Varios modos de creación de líneas, arcos, círculos, elipses, paralelas, ángulos bisectores, ...
- Formato DXF (DXF 2004).
- Muchas fuentes de texto CAD.
- Dimensiones en distancias, ángulos, diámetros, tolerancias, etc.
- Rellenos sólidos y rallados.
- Soporte completo para Capas y Bloques (Inserciones).

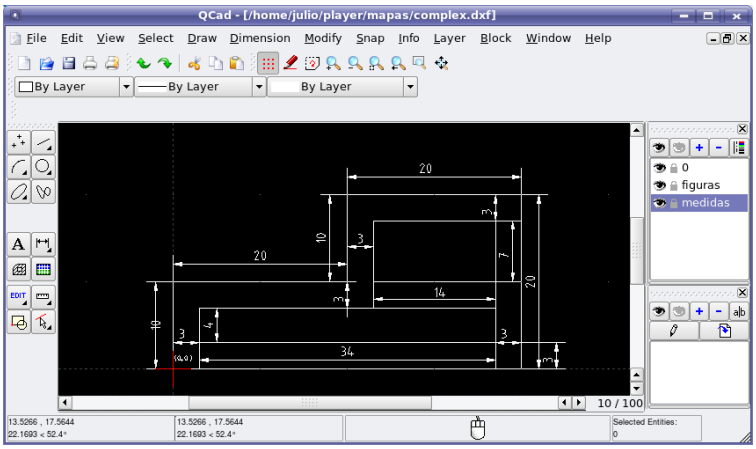


Figura 8.6: Interfaz gráfica del programa Qcad.

- Herramientas potentes de selección y modificación (mover, rotar, espejo, recortar, estirar, etc.).
- Ajuste a objetos (extremos, centros, intersecciones, etc.).
- Consola para inserción de coordenadas y ejecución de instrucciones.
- Múltiples niveles de *deshacer/rehacer*.
- Soporte para varias unidades, incluyendo métrica, imperial, grados, radianes, etc.

- Importación y exportación de mapas de bits (JPEG, PNG, etc.).
- Creación de ficheros PS que pueden portarse fácilmente a PDF.
- Interfaz de usuario traducida a múltiples idiomas: Alemán, Catalán, Checo, Danés, Eslovaco, Español, Estonio, Francés, Griego, Holandés, Húngaro, Inglés, Italiano, Polaco, Ruso y Turco.

8.3.0.7. Gimp

GIMP [65] (GNU Image Manipulation Program) es un programa editor de imágenes del proyecto GNU [14]. Se publica bajo la licencia GNU General Public License. Se trata de la alternativa más firme del software libre al popular programa de retoque fotográfico Photoshop. La primera versión se desarrolló para sistemas Unix y fue pensada especialmente para GNU/Linux, sin embargo actualmente existen versiones totalmente funcionales para Windows y para Mac OS X. La biblioteca de controles gráficos GTK [50] , desarrollada para GIMP, dio origen al entorno de ventanas de GNOME [50].

GIMP sirve para procesar gráficos y fotografías digitales. Los usos típicos incluyen la creación de gráficos y logos, el cambio de tamaño y recorte de fotografías, el cambio de colores, la combinación de imágenes usando un paradigma de capas, la eliminación de elementos no deseados de las imágenes y la conversión entre distintos formatos de imágenes. También se puede utilizar el GIMP para crear imágenes animadas sencillas. En la figura 8.7 puede observarse la paleta de herramientas de Gimp.

Figura 8.7: Paleta de herramientas del programa Gimp.

8.3.0.8. Openoffice Calc

OpenOffice.org Calc es una hoja de cálculo de código abierto y software libre compatible con Microsoft Excel. Es parte de la suite ofimática OpenOffice.org. Como con todos los componentes de la suite OpenOffice.org, Calc puede usarse a través de una variedad de plataformas, incluyendo Mac OS X, Windows, GNU/Linux, FreeBSD y Solaris.

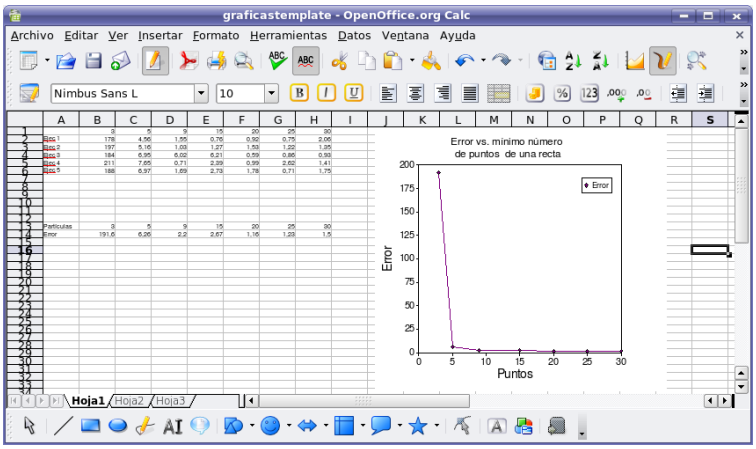


Figura 8.8: Interfaz gráfica del programa Openoffice Calc.

Calc es una hoja de cálculo similar a Microsoft Excel, con un rango de características más o menos equivalente. Su tamaño es mucho menor y proporciona un número de características no presentes en Excel, incluyendo un sistema que automáticamente define series para representar gráficamente basado en la disposición de los datos del usuario. Calc también es capaz de escribir hojas de cálculo como archivos PDF. En la figura 8.8 puede observarse su interfaz gráfica.

8.3.0.9. Player/Stage

Player [51] es es una capa de abstracción hardware (HAL) para dispositivos robóticos. Puede considerarse como un sistema operativo para robots. Player define una serie de interfaces estándares, indicando cada una, una manera de interactuar con un tipo de dispositivo hardware. Por ejemplo la interfaz *position2d*, se encarga de interactuar con robots móviles terrestres. Esta interfaz permite ejecutar comandos de movimiento (objetivos de velocidad o posición), y obtener el estado del robot controlado (velocidad y posición actuales). Hay muchos controladores que implementan esta interfaz, como *p2os*, *obot*, etc, cada uno enfocado al control de un tipo de robot específico.

Player ofrece mecanismos de trasporte que permiten intercambiar datos entre distintos controladores y programas de control que se ejecutan en distintas máquinas. El mecanismo de transporte más empleado es el cliente/servidor sobre sokets TCP. Los controladores se ejecutan dentro del servidor Player, y los programas de control de los usuarios se ejecutan en clientes de dicho servidor. Existen librerías cliente para diversos lenguajes de programación (C++, Tcl, Java y Python), para facilitar el desarrollo de dichos programas de control. Además Player permite visualizar una recreación virtual de una situación, ya sea real o simulada, tal como se aprecia en la figura 8.9.

La manera más habitual de utilizar Player es ejecutando el servidor en un robot físico, y accediendo a los dispositivos del robot mediante un programa cliente de dicho servidor.

Stage es un simulador de robots en entornos 2D. Gracias a Stage es posible simular el

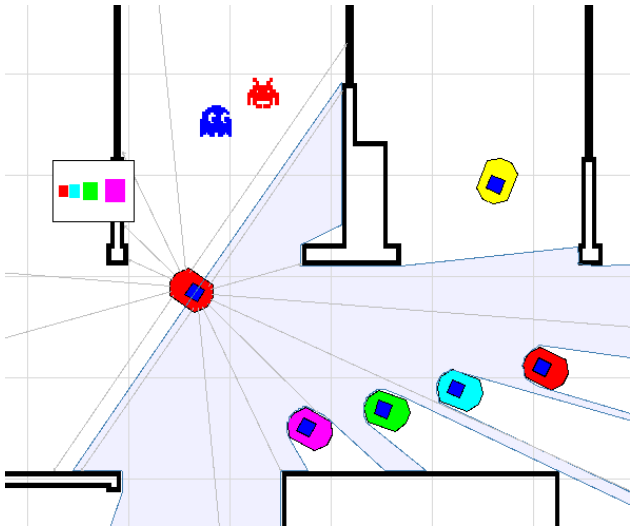


Figura 8.9: Recreación virtual de una situación, según Player.

comportamiento de robots de un modo muy realista. Stage simula una población de robots móviles, sensores, y objetos, en un entorno bidimensional, y pone a disposición del investigador la capacidad de simular gran cantidad de dispositivos reales.

Entre los dispositivos que es posible manejar/simular con Player/Stage encontramos aquellos indicados para el desarrollo del proyecto:

- Robots de la familia Pioneer, entre ellos existen plataformas diferenciales. El controlador *p2os* permite controlar el tipo de robots de que se dispone en el laboratorio (Pioneer P3-DX) [23].

- Medidores de rango láser SICK. Son los medidores de rango laser más difundidos. Existen diversos controladores según el modelo concreto, como el *sicklms200* para el modelo SICK LMS-200 del que se dispone en el laboratorio, o el *sickpls*.

Existen dos controladores específicos cuya finalidad es grabar y reproducir archivos de registro (denominados *logs*), es decir, con ellos es posible guardar información sobre lecturas de sensores y odometría, y recuperarla posteriormente. Esto es interesante para la prueba de algoritmos fuera de linea. En nuestro caso nos interesa fundamentalmente la capacidad de guardar esta información. Además Player incluye en la distribución estándar una aplicación, *playervcr*, destinada a facilitar las labores de registro. Para poder emplearla hay que declarar un controlador *writelog* y/o *readlog*, en el archivo de configuración, lo cual se hace como se muestra en la figura 8.10

Los archivos de registro Player tienen un formato de texto sencillo. Cada mensaje ocupa una linea. Las líneas que comienzan por # son ignoradas. Cada nuevo mensaje comienza con una serie de metadatos comunes:

```
# Log data from laser:0 position2d:0 to "mydata.log"
driver
(
    name "writelog"
    filename "/home/julio/mydata.log"
    requires ["laser:0" "position2d:0"]
    provides ["log:0"]
    alwayson 1
    autorecord 1
)
```

Figura 8.10: Declaración de un controlador writelog, para la creación de un archivo de registro Player/Stage

time host robot interface index type subtype

Los campos son:

- time(double): Marca temporal en segundos.
- host(uint): La parte del host de la dirección del dispositivo Player.
- robot(uint): La parte del robot de la dirección del dispositivo Player.
- interface(string): La parte de la interface de la dirección del dispositivo Player.
- index(string): La parte del índice de la dirección del dispositivo Player.
- type(uint): El tipo de mensaje.
- subtype(uint): El subtipo de mensaje.

Siguiendo a esta meta-información viene la verdadera información del mensaje, siguiendo un formato específico para cada tipo y subtipo de mensaje.

Archivos de configuración. Player necesita saber qué controlador instanciar y a qué direcciones debe ser asociado cada uno. Esta información se especifica habitualmente en un archivo de configuración Player. En la figura 8.11 se muestra la sintaxis básica de declaración de controladores.

Las opciones independientes del tipo de dispositivo son:

- name. El nombre del controlador a instanciar.
- plugin. El nombre de la librería compartida (plugin) que contiene el controlador. Esta opción es obligatoria.

```
driver
(
    opcion_1 valor # comentario_1
    opcion_2 valor # comentario_2
    ...
    opcion_n valor # comentario_n
)
```

Figura 8.11: Sintáxis básica para la declaración de un controlador

- provides. La(s) dirección(es) para acceder al controlador. Esta opción es obligatoria.
- requires. La dirección del dispositivo al que el controlador está suscrito.
- alwayson. Si vale 1 indica que el controlador será configurado en cuanto arranque el servidor Player, sin esperar a que algún cliente se conecte.

8.4. Fuentes de datos

Debido a que el software desarrollado realiza procesamiento fuera de línea (*off-line*), no en tiempo real, de datos recopilados previamente mdiante sistemas robóticos reales, o fruto de la simulación, las distintas fuentes de datos empleadas son archivos de registro (*log* es el anglicismo empleado habitualmente), generados con distintas herramientas, como Player/Stage o Carmen [64]. Es necesario destacar que aún no existe un consenso con respecto al formato que deben tener los archivos de registro, y puede decirse que cada grupo de investigación emplea el suyo propio.

Con el objeto de probar y evaluar el algoritmo en el mayor rango de situaciones posible, nos hemos planteado las siguientes fuentes de datos:

- Registro de simulaciones en entornos sintéticos, realizadas con Player/Stage.
- Registro de recorridos reales realizados en el laboratorio del IUSIANI, con la ayuda de Player/Stage.
- Registro de recorridos reales realizados por otros investigadores, que se encuentran disponibles en Internet.

En cuanto a los registros de recorridos reales realizados por otros investigadores, hemos acudido a un repositorio de datos para SLAM denominado *radish* (`http://radhish.sourceforge.net`), en el que se encuentran infinidad de archivos de registro, junto con información de cómo fueron generados, así como los mapas resultantes de los recorridos.

En el apéndice A puede encontrarse información detallada sobre los formatos de los archivos de registro, mientras que en la sección 9.2.2 se expone el diseño elaborado para su procesamiento.

8.5. Resultados

Probar el algoritmo consiste en verificar el funcionamiento en diferentes circunstancias. Se debe experimentar en diversos entornos y comprobar si los mapas obtenidos podrían ser empleados para tareas tales como la navegación o la planificación de caminos. Se trata pues de una calificación cualitativa, por lo que la mejor forma de presentar los resultados es mediante el mapa de rejilla obtenido que, comparado con la realidad, permitirá calificar la ejecución. Como ejemplo se muestra en la figura 8.12 un mapa sintético, y un mapa de rejilla obtenido con nuestra aplicación.

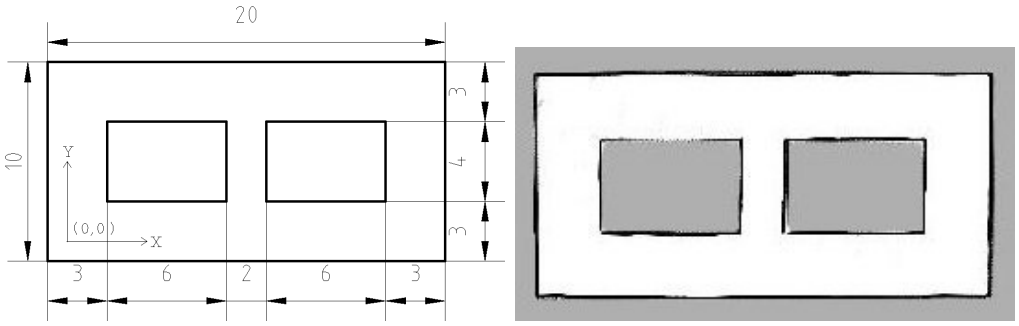


Figura 8.12: Mapa sintético, a la izquierda, y mapa de rejilla obtenido con 100 partículas, a la derecha, de un recinto con dos bucles. Unidades en metros.

Evaluar el algoritmo significa profundizar más en el conocimiento, tratando de encontrar sus fortalezas y sus debilidades. Para esto se debe experimentar en diversos tipos de entornos, analizando el comportamiento de los distintos parámetros del algoritmo. Uno de

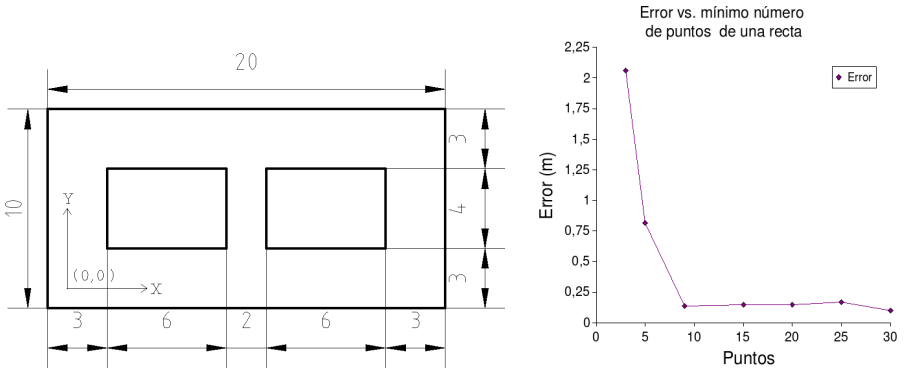


Figura 8.13: Mapa sintético, a la izquierda, y evolución del error frente a la longitud mínima de una recta válida, a la derecha.

los resultados más importantes de la evaluación debe ser una configuración media adecuada, que permita un ajuste detallado sin partir desde cero, además de unas directrices de configuración, en función de la tipología del entorno. Para obtener esta información una buena

herramienta son las gráficas que muestren el error del mapa obtenido o el número de hitos detectados, en función de un determinado parámetro del algoritmo. Como ejemplo se muestra en la figura 8.13 un mapa y la evolución para ese mapa del error frente al número mínimo de puntos de una recta.

Una de las pretensiones del presente proyecto es arrojar un poco de luz sobre la implementación del algoritmo, enfatizando ciertos detalles que habitualmente no tienen cabida en los artículos publicados. En este sentido, una justificación experimental de los distintos valores de configuración del algoritmo puede ser de mucha ayuda de cara a futuras investigaciones, proporcionando un punto de partida y una base sólida para la interpretación de resultados.

Capítulo 9

Diseño y desarrollo

9.1. Introducción

Como dicta la ingeniería del software, tras la fase de análisis en la que se determina qué *problema* se quiere solucionar y en qué condiciones, se realiza la fase de diseño, en la que se determina cómo se va a llevar a cabo dicha solución [62]. Tras la fase de diseño se pasa a la fase de desarrollo en la que realmente se construye la solución. La fase de diseño es crucial para conseguir un software de calidad. Un buen diseño debe basarse en la búsqueda de un software reutilizable, flexible, de fácil mantenimiento, fácil de comprender, etc. Esta es una tarea compleja que debe hacer uso de toda la experiencia acumulada en esta rama de la ingeniería, fundamentalmente mediante el uso de patrones de diseño [5]. Además, como ingenieros de software, debemos facilitar la comunicación mediante el uso de herramientas estandarizadas, como el lenguaje UML [19].

A lo largo de este capítulo vamos a estudiar el diseño de la aplicación que constituye la parte fundamental del entorno para la prueba y evaluación del algoritmo FastSLAM, así como el modo en que se ha llevado a cabo la planificación y el desarrollo de todo el proyecto.

En primer lugar se desarrolla en la sección 9.2.1 la arquitectura de la aplicación. Seguidamente, en las secciones 9.2.2, 9.2.3, y 9.2.4 se analizan las principales líneas de diseño de las distintas capas que integran el núcleo de la aplicación, pasando en la sección 9.2.5 a analizar el diseño de la capa de servicios. Para terminar con el diseño, se analiza en las secciones 9.2.6, y 9.2.7 el diseño de la interfaz gráfica de usuario y la interfaz de línea de comandos de la aplicación, respectivamente. Por último se expone en la sección 9.3 la planificación y el desarrollo del proyecto.

9.2. Diseño

9.2.1. Diseño arquitectónico

La arquitectura de la aplicación se divide en un núcleo de ejecución del propio algoritmo FastSLAM, organizado según una arquitectura en capas, una capa de servicios que recubre

todo el núcleo y que encapsula detalles de la ejecución, configuración y visualización del proceso de SLAM, y un subsistema de interfaz de usuario (user interface o UI) que contiene aquellas clases que permiten construir tanto interfaces gráficas como de línea de comandos.

Para el núcleo del sistema, el encargado de ejecutar el algoritmo FastSLAM, se emplea un modelo de arquitectura denominado *Arquitectura en capas* [63]. El paradigma de este modelo lo encontramos el el modelo *OSI* de redes de computadores [62]. Tal como se muestra en la figura 9.1 las capas se encuentran apiladas, de manera que cada capa se ocupa de una parte de la funcionalidad del sistema, y se comunica únicamente con las capas situadas inmediatamente encima y debajo de ella. Además, de cara a la implementación de una aplicación que proporcione sencillez a la hora de configurar los distintos parámetros del algoritmo, así como potentes visualizaciones gráficas, se introduce una capa que recubre todo el núcleo y que proporciona servicios de configuración, ejecución (coordinación de las distintas capas) y visualización. De este modo será posible, mediante la comunicación con la capa de servicios, desarrollar rápidamente aplicaciones con diversas interfaces gráficas (GUI, graphical user interface) o interfaces de línea de comandos.

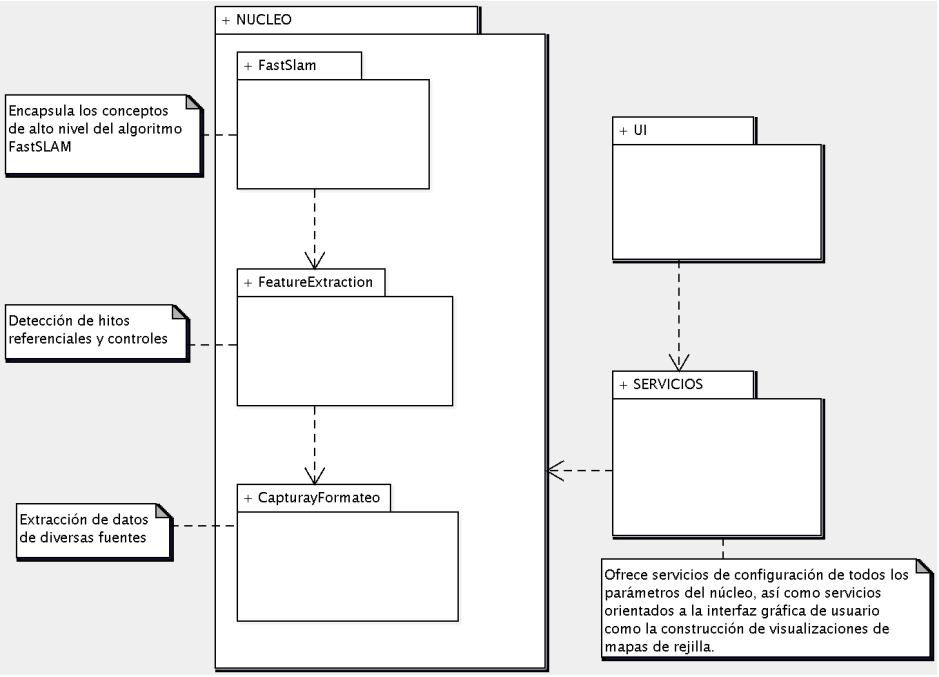


Figura 9.1: Arquitectura de la aplicación

La arquitectura del núcleo consta de tres capas, de modo que la capa inferior se encuentra próxima al hardware y la obtención de datos, mientras que la superior abstrae la ejecución propia del algoritmo FastSLAM 1.0:

- Capa de *Captura y formateo de datos*: Esta capa se encarga de suministrar lecturas de los distintos sensores a las capas superiores. Su misión es independizar al resto del sistema del tipo de fuente de datos.

- Capa de *Extracción de características*: A partir de los datos proporcionados por la capa de captura y formateo de datos, esta capa detecta hitos referenciales o características (esquinas y rectas) en los datos sensoriales. Además, extrae los controles ejecutados por el robot, a partir de los datos de odometría.

- Capa *FastSLAM*: Esta capa aplica el algoritmo FastSLAM 1.0, partiendo de las características detectadas por la capa de Extracción.

La tipología del problema, en el que intervienen desde fuentes hardware de datos, hasta conceptos probabilísticos de muy alto nivel, aconsejan la utilización de la arquitectura en capas. De este modo es posible aislar los conceptos matemáticos, de las fuentes de datos, y éstas de las distintas características que pueden contener dichos datos. Un adecuado diseño de las interfaces entre las distintas capas posibilitará la aplicación del software a distintas configuraciones robóticas y tipos de entornos.

Por su parte, la capa de servicios va a permitir configurar completamente el algoritmo, proporcionando servicios de carga y almacenamiento de configuraciones en archivos, modificación *manual* de parámetros, y ejecución tanto paso a paso, como en lotes. Además ofrece servicios de construcción de mapas de rejilla, muy importantes de cara a la visualización y comprensión del proceso de SLAM.

En la capa UI, la más próxima al usuario, se encuentran todas aquellas clases necesarias para el desarrollo de interfaces de usuario, tanto gráficas, como de línea de comandos.

9.2.2. Captura de datos

El subsistema de captura de datos constituye la capa inferior de la arquitectura. Este subsistema es el más próximo al hardware que genera la información que alimenta todo el sistema. Gracias a esta capa es posible independizar al resto del sistema de las particularidades de las fuentes de datos. Este subsistema es capaz de manejar diversas fuentes de datos (archivos con un formato específico, dispositivos de red, etc, ...), entregando la información a la capa inmediatamente superior, la capa de extracción de características, en un formato estandarizado. Esta información es de dos tipos:

- *Barrido láser*. Se trata de la medida proporcionada por un sensor de rango láser. Un barrido láser puede expresarse en coordenadas polares o cartesianas. En nuestra implementación, un barrido es una lista de puntos en coordenadas cartesianas con respecto al sensor. Además es necesario mantener una marca temporal que permita conocer en qué momento se realizó la lectura del barrido. Esta información se modela, tal como puede apreciarse en la figura 9.2, mediante la clase *Scan*.

- *Odometría*. Se trata de información sobre el estado interno de la plataforma robótica. La información odométrica puede representarse de diversas maneras, como por ejemplo, mediante el ángulo girado por cada una de las ruedas o las velocidades de traslación y rotación. En nuestro caso, representamos la información odométrica como la velocidad de traslación y la velocidad de rotación de la plataforma robótica móvil, además de

una marca de tiempo que señala el momento en que se ha realizado la medida (Ver Figura 9.2). Esta información se modela mediante la clase *Odometria*.

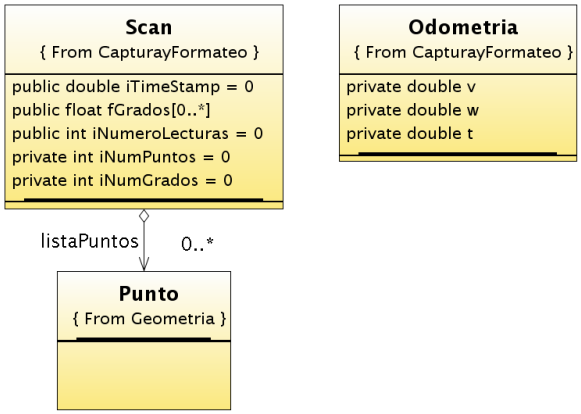


Figura 9.2: Clases *Scan* y *Odometria*.

Para que el subsistema sea capaz de manejar distintos generadores de datos, y por lo tanto sea flexible, se define la interfaz *fuente* que define los servicios que debe prestar una fuente de datos para poder ser incorporada al sistema. Los principales servicios que debe proporcionar una fuente son la capacidad de proporcionar información sobre barridos láser y odometría, en forma de líneas de texto plano.

Como no es posible que cada fuente devuelva las líneas de barrido láser y odometría con el mismo formato, se define un formato de línea de barrido láser y de línea de odometría estandarizados. De este modo existirá una clase abstracta *Conversor*, cuyas particularizaciones se encargarán de convertir diversos formatos de entrada al formato de línea intermedio estándar. Por otro lado existe una clase Procesador que toma lecturas en formato estandarizado y devuelve objetos de las clases *Scan* y *Odometria*. Para flexibilizar todo el diseño se aplica el patrón *Factory* [5] a los conversores (Ver Figura 9.3), de modo que el Procesador incorpora un Conversor particularizado según sus necesidades. El patrón Factory permite que la clase Procesador no tenga que conocer los tipos concretos de Conversores que existen, con lo que se consigue un diseño menos acoplado. Este diseño permite incorporar nuevas fuentes de datos de un modo simple y flexible.

En principio, tal como se observa en la figura 9.4, las diversas fuentes de datos y sus correspondientes conversores, se corresponden con archivos de texto que contienen la información en diferentes formatos (ver apéndice A). Para definir una forma homogénea de tratamiento de archivos de registro de datos que vayan a ser fuentes, se define la clase abstracta *Archivo*. Esta clase especifica un mecanismo de lectura independiente de líneas de odometría y líneas de barrido láser. Los archivos de registro contienen entremezcladas líneas de odometría y líneas de barrido láser. Entre una línea con un tipo de información, y la siguiente con el mismo tipo de información, pueden aparecer varias líneas del otro tipo. Un efecto indeseable

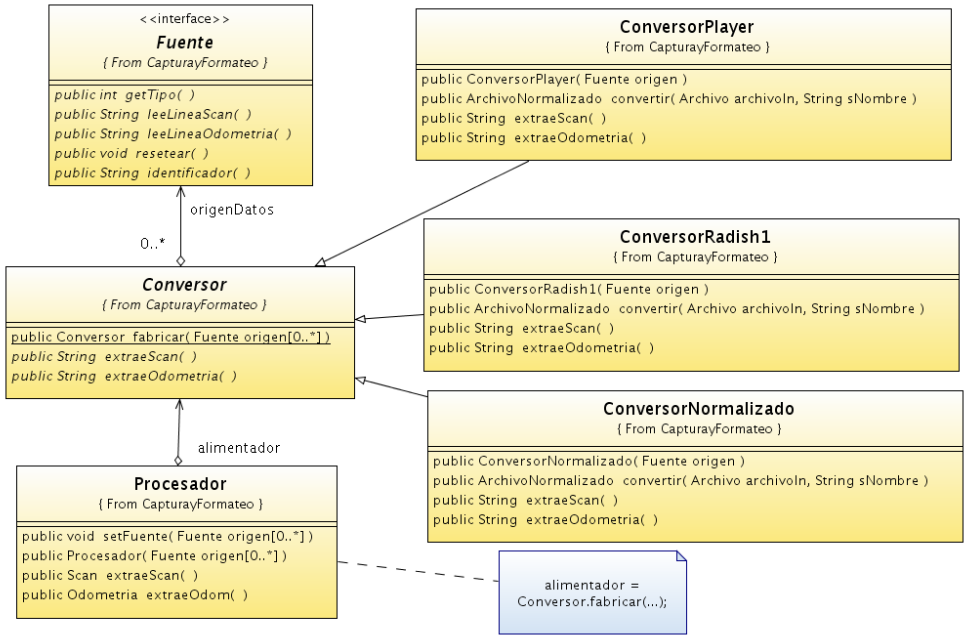


Figura 9.3: Implementación del patrón Factory

sería que al avanzar hasta la siguiente línea de un determinado tipo de información, se descartaran las líneas intermedias del otro tipo. El mecanismo de acceso independiente definido en la clase Archivo, permite que al solicitar una línea de información de un tipo, se nos proporcione la siguiente, independientemente de hasta dónde se haya avanzado en la lectura de líneas del otro tipo.

Un uso típico de esta capa, mostrado con el diagrama de interacción de la figura 9.5 incluye las siguientes etapas:

1. Creación de un Procesador

2. Establecimiento de la Fuente, ya sea vía el constructor, o con una llamada a setFuente(Fuente origen).

3. Llamadas sucesivas a extraeScan(), extraeOdometria()

Internamente el procesador utiliza el patrón *Factory* para obtener un Conversor según el tipo del objeto Fuente con el que ha sido inicializado. Una llamada a uno de los métodos "*extrae*" del Procesador provoca una llamada a uno de los métodos "*extrae*" del conversor específico, que llamará a leeLineaScan() o leeLineaOdometria() de la fuente, y tratará convenientemente la cadena obtenida, para devolver una cadena normalizada al Procesador, que creará el Scan o la Odometria correspondientes.

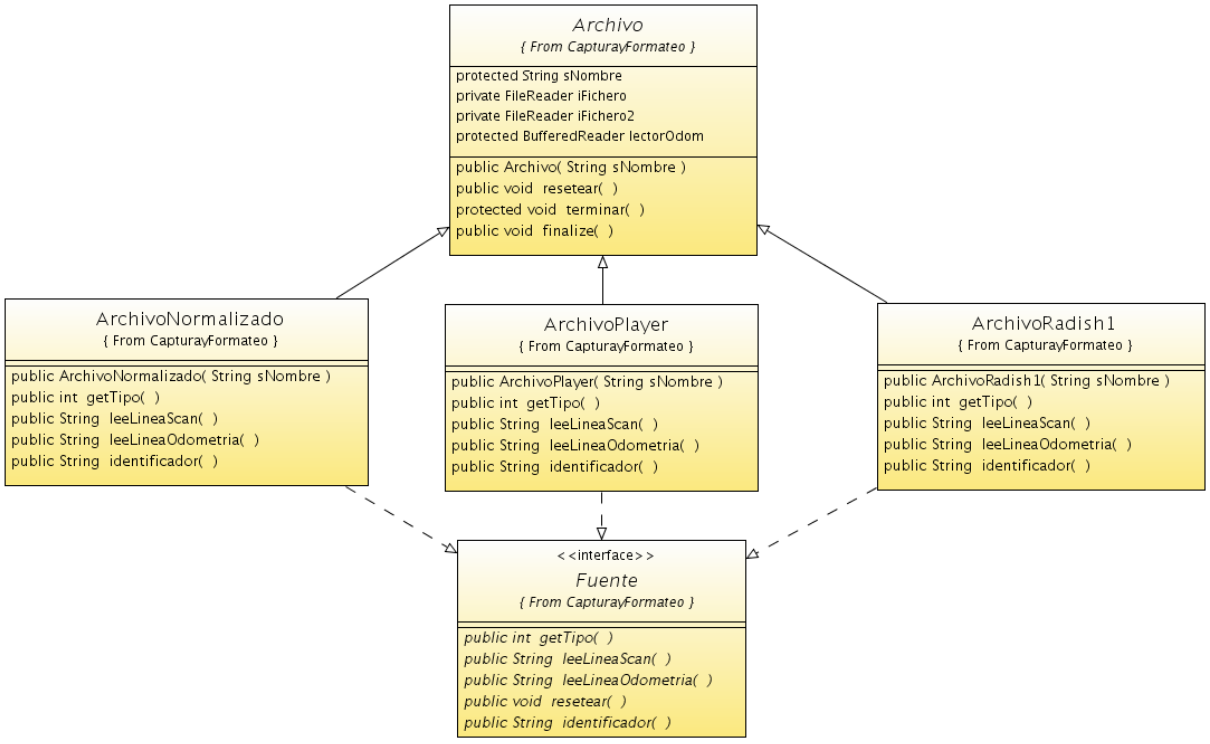


Figura 9.4: Estructura de las fuentes de información.

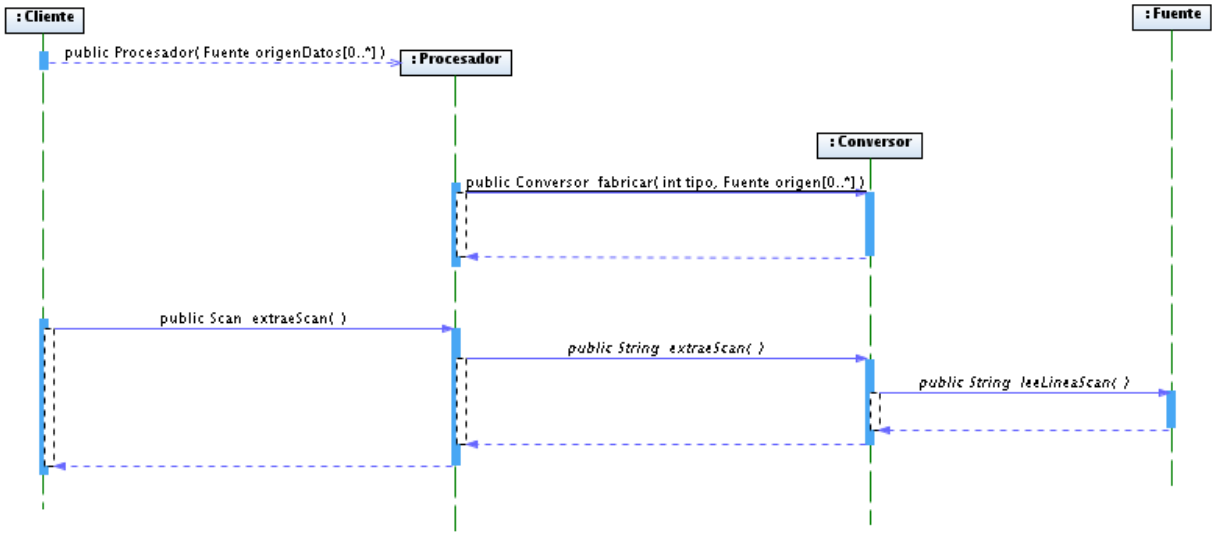


Figura 9.5: Extracción de un Scan Láser.

9.2.3. Extracción de características

El subsistema de extracción de características forma la capa intermedia de la arquitectura del sistema. Esta capa emplea la información de barridos láser y odometría, proporcionada por la capa de *captura y formateo de datos*, para obtener los hitos referenciales presentes en el entorno (rectas, esquinas, etc,...) y los controles ejecutados por la plataforma robótica. Este subsistema aisla la lógica del algoritmo de los detalles de los diversos métodos de extracción de características y cálculo de controles. Así este subsistema consume objetos de las clases *Scan* y *Odometria* (ver figura 9.2), los procesa, y entrega dos tipos de información a la capa superior, la capa de *FastSLAM*:

- *Observaciones*. Conjuntos de hitos referenciales extraídos de un barrido láser. En nuestro caso estos hitos son rectas y/o esquinas.
- *Controles*. *Órdenes* ejecutadas por el robot durante un cierto periodo de tiempo. En nuestro caso se trata de la velocidad lineal y angular mantenida durante el periodo.

Una observación es un conjunto de hitos referenciales o características extraídas de un barrido láser. Una observación se modela como un *vector de hitos referenciales*. Dado que la capa *FastSLAM* puede manejar todo tipo de características, y para independizarla de los detalles de éstas, además de para facilitar la incorporación de nuevos tipos, se define una interfaz común, denominada *ICarateristica* (Ver Figura 9.6). Los servicios básicos que debe

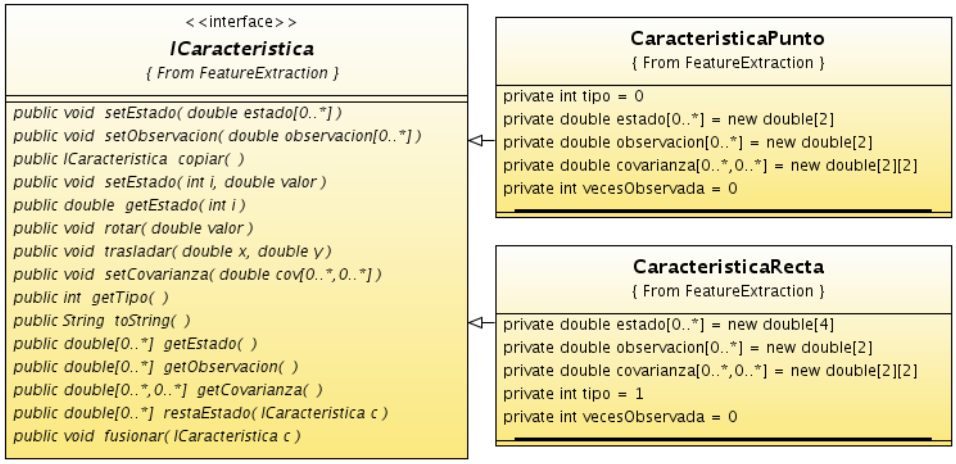


Figura 9.6: Jerarquía de clases para representar características.

proporcionar una característica son su traslación, su rotación, y una operación de resta. De este modo tanto el filtro de partículas, como los filtros de Kalman de la capa *FastSLAM* no tendrán que tener en cuenta que la característica detectada sea una esquina o una línea recta. Encontramos el concepto de características en dos lugares. Por una lado como información

resultante del subsistema de extracción de características, y, por otro, formando el mapa que mantiene cada partícula en el subsistema FastSLAM.

Un control representa las *órdenes* ejecutadas por un robot durante un cierto periodo de tiempo. En concreto un control contiene información sobre la velocidad de traslación, la velocidad de rotación, y el tiempo durante el que se ha estado ejecutando la orden.

Otro concepto fundamental dentro de este subsistema es el *extractor de características*. Dado que es posible extraer distintos tipos de hitos del entorno, se diseña una jerarquía de extractores, cada uno especializado en un tipo o tipos de hitos. La clase abstracta *Extractor* es la que se va a manejar directamente en la aplicación. Esta clase se concreta en principio en un extractor de rectas (ExRectas) y un extractor de esquinas o puntos (ExPuntos). La clase ExRectas puede implementar distintos métodos de extracción por lo que se define una jerarquía de métodos encapsulados en clases, es decir se aplica el patrón *Strategy* [5], de modo que ExRectas contendrá un atributo de tipo MetodoExtraccionRectas, clase abstracta de la que derivan los distintos métodos. ExPuntos extrae las esquinas mediante la intersección de rectas, por lo que también necesitará de un MetodoExtraccionRectas. La relación entre todas las clases descritas se muestra en la figura 9.7.

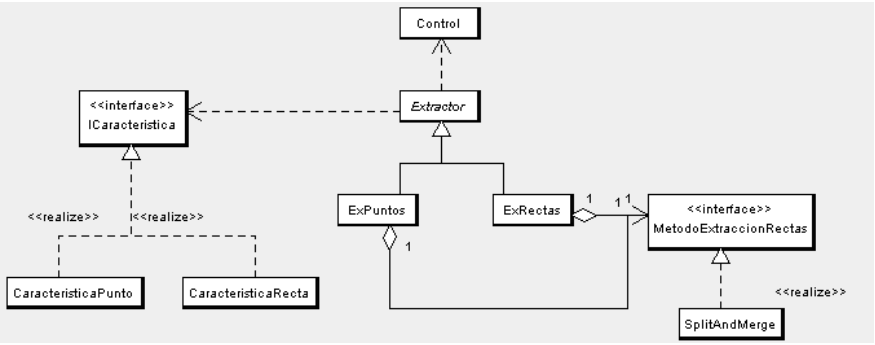


Figura 9.7: Diseño del subsistema Feature Extraction, sin detalles de las clases.

En la figura 9.8 se muestra la relación existente entre las clases más relevantes del subsistema de extracción de características, y las clases que modelan la información de entrada que necesita este subsistema, integrantes del subsistema de captura y formateo de datos. Un Extractor va a tomar información de la capa de *Captura y Formateo* en forma de objetos de las clases Scan y Odometria, y va a producir controles y observaciones, en forma de objetos de las clases Control e ICaracterística.

Un ejemplo de uso de la capa de extracción de características se muestra en la figura 9.9. Se supone que en esta situación la clase Cliente tiene acceso a un objeto Procesador representante de la capa de *Captura y Formateo* de datos. Así en primer lugar se crea un extractor de rectas, seguidamente se solicita un Scan al objeto Procesador, y por último se envía el Scan obtenido al extractor de rectas, para obtener un vector de ICaracterísticas que contenga las rectas detectadas.

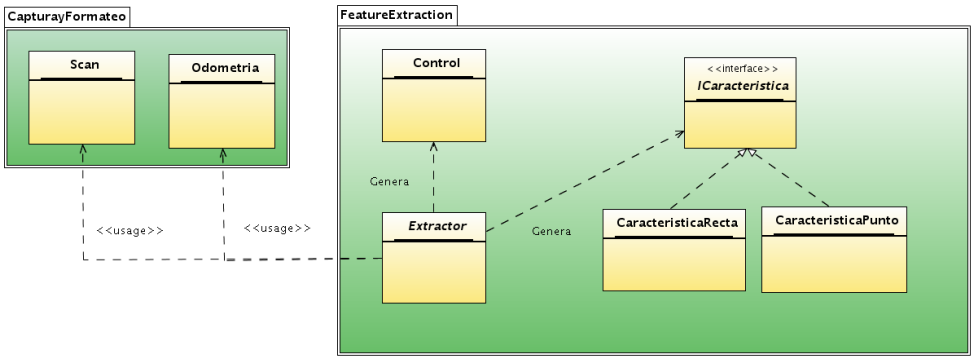


Figura 9.8: Subsistemas Captura y formateo de datos, y Extracción de características.

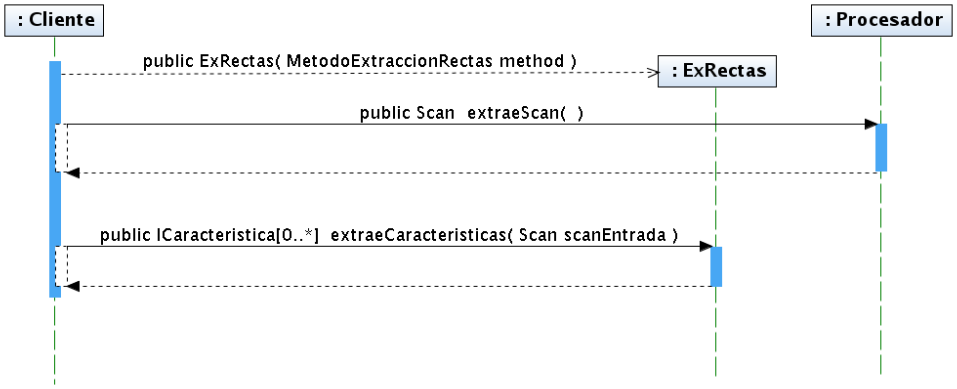


Figura 9.9: Diagrama de interacción, ejemplo de uso de la capa de extracción de características

9.2.4. FastSLAM

El subsistema FastSLAM constituye la capa superior del sistema. Se trata del subsistema más alejado de las fuentes de datos y más próximo a los conceptos de la construcción automática de mapas. Como información de entrada, el subsistema FastSLAM necesita observaciones, es decir, vectores de objetos de la clase ICaracteristica (ver figura 9.6), y controles, es decir, objetos de la clase Control. A partir de esta información, el subsistema computa el algoritmo FastSLAM 1.0, y como resultado de cada iteración obtenemos un conjunto de partículas. Cada partícula contiene un peso, un mapa, que no es más que un vector de objetos de la clase ICaracteristica, y un registro histórico de poses, necesario para construir un mapa de rejilla. Lo habitual es considerar la partícula con la mayor puntuación como aquella que refleja la configuración de la realidad. Además encontramos en este subsistema los modelos de observación y movimiento, correspondientes a distintos sensores y plataformas robóticas respectivamente, y los filtros de Kalman extendidos.

A continuación se muestra una lista con los conceptos que se integran en esta capa. Cada uno de los conceptos será desarrollado en profundidad más adelante:

- *Observaciones*. Forman parte de la información de entrada que necesita el subsistema. Son vectores de objetos de la clase ICaracteristica, provenientes de la capa de *Extracción de Características*.
- *Controles*. Forman parte de la información de entrada que necesita el subsistema. Son objetos de la clase Control, provenientes de la capa de *Extracción de Características*.
- *Partícula*. Una partícula es una hipótesis sobre la pose de un robot y su mapa.
- *Filtro de partículas*. Se trata de una herramienta probabilística para representar procesos estocásticos.
- *Pose*. La posición en coordenadas cartesianas, y el ángulo de orientación del robot.
- *Mapa de hitos*. Cada partícula mantiene un vector con los hitos referenciales que ha detectado en el entorno, almacenados en coordenadas del mundo.
- *Modelo de observación*. Modelo que expresa qué observación produce el sensor a partir de un punto real del mundo.
- *Modelo de movimiento*. Modelo que expresa cómo cambia la pose del robot cuando se ejecutan unos controles.
- *Filtro de Kalman Extendid*o o *EKF*. Herramienta probabilística para la estimación de procesos estocásticos.
- *Remuestreo*. Proceso por el cual se selecciona un subconjunto del conjunto de partículas de un filtro, generalmente aquellas que tengas un mayor peso, con el objeto de evitar la degeneración del filtro. En nuestro caso algunas de las partículas seleccionadas se *clonan*, para mantener constante el número de partículas del filtro.

Para modelar el filtro de partículas se diseña la clase *FiltroParticulas*. Esta clase se concibe como un gestor del algoritmo, de modo que proporciona métodos para incorporar observaciones y controles, o para realizar la asociación de datos. En la figura 9.10 se muestran las relaciones básicas que modelan el filtro de partículas. El filtro debe tener en todo momento acceso a la última observación y los últimos controles ejecutados, para poder gestionar adecuadamente la ejecución del algoritmo. Además el filtro tiene un vector de partículas y acceso directo a la mejor partícula del filtro. Cada partícula contiene, entre otras cosas, un mapa, la última pose s, y un histórico de poses.

El objeto del registro histórico de poses es la obtención de un mapa de rejilla, por ello en él no aparecen todas las poses calculadas para una partícula. Tal como se observa en la figura 9.11, en ocasiones se toman lecturas de controles sin que se haya tomado una lectura de barrido láser, como por ejemplo cuando en el instante T2. Las poses resultantes de estos controles *huérfanos* no se anotan en el histórico de poses, ya que no aportan información para

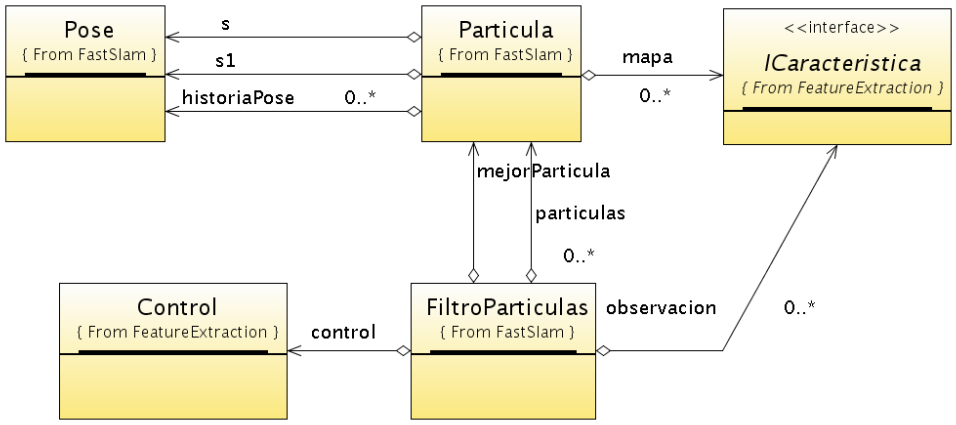


Figura 9.10: Clases básicas relacionadas con el filtro de partículas.

la obtención del mapa de rejilla. Dado que los tiempos en los que se registra la información odométrica y los barridos láser casi nunca coinciden, es necesario interpolar la información odométrica para calcular la pose que se corresponde con un determinado barrido, como ocurre con la odometría registrada en el instante T3, que se interpola al instante anterior T3', para calcular la pose desde la que se ha realizado la lectura del corresponediente barrido láser.

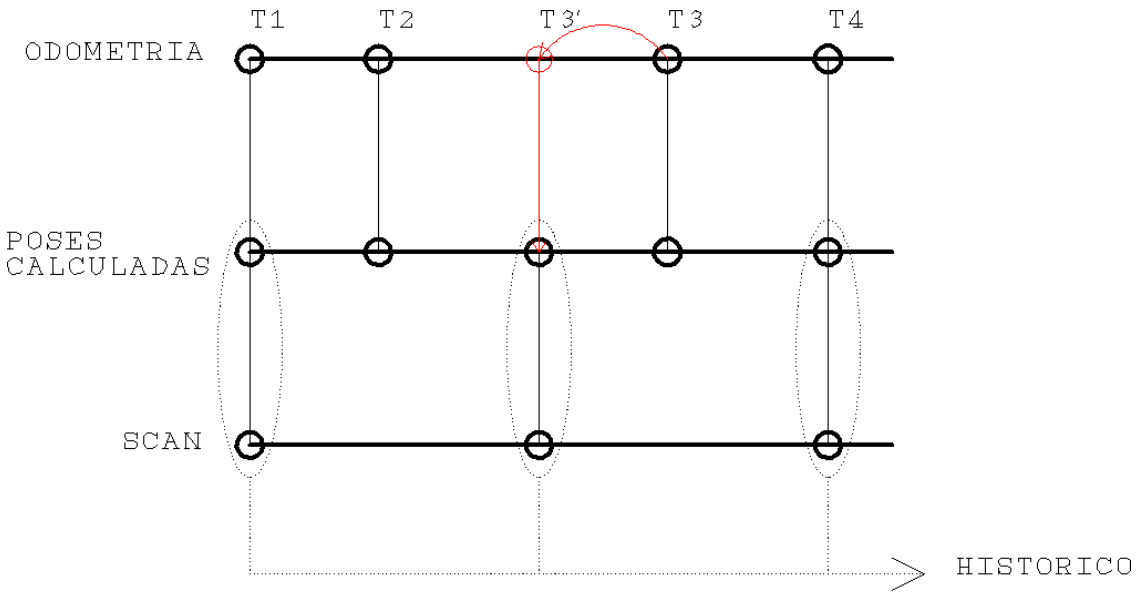


Figura 9.11: Detalle de la incorporación de datos al registro histórico de poses de cada partícula.

Para poder realizar adecuadamente la gestión del algoritmo, el filtro de partículas necesita tener acceso a los modelos de movimiento y observación, además de a filtros de Kalman Extendidos, y a un remuestreador.

Un modelo de movimiento se corresponde con un tipo de plataforma robótica. Con el objeto de poder incorporar en el futuro diversas plataformas robóticas, se ha diseñado una jerarquía de modelos de movimiento, que puede observarse en la figura 9.12. La cima de la jerarquía es la clase abstracta *Robot* que define que un modelo de observación debe proporcionar una función de transición de estado con y sin ruido añadido, y el jacobiano de dicha función. Aunque únicamente se implementa el modelo de movimiento de una plataforma diferencial, esta jerarquía flexibiliza desarrollos futuros.

Un modelo de observación se corresponde con un tipo de sensor y está íntimamente relacionado con los hitos referenciales que es posible obtener mediante sus datos. Con el objeto de poder incorporar en el futuro diversos tipos de sensores se ha diseñado una jerarquía de modelos de observación, que puede observarse en la figura 9.12. La cima de la jerarquía es la clase abstracta *Sensor* que define que un modelo de observación debe proporcionar una función de observación con y sin ruido añadido, y el jacobiano de dicha función. Además debe proporcionar la función inversa de la de observación (ver figura 7.27), que permite trasladar las coordenadas de una observación desde un sistema centrado en el robot, hasta un sistema global, y de este modo incorporar un nuevo hito al mapa. Hay que tener en cuenta que el modelo de observación varía en función del tipo de la característica que se esté observando, de modo que las clases que hereden de Sensor deben poseer mecanismos internos para tratar específicamente cada tipo de hito. Por ello la interfaz ICaracterística proporciona el método getTipo() que devuelve un entero identificando el tipo de característica concreto (ver Figura 9.6).

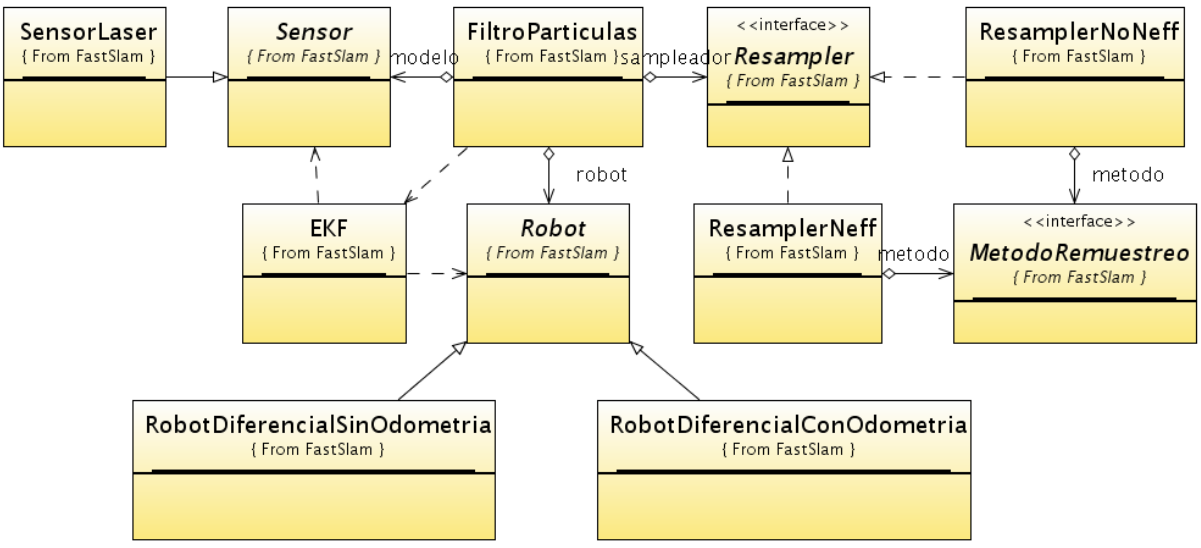


Figura 9.12: Jerarquías de los modelos de observación y movimiento.

El filtro de Kalman Extendido que se ha diseñado no es un EKF genérico, sino uno diseñado específicamente para estimar el estado de objetos de tipo ICaracterística. La aplicación de un filtro de Kalman Extendido en estas condiciones requiere información sobre un modelo de movimiento, un modelo de observación, una pose, una ICaracteristica observada, y la última estimación de la ICaracterística observada. El filtrado modifica la propia ICaracteristica pasada como última estimación. La relación de la clase *EKF* con todos los conceptos comentados se muestra en la figura 9.13.

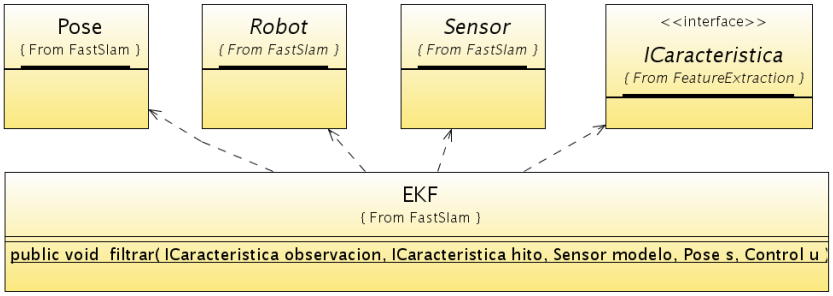


Figura 9.13: Filtro de Kalman Extendido, y sus relaciones.

Por último, y con el objeto de posibilitar la incorporación de diversos tipos de remuestreadores se ha creado una jerarquía de éstos, que puede observarse en la figura 9.12. La clase raíz de la jerarquía es la clase *Resampler*. Para flexibilizar aún más el diseño se implementa el método concreto de remuestreo mediante un patrón *Strategy* de modo que los distintos remuestreadores puedan especificar tanto las condiciones para realizar el remuestreo como el método para llevarlo a cabo. Como ejemplo hemos incluido en el diseño los remuestreadores *ResamplerNeff* que depende del número efectivo de partículas o N_{eff} (más información en la sección 6.3) como umbral para el remuestreo, y el *ResamplerNoNeff* que remuestrea siempre, independientemente del N_{eff}. Ambos remuestreadores emplean el algoritmo de remuestreo secuencial (ver figura 9.12).

9.2.5. Subsistema de servicios

La capa de servicios no es una capa fundamental para la ejecución del algoritmo, es decir, sin ella se podría también llevar a cabo. Esta capa se sitúa sobre el núcleo de la aplicación, proporcionando un nivel superior de abstracción. Su uso acelera enormemente el desarrollo, ya que encapsula detalles de coordinación entre las distintas clases. Los servicios básicos que ofrece este subsistema son servicios de configuración, de ejecución, y de visualización. De este modo será posible, mediante la comunicación con la capa de servicios, desarrollar rápidamente aplicaciones con diversas interfaces gráficas (GUI, graphical user interface) o interfaces de línea de comandos.

Para coordinar el funcionamiento de todas las clases proporcionando servicios de ejecución, se ha diseñado una clase denominada *Algoritmo*. Aunque es posible implementar el

algoritmo FastSLAM directamente gestionando *manualmente* todas las clases implicadas, es preferible encapsular todos los pasos en una única clase que ofrezca el servicio de ejecutar el algoritmo. Tal como se muestra en la figura 9.14, para beneficiarnos de la clase *Algoritmo* debemos crear un objeto de la clase *Procesador* de la capa de *captura y formateo de datos*, un objeto de la clase *Extractor* de la capa de *extracción de características*, y un objeto de la clase *FiltroParticulas* de la capa *FastSLAM*. Con estos objetos se crea el objeto de la clase *Algoritmo* al que se solicita la ejecución de un determinado número de iteraciones, tras las cuales podremos obtener informaciones tales como el mapa de la mejor partícula, o podremos construir el mapa de rejilla de cualquiera de las partículas que integran el filtro.

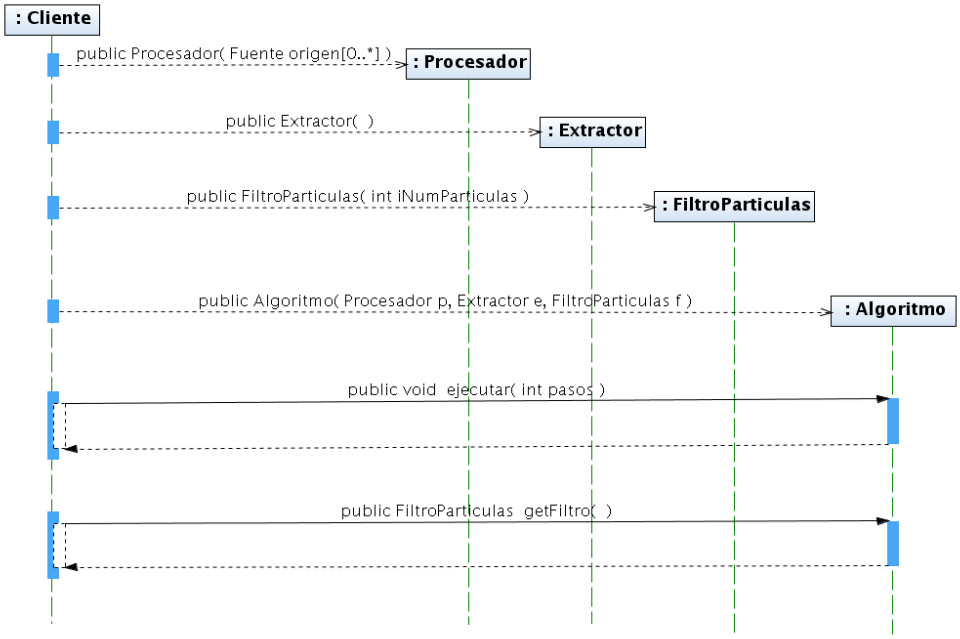


Figura 9.14: Uso de la clase *Algoritmo*.

Para gestionar los servicios de configuración se integran en una única clase, denominada *Configuracion* todos los aspectos parametrizables del algoritmo. Además se diseña una clase que ofrece servicios de lectura y escritura de configuraciones a disco, denominada *ArchivoConfiguracion*. Una configuración no solo encierra los parámetros configurables, sino que también encapsula la ejecución mediante un objeto Algoritmo, tal como se muestra en la figura 9.15

Lo que denominamos servicios de visualización se refiere a la clase *Grid*. Esta clase implementa el concepto del mapa de rejilla que, además de ser una herramienta fundamental para la navegación, resulta muy útil para la visualización del proceso.

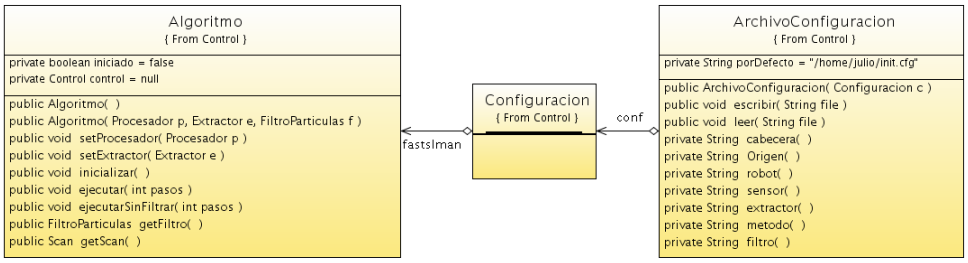


Figura 9.15: Relación entre las clases *Configuracion*, *ArchivoConfiguracion*, y *Algoritmo*

9.2.6. Interfaz gráfica

La interfaz gráfica de usuario, en adelante *GUI* (del inglés *Graphical User Interface*), se diseña en base a los casos de uso identificados en la Sección 8.2. El paradigma empleado en el diseño es el denominado WIMP (del inglés Windows, Icons, Menus, Pointing devices), típico de la gran mayoría de aplicaciones de escritorio. Este paradigma define una serie de usos habituales de los distintos elementos de la interfaz gráfica (botones, menús, etiquetas, etc, ...), que se han tratado de respetar.

En el diseño de interfaces gráficas de usuario es muy útil el empleo de dos patrones de diseño, el patrón *comando* [5], y el patrón *observador-observable* [5]. Mediante el uso de estos patrones se facilita el desarrollo de interfaces flexibles y dinámicas.

En la figura 9.16 se muestra la ventana principal del programa. Esta ventana se encuentra dividida en tres áreas, además de la habitual barra de menús:

- Área de consulta de configuración.
- Área de visualización.
- Panel de control.

9.2.6.1. Área de consulta de configuración

Se traza de la zona superior de la ventana principal. Como se muestra en la figura 9.17 en ella pueden consultarse de modo rápido los distintos valores de los parámetros que conforman la configuración actual de la aplicación. Es fundamental que el investigador tenga acceso rápido a esta información para poder valorar la situación y tomar decisiones. Este área se encuentra dividida en compartimentos *temáticos* que contienen información sobre la configuración de aspectos concretos del algoritmo:

- Origen de los datos.
- Modelo de movimiento.
- Filtro de partículas.

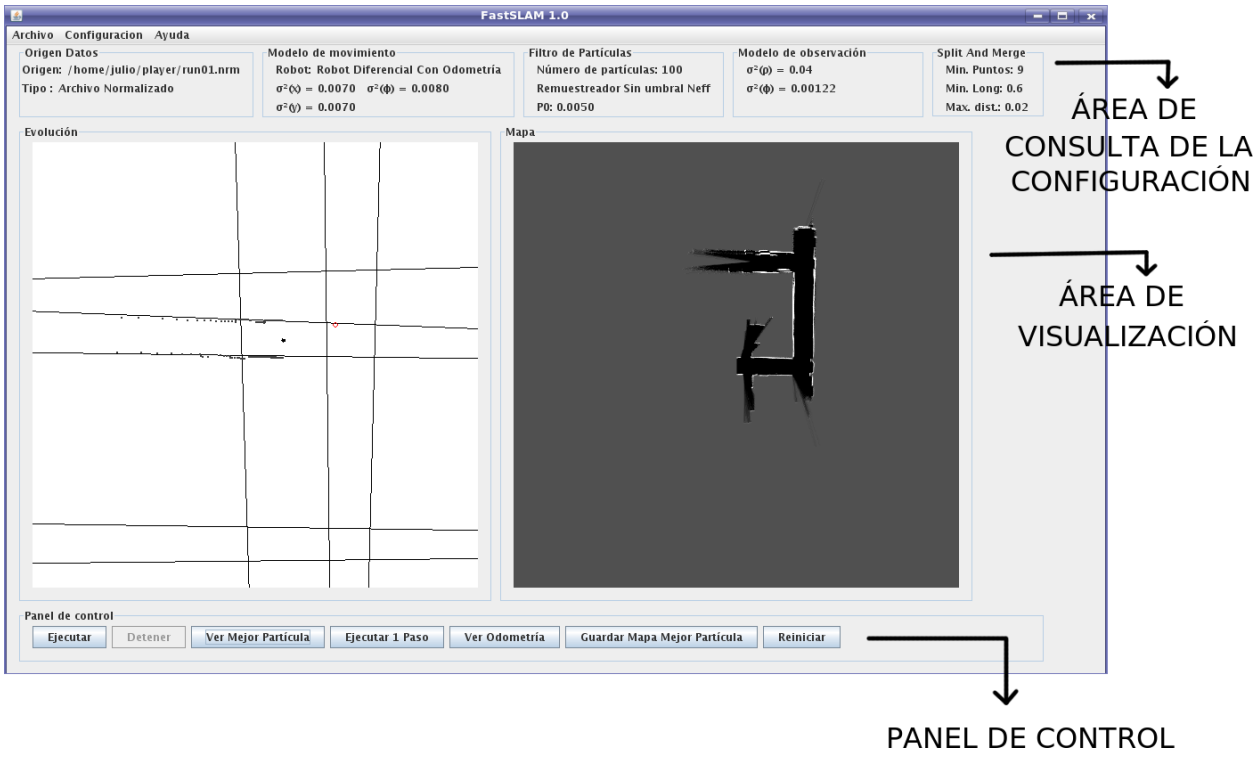


Figura 9.16: Ventana principal del programa, con las distintas zonas destacadas.

- Modelo de observación.
- Algoritmo Split and Merge.

Figura 9.17: Área de consulta de configuración.

9.2.6.2. Área de visualización

Se encuentra en la zona media de la ventana principal, ya que se considera la parte más importante de la interfaz gráfica. Tal como aparece en la figura 9.18, en ella se muestra a la izquierda el último barrido láser, junto con la nube de partículas y los hitos presentes en el mapa de la mejor partícula, y a la derecha el mapa de rejilla formado por la contribución de la mejor partícula de cada iteración. Esta zona de visualización es vital para poder entender las debilidades y fortalezas del algoritmo, ya que permite ver lo que realmente estaría percibiendo el robot, la forma de realizar las asociaciones de datos, la posible divergencia del filtro, etc.

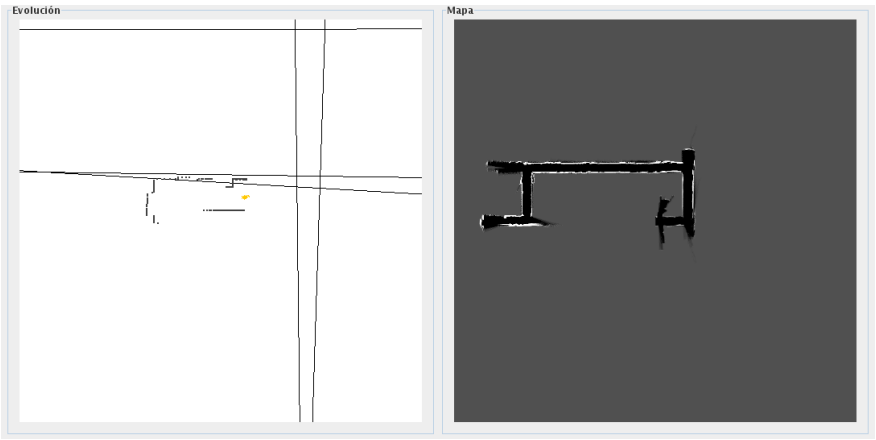


Figura 9.18: Área de visualización.

9.2.6.3. Panel de control

Se encuentra en la zona inferior de la ventana. En la figura 9.19 puede observarse en detalle. Contiene botones que dan acceso a las acciones relacionadas con la ejecución más importantes. Estas son el inicio/pausa de la ejecución, la ejecución de un solo paso, la obtención del mapa basado en odometría, o el mapa de rejilla de la mejor partícula. A continuación

Figura 9.19: Panel de control.

se describe la utilidad de los distintos botones:

- **Ejecutar**. Inicia/reanuda la ejecución del algoritmo. Durante la ejecución el área de visualización evoluciona de acuerdo al algoritmo.
- **Detener**. Produce la pausa de la ejecución.
- **Ver mejor partícula**. Despliega la ventana denominada *Mapa de rejilla de la mejor partícula*, que muestra el mapa de rejilla construido con la información de la que en el momento de la ejecución es la mejor partícula.
- **Ejecutar 1 paso**. Ejecuta una iteración del algoritmo. El área de visualización evoluciona de acuerdo a la iteración ejecutada.
- **Ver Odometría**. Muestra la ventana denominada *Mapa de rejilla basado en odometría*, que muestra el mapa de rejilla obtenido teniendo en cuenta únicamente la información odométrica.

- **Guardar Mapa Mejor Partícula**. Muestra un cuadro de diálogo de *guardar archivo*, para seleccionar un archivo en el que almacenar información sobre los hitos de la que es la mejor partícula en ese momento.
- **Reiniciar**. Sitúa la simulación en la situación inicial.

9.2.6.4. Barra de menús

Vamos a mostrar la estructura y función de cada uno de los menús y opciones que los integran:

- **Archivo**
  - **Seleccionar Origen**. Muestra la ventana denominada *Selección de origen*, que permite seleccionar el tipo y el archivo concreto origen, que contiene los datos registrados sobre un recorrido de un robot en un entorno.
  - **Salir**. Finaliza la ejecución de la aplicación.
- **Configuración**
  - **Configuración Manual**. Muestra la ventana denominada *Configuración manual*, que permite ajustar los parámetros del algoritmo.
  - **Cargar Configuración**. Muestra un cuadro de diálogo de *abrir archivo*, para indicar el archivo de configuración a cargar.
  - **Guardar Configuración**. Muestra un cuadro de diálogo de *guardar archivo*, para indicar el archivo en el que guardar la configuración.
  - **Establecer como predeterminada**. Permite establecer la configuración actual como configuración predeterminada, es decir, la que se carga al iniciar la aplicación. Antes de cambiar la configuración predeterminada se muestra el aviso de la figura 9.20.

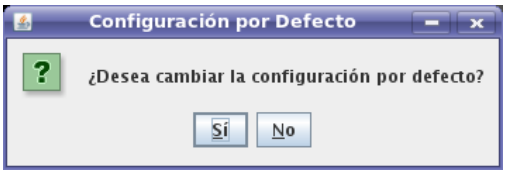


Figura 9.20: Aviso mostrado antes de cambiar la configuración predeterminada.

9.2.6.5. Ventana de Configuración Manual

Esta ventana permite modificar todos los parámetros del algoritmo. Como puede observarse en la figura 9.21, se encuentra dividida en los siguientes compartimentos temáticos:

Figura 9.21: Ventana de configuración manual.

- Filtro de partículas
- Modelo de movimiento.
- Modelo de observación.
- Algoritmo Split and Merge.
- Remuestreo.

9.2.6.6. Ventana de Selección de Origen

Esta ventana se muestra en la figura 9.22. Permite seleccionar un tipo y un archivo origen de datos para le ejecución del algoritmo. Los tipos se seleccionan desde una lista desplegable, mientras que el archivo se determina mediante un cuadro de diálogo de *abrir archivo*.

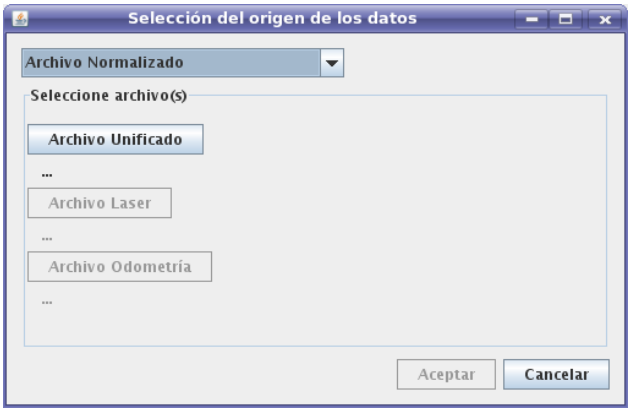
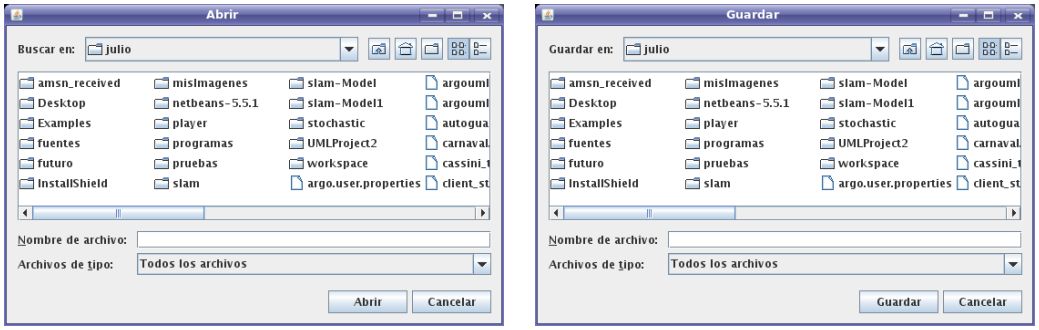


Figura 9.22: Ventana de Selección de Origen.

9.2.6.7. Otros cuadros de diálogo

En la figura 9.23 se muestran los cuadros de dialogo empleados para abrir o guardar en un archivo. Dentro de la aplicación se emplean para cargar un archivo origen, y para la carga y almacenamiento de configuraciones. Este tipo de cuadros de diálogo son muy empleados por casi todas las aplicaciones, por lo que son proporcionados por el propio lenguaje de programación, no ha sido necesario, pues, diseñarlos, sino únicamente integrarlos en el sistema.



(a) Abrir archivo (b) Guardar archivo

Figura 9.23: Cuadros de dialogo para abrir y guardar archivos.

Cuando se intenta guardar una configuración en un archivo que ya existe, se muestra el cuadro de diálogo de la figura 9.24, mediante el que el usuario puede decidir sobreescribir el archivo o bien volver al cuadro de diálogo *guardar archivo* y para seleccionar otro destino.

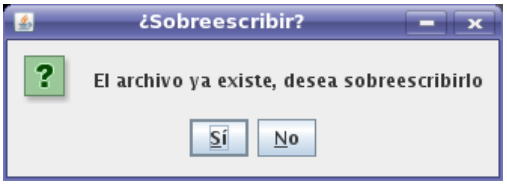


Figura 9.24: Advertencia de sobreescritura.

9.2.7. Interfaz de línea de comandos

En general, las interfaces de línea de comandos están dirigidas a usuarios expertos que buscan rapidez y flexibilidad de uso. Hay que tener en cuenta que para ciertas tareas, como la ejecución de un número de simulaciones con el objeto de recopilar datos, la interfaz gráfica no aporta ninguna ventaja y consume recursos del sistema que ralentizan la ejecución. Además, una vez familiarizado con el uso de los comandos, un usuario es más rápido y productivo mediante la línea de comandos que usando una interfaz gráfica.

Mediante la interfaz de línea de comandos nuestra aplicación va a proporcionar la capacidad de lanzar simulaciones en lotes, de modo que se puede especificar el parámetro que se desea variar y los valores a probar. Como resultado se van a obtener una serie de archivos que contienen información acerca de los hitos de los mapas construídos, y archivos con parámetros de calidad de los mapas construídos. Evidentemente, esta funcionalidad sólo tiene sentido para mapas de los que se disponga un mapa de referencia, es decir, para los que se disponga de las características de los hitos reales presentes en el mapa.

A continuación se describe el uso de la interfaz de línea de comandos. Se ha diseñado un único comando en el que se integra toda la funcionalidad necesaria, que permite su ajuste mediante parámetros. El prototipo de invocación del comando es:

```
BatchSLAM [-h]<Config><Salida><Referencia>
   <Parámetro><Valores>
```

Los corchetes `[]` indican que el parámetro es opcional, mientras que los símbolos mayor/menor `<>` indican la obligatoriedad del parámetro:

- `-h`, muestra la ayuda.
- Config, ruta del archivo de configuración.
- Salida, ruta del archivo de salida.
- Referencia, ruta del archivo de referencia.
- Parámetro, el parámetro a modificar en las distintas simulaciones. Los parámetros pueden ser
  - `p`, número de partículas.
  - `dmax`, distancia máxima de un punto a la recta de la que forma parte.
  - `mlon`, longitud mínima de una recta.
  - `mpuntos`, número mínimo de puntos de una recta.
  - `sensor`, componentes σ_ρ^2,σ_θ^2 de la covarianza del modelo de observación.
  - `mov`, componentes σ_x^2,σ_θ^2,σ_x^2 de la covarianza del modelo de movimiento.
  - `p0`, similitud de una nueva observación.
  - `col`, umbrales ρ_{max},θ_{max} de colinealidad.
  - `es`, umbral de distancia de una esquina a las rectas cuya intersección al originan.
- Valores, lista de los valores que ha de tomar el parámetro. Para todas las opciones se especifica un valor por simulación, salvo para el parámetro `sensor`, que debe recibir la varianza del rango ρ y la varianza del ángulo θ.

9.3. Planificación y evolución temporal

En el desarrollo del proyecto se ha utilizado un enfoque ingenieril conocido como *ciclo de vida incremental* [62]. El ciclo de vida incremental va produciendo sucesivas versiones del producto en las que se va añadiendo funcionalidad. Además permite adaptarse a nuevos requerimientos. Los distintos enfoques de la ingeniería del software aportan una disciplina al desarrollo de proyectos, que redunda en una disminución de los riesgos, y un aumento en la calidad del producto. El ciclo de vida incremental comprende las siguientes fases:

- Análisis
- Diseño
- Implementación
- Prueba

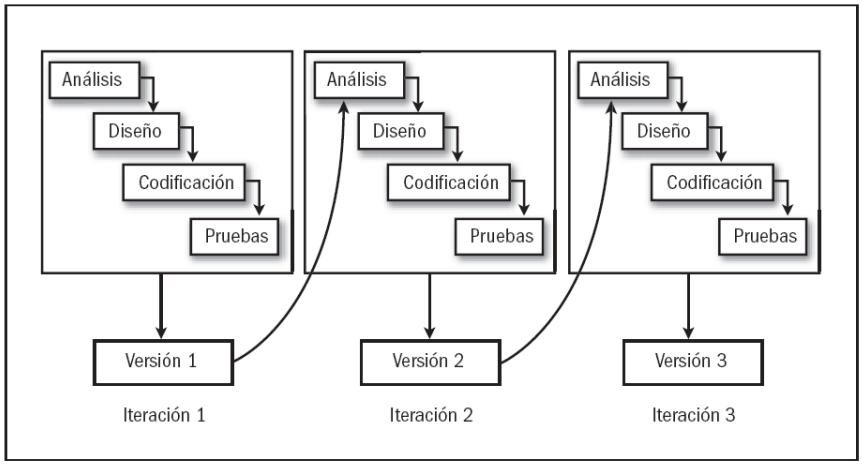


Figura 9.25: Ciclo de vida Iterativo

Cada una de las iteraciones comprende todas las fases (ver Figura 9.25), y han venido acompañadas de la elaboración de prototipos que han permitido refinar el funcionamiento, y realizar pruebas del código generado, desde los primeros momentos. Además, como complemento a la documentación de análisis, diseño e implementación, se han utilizado diversos diagramas UML, tales como diagramas de clases, de actividades o de interacción.

El proyecto ha sido realizado a través de la serie de etapas que se enumeran a continuación:

1. Estudio y análisis del problema (90h)
   a) Estudio y análisis del problema del SLAM.
   b) Estudio y análisis del algoritmo FastSLAM 1.0.

   c) Análisis y diseño del entorno a desarrollar para la prueba y evaluación del algoritmo.

   d) Especificación de experimentos para la evaluación y prueba del algoritmo.

2. Desarrollo e implementación (120h)

   a) Implementación del algoritmo FastSLAM 1.0.

   b) Implementación del entorno para la prueba y evaluación del algoritmo.

3. Evaluación y prueba (60h)

   a) Realización de los diferentes experimentos definidos.

   b) Análisis cualitativo y cuantitativo de los resultados obtenidos.

   c) Reflexión y obtención de conclusiones en base a los resultados.

4. Documento de Proyecto final de carrera. (120h)

   a) Integración de la documentación generada en cada fase en la memoria final.

   b) Preparación de la defensa del proyecto.

Capítulo 10

Prueba y evaluación

10.1. Introducción

Para poder aplicar adecuadamente un método de construcción automática de mapas, y en general cualquier método parametrizable, es necesario conocer la influencia de los distintos parámetros sobre el funcionamiento del mismo. Además, siempre es deseable disponer de una serie de directrices que permitan ajustar los parámetros a las condiciones de funcionamiento. En el caso de la construcción automática de mapas las condiciones de funcionamiento se refieren a la tipología y distribución de los hitos presentes en el entorno (longitud de las rectas, ángulos que forman entre sí, alineamiento, etc, ...), así como el tamaño del mismo. Todo esto suscita la necesidad de un proceso de prueba y evaluación del método, que en nuestro caso va a llevarse a cabo gracias al entorno definido en el capítulo 8.

Los parámetros que afectan a nuestra implementación del método FastSLAM [40] para la construcción automática de mapas, pueden dividirse en tres grupos. Por un lado encontramos los parámetros que afectan a la extracción de hitos referenciales, concretamente los parámetros del algoritmo de detección de segmentos rectilíneos *Split & Merge* [7, 44]:

- Mínimo número de puntos de un segmento válido (P_{min}).
- Longitud mínima de un segmento válido (L_{min}).
- Máxima distancia a una recta de un punto integrante de la misma (D_{max}).
- Umbrales máximos de colinealidad (ρ_{max}, θ_{max}).

y los que afectan a la detección de esquinas,

- Distancia máxima de una esquina a las rectas que la determinan, para considerarla verdadera (E_{max}).

Por otro lado encontramos parámetros que afectan al los modelos de movimiento y observación:

- Matriz de covarianza del ruido del modelo de observación ($\begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix}$).

- Matriz de covarianza del ruido del modelo de movimiento ($\begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix}$).

Por último encontramos parámetros directamente relacionados con el filtro de partículas:

- Número de partículas (Par).
- Umbral para el número de partículas efectivas (N_{eff}).
- Probabilidad asociada a una nueva observación (P_0).

Por otro lado, dadas las enormes dimensiones del problema del SLAM, y su posibilidad de aplicación a infinidad de entornos, tanto exteriores como interiores, restringimos nuestro estudio a entornos interiores de tipo oficina, determinados por una alta presencia de segmentos rectilíneos (paredes, escritorios, etc), y generalmente estructurados siguiendo cierto orden. Elegimos este acercamiento pues el enfrentar el método a entornos menos ambiciosos va a permitir poner de manifiesto sus características, además de existir infinidad de aplicaciones para sistemas robóticos móviles en este tipo de entornos.

En el presente capítulo vamos a describir los entornos de prueba así como los experimentos realizados para determinar la influencia de cada uno de los parámetros que afectan al método.

En la sección 10.2 se describen los distintos entornos de prueba que se emplean a lo largo del capítulo, destacando sus características más importantes, y la finalidad de su uso. A continuación, en la sección 10.3 se describe el proceso de estimación y medida de la calidad del mapa obtenido tras una simulación. Después de esto se analiza, en la sección 10.4, la influencia de los parámetros relacionados con la extracción de características. En la sección 10.5 se estudian los efectos de los ruidos en los modelos de observación y movimiento, pasando a estudiar en la sección 10.6 distintos aspectos relacionados con los filtros de partículas. Finalmente, en la sección 10.7 se ofrecen una serie de recomendaciones para el ajuste de todos los parámetros de nuestra implementación del algoritmo.

10.2. Entornos de prueba

Los entornos de prueba empleados se dividen en tres categorías:

- **Entornos sintéticos**. Mapas diseñados con la herramienta Qcad, cuyo recorrido ha sido simulado empleando la herramienta Player/Stage. Los mapas sintéticos permiten un control absoluto sobre la configuración del entorno. La herramienta Player/Stage simula adecuadamente los sensores de rango láser SICK, sin embargo presenta alguna limitaciones en cuanto a la simulación del ruido en la odometría, de modo que se ha optado por realizar simulaciones con una odometría perfecta añadiendo ruido a posteriori. En concreto se ha considerado que las velocidades lineal v_i y angular w_i están afectadas por un error Gaussiano de media nula y varianza del 10 % de la propia velocidad medida en cada momento.

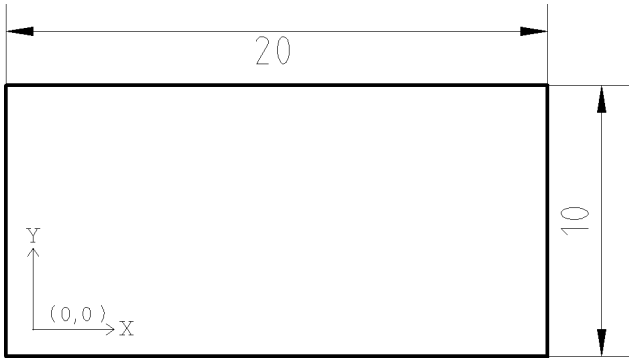
$$\begin{aligned} v_i &= v_i + N(0, v_i * 0{,}1) \\ w_i &= w_i + N(0, w_i * 0{,}1) \end{aligned}$$

Este valor de varianza ha sido elegido por ser el que permite obtener mapas correctos con un uso de 100 partículas aproximadamente, y unos ruidos próximos a los empleados en los mapas reales. Valores menores producen mapas basados en la odometría muy poco distorsionados, mientras que valores superiores, exigen el uso de demasiadas partículas para ser correctamente cartografiados.

- **Entornos reales propios**. Tanto el recorrido como la configuración del entorno han sido realizados físicamente por nosotros. Este tipo de entornos permiten comprobar el funcionamiento del algoritmo en situaciones reales sobre las que se tiene cierto control.
- **Entornos reales ajenos**. Tanto el recorrido como la configuración del entorno han sido realizados por terceros que ponen a disposición pública los archivos de registro obtenidos. Este tipo de entornos permite comprobar el funcionamiento en situaciones reales fuera de nuestro control. En este sentido suponen la mayor prueba de funcionamiento del algoritmo.

A continuación se describe cada uno de los entornos empleados, indicando su procedencia, tipología, y objetivo de uso. En el caso de entornos sintéticos se muestra, además, el mapa diseñado y las características presentes en el mismo. Para los entornos reales se muestra un mapa de rejilla, ya sea el proporcionado por terceros, o el mejor obtenido por nosotros mismos.

10.2.0.1. Recinto básico



(a)

Rectas		Esquinas	
ρ (m)	θ (grados)	x (m)	y (m)
1	-90	-1	-1
9	90	-1	9
1	180	19	-1
19	0	19	9

(b)

Figura 10.1: (a) Mapa *Recinto básico*. Medidas en metros. (b) Rectas y esquinas presentes en el mapa.

Tipología. Recinto rectangular sin elementos en su interior. Es el recinto más sencillo. Se encuentra formado por grandes rectas que forman ángulos de 90 grados, fácilmente detectables.

Objetivo. Servir de referencia para comparar con mapas más complejos. Este mapa es el más sencillo de todos los entornos sintéticos empleados. Sin embargo, esto no significa que sea el más favorable para realizar el proceso de SLAM, ya que la escasez de características puede provocar que en algún momento el algoritmo no funcione correctamente.

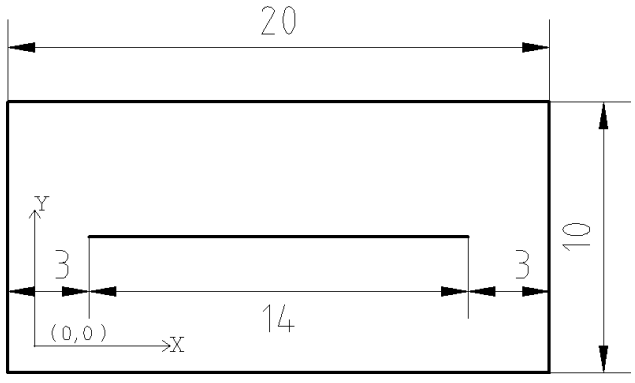
Origen. Sintético. Mapa realizado con la herramienta CAD *Qcad*, y recorrido simulado con la herramienta *Player/Stage*.

10.2.0.2. Lineamedia

Tipología. Recinto rectangular básico con una pared en su interior. Lo constituyen rectas grandes que forman ángulos de 90 grados, fácilmente detectables.

Objetivo. Analizar el comportamiento frente a bucles grandes y sencillos. Este mapa añade únicamente una recta al mapa *Recinto básico*, no obstante, esta recta provoca la aparición de una estructura fundamental en entornos interiores; el bucle.

Origen. Sintético. Mapa realizado con la herramienta CAD *Qcad*, y recorrido simulado con la herramienta *Player/Stage*.



(a)

Rectas		Esquinas	
ρ (m)	θ (grados)	x (m)	y (m)
1	-90	19	-1
9	90	-1	-1
19	0	-1	9
1	180	19	9
4	90		

(b)

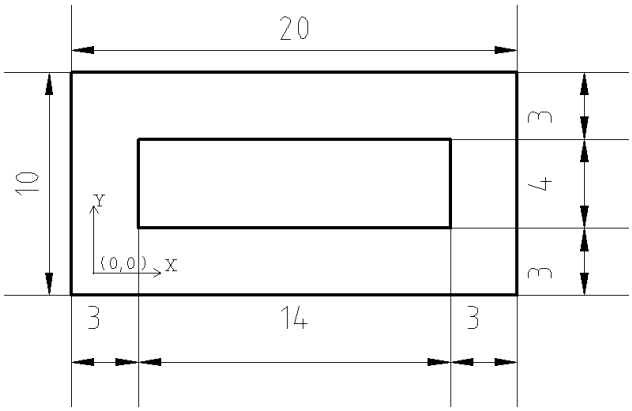
Figura 10.2: (a) Mapa *Lineamedia*, caracterizado por un recinto rectangular con una recta en su interior. Medidas en metros. (b) Rectas y esquinas presentes en el mapa.

10.2.0.3. Bigloop

Tipología. Recinto rectangular básico con un bucle en su interior. Tanto el recinto como el bucle son bloques rectangulares constituídos por grandes rectas que forman ángulos de 90 grados, fácilmente detectables.

Objetivo. Analizar el comportamiento frente a bucles de gran tamaño con un nivel de complejidad superior al mapa *Lineamedia*.

Origen. Sintético. Mapa realizado con la herramienta CAD *Qcad*, y recorrido simulado con la herramienta *Player/Stage*.



(a)

Rectas		Esquinas	
ρ (m)	θ (grados)	x (m)	y (m)
1	-90	2	2
2	90	19	-1
2	0	16	2
19	0	19	9
16	0	16	6
9	90	-1	9
6	90	2	6
1	180	-1	-1

(b)

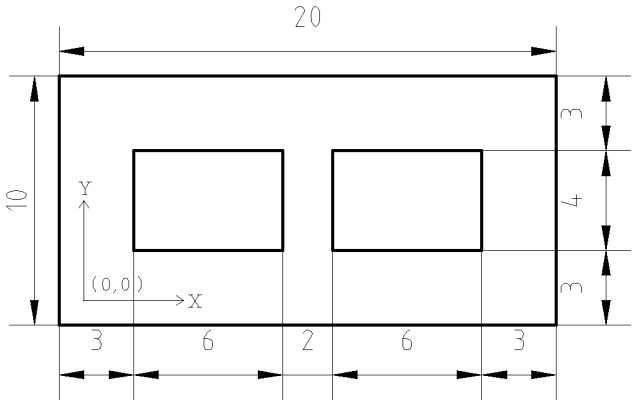
Figura 10.3: (a) Mapa *Bigloop*. Medidas en metros. (b) Rectas y esquinas presentes en el mapa.

10.2.0.4. Bigloop2

Tipología. Recinto rectangular básico con dos bucles de tamaño medio en su interior. Tanto el recinto como los bucles son bloques rectangulares. Las rectas que constituyen el recinto son grandes, mientras que las de los bucles son de un tamaño correspondiente a la mitad del tamaño de las rectas del recinto. Las rectas forman 90 grados, y en general son fácilmente detectables.

Objetivo. Analizar el comportamiento frente a entornos con múltiples bucles de tamaño menor a los de los mapas *Lineamedia* o *Bigloop*.

Origen. Sintético. Mapa realizado con la herramienta CAD *Qcad*, y recorrido simulado con la herramienta *Player/Stage*.



(a)

Rectas		Esquinas	
ρ (m)	θ (grados)	x (m)	y (m)
1	-90	2	2
2	90	19	-1
2	0	16	2
19	0	19	9
16	0	16	6
9	90	-1	9
6	90	2	6
1	180	-1	-1
10	0	10	6
8	0	8	2
		8	6
		10	2

(b)

Figura 10.4: (a) Mapa *Bigloop2*. Medidas en metros. (b) Rectas y esquinas presentes en el mapa.

10.2.0.5. Complex

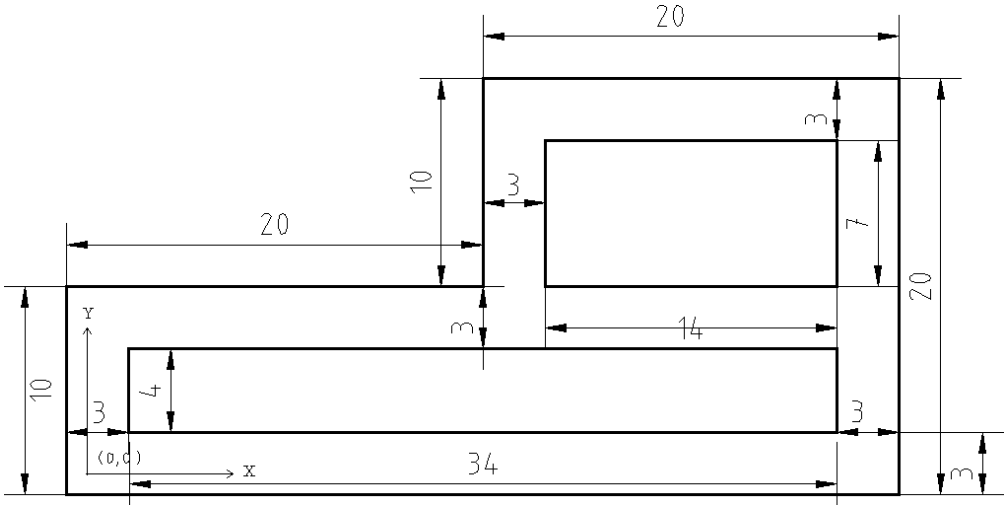


Figura 10.5: (a) Mapa *Complex*. Medidas en metros.

Tipología. Recinto no rectangular con bucles interiores rectangulares. Constituido por rectas de diversos tamaños, siempre formando ángulos de 90 grados. En general todas las rectas son fácilmente detectables.

Objetivo. Analizar el comportamiento en entornos con cierto grado de complejidad, en los que encontramos bucles de diversos tamaños no alineados.

Origen. Sintético. Mapa realizado con la herramienta CAD *Qcad*, y recorrido simulado con la herramienta *Player/Stage*.

Rectas		Esquinas	
ρ (m)	θ (grados)	x (m)	y (m)
1	-90	-1	-1
1	180	-1	9
9	90	19	9
19	0	19	19
19	90	39	19
39	0	39	-1
2	0	2	2
36	0	2	6
2	90	36	2
6	90	22	9
22	0	22	16
16	90	36	6
		36	16
		36	9

Tabla 10.1: Rectas y esquinas presentes en el mapa *Complex*.

10.2.0.6. USC SAL building

Tipología. Gran bucle central con habitaciones a su alrededor. Edificio SAL de la Universidad del Sur de California (University of Southern California (`http://www.usc.edu/`)).

Objetivo. Analizar el comportamiento del método en entornos reales, como complemento a los entornos simulados. Comprobar el funcionamiento frente a datos reales fuera de nuestro control.

Origen. Repositorio radish (`http://radish.sourceforge.net`).

Al ser éste un mapa obtenido en un recorrido real de un entorno al que no se tiene acceso, no se dispone de datos acerca de los elementos geométricos reales que lo integran. Por ello, la comparación con los mapas obtenidos por nuestro algoritmo se realizará por inspección visual de los mapas de rejilla, determinando la divergencia o no, y el grado de similitud, a grosso modo. La experimentación con datos reales, fuera de nuestro control, supone una prueba definitiva para nuestra implementación del algoritmo, y proporciona una idea más clara acerca de sus capacidades funcionales, debilidades, y fortalezas.

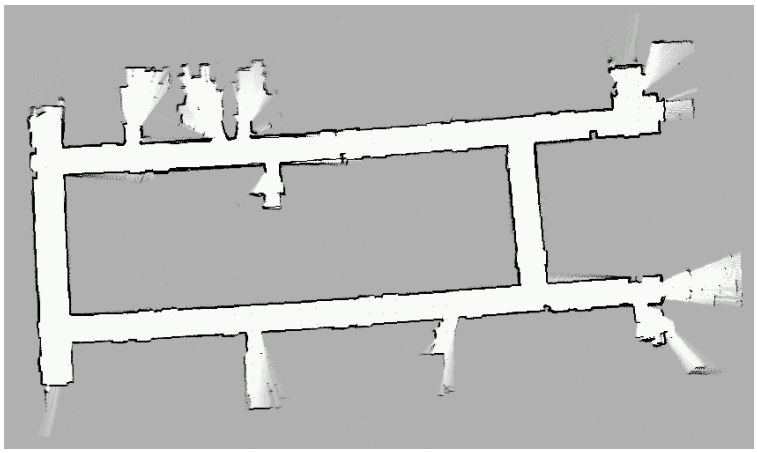


Figura 10.6: Mapa de rejilla proporcionado por junto con los archivos de registro. Dimensiones aproximadas 40 x 15 metros.

Para este recinto se dispone de dos recorridos distintos, lo que añade aun más riqueza a las pruebas.

10.2.0.7. Simulación IUSIANI

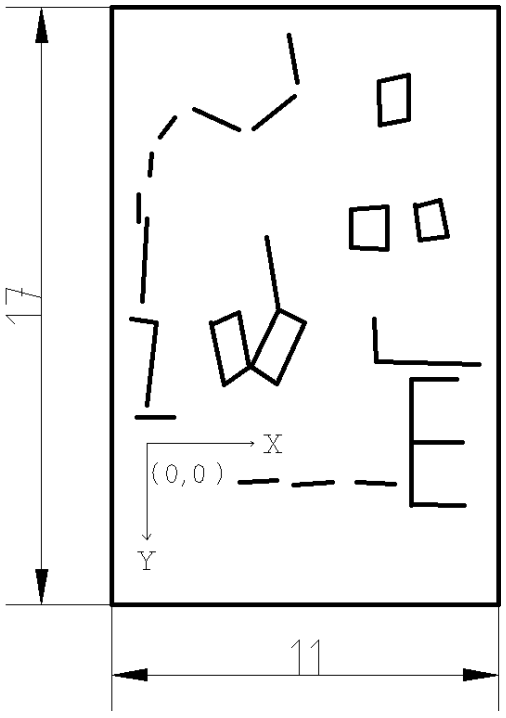


Figura 10.7: Mapa del laboratorio del IUSIANI. Dimensiones en metros.

Tipología. Simulación del Laboratorio del Instituto Universitario de Sistemas Inteligentes y Aplicaciones Numéricas en Ingeniería (IUSIANI) de la Universidad de Las Palmas de Gran Canaria (`http://iusiani.ulpgc.es`). Mapa constituido por distintos tipos de rectas distribuídas de modo irregular y que forman diversos ángulos entre si, al contrario que el resto de mapas simulados en los que las rectas siempre forman ángulos de 90 grados.

Objetivo. Analizar desde un punto de vista cualitativo el comportamiento del método frente a la frecuencia de remuestreo.

Origen. Sintético. Mapa realizado con la herramienta CAD *Qcad*, y recorrido simulado con la herramienta *Player/Stage*.

Debido a que este mapa es una representación de una situación real del laboratorio del IUSIANI, de la que no se conocen las medidas exactas, no se van a considerar ni las dimensiones ni las posiciones de los elementos que lo integran. Además, dado que su objetivo es una evaluación cualitativa, bastará con realizar una inspección visual para determinar los resultados, en cuanto a la calidad del mapa de rejilla obtenido.

10.2.0.8. IUSIANI

Tipología. Laboratorio IUSIANI. Mapa real constituido por distintos tipos de rectas distribuídas de modo irregular y que forman diversos ángulos entre si, al contrario que el resto de mapas simulados en los que las rectas siempre forman ángulos de 90 grados.

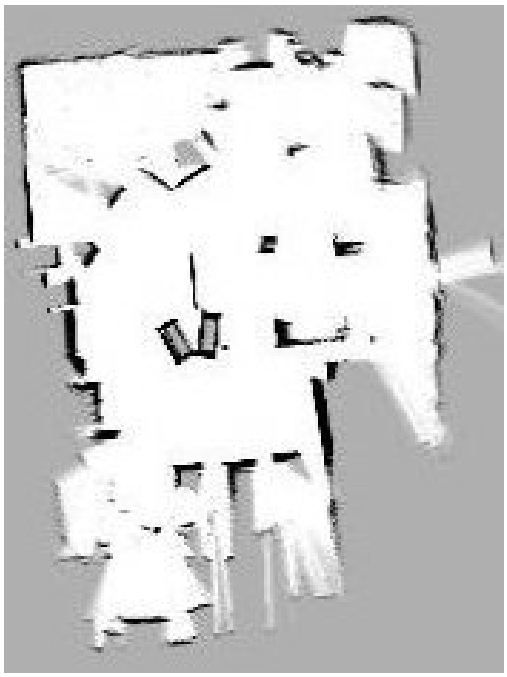


Figura 10.8: Mapa del laboratorio del IUSIANI. Dimensiones aproximadas 8 x 13 metros.

Objetivo. Analizar desde un punto de vista cualitativo el comportamiento del método frente a un entorno real bajo nuestro control.

Origen. Mapa obtenido en un recorrido real empleando una plataforma robótica Pioneer DX3 y un medidor de rango láser SICK LSM-200, registrando los datos mediante la herramienta *Player/Stage*.

Debido a que este mapa es una representación de una situación real del laboratorio del IUSIANI, de la que no se conocen las medidas exactas, no se van a considerar ni las dimensiones ni las posiciones de los elementos que lo integran. Además, dado que su objetivo es una evaluación cualitativa, bastará con realizar una inspección visual para determinar los resultados, en cuanto a la calidad del mapa de rejilla obtenido.

10.3. Medidas

Para evaluar el impacto de un determinado parámetro en el algoritmo se van a emplear dos tipos de resultados. Por un lado, considerando el mapa como una colección de hitos referenciales, se va a calcular la diferencia entre el mapa obtenido y el mapa *real* o de referencia. Denominamos a esta diferencia el *error* del mapa obtenido. Por otro lado, se va a comparar visualmente el mapa de rejilla obtenido, con el mapa real, analizando cuestiones como el cierre de bucles, o los ángulos que forman las rectas.

Para calcular el error se procede de la siguiente manera:

- Se fijan todos los parámetros de la simulación, excepto el que se esté estudiando. Lo que se busca es una configuración media que funcione bien, esto es, que permita obtener mapas coherentes, de modo que el modificar alguno de los parámetros permita observar cambios apreciables en los mapas obtenidos. Este proceso no es exhaustivo y su fin es encontrar una configuración que permita ejecutar la simulación de todos los mapas en unas mismas condiciones.

- Para cada mapa y valor concreto del parámetro a estudiar se realizan 5 simulaciones del proceso de SLAM. El motivo de emplear un número de simulaciones limitado se debe al hecho de que un conjunto completo de experimentos tarda aproximadamente 76 horas en realizarse, con lo que realizar por ejemplo 100 simulaciones supondría dos meses de computación.

- Para cada una de las 5 simulaciones s se calcula el error como la suma del error cada uno de los hitos referenciales i detectados dividido entre el número total de hitos detectados N.

$$error_s = \frac{1}{N} \sum_{i=1}^{N} error_i$$

El error para las esquinas se toma como la distancia euclídea

$$error_i = \sqrt{(x_i - x_m)^2 + (y_i - y_m)^2}$$

donde el subíndice i corresponde al hito detectado, y el subíndice m al hito real, es decir, aquel con cuyo emparejamiento se minimiza el error. Para las características de tipo línea se toma igualmente como medida del error la distancia euclídea, pero transformando las coordenadas polares (ρ, θ) de las rectas a un punto en coordenadas rectangulares.

$$x = \rho \cos\theta \tag{10.1}$$
$$y = \rho \sin\theta \tag{10.2}$$
$$error_i = \sqrt{(x_i - x_m)^2 + (y_i - y_m)^2} \tag{10.3}$$

donde el subíndice i corresponde al hito detectado, y el subíndice m al hito real. Por último el error para cada combinación mapa e, valor del parámetro v_p, se calcula como la media de los errores de cada una de las N simulaciones

$$error(e, v_p) = \frac{\sum_{s=1}^{N} error_s}{N}$$

- En caso de que se detecten más hitos referenciales que los presentes en el mapa real, se calculará el error para cada uno de ellos, asociándolo al hito real que lo haya generado.
- Con los errores registrados para cada pareja (e, v_p), se realiza una gráfica para exponer los resultados.

Estas medidas de error no pretenden ser una medida absoluta, de modo que sirvan para comparar nuestros resultados con los obtenidos con otros algoritmos, sino que pretenden poner de manifiesto el comportamiento de nuestra implementación con respecto a diversos parámetros. La intención es destacar la tendencia del error, no su valor numérico concreto.

Además de gráficas de error se van a mostrar gráficas de hitos referenciales detectados en función de determinados parámetros. Para determinar el número de hitos detectados se calcula la media del número de hitos detectados en cada una de las N simulaciones que se realizan para cada valor concreto de un parámetro.

Combinando el error con la observación del mapa de rejilla obtenido, y con los hitos detectados, obtendremos una información completa sobre el comportamiento del algoritmo frente a un determinado parámetro. Así, se mostrarán gráficas de error e hitos detectados, en aquellas situaciones en las que los resultados tengan sentido, mientras que se comparan mapas de rejilla cuando éstos ofrezcan una mejor visualización del proceso.

10.3.1. Significado de los errores

A la hora de analizar los errores cabe preguntarse: ¿Es adecuado un mapa en el que el error medio por hito ronda los 15 o 20 cm? La respuesta en general no es sencilla y depende

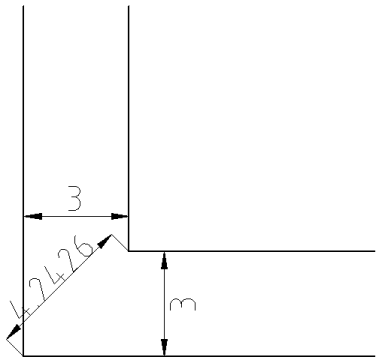


Figura 10.9: Detalle de la consideración del grosor de una pared. Unidades en píxels.

en gran medida de lo que se desee realizar con el mapa obtenido, y de las condiciones en las que se ha obtenido.

En primer lugar hay que tener en cuenta que el error computado no es únicamente imputable a la bondad o no de los hitos detectados en el proceso de SLAM. El error tiene además aportaciones de los siguientes factores:

- Cuantización del espacio realizada por el software de simulación Player/Stage. En base a la imgen proporcionada y las dimensiones indicadas el simulador construye una representación interna con una resolución de 2 cm por pixel. El error que induce esta cuantización no es cuantificable pues no conocemos los detalles internos de la herramienta.

- Las posiciones teóricas de los hitos de los mapas (las que sirven de referencia para el cálculo del error) no tienen en cuenta el grosor que tienen las paredes al representar los mapas como imágenes, paso necesario para poder usar la herramienta Player/Stage. Se ha intentado disminuir al máximo el grosor de las líneas que representan las paredes, logrando dejarlas entre 2 y 3 píxels. Esto, dependiendo de la escala de los diversos mapas, puede llegar a suponer hasta 13 cm de error inducido:

  - Un mapa de 1024x768 píxels representa entornos de 40x20 m y de 20x10 m, de modo que obtenemos una resolución por pixel de 3,2 y 1,8 cm respectívamente.
  - Si las paredes tienen un grosor de 3 pixels el error máximo que pueden inducir es la longitud de la hipotenusa de un triángulo rectángulo cuyos catetos miden 3 pixels: $\sqrt{(3*3{,}2)^2+(3*3{,}2)^2} = 13,57$ cm para entornos de 40x20 m y $\sqrt{(3*1{,}8)^2+(3*1{,}8)^2} = 7,63$ cm.

- El mapa obtenido es el mapa más probable, no el mejor mapa. La mejor partícula en un momento determinado es aquella que mejor *explica* la última observación, no aquella que minimiza el error. No debe olvidarse que el verdadero mapa viene representado por todas las partículas, ya que estas son una representación de la distribución probabilística de los distitos mapas y poses.

Con los datos expuestos, un mapa perfecto, comparado con uno teórico, podría llegar a acumular de 7 a 14 cm de error por hito en el peor de los casos, sin considerar el error de cuantización. Bajo esta perspectiva, se comprobará, que los mapas obtenidos son desde luego aptos para la navegación. En cualquier caso los experimentos realizados han considerado entornos de grandes dimensiones que se recorren con una velocidad máxima de 0,5 m/s (1,8 Km/h), realizando trayectos de más de 200 m, con lo que errores reales en torno a 10 cm en la posición de un hito (que como se verá en las siguiente secciones son los que se registran) no pueden ser considerados inaceptables.

En cualquier caso el objetivo de este proyecto es determinar la tendencia de la calidad de los mapas obtenidos en función de distintos parámetros, ya sea mediante el análisis de la tendencia de errores o por inspección visual de mapas de rejilla.

10.4. Estudio de la influencia de los parámetros de la extracción de características

La extracción de hitos referenciales a partir de información sensorial es vital para el funcionamiento de los algoritmos de SLAM. Por ello es necesario conocer en profundidad las características del método empleado y la influencia de los distintos parámetros que pueda contener. En nuestra implementación del algoritmo *Split & Merge* (ver sección 4.2), para la extracción de rectas a partir de barridos láser, existen los siguientes atributos parametrizables:

- Distancia máxima a una recta de un punto integrante de la misma (D_{max}).
- Longitud mínima de un segmento (L_{min}).
- Mínimo número de puntos que componen un segmento (P_{min}).
- Umbrales máximos de colinealidad (ρ_{max}, θ_{max}).

además en cuanto a la detección de esquinas es posible especificar:

- Distancia máxima de una esquina a las rectas que la determinan, para considerarla verdadera (E_{max}).

En general, una buena configuración dependerá de las condiciones del entorno que se esté percibiendo, y de un balance adecuado entre los distintos parámetros. Por ejemplo, si las rectas presentes en el entorno son todas menores de 0,5 m, se debe fijar la longitud mínima de una recta en torno a 0,4 m. Para evitar la detección de rectas falsas a la que puede conducir un valor bajo de L_{min}, se puede incrementar el número de puntos mínimo de una recta válida, de modo que se impone la restricción de que las rectas consideradas verdaderas sean soportadas por una mayor evidencia.

En la presente sección vamos a determinar la influencia de los atributos parametrizables relacionados con la extracción de características en el proceso de SLAM, además de

determinar unos valores típicos para ellos, que sirvan como punto de referencia a la hora de ajustar el algoritmo a un entorno concreto. Para cada uno de los parámetros se van a realizar simulaciones tanto en entornos sintéticos, como en entornos reales, según convenga, ya que ambos tipos de entornos permiten apreciar de diferente manera la influencia de los mencionados parámetros sobre el proceso de SLAM.

De estos parámetros, los de colinealidad, esto es, ρ_{max} y θ_{max}, y el umbral para la detección de esquinas E_{max}, no presentan prácticamente influencia sobre el proceso, por lo que una vez determinados unos valores medios aconsejables se emplearán éstos para el resto de simulaciones, no considerándolos como parámetros del algoritmo.

De no especificarse lo contario debe entenderse que todas las simulaciones se realizan empleando:

- Detección de rectas y esquinas.
- $\rho_{max} = 0{,}1$ m
- $\theta_{max} = 0{,}07$ radianes
- $E_{max} = 0{,}1$ m

En primer lugar, en la sección 10.4.1, se estudia la influencia de D_{max}, pasando a estudiar el efecto de L_{min} en la sección 10.4.2 y de P_{min} en la sección 10.4.3. Además se estudia el efecto de ρ_{max} y θ_{max} en la sección 10.4.4, y de E_{max} en la sección 10.4.5. Se finaliza en la sección 10.4.6 con una recomendación de ajuste de los parámetros del algoritmo.

10.4.1. Distancia máxima a una recta de un punto integrante (D_{max})

A priori cabe pensar que valores muy pequeños de D_{max} provocarán la detección de pocas rectas, al exigirse que los puntos que las forman estén muy alineados. Esto puede manifestarse de dos maneras:

- No se detectan todos los hitos presentes en el entorno. En el tipo de recorridos realizados, en los que se recorren varias veces los bucles, se detectan correctamente todas las rectas. No ocurre así con las esquinas, cuya extracción requiere de la detección en el mismo barrido láser de las rectas cuyas intersecciones las originan.
- Incluso detectando todos los hitos presentes en el entorno, puede darse el caso de que la observación de hitos no sea frecuente, es decir, no siempre será posible detectar los hitos que efectivamente se están percibiendo. Esto originará un empeoramiento en los procesos de asociación de datos, localización de hitos mediante filtros de Kalman, y remuestreo, con lo que es esperable un aumento en el error del mapa obtenido.

Por su parte, valores elevados de D_{max} provocarán la detección errónea de rectas, al relajarse la exigencia de alineamiento de los puntos integrantes de las mismas. Hay que tener

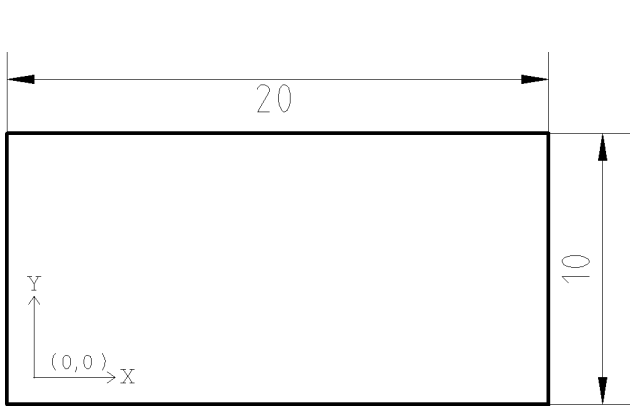
en cuenta que el ruido que afecta al sensor de rango láser está íntimamente relacionado con el valor de D_{max}. Cuanto mayor sea el ruido que afecta al modelo de observación, mayor deberá ser D_{max} de modo que se permita la correcta detección de rectas provenientes de datos ruidosos. Se van a exponer primero una serie de experimentos llevados a cabo con entornos sintéticos y recorridos simulados, mostrando posteriormente los resultados en un entorno y recorrido reales.

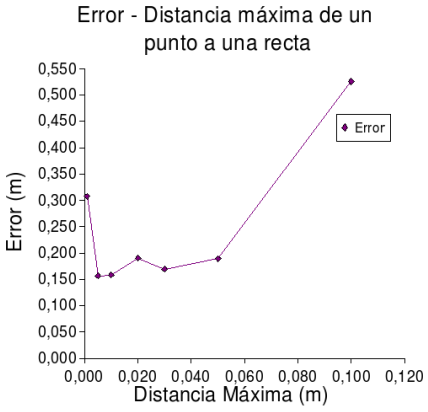
Los parámetros empleados en la simulación del proceso de SLAM en entornos sintéticos se muestran en la tabla 10.2. Entre otras cosas esta configuración establece que se remuestrea siempre, independientemente del número de partículas efectivas N_{eff}. Además, como punto más destacable, se considera una longitud mínima de una recta válida de 40 cm, a pesar de que las rectas en todos los entornos sintéticos superan todas los 2 m. Esto es así para realzar el efecto de D_{max} que queda enmascarado si L_{min} es elevado, ya que es más complicado detectar rectas falsas, si se exige que las rectas válidas sean largas.

Filtro			Split & Merge		Láser	Robot
N_{eff}	P_0	Par	P_{min}	L_{min}	Covarianza	Covarianza
∞	0,005	100	9	0,4	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$	$\begin{pmatrix} 0{,}007 & 0 & 0 \\ 0 & 0{,}007 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

Tabla 10.2: Parámetros de las simulaciones para el estudio de D_{max} en entornos reales.

En la figura 10.10 se muestra, a la izquierda, el mapa de un recinto básico, y a la derecha la evolución del error frente a D_{max}. Se observa como un valor de $D_{max} = 1$ mm produce



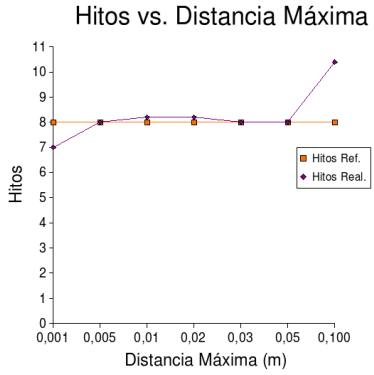


D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Error	0,308	0,156	0,158	0,190	0,169	0,189	0,526
σ^2_{error}	0,0037	0,0030	0,0046	0,0117	0,0028	0,0035	0,0031

Figura 10.10: Mapa real, a la izquierda, y evolución del error frente a D_{max}, a la derecha.

un error alto en comparación con los siguientes. Esto se debe a que al ser la exigencia de alineamiento elevada, no se detectan hitos en todos los barridos, por lo que los procesos de asociación de datos y remuestreo no tienen materia prima con la que trabajar, de modo que no se *ajustan* adecuadamente los hitos mediante los filtros de Kalman, ni se seleccionan adecuadamente las mejores partículas durante el remuestreo. Puede apreciarse como valores de D_{max} entre 5 mm y 5 cm producen un error más o menos estable, mientras que valores más allá de 5 cm producen un incremento sostenido del error. Estos resultados, en los que una exigencia de alineamiento grande, es decir D_{max} pequeño, no parece tener una influencia negativa sobre el error, son acordes a un entorno simulado compuesto por pocas rectas, cuyos puntos se encuentran muy alineados al haber sido diseñadas como rectas perfectas.

Sin embargo esta información no es suficiente para determinar el rango de valores adecuados para D_{max}. Conocemos el número de hitos presentes en el mapa real, 4 rectas y 4 esquinas (ver figura 10.1). Distintos valores de D_{max} van a provocar que en ocasiones se detecte un número de características erróneo, ya sea superior o inferior al real. En la figura 10.11 se muestra el número total de hitos detectados frente a D_{max}. Para valores de D_{max} inferiores a 5 mm el número de hitos detectados es inferior al número de hitos reales. Esto se debe a la excesiva exigencia de alineación de los puntos integrantes de un hito. Entre 5 mm y 5 cm el número de hitos detectados coincide o se aproxima al número real, y para valores por encima de 5 cm se detectan más hitos que los reales. Así, valores entre 5 mm y 5 cm son adecuados para D_{max}.



D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Hitos detectados	7,0	8,0	8,2	8,2	8,0	8,0	10,4

Figura 10.11: Número de hitos detectados frente a D_{max} para el mapa *Recinto básico*.

Finalmente en la figura 10.12 podemos comparar el mapa obtenido mediante FastSLAM con $D_{max} = 2$ cm, y el mapa obtenido empleando únicamente información odométrica.

Se lleva a cabo el mismo experimento para el mapa de un recinto básico con una pared interior. Tanto el mapa como la evolución del error se muestran en la figura 10.13. Se observa una primera zona hasta $D_{max} = 5$ mm dónde el error es más elevado, fruto de la detección escasa de hitos. Entre 1 y 2 cm se producen los mejores resultados, mientras que para valores

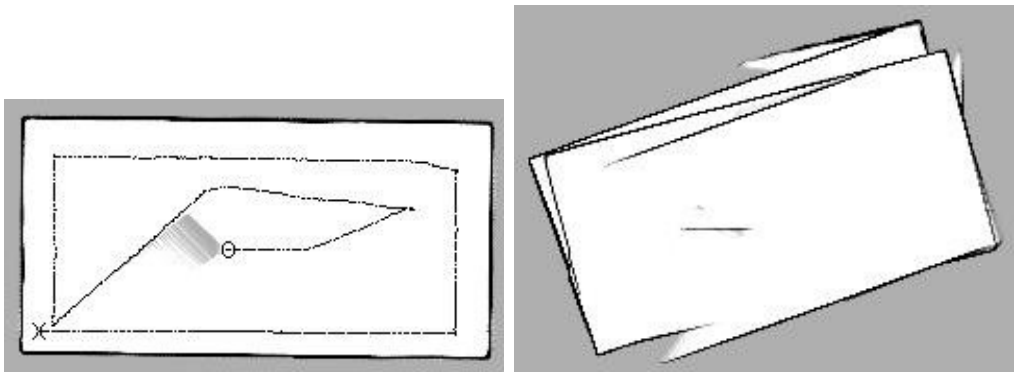
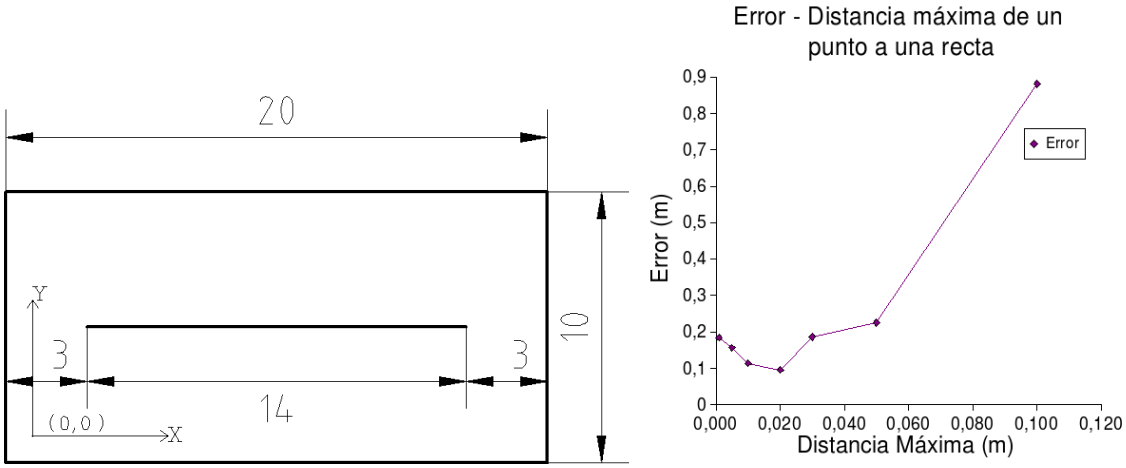


Figura 10.12: A la izquierda mapa de rejilla y recorrido para $D_{max} = 2$ cm, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

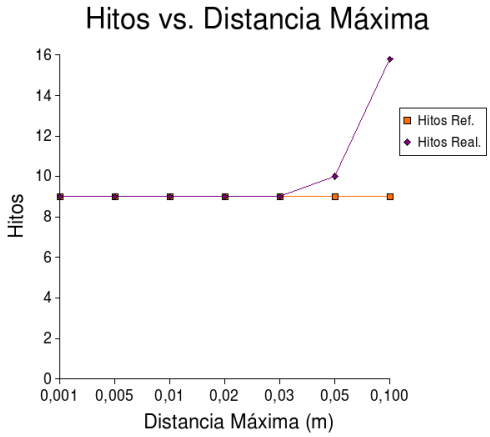
superiores a 2 cm el error se dispara.



D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Error	0,18	0,16	0,11	0,1	0,19	0,22	0,88
σ^2_{error}	0,0050	0,0069	0,0023	0,0012	0,0037	0,0029	0,0025

Figura 10.13: Mapa real, a la izquierda, y evolución del error frente a D_{max} de un punto a la recta de la que forma parte, a la derecha.

Los resultados en cuanto al número de hitos detectados se muestran en la figura 10.14. En este caso el rango de valores de D_{max} para el que se detecta el número correcto de características, que en este mapa es 9 (ver figura 10.2), va de 0 hasta 3 cm. Así combinando la información del error y de los hitos detectados se concluye que el rango de D_{max} adecuado comprende de 1 a 2 cm.



D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Hitos detectados	9,0	9,0	9,0	9,0	9,0	10,0	15,8

Figura 10.14: Número de hitos detectados frente a D_{max} para el mapa *Lineamedia*.

Por último en la figura 10.15 podemos comparar el mapa obtenido mediante FastSLAM con $D_{max} = 2$ cm, y el mapa obtenido empleando únicamente información odométrica. Es evidente la mejoría que se produce al aplicar el proceso de SLAM.

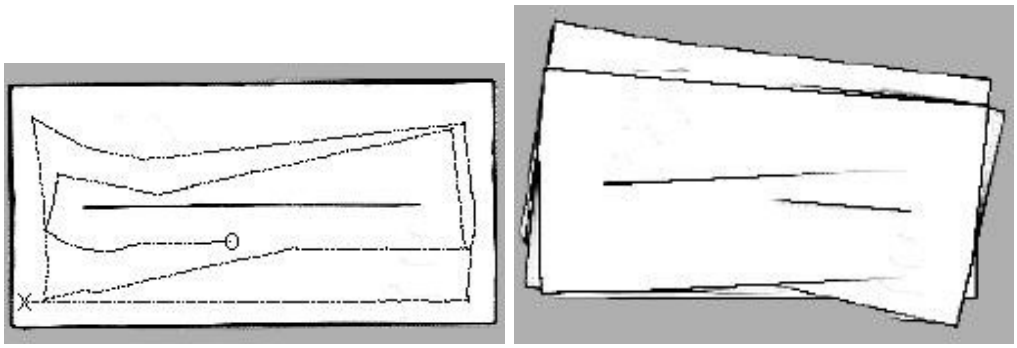
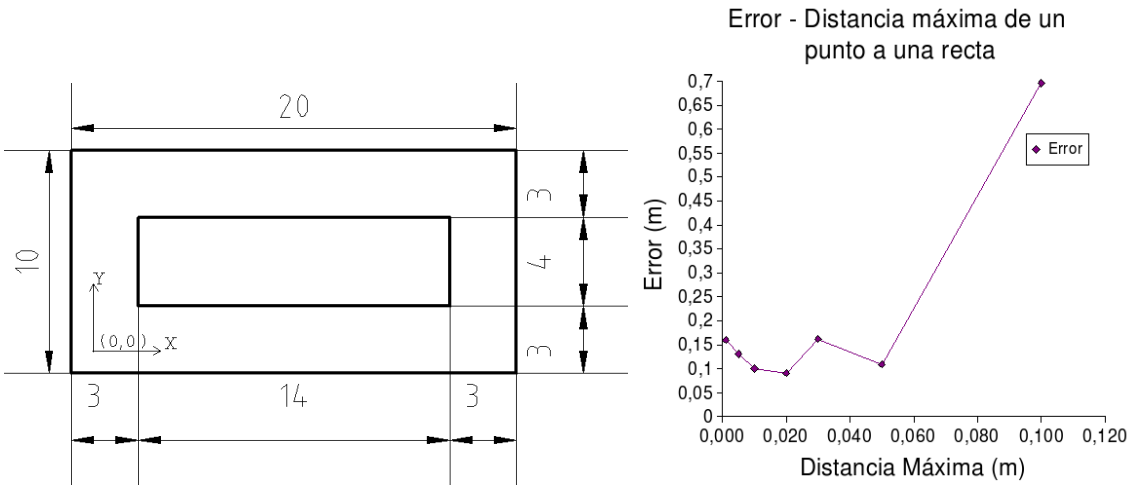


Figura 10.15: A la izquierda mapa de rejilla y recorrido para $D_{max} = 2$ cm, X punto inicial, O punto fina. A la derecha, mapa de rejilla basado en la odometría.

Se reproduce el mismo experimento para el mapa de un recinto básico con un bucle interior. Tanto el mapa como la evolución del error se muestran en la figura 10.16. Los resultados obtenidos son totalmente análogos a los obtenidos en los casos anteriormente estudiados. El rango de valores de D_{max} debe encontrarse entre 1 y 5 cm.

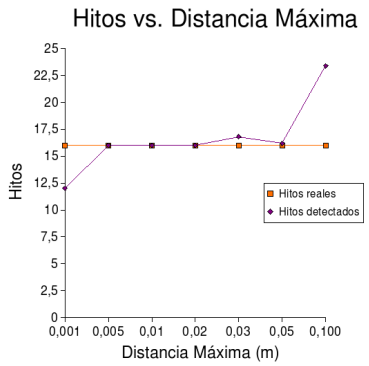
Los resultados en cuanto al número de hitos detectados se muestran en la figura 10.17. En este caso el rango de valores de D_{max} para el que se detecta el número correcto de características, en este caso 16 (ver figura 10.3) va de 0,5 hasta 5 cm. Así combinando la información del error y de los hitos detectados se concluye que el rango de D_{max} adecuado



D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Error	0,16	0,13	0,1	0,09	0,16	0,11	0,7
σ^2_{error}	0,0035	0,0005	0,0008	0,0008	0,0051	0,0042	0,0054

Figura 10.16: Mapa real, a la izquierda, y evolución del error frente a la distancia máxima de un punto a la recta de la que forma parte, a la derecha.

comprende de 1 a 5 cm.



D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Hitos detectados	12,0	16,0	16,0	16,0	16,8	16,2	23,4

Figura 10.17: Número de hitos detectados frente a D_{max} para el mapa *Bigloop*.

En la figura 10.18 podemos comparar el mapa obtenido mediante FastSLAM con $D_{max} =$ 2 cm, y el mapa obtenido empleando únicamente información odométrica.

Siguiendo en una línea ascendente en la complejidad de los entornos estudiados, se pasa a analizar un entorno con dos bucles interiores. Se observa en la figura 10.19 que los mejores resultados se obtienen para valores de D_{max} entre 0 y 5 cm. Más allá el error cometido

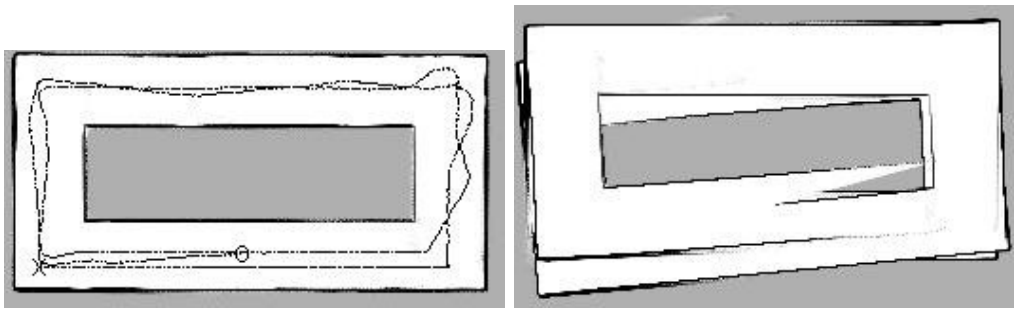
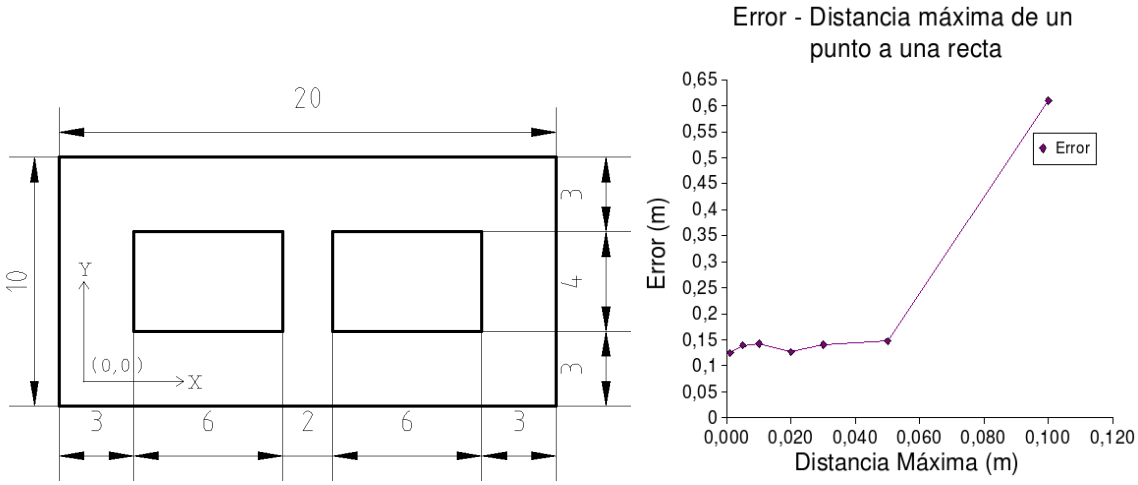


Figura 10.18: A la izquierda mapa de rejilla y recorrido para $D_{max} = 2$ cm, X punto inicial, O punto fina. A la derecha, mapa de rejilla basado en la odometría.

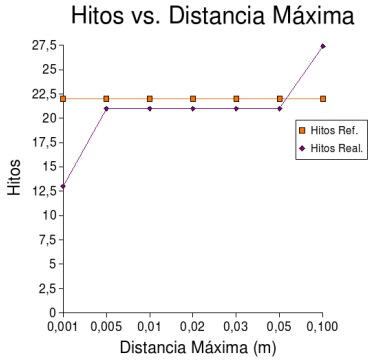
comienza a aumentar. El aumento del error con D_{max} se debe a que pueden producirse falsas detecciones, dado que la exigencia de alineación de los puntos integrantes de una recta se debilita.



D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Error	0,12	0,14	0,14	0,13	0,14	0,15	0,61
σ^2_{error}	0,0015	0,0043	0,0042	0,0020	0,0024	0,0016	0,0017

Figura 10.19: Mapa real, a la izquierda, y evolución del error frente a la distancia máxima de un punto a la recta de la que forma parte, a la derecha.

Los resultados en cuanto al número de hitos detectados se muestran en la figura 10.20. La complejidad del entorno y el tipo de recorrido realizado provocan que en nigún caso se detecten todos los hitos. En este caso el rango de valores de D_{max} para el que se detecta el número de hitos más cercano al correcto, para este mapa es 22 (ver figura 10.4) va de 5 mm hasta 5 cm. Así combinando la información del error y de los hitos detectados se concluye que el rango de D_{max} adecuado comprende de 5 mm a 5 cm.



D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Hitos detectados	13,0	21,0	21,0	21,0	21,0	21,0	27,4

Figura 10.20: Número de hitos detectados frente a D_{max} para el mapa *Bigloop2*.

En la figura 10.21 podemos comparar el mapa obtenido mediante FastSLAM con $D_{max} =$ 2 cm, y el mapa obtenido empleando únicamente información odométrica.

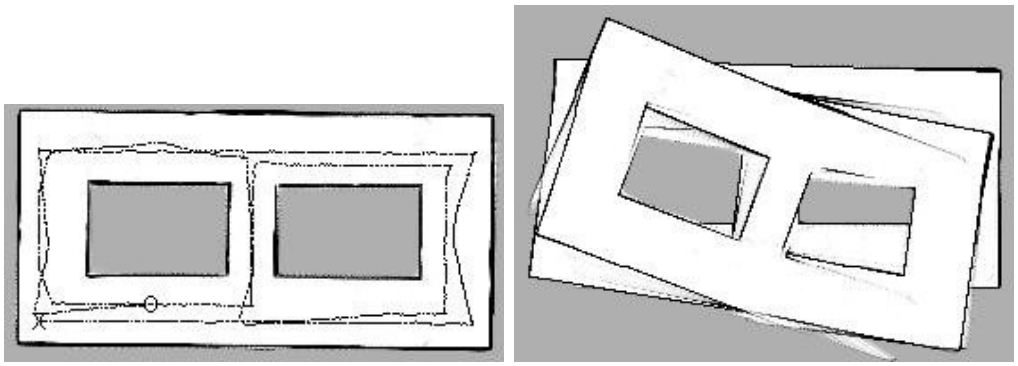
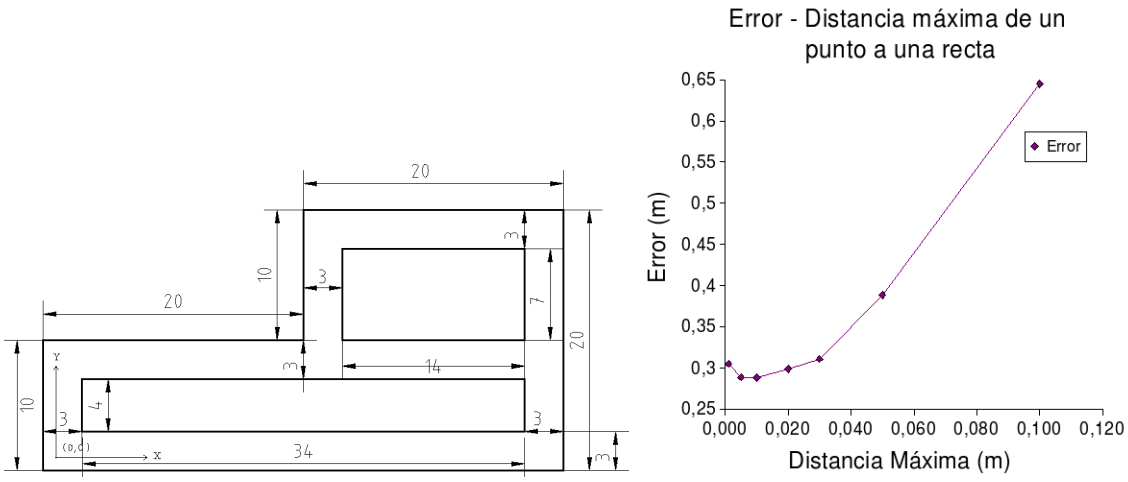


Figura 10.21: A la izquierda mapa de rejilla y recorrido para $D_{max} = 2$ cm, X punto inicial, O punto fina. A la derecha, mapa de rejilla basado en la odometría.

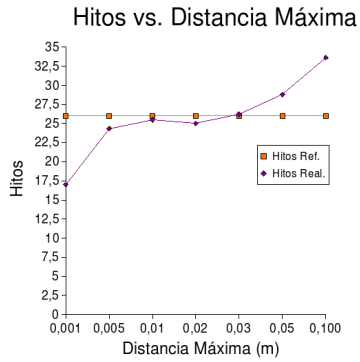
Como último entorno sintético, se pasa a analizar un entorno con bucles internos complejos. Se observa en la figura 10.22 que los mejores resultados se obtienen para valores de D_{max} entre 0 y 3 cm. Más allá el error cometido comienza a aumentar. Mientras que para el resto de mapas sintéticos, los valores mínimos de error se sitúan en torno a 10 o 15 cm, para este mapa el valor asciende a 30 cm. Esto se debe fundamentalmente a la presencia del pasillo inferior, con una longitud de 40 m, que dificulta enormente la localización del robot mientras lo recorre. Si bien en todo momento puede observar las paredes superior e inferior del pasillo, no tiene ninguna referencia para determinar su posición en el eje X. De este modo al llegar al final del pasillo las diferentes partículas habrán acumulado un error imposible de determinar. Este problema se presenta en mayor o menor medida en todos los mapas, pero en este caso su efecto es más acusado.



D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Error	0,31	0,29	0,29	0,3	0,31	0,39	0,64
σ^2_{error}	0,0006	0,0020	0,0049	0,0037	0,0112	0,0210	0,0138

Figura 10.22: Mapa real, a la izquierda, y evolución del error frente a la distancia máxima de un punto a la recta de la que forma parte, a la derecha.

Los resultados en cuanto al número de hitos detectados se muestran en la figura 10.23. En este caso el rango de valores de D_{max} para el que se detecta el número correcto de características, que para este mapa asciende a 26 (ver figura 10.1), va de 1 hasta 3 cm. Así combinando la información del error y de los hitos detectados se concluye que el rango de D_{max} adecuado comprende de 1 a 3 cm.



D_{max}	0,001	0,005	0,01	0,02	0,03	0,05	0,1
Hitos detectados	17,0	24,3	25,5	25,0	26,2	28,8	33,6

Figura 10.23: Número de hitos detectados frente a D_{max} para el mapa *Complex*.

En la figura 10.24 podemos comparar el mapa obtenido mediante FastSLAM con $D_{max} =$ 2 cm, y el mapa obtenido empleando únicamente información odométrica.

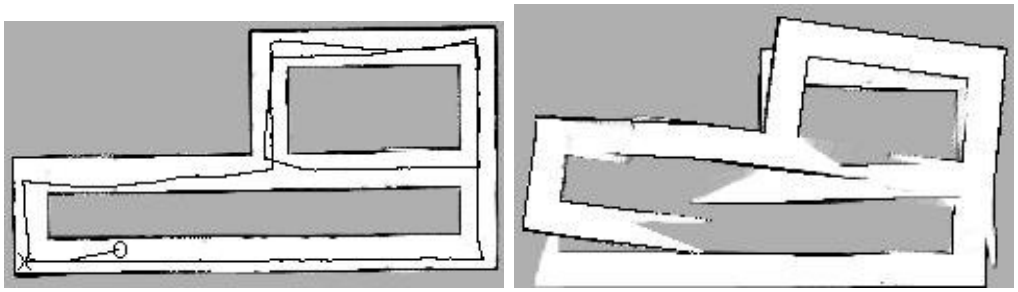


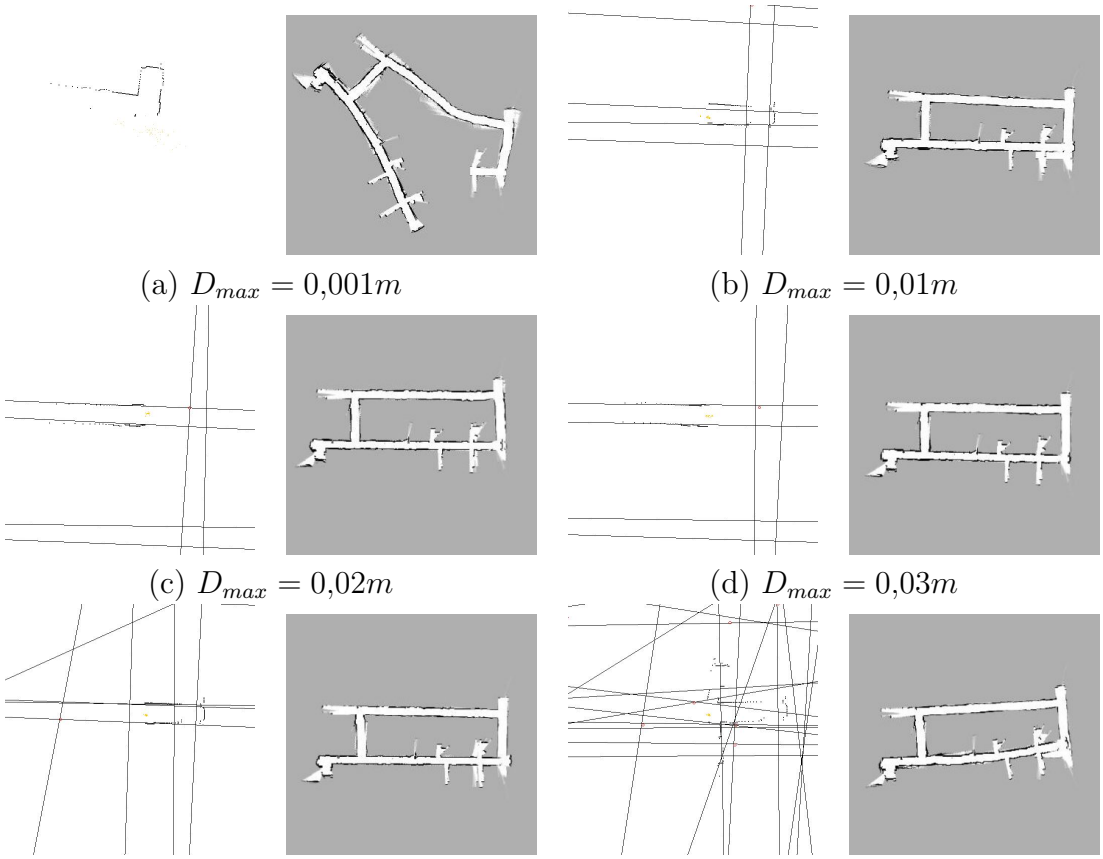
Figura 10.24: A la izquierda mapa de rejilla y recorrido para $D_{max} = 2$ cm, X punto inicial, O punto fina. A la derecha, mapa de rejilla basado en la odometría.

En condiciones reales las rectas detectadas pueden contener error, con lo que valores demasiado pequeños de D_{max} pueden conducir a que no se detecten rectas, ya que los puntos detectados no están tan estrictamente alineados. Para mostrar este efecto realizamos la simulación con el mapa *USC SAL Building*, cuyos datos son fruto de un recorrido real. Los parámetros empleados en esta simulación se muestran en la tabla 10.3.

Filtro			Split & Merge		Láser	Robot
N_{eff}	P_0	Par	P_{min}	L_{min}	Covarianza	Covarianza
∞	0,005	50	15	1,5	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$	$\begin{pmatrix} 0{,}01 & 0 & 0 \\ 0 & 0{,}01 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

Tabla 10.3: Parámetros de las simulaciones para el estudio de D_{max} en el mapa *USC SAL Building*.

Como puede verse en la figura 10.25 (a), un valor de $D_{max} = 0{,}001$ m provoca que no se detecte ninguna característica, con lo que el método es incapaz de trabajar. A medida que se aumenta el valor de D_{max} los mapas obtenidos se ajustan más a la realidad, siendo el mejor mapa el obtenido con $D_{max} = 0{,}03$ m (figura 10.25 (d)). A partir de ahí el aumento de D_{max} provoca la detección de muchos hitos ruidosos que conducen a un mal funcionamiento del algoritmo.



(a) $D_{max} = 0{,}001m$ (b) $D_{max} = 0{,}01m$

(c) $D_{max} = 0{,}02m$ (d) $D_{max} = 0{,}03m$

(e) $D_{max} = 0{,}05m$ (f) $D_{max} = 0{,}1m$

Figura 10.25: Barridos láser y mapas de rejilla obtenidos en simulaciones para distintos valores de D_{max}.

10.4.2. Longitud mínima de una recta (L_{min})

La longitud mínima para considerar una recta válida es un parámetro fundamental del algoritmo de detección de segmentos rectilíneos *Split & Merge*. Una longitud demasiado pequeña provocará la detección de muchas rectas falsas, mientras que una longitud excesivamente grande provocará la detección de pocos hitos referenciales, lo que probablemente conducirá al proceso de SLAM a la obtención de un mapa erróneo por la falta de información sobre la que apoyarse. Por otro lado la longitud mínima de una recta se debe ajustar a las características de las rectas presentes en el entorno.

Otro aspecto a tener en cuenta es el alcance del medidor de rango láser. En la figura 10.26 se muestra un esquema de las rectas detectadas por un robot en un pasillo con una anchura de 12 m, con un medidor de rango láser cuyo alcance es de 8 metros. Las rectas detectadas serían de 5 metros, sin embargo esto es en codiciones ideales, los ruidos presentes en los datos hacen que en realidad las rectas detectadas sean menores, como se verá a lo largo de la presente sección.

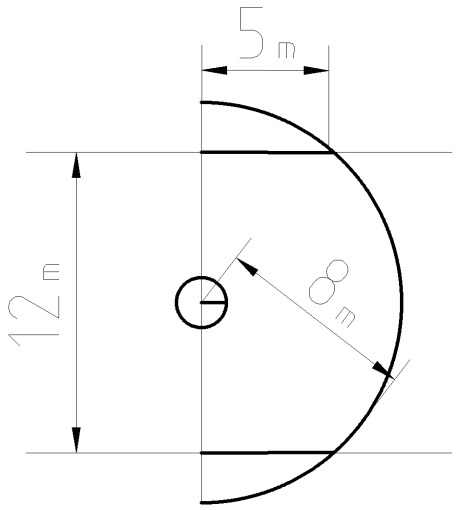


Figura 10.26: Robot en un pasillo, y alcance del medidor de rango láser.

Los parámetros empleados en la simulación se muestran en la tabla 10.4. Destacar de esta configuración que establece que se remuestree siempre, y considera una $D_{max} = 2$ cm, uno de los valores dentro del rango de valores adecuados determinados en la sección anterior.

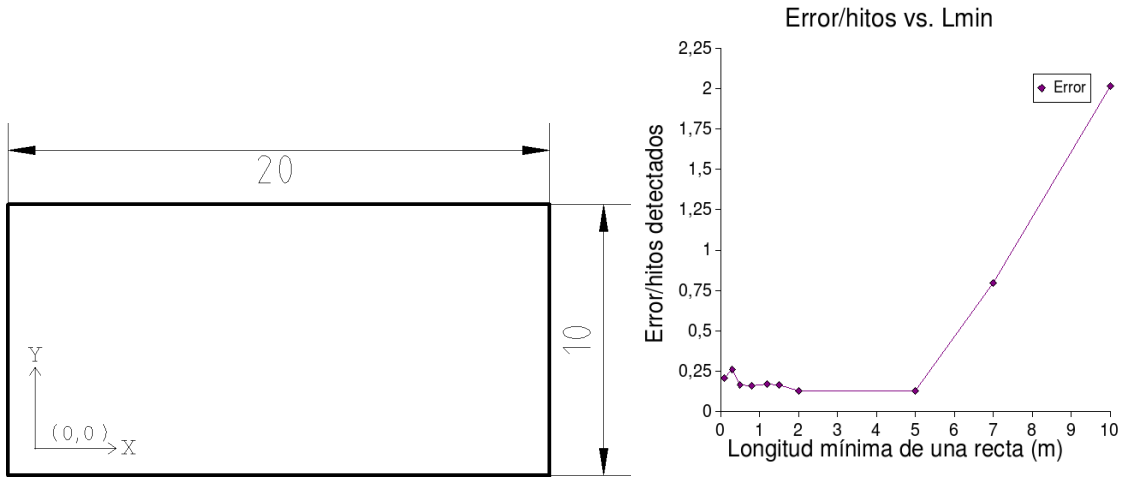
Filtro			Split & Merge		Láser	Robot
N_{eff}	P_0	Par	P_{min}	D_{max}	Covarianza	Covarianza
∞	0,005	100	15	0,02	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$	$\begin{pmatrix} 0{,}01 & 0 & 0 \\ 0 & 0{,}01 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

Tabla 10.4: Parámetros de las simulaciones para el estudio de L_{min} en entornos sintéticos.

En la figura 10.27 puede observarse el mapa de un recinto básico y la evolución del error frente a L_{min}.

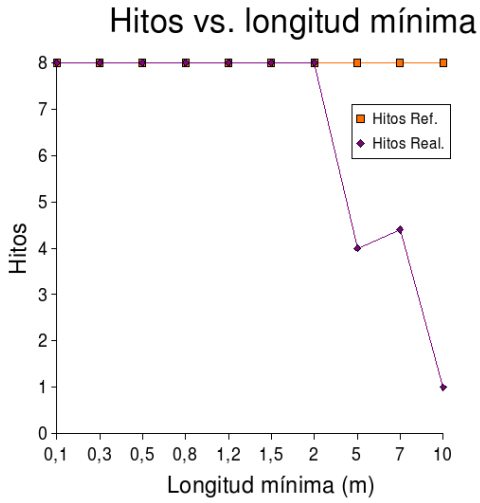
Para valores de L_{min} entre 0,5 y 5 m el error permanece estable. Valores superiores a 5 m provocan un incremento en el error. El aumento del error que se observa por debajo de 0,5 m se debe a la detección de rectas ruidosas que, sin llegar a ser consideradas como nuevas observaciones por el proceso de asociación de datos, distorsionan levemente el mapa obtenido, al incorporarse su información mediante los filtros de Kalman.

Además se muestra en la figura 10.28 la evolución de la cantidad de hitos detectados en función de L_{min}. Como puede observarse a partir de valores de 2 m empieza a fallar la detección de hitos, llegando al caso extremo de sólo detectar 1 hitos cuando L_{min} toma el valor de 10 m. El rango de alcance del sensor láser limita el tamaño de las rectas que pueden ser detectas. Teóricamente un rango de 8 m de alcance del sensor láser permitiría percibir rectas con longitudes próximas a 16 m (el diámetro de la circunferencia que establece el rango máximo). Este es un caso teórico en el que el robot debería aproximarse infinítamente



L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Error	0,2	0,26	0,17	0,16	0,17	0,16	0,13	0,13	0,8	2,02
σ^2_{error}	0,0071	0,0110	0,0045	0,0056	0,0014	0,0025	0,0019	0,0014	0,2044	1,2715

Figura 10.27: Mapa real, a la izquierda, unidades en metros. Evolución del error frente a L_{min}, a la derecha.



L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Hitos	8	8	8	8	8	8	8	4	4,4	1

Figura 10.28: Número de hitos detectados frente a L_{min} para el mapa *Recinto básico*.

al hito, observándolo en una dirección perpendicular. En la realidad, tal como se desprende de los resultados, lo habitual es observar los hitos desde cierta distancia, de modo que en la

mayoria de los casos se observan fragmentos no superiores a 2 m.

Combinando la información sobre error y sobre número de hitos detectados se obtiene que el rango adecuado para L_{min}, se encuentra entre 0,5 y 2 m.

Finalmente en la figura 10.29 podemos comparar el mapa obtenido mediante FastSLAM con $L_{min} = 1$ m, y el mapa obtenido empleando únicamente información odométrica.

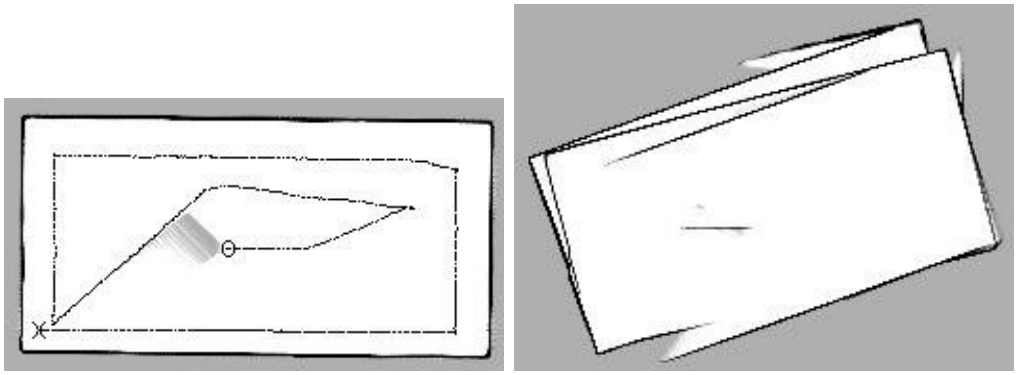
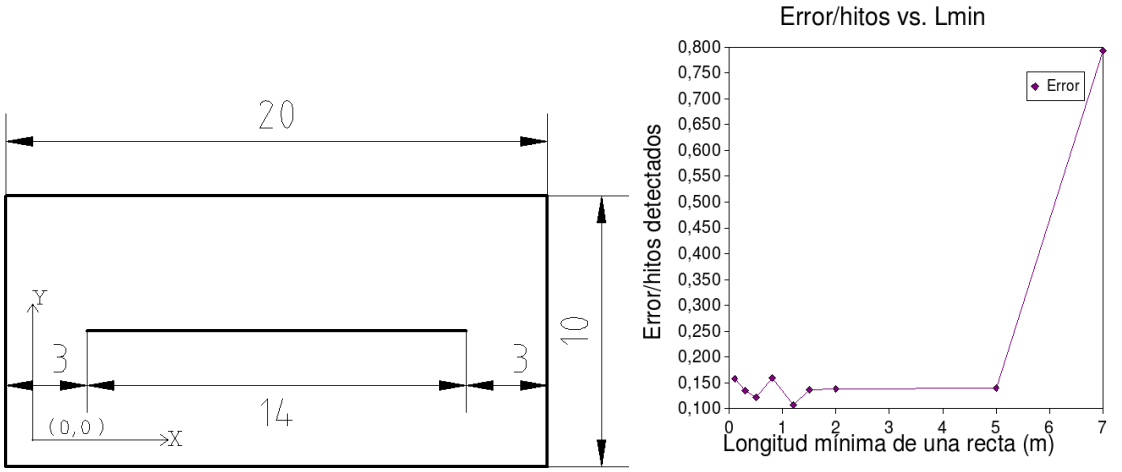


Figura 10.29: A la izquierda mapa de rejilla y recorrido para $L_{min} = 1$ m, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

En la figura 10.30 puede observarse el mapa de un recinto básico con una pared central, denominado *Lineamedia*. Además en la misma figura se muestra la evolución del error frente



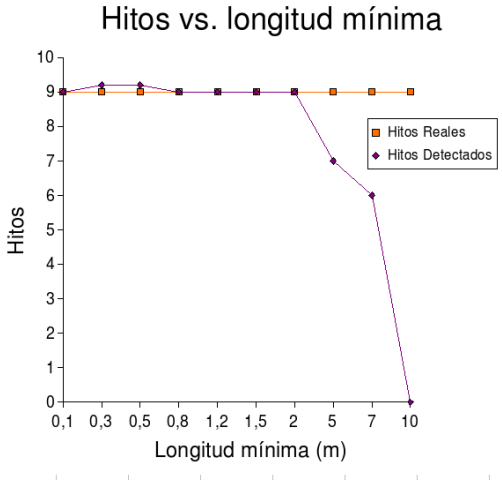
L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Error	0,158	0,136	0,121	0,160	0,106	0,136	0,138	0,139	0,793	∞
σ^2_{error}	0,0072	0,0033	0,0005	0,0029	0,0019	0,0018	0,0005	0,0023	0,0727	0

Figura 10.30: Mapa real, a la izquierda, unidades en metros. Evolución del error frente a L_{min}, a la derecha.

a L_{min}.

En este caso valores de 0 a 5 m producen bueno resultados, si bien los menores errores se observan para valores entre 0,8 y 1,2 m. Valores superiores a 7 m producen un error *infinito*, producto del hecho de no detectar ningún hito referencial.

Como complemento se muestra en la figura 10.31 la evolución de la cantidad de hitos detectados en función de L_{min}. Como puede observarse, para valores superiores a 2 m, falla



L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Hitos	9	9,2	9,2	9	9	9	9	7	6	0

Figura 10.31: Número de hitos detectados frente a L_{min} para el mapa *Lineamedia*.

la detección de hitos, llegando al caso extremo de no detectar ningún hito cuando L_{min} toma el valor de 10 m.

Combinando la información sobre error y sobre número de hitos detectados se obtiene que el rango adecuado para L_{min}, se encuentra entre 0 y 2 m. Finalmente en la figura 10.32

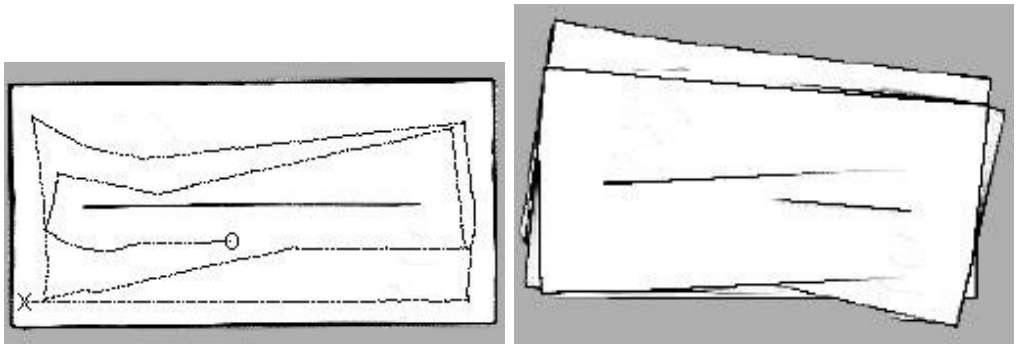
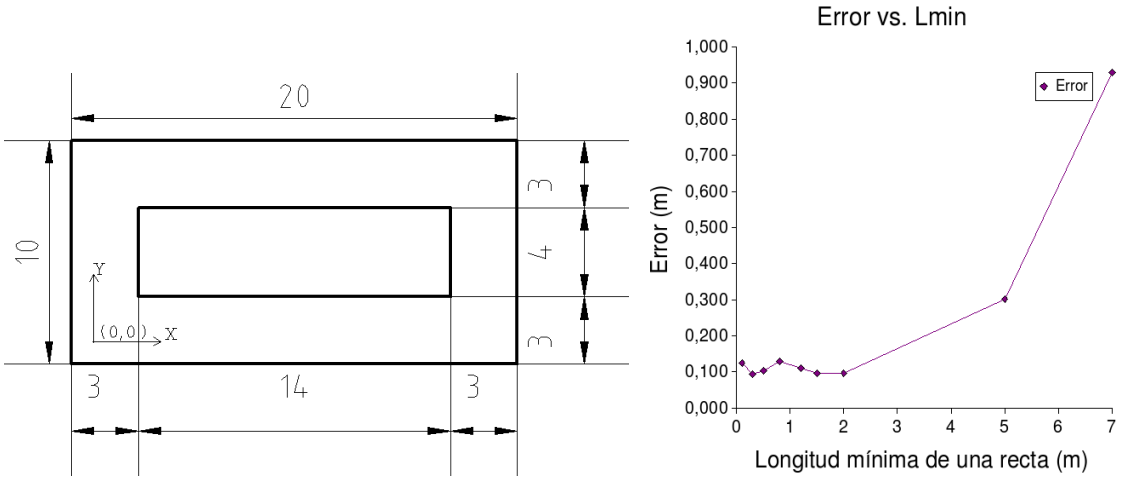


Figura 10.32: A la izquierda mapa de rejilla y recorrido para $L_{min} = 1$ m, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

podemos comparar el mapa obtenido mediante FastSLAM con $L_{min} = 1$ m, y el mapa obtenido empleando únicamente información odométrica.

En la figura 10.33 se muestra el mapa de un recinto con bucle de grandes dimensiones, denominado *Bigloop* y la evolución del error frente a L_{min}.



L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Error	0,125	0,093	0,104	0,130	0,111	0,097	0,097	0,301	0,929	∞
σ^2_{error}	0,0012	0,0003	0,0002	0,0013	0,0008	0,0025	0,0010	0,0135	0,1054	0

Figura 10.33: Mapa real, a la izquierda, unidades en metros. Evolución del error frente a L_{min}, a la derecha.

En la gráfica de evolución del error se distinguen básicamente dos zonas. Una zona de estabilidad, hasta 2 m, y una zona de crecimiento del error para valores por encima de 2 m.

En figura 10.34 puede consultarse la evolución de la cantidad de hitos detectados en función de L_{min}. Como puede observarse, a partir de valores superiores a 2 m falla la detección de hitos, llegando al caso extremo de no detectar ningún hito cuando L_{min} toma el valor de 10 m.

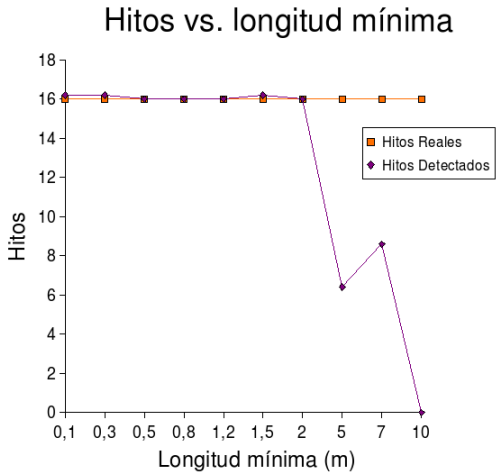
Combinando la información sobre error y sobre número de hitos detectados se obtiene que el rango adecuado para L_{min}, se encuentra entre 0 y 2 m.

Finalmente en la figura 10.35 podemos comparar el mapa obtenido mediante FastSLAM con $L_{min} = 1$ m, y el mapa obtenido empleando únicamente información odométrica.

En la figura 10.36 puede observarse el mapa de un recinto con dos bucles de tamaño medio, denominado *Bigloop2*. Además en la misma figura se muestra la evolución del error frente a L_{min}.

En la gráfica de evolución del error se distinguen básicamente dos zonas. Una zona de estabilidad, hasta 2 m, y una zona de crecimiento del error para valores por encima de 2 m.

Además se muestra en la figura 10.37 la evolución de la cantidad de hitos detectados en función de L_{min}. Como puede observarse, a partir de valores superiores de 2 m falla la



L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Hitos	16,2	16,2	16	16	16	16,2	16	6,4	8,6	0

Figura 10.34: Número de hitos detectados frente a L_{min} para el mapa *Bigloop*.

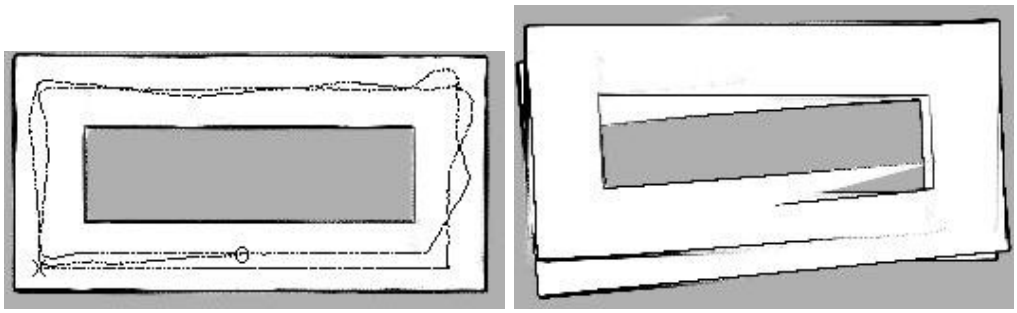


Figura 10.35: A la izquierda mapa de rejilla y recorrido para $L_{min} = 1$ m, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

detección de hitos, llegando al caso extremo de detectar alrededor de 1 hito cuando L_{min} toma el valor de 10 m.

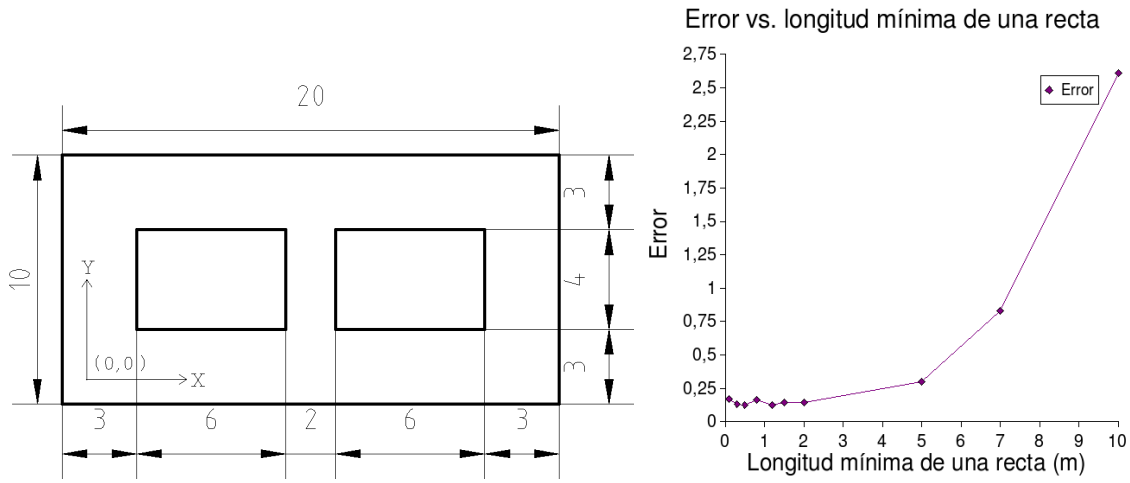
Combinando la información sobre error y sobre número de hitos detectados se obtiene que el rango adecuado para L_{min}, se encuentra entre 0 y 2 m.

Finalmente en la figura 10.38 podemos comparar el mapa obtenido mediante FastSLAM con $L_{min} = 1$ m, y el mapa obtenido empleando únicamente información odométrica.

En la figura 10.39 puede observarse el mapa de un recinto con bucles complejos de grandes dimensiones, denominado *Complex*. Además en la misma figura se muestra la evolución del error frente a L_{min}.

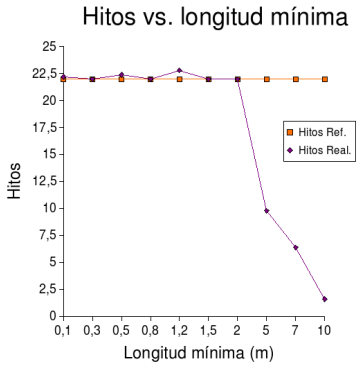
En la gráfica de evolución del error se distinguen básicamente dos zonas. Una zona de estabilidad, hasta 2 m, y una zona de crecimiento del error para valores por encima de 2 m.

Además se muestra en la figura 10.40 la evolución de la cantidad de hitos detectados



L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Error	0,17	0,13	0,13	0,16	0,13	0,14	0,14	0,3	0,83	2,61
σ^2_{error}	0,0040	0,0024	0,0018	0,0044	0,0007	0,0029	0,0034	0,0495	0,1398	0,8465

Figura 10.36: Mapa real, a la izquierda, unidades en metros. Evolución del error frente a L_{min}, a la derecha.



L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Hitos	22,2	22	22,4	22	22,8	22	22	9,8	6,4	1,6

Figura 10.37: Número de hitos detectados frente a L_{min} para el mapa *Bigloop2*.

en función de L_{min}. Como puede observarse, a partir de valores superiores de 2 m falla la detección de hitos, llegando al caso extremo de detectar alrededor de 1 hito cuando L_{min} toma el valor de 10 m.

Finalmente en la figura 10.41 podemos comparar el mapa obtenido mediante FastSLAM con $L_{min} = 1$ m, y el mapa obtenido empleando únicamente información odométrica.

Se observa una tendencia general que divide la evolución del error en tres regiones. En

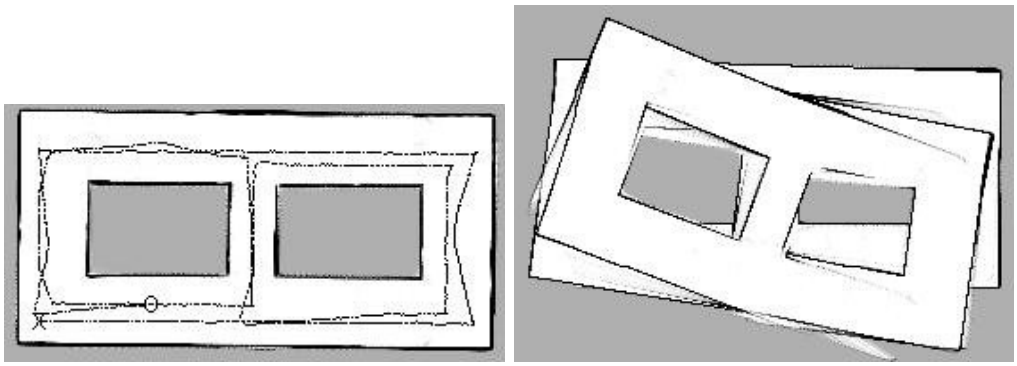
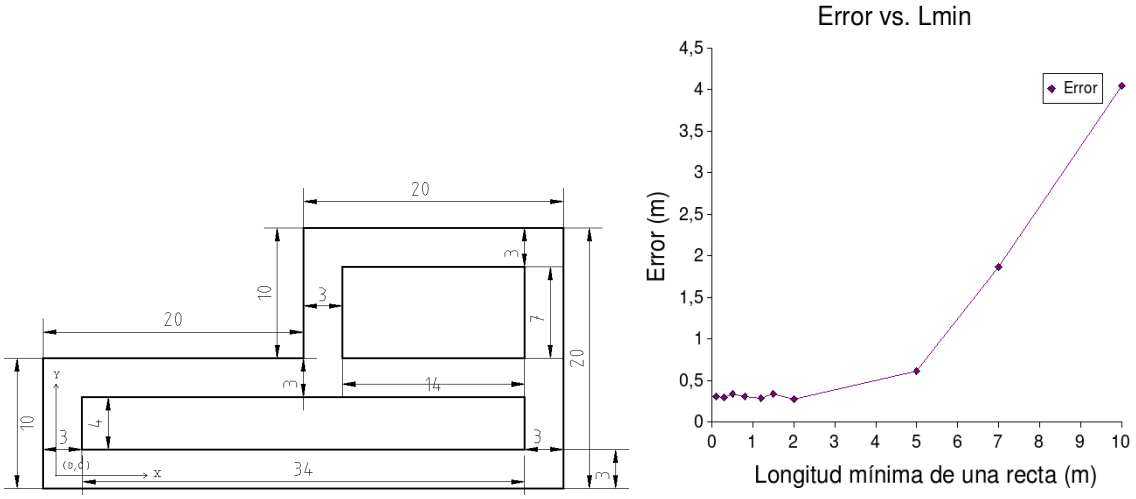


Figura 10.38: A la izquierda mapa de rejilla y recorrido para $L_{min} = 1$ m, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

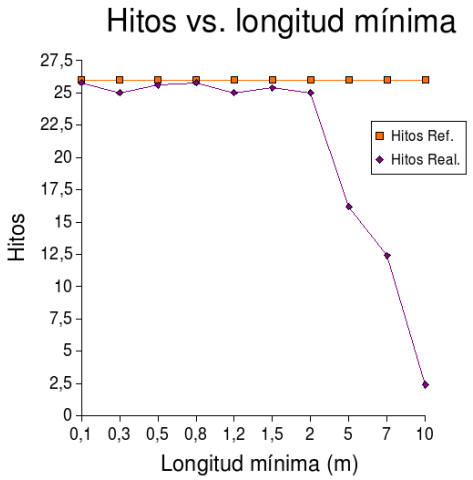


L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Error	0,31	0,3	0,34	0,31	0,29	0,34	0,27	0,61	1,87	4,05
σ^2_{error}	0,0153	0,0074	0,0082	0,0170	0,0113	0,0003	0,0029	0,0649	0,3663	3,0204

Figura 10.39: Mapa real, a la izquierda, unidades en metros. Evolución del error frente a L_{min}, a la derecha.

primer lugar valores de L_{min} inferiores a 0,3 metros dan lugar a un error muy elevado, debido a la detección de rectas falsas. A partir de 2 metros se observa una tendencia hacia el aumento del error, pues cada vez se detectan menos rectas, y el proceso de SLAM tiene menos evidencias para desarrollarse adecuadamente. La franja de valores entre 0,5 y 2 metros produce el error mínimo en todos los mapas.

En principio, estos valores están en consonancia con el tipo de rectas presentes en los entornos, la gran mayoría de ellas de grandes dimensiones. En cualquier caso, valores en



L_{min}	0,1	0,3	0,5	0,8	1,2	1,5	2	5	7	10
Hitos	25,8	25	25,6	25,8	25	25,4	25	16,2	12,4	2,4

Figura 10.40: Número de hitos detectados frente a L_{min} para el mapa *Complex*.

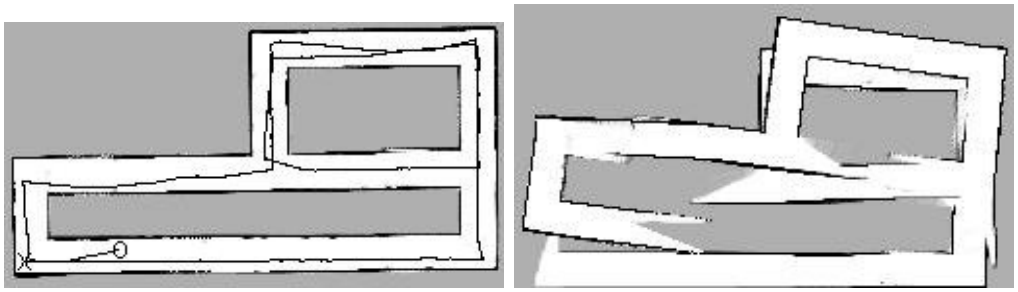


Figura 10.41: A la izquierda mapa de rejilla y recorrido para $L_{min} = 1$ m, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

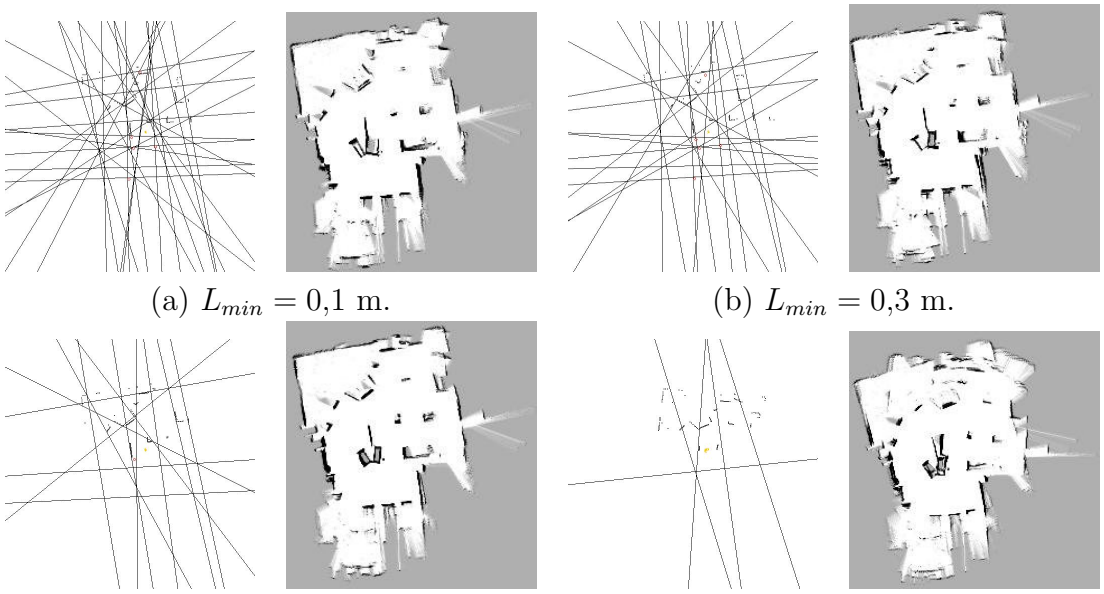
torno a 1 metro son adecuados para todos los mapas y, en general, para entornos de oficina.

Para continuar estudiando el efecto de L_{min} sobre el proceso de SLAM, se estudia ahora el mapa de rejilla obtenido sobre un recorrido real de laboratorio del IUSIANI. La razón de emplear este entorno es, además de para mostrar el efecto sobre un entorno y recorrido reales, el hecho de que en él se encuentran rectas de muy diversos tamaños, con lo que se enriquecen los resultados. Los parámetros empleados en las simulaciones se muestran en la tabla 10.5.

Cuando se fija un valor de L_{min} excesivamente pequeño, como en la figura 10.42 (a) y (b), se produce la detección de una cantidad elevada de rectas, que en su mayoría constituyen ruido. El mapa de la figura 10.42 (c), obtenido con un valor de 0,6 m, es el que refleja más fielmente la realidad, además se observa que la detección de rectas se ha moderado. Por último, si el tamaño de L_{min} es excesivo para la longitud de las rectas presentes en el

Filtro			Split & Merge		Láser	Robot
N_{eff}	P_0	Par	P_{min}	D_{max}	Covarianza	Covarianza
∞	0,005	100	15	0,02	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$	$\begin{pmatrix} 0{,}005 & 0 & 0 \\ 0 & 0{,}005 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

Tabla 10.5: Parámetros de las simulaciones para el estudio de L_{min} en un recorrido real del labororio del IUSIANI.



(a) $L_{min} = 0{,}1$ m. (b) $L_{min} = 0{,}3$ m.

(c) $L_{min} = 0{,}6$ m. (d) $L_{min} = 1$ m.

Figura 10.42: Evolución del error frente al número mínimo de puntos de una recta válida, para un recorrido por el laboratiro del IUSIANI.

entorno, con en la figura 10.42 (d), se detecta un numero insuficiente de hitos, y el algoritmo de SLAM no es capaz de trabajar correctamente.

10.4.3. Mínimo número de puntos de una recta (P_{min})

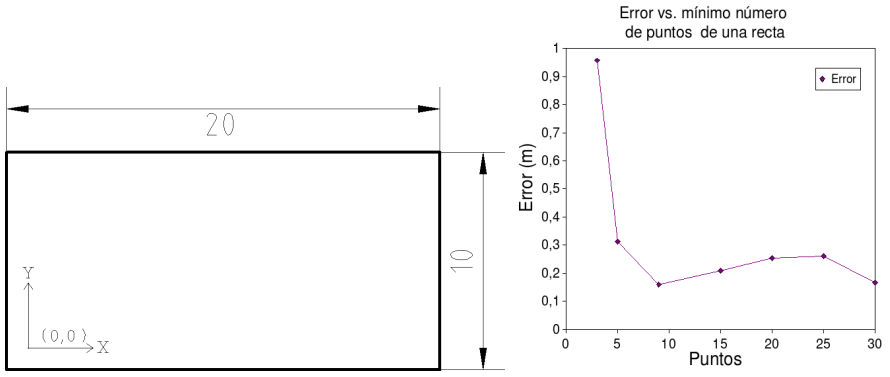
El número mínimo de puntos necesario para considerar una recta válida es un parámetro fundamental del algoritmo de detección de segmentos rectilíneos *Split & Merge*. Un número excesivamente pequeño puede conducir a la detección de rectas falsas, mientras que un valor excesivo provocará la detección de pocas rectas, lo que conducirá al proceso de SLAM a la obtención de un mapa erróneo. Este parámetro puede compensar los efectos de una longitud mínima demasiado pequeña, si se establece un valor suficientemente alto, de modo que aunque las rectas permitidas sean pequeñas en longitud, el obligar a que estén compuestas por un

número considerable de puntos, aumenta la probabilidad de que se trate de rectas reales y no de ruido. Los parámetros empleados en la simulación se muestran en la tabla 10.6.

Filtro			Split & Merge		Láser	Robot
N_{eff}	P_0	Par	L_{min}	D_{max}	Covarianza	Covarianza
∞	0,005	100	0,4	0,02	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$	$\begin{pmatrix} 0{,}01 & 0 & 0 \\ 0 & 0{,}01 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

Tabla 10.6: Parámetros de las simulaciones para el estudio de P_{min} en entornos sintéticos.

En la figura 10.43 puede observarse el mapa de un recinto básico. Además en la misma figura se muestra la evolución del error frente a P_{min}.

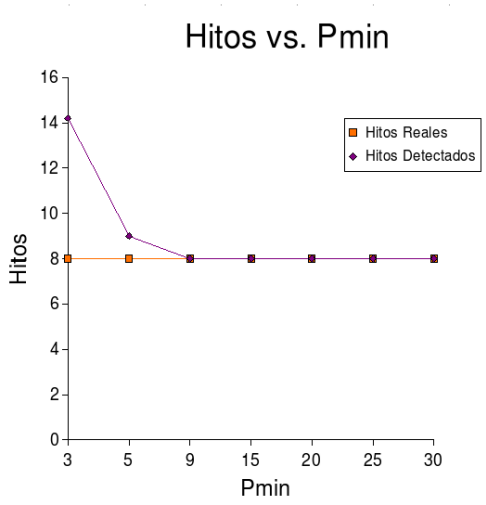


P_{min}	3	5	9	15	20	25	30
Error	0,96	0,31	0,16	0,21	0,25	0,26	0,17
σ^2_{error}	0,0062	0,0023	0,0053	0,0035	0,0092	0,0114	0,0047

Figura 10.43: Mapa real, a la izquierda, unidades en metros. Error frente a P_{min} a la derecha

Como cabe esperar valores bajos de P_{min} conducen a errores muy elevados. A partir de 9 puntos el error se estabiliza en su valor mínimo, por lo que exigir un mayor número de puntos mínimos a las rectas no aporta ningún beneficio.

Sin embargo esta información no es suficiente para determinar el rango de valores adecuados para P_{min}. Conocemos el número de hitos presentes en el mapa real, 4 rectas y 4 esquinas. Distintos valores de P_{min} van a provocar que en ocasiones se detecte un número de características erróneo, ya sea superior o inferior. En la figura 10.44 se observa el número total de hitos detectados frente a P_{min}. Para valores de P_{min} inferiores a 9 puntos el número de hitos detectados es superior al número de hitos reales. De ahí en adelante se detecta el número de hitos correcto, incluso para valores tan elevados como 30. Esto se debe al hecho de ser un entorno sintético con rectas grandes.



P_{min}	3	5	9	15	20	25	30
Hitos	14,2	9	8	8	8	8	8

Figura 10.44: Número de hitos detectados frente a P_{min} para el mapa *Recinto básico*.

Combinando la información sobre error y sobre número de hitos detectados se obtiene que el rango adecuado para P_{min}, se encuentra a partir de 9 puntos.

Finalmente en la figura 10.45 podemos comparar el mapa obtenido mediante FastSLAM con $P_{min} = 15$ puntos, y el mapa obtenido empleando únicamente información odométrica.

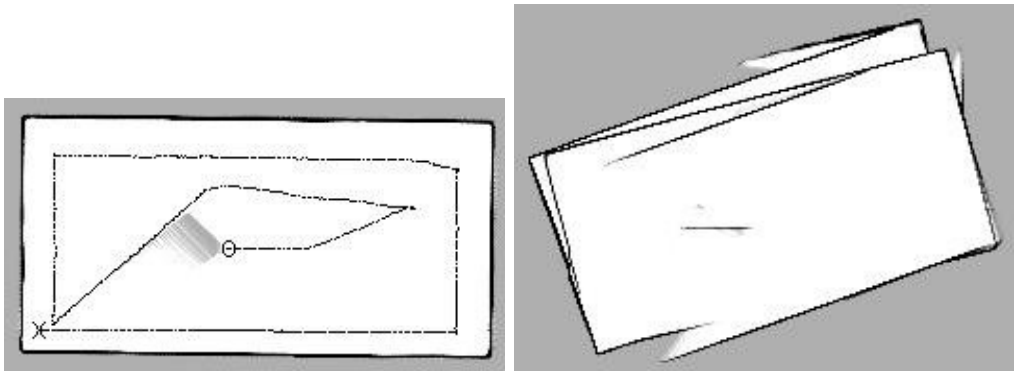
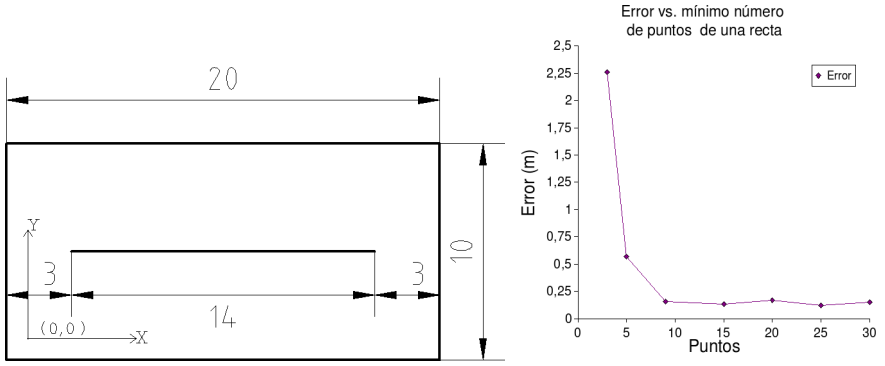


Figura 10.45: A la izquierda mapa de rejilla y recorrido para $P_{min} = 15$ puntos, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

En la figura 10.46 puede observarse el mapa de un recinto básico con una pared central, denominado *Lineamedia*. Además en la misma figura se muestra la evolución del error frente a P_{min}.

Como cabe esperar valores bajos de P_{min} conducen a errores muy elevados. A partir de 9 puntos el error se estabiliza en su valor mínimo, por lo que exigir un mayor número de

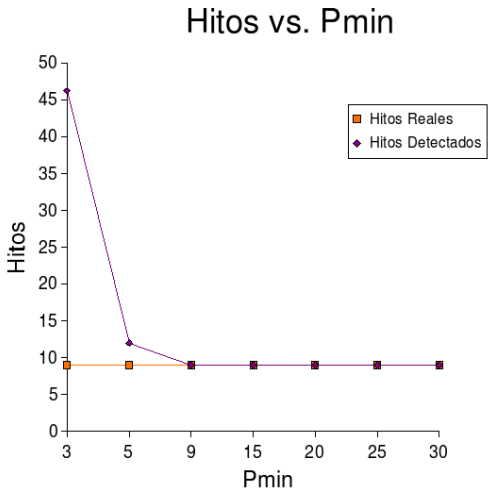


P_{min}	3	5	9	15	20	25	30
Error	2,26	0,57	0,16	0,14	0,17	0,12	0,15
σ^2_{error}	0,0026	0,0074	0,0017	0,0046	0,0053	0,0041	0,0045

Figura 10.46: Mapa real, a la izquierda, unidades en metros. Error frente a P_{min} a la derecha

puntos mínimos a las rectas no aporta ningún beneficio.

Los resultados en cuanto al número de hitos detectados se muestran en la figura 10.47. Valores a partir de 9 puntos producen la detección del número correcto de hitos. Así combinando la información del error y de los hitos detectados se concluye que el rango de P_{min} adecuado se sitúa a partir de 9 puntos.



P_{min}	3	5	9	15	20	25	30
Hitos	46,2	12	9	9	9	9	9

Figura 10.47: Número de hitos detectados frente a P_{min} para el mapa *Lineamedia*.

Finalmente en la figura 10.48 podemos comparar el mapa obtenido mediante FastSLAM

con $P_{min} = 15$ puntos, y el mapa obtenido empleando únicamente información odométrica.

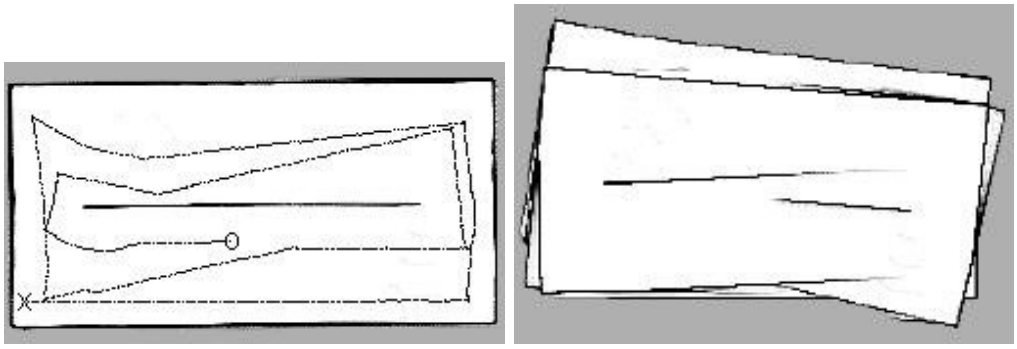
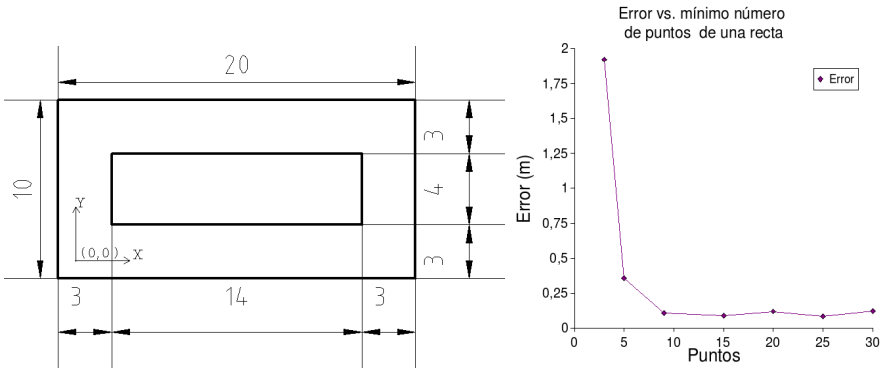


Figura 10.48: A la izquierda mapa de rejilla y recorrido para $P_{min} = 15$ puntos, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

En la figura 10.49 puede observarse el mapa de un recinto con bucle de grandes dimensiones, denominado *Bigloop*. Además en la misma figura se muestra la evolución del error frente a P_{min}.



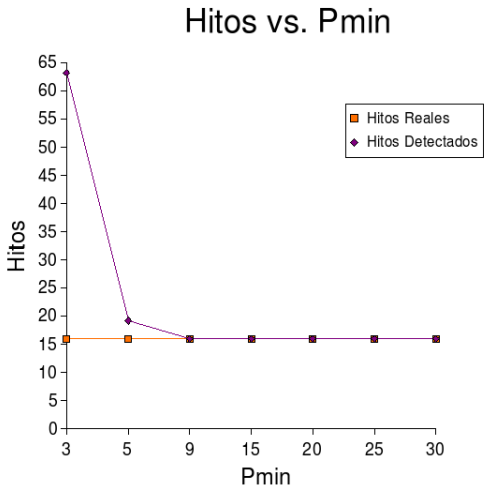
P_{min}	3	5	9	15	20	25	30
Error	1,92	0,36	0,11	0,09	0,12	0,09	0,12
σ^2_{error}	0,0032	0,0037	0,0009	0,0010	0,0012	0,0004	0,0002

Figura 10.49: Mapa real, a la izquierda, unidades en metros. Error frente a P_{min} a la derecha

Como cabe esperar valores bajos de P_{min} conducen a errores muy elevados. A partir de 9 puntos el error se estabiliza en su valor mínimo, por lo que exigir un mayor número de puntos mínimos a las rectas no aporta ningún beneficio. A partir de 25 puntos el error tiende a crecer.

Los resultados en cuanto al número de hitos detectados se muestran en la figura 10.50. Valores a partir de 9 puntos producen la detección del número correcto de hitos. Así com-

binando la información del error y de los hitos detectados se concluye que el rango de P_{min} adecuado se sitúa entre 9 y 25 puntos.



P_{min}	3	5	9	15	20	25	30
Hitos	63,2	19,2	16	16	16	16	16

Figura 10.50: Número de hitos detectados frente a P_{min} para el mapa *Bigloop*.

Finalmente en la figura 10.51 podemos comparar el mapa obtenido mediante FastSLAM con $P_{min} = 15$ puntos, y el mapa obtenido empleando únicamente información odométrica.

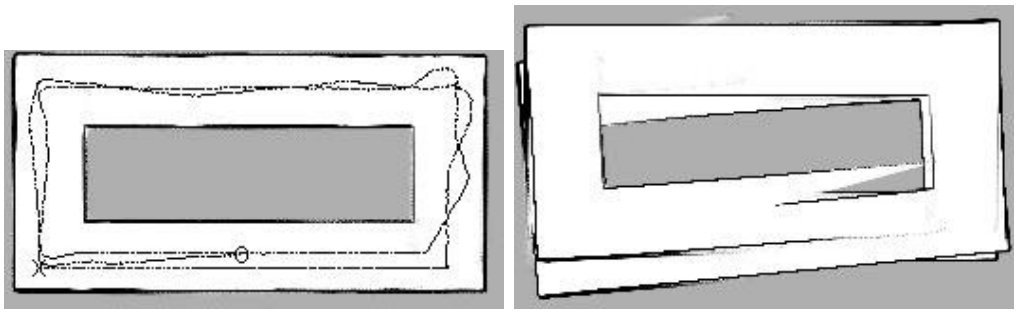
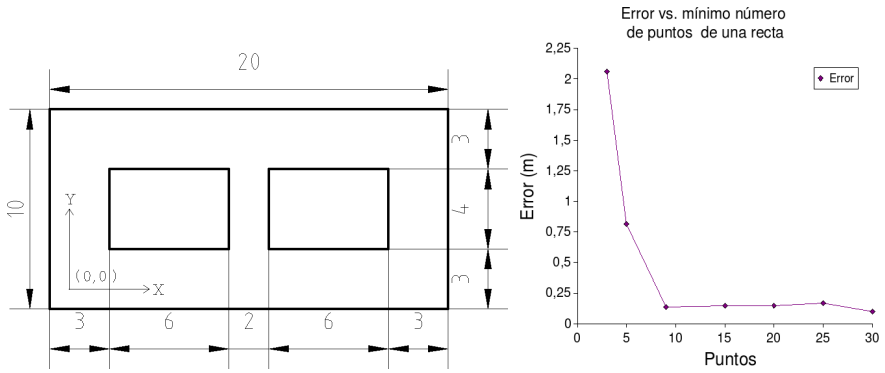


Figura 10.51: A la izquierda mapa de rejilla y recorrido para $P_{min} = 15$ puntos, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

En la figura 10.52 puede observarse el mapa de un recinto con dos bucles de tamaño medio, denominado *Bigloop2*. Además en la misma figura se muestra la evolución del error frente a P_{min}.

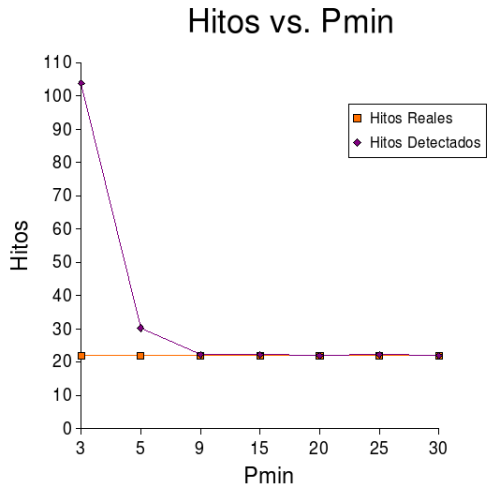
Como cabe esperar valores bajos de P_{min} conducen a errores muy elevados. A partir de 9 puntos el error se estabiliza en su valor mínimo, por lo que exigir un mayor número de puntos mínimos a las rectas no aporta ningún beneficio.



P_{min}	3	5	9	15	20	25	30
Error	2,06	0,81	0,14	0,15	0,15	0,17	0,1
σ^2_{error}	0,0026	0,0042	0,0004	0,0043	0,0036	0,0031	0,0010

Figura 10.52: Mapa real, a la izquierda, unidades en metros. Error frente a P_{min} a la derecha

Los resultados en cuanto al número de hitos detectados se muestran en la figura 10.53. Valores a partir de 9 puntos producen la detección del número correcto de hitos. Así combinando la información del error y de los hitos detectados se concluye que el rango de P_{min} adecuado se sitúa a partir de 9 puntos.



P_{min}	3	5	9	15	20	25	30
Hitos	103,8	30,2	22,2	22,2	22	22,2	22

Figura 10.53: Número de hitos detectados frente a P_{min} para el mapa *Bigloop2*.

Finalmente en la figura 10.54 podemos comparar el mapa obtenido mediante FastSLAM con $P_{min} = 15$ puntos, y el mapa obtenido empleando únicamente información odométrica.

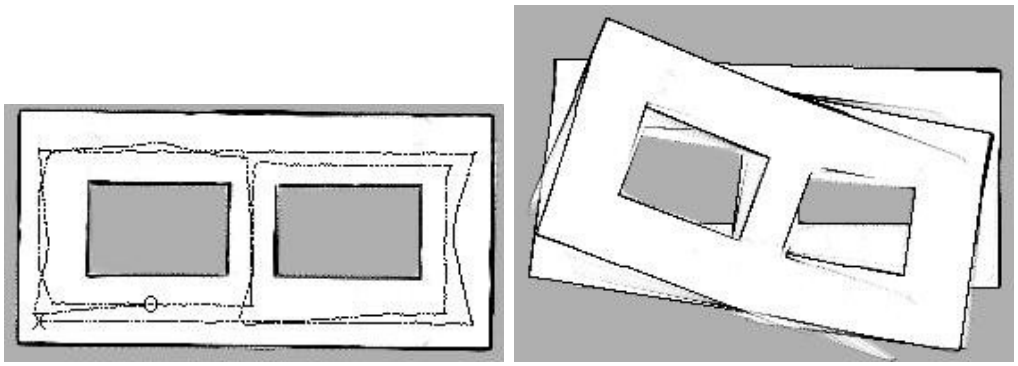
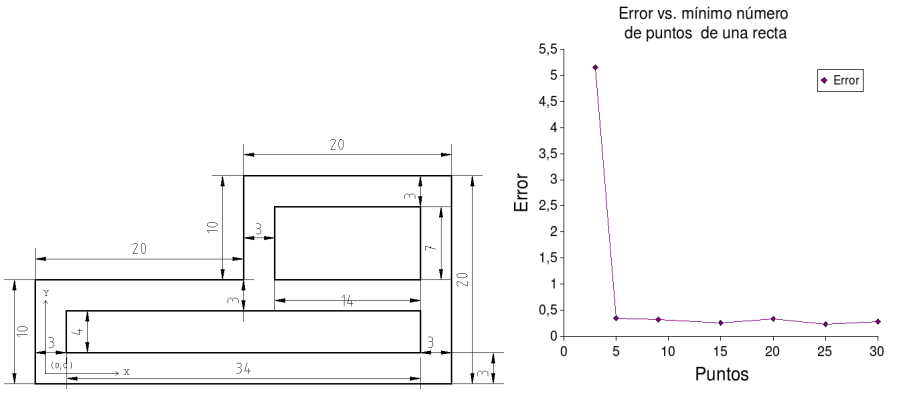


Figura 10.54: A la izquierda mapa de rejilla y recorrido para $P_{min} = 15$ puntos, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

En la figura 10.55 puede observarse el mapa de un recinto con bucles complejos de grandes dimensiones, denominado *Complex*. Además en la misma figura se muestra la evolución del error frente a P_{min}.

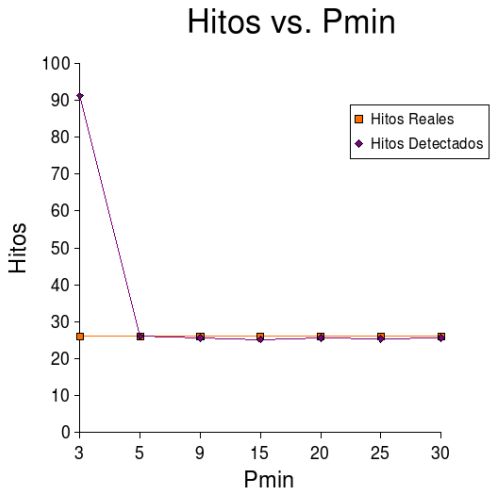


P_{min}	3	5	9	15	20	25	30
Error	5,16	0,34	0,32	0,26	0,33	0,23	0,29
σ^2_{error}	0,0232	0,0039	0,0270	0,0018	0,0066	0,0009	0,0204

Figura 10.55: Mapa real, a la izquierda, unidades en metros. Error frente a P_{min} a la derecha

Como cabe esperar valores bajos de P_{min} conducen a errores muy elevados. A partir de 9 puntos el error se estabiliza en su valor mínimo, por lo que exigir un mayor número de puntos mínimos a las rectas no aporta ningún beneficio.

Los resultados en cuanto al número de hitos detectados se muestran en la figura 10.56. Valores entre 9 y 20 puntos producen la detección del número correcto de hitos. Así combinando la información del error y de los hitos detectados se concluye que el rango de P_{min} adecuado se sitúa entre 9 y 20 puntos.



P_{min}	3	5	9	15	20	25	30
Hitos	91,2	26	25,6	25,2	25,6	25,4	25,6

Figura 10.56: Número de hitos detectados frente a P_{min} para el mapa *Complex*.

Finalmente en la figura 10.57 podemos comparar el mapa obtenido mediante FastSLAM con $P_{min} = 15$ puntos, y el mapa obtenido empleando únicamente información odométrica.

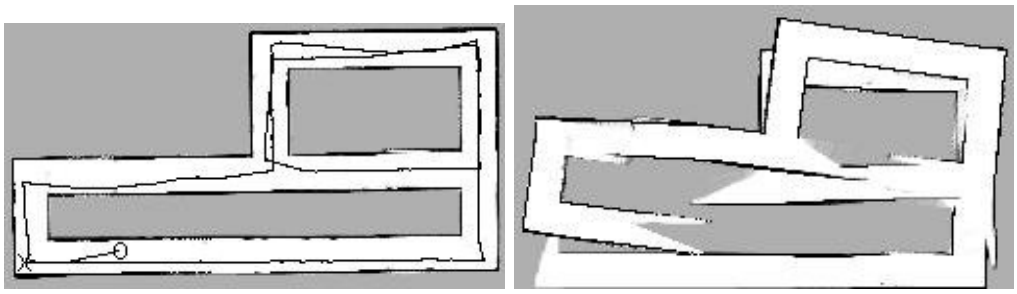


Figura 10.57: A la izquierda mapa de rejilla y recorrido para $P_{min} = 15$ puntos, X punto inicial, O punto final. A la derecha, mapa de rejilla basado en la odometría.

En todos los casos se observa que valores por encima de 9 puntos producen el menor error. Al ser estos entornos de origen sintético, las rectas que contienen están compuestas por una gran cantidad de puntos, por lo que valores tan grandes como $P_{min} = 30$ sigue obteniendo un error bajo.

Para ilustrar el comportamiento del algoritmo en entornos reales se muestra en la figura 10.58 un detalle de las rectas detectadas, así como el mapa de rejilla obtenido, en un recorrido real del laboratorio del IUSIANI. Los detalles sobre la obtención del mapa se describen en la sección 10.2. Los parámetros empleados en la simulación se muestran en la tabla 10.7.

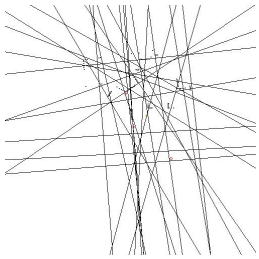
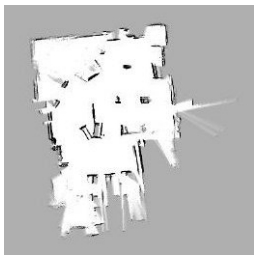
Como se observa en la figura 10.58 (a), un número de puntos bajo conduce a la detección

Filtro			Split & Merge		Láser	Robot
N_{eff}	P_0	Par	L_{min}	D_{max}	Covarianza	Covarianza
∞	0,005	50	0,6	0,02	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$	$\begin{pmatrix} 0{,}005 & 0 & 0 \\ 0 & 0{,}005 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

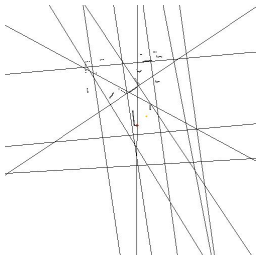
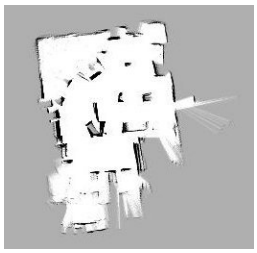
c

Tabla 10.7: Parámetros de las simulaciones para el estudio de P_{min} un recorrido real del laboratorio del IUSIANI.

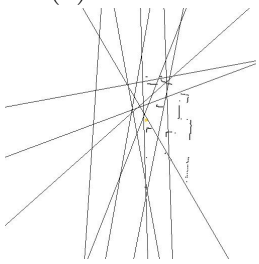
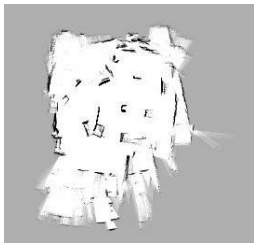
de infinidad de rectas. A media que se aumenta el valor de P_{min}, las rectas detectadas se corresponden más con la realidad como puede observarse en la figura 10.58 (b), hasta llegar a valores excesivamente altos que provocan una detección de rectas muy escasa, y por lo tanto una construcción errónea del mapa, tal como puede verse en la figura 10.58 (c).

(a) Mínimo número de puntos 3.

(b) Mínimo número de puntos 17.

(c) Mínimo número de puntos 40.

Figura 10.58: Mapas de rejilla e hitos detectados para distinto número mínimo de puntos de una recta válida, en un recorrido por el laboratiro del IUSIANI.

10.4.4. Umbrales máximos de colinealidad (ρ_{max}, θ_{max})

Para estudiar el efecto de los umbrales máximos de colinealidad sobre la detección de hitos y el proceso de SLAM, se debe emplear un entorno en el que las rectas no sean perfectas, de modo que de hecho tenga lugar el proceso de fusión de segmentos. Los entornos sintéticos que hemos diseñado carecen de estas características por lo que dentro de un barrido láser los segmentos se detectan completos y no tiene lugar la fusión de segmentos colineales.

No obstante, para mostrar realmente que estos parámetros no tienen influencia sobre los entornos sintéticos se han realizado una serie de simulaciones sobre el mapa *Bigloop2* (ver figura 10.59). Los parámetros empleados en todas las simulaciones se muestran en la tabla 10.8.

Filtro			Split & Merge			Robot	Láser
N_{eff}	Par	$P0$	P_{min}	D_{max}	L_{min}	Covarianza	Covarianza
∞	100	0.005	9	0.02	0.4	$\begin{pmatrix} 0{,}007 & 0 & 0 \\ 0 & 0{,}007 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$

Tabla 10.8: Parámetros de la simulación para el estudio de $(\rho_{max}, \theta_{max})$ en el mapa *Bigloop2*.

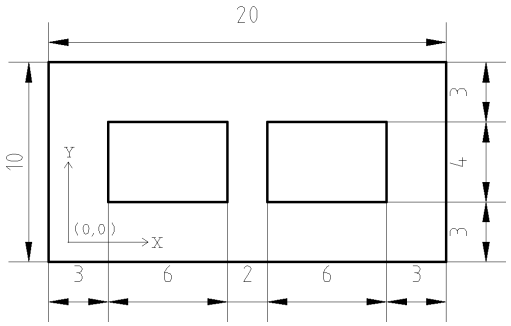
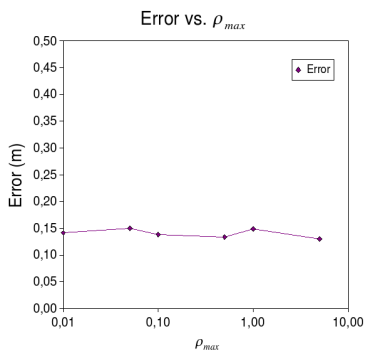


Figura 10.59: Mapa *Bigloop2*

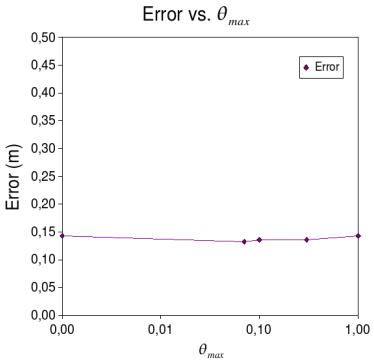
Tanto los errores como el número de hitos detectados se muestran en la figura 10.60. en las figuras (a) y (c) se muestra el resultado de la variación del umbral de colinealidad ρ_{max}, manteniendo $\theta_{max} = 0{,}07$ radianes. Por otro, ne las figuras (b) y (d) se muestra el resultado de la variación de θ_{max}, manteniendo $\rho_{max} = 0{,}1$ m. Como puede observarse, ni el error ni el número de hitos detectados dependen de los umbrales de colinealidad, siempre y cuando se mantengan en límites razonables

Por esto para estudiar estos parámetros vamos a emplear el mapa *USC SAL Building*, cuyos datos son fruto de un recorrido real. Los parámetros empleados en todas las simulaciones se muestran en la tabla 10.9.

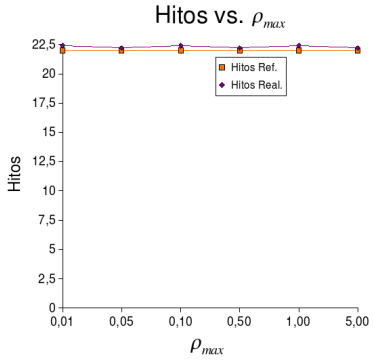
En primer lugar se realizan una serie de simulaciones manteniendo un valor de $\theta_{max} = 0{,}07$ radianes, aproximadamente 4 grados, y se varía ρ_{max}, que toma los valores [0.001, 0.01, 0.03, 0.05, 0.5, 1]. No se muestran los mapas de rejilla ya que todos ellos han sido correctos. Sin embargo, el número de hitos detectados varía considerablemente, tal como muestra la figura 10.61. En dicha figura se incluyen por un lado las rectas presentes en el mapa y por otro las esquinas. Las rectas presentes en el mapa final se mantienen prácticamente constantes. Esto se debe a que el algoritmo de asociación de datos realiza lo que podríamos llamar una fusión



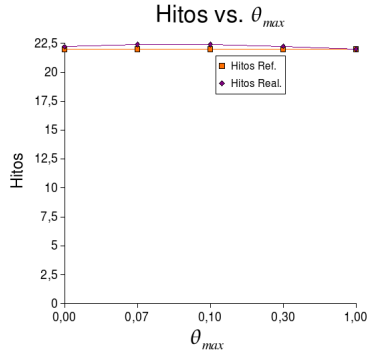
(a) Error frente a distintos valores de ρ_{max}. $\theta_{max} = 0{,}07$ radianes.



(b) Error frente a distintos valores de θ_{max}. $\rho_{max} = 0{,}1$ m.



(c) Hitos frente a distintos valores de ρ_{max}. $\theta_{max} = 0{,}07$ radianes.



(d) Hitos frente a distintos valores de θ_{max}. $\rho_{max} = 0{,}1$ m.

Figura 10.60: Error e hitos detectados para distintos valores de los umbrales de colinealidad, para el mapa *Bigloop2*.

de datos entre las observaciones y el mapa, de modo que todas aquellas rectas muy similares que se han detectado en el barrido láser terminan siendo asociadas con el mismo hito en el mapa, y por ello no hay una proliferación de rectas en el mismo.

En cuanto a las esquinas, mostradas en la figura 10.61, se aprecia una detección excesiva para valores de ρ_{max} de 1 mm, 1 cm y 3 cm, a partir de 5 cm el número de esquinas detectadas se estabiliza. Consideramos que un valor de 10 cm es adecuado. Permitir que se fusionen rectas que están separadas por más de 10 cm puede conducir a errores, aunque en estas simulaciones no se detecte este efecto. En cuanto a las esquinas detectadas el algoritmo de asociación de datos no tiene capacidad de actuación, ya que las esquinas se calculan con las rectas detectadas en el barrido láser, no con las incorporadas al mapa.

Filtro			Split & Merge			Robot	Láser
N_{eff}	Par	$P0$	P_{min}	D_{max}	L_{min}	Covarianza	Covarianza
∞	50	0.005	15	0.02	0.6	$\begin{pmatrix} 0{,}01 & 0 & 0 \\ 0 & 0{,}01 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$

Tabla 10.9: Parámetros de la simulación para el estudio de $(\rho_{max}, \theta_{max})$ en el mapa *USC SAL Building*.

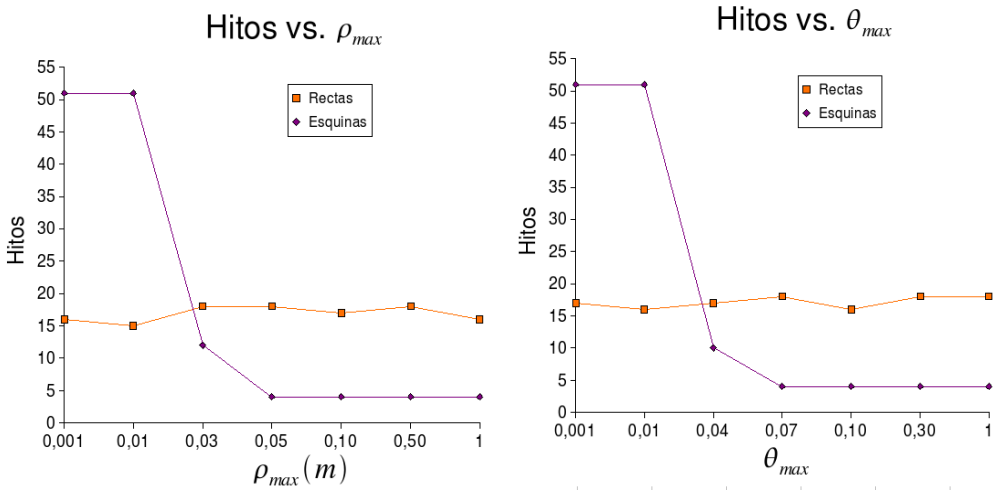


Figura 10.61: Rectas y esquinas detectadas para distintos valores de ρ_{max}, manteniendo $\theta_{max} = 0{,}07$ radianes, a la izquierda, y para distintos valores de θ_{max}, manteniendo $\rho_{max} = 10$ cm, a la derecha.

10.4.5. Estudio de la influencia de los parámetros de la detección de esquinas

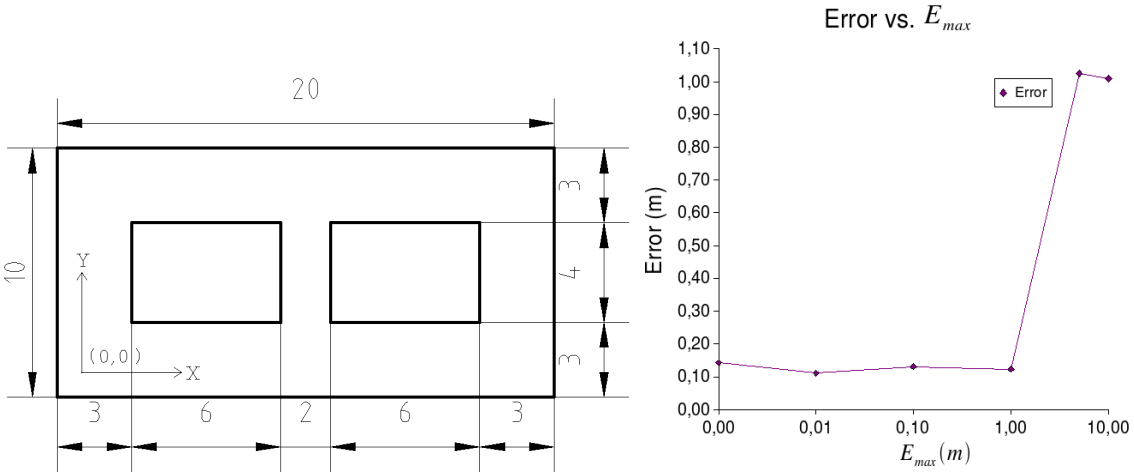
Las esquinas son un hito referencial que viene a reforzar la información proporcionada por los hitos principales en nuestra implementación que son las rectas. Es evidente que sin una correcta detección de rectas no es posible una correcta detección de esquinas. El algoritmo es capaz de funcionar sin la detección de esquinas; ahora bien, allí dónde es posible incluir la información proporcionada por estos hitos el proceso de SLAM cuenta con más evidencias y es más robusto. El umbral de distancia máxima de una esquina a cada una de las rectas cuya intersección la origina, que denominamos E_{max}, se emplea para discernir las denominadas intersecciones o esquinas reales de las ficticias (ver sección 4.3). Valores pequeños de E_{max} imposibilitarán la detección de algunas de las esquinas reales, al ser muy restrictivas las condiciones, lo que no conlleva necesariamente un aumento del error, ya que no se detecta ninguna esquina falsa. Por otro lado, valores excesivamente grandes provocarán la detección de esquinas falsas como esquinas reales, con lo que error crecerá considerablemente. Para

confirmar los efectos comentados es apto cualquier mapa en el que se detecten esquinas, aunque es preferible un mapa en el que existan rectas que no sean paralelas o formen ángulos rectos. De todos modos el propio error en el proceso de SLAM conduce a que se den estas condiciones. Por ello empleamos el mapa sintético *Bigloop2*. Los parámetros empleados en todas las simulaciones se muestran en la figura 10.10.

Filtro			Split & Merge			Robot	Láser
N_{eff}	Par	$P0$	P_{min}	D_{max}	L_{min}	Covarianza	Covarianza
∞	100	0.005	9	0.02	0.4	$\begin{pmatrix} 0{,}007 & 0 & 0 \\ 0 & 0{,}007 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$

Tabla 10.10: Parámetros de las simulaciones para el estudio de E_{max} en el mapa *Bigloop2*.

En la figura 10.62 se muestra el mapa *Bigloop2* y la evolución del error frente a distintos valores de E_{max}. Los resultados obtenidos confirman los efectos comentados anteriormente. Valores bajos del umbral no tienen una influencia sobre el error, al no introducir hitos ruidosos en el mapa, y al no ser las esquinas unos hitos fundamentales para el proceso. Por otro lado valores elevados del umbral conducen a un incremento del error, ya que se incluyen en el mapa hitos que no tienen una correspondencia en la realidad.

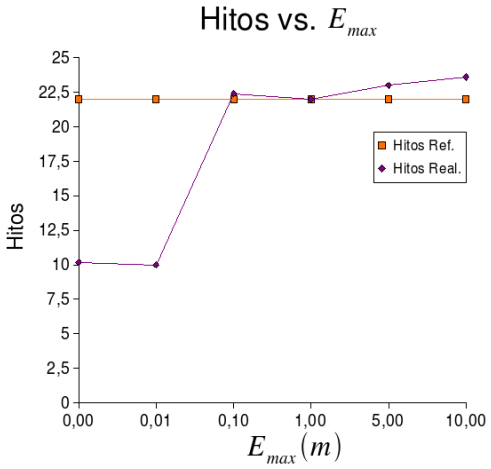


E_{max}	0,001	0,01	0,10	1,00	5,00	10,00
Error	0,14	0,11	0,13	0,12	1,03	1,01
σ^2_{error}	0,0007	0,0001	0,0002	0,0001	0,0003	0,0007

Figura 10.62: Mapa real, a la izquierda, y evolución del error frente a la distancia máxima de un punto a la recta de la que forma parte, a la derecha.

Por otro lado la evolución del número de hitos detectados, mostrada en la figura 10.63 también confirma los resultados esperados. Cuando E_{max} es pequeño dejan de detectarse es-

quinas presentes en la realidad, mientras que para valores altos, se detectan esquinas ficticias, y el número total de hitos detectados se eleva.



E_{max}	0,001	0,01	0,10	1,00	5,00	10,00
Hitos detectados	10,2	10,0	22,4	22,0	23,0	23,6

Figura 10.63: Número de hitos detectados frente a E_{max} para el mapa *Bigloop2*.

10.4.6. Ajustes recomendados

A la vista de los resultados obtenidos se proponen los siguientes ajustes para los parámetros del algoritmo de detección de segmentos rectilíneos *Split & Merge*:

- D_{max}. 2 o 3 centímetros son valores que han obtenidos buenos resultados tanto en entornos sintéticos como reales.
- L_{min}. Valores a partir de 0,5 metros y no superiores a 2 metros producen resultados adecuados. No obstante este parámetro depende enormemente del tipo de rectas presentes en el entorno.
- P_{min}. Valores en torno a 15 puntos, proporcionan buenos resultados, tanto en entornos reales como sintéticos.
- ρ_{max}. 10 cm.
- θ_{max}. 0.07 radianes $\approx$ 4 grados.
- E_{max}. 10 cm.

Los valores de D_{max}, L_{min} y P_{min}, deben ser utilizados como un primer ajuste grueso, previo a una ajuste fino adaptado a las condiciones del entorno. Por su parte los umbrales

de colinealidad ρ_{max} y θ_{max}, el el umbral de detección de esquinas E_{max}, pueden usarse sin ajustes posteriores, debido a la baja influencia que tienen sobre el proceso, y a que los valores recomendados se encuentran situados en el medio de amplias zonas de estabilidad.

10.5. Estudio de la influencia del ruido de los modelos de movimiento y observación

Los ruidos que afectan a los modelos de movimiento y observación tienen distinta influencia en el método FastSLAM. El ruido del modelo de movimiento afecta a la incertidumbre en la predicción de la pose del robot cada vez que se ejecutan unos controles, mientras que el ruido del modelo de observación afecta a la incertidumbre de las propiedades de los hitos detectados. Tanto el error en el movimiento como en la observación deberían estar disponibles, ya sea por datos proporcionados por los fabricantes de los sensores, o por ajustes de calibración. Sin embargo, en muchas ocasiones esto no es así, por lo que se debe experimentar con distintos valores.

El ruido que afecta a los modelos se considera un ruido blanco, con una distribución de Gauss de media nula y matriz de covarianza C [31]. Para el caso del modelo de movimiento la matriz de covarianza es de la forma,

$$C = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix}$$

mientras que para el modelo de observación tenemos,

$$C = \begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix}$$

donde σ_c^2 es la varianza de la componente c del ruido. El hecho de que las matrices de covarianza sean diagonales indica que las distintas componentes del ruido son independientes.

Uno de los problemas conocidos del algoritmo FastSLAM, en su versión 1.0 es su mal funcionamiento en situaciones en las que el ruido de observación es mucho menor que el ruido de movimiento [40]. En estos casos el algoritmo tiende a detectar hitos falsos y a realizar asociaciones de datos erróneas que conducen a la divergencia del método.

En la presente sección vamos a estudiar la influencia de los ruidos de movimiento y observación en el funcionamiento del método FastSLAM. En apartado 10.5.1 se estudia el efecto del ruido en el modelo de movimiento, mientras que en el siguiente (10.5.2) se estudia el efecto del ruido de observación.

10.5.1. Ruido de movimiento

El ruido de movimiento depende de la plataforma robótica que se esté desplazando en el entorno que se está cartografiando, así como de las propiedades de la superficie por la

que se desplaza, como el hecho de ser deslizante o no, el agarre de la plataforma al suelo, etc. Las posibilidades de simulación que ofrece la herramienta Player/Stage no son amplias en este sentido, de modo que únicamente es posible introducir cierto ruido aleatorio en la odometría. Por lo general el ruido presente en plataformas reales, suele estar sesgado hacía ciertos elementos, como por ejemplo un mayor deslizamiento de una de las ruedas, o la lectura incorrecta de alguno de los encoders, pero raramente puede considerarse su acción aleatoria. Por ello para estudiar la influencia del ruido de movimiento considerado en el modelo, se va a emplear únicamente un entorno y recorrido reales. En concreto se va a emplear el mapa denominado *USC SAL building*, mostrado en la figura 10.64. Este mapa se corresponde con datos de un recorrido real, obtenidos del repositorio de datos para SLAM radish (`http://radish.sourceforge.net`).

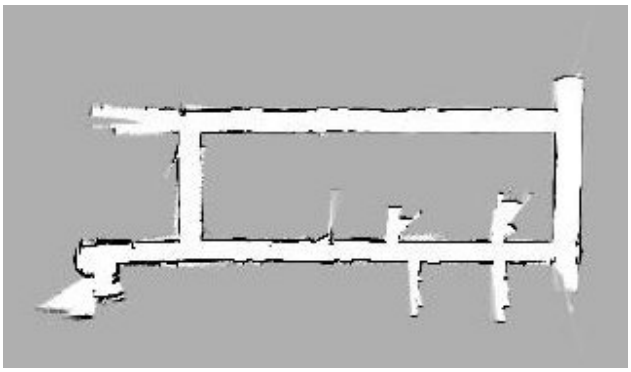


Figura 10.64: Mapa de rejilla del *USC SAL building*

Los parámetros comunes a todas las simulaciones se muestran en la figura 10.11.

Filtro			Split & Merge			Láser
N_{eff}	P0	N_p	P_{min}	D_{max}	L_{min}	Covarianza
∞	0,005	50	15	0,02	1,0	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$

Tabla 10.11: Parámetros de las simulaciones para el estudio del ruido de movimiento en el mapa *USC SAL Building*.

En la figura 10.65 se muestra el mapa construido en base a la odometría, que puede ser interpretado como el mapa obtenido al considerar el ruido en el modelo de movimiento, nulo. Como puede observarse la información odométrica es claramente insuficiente para construir un mapa fiel a la realidad.

Para hacer más evidentes los efectos del ruido del modelo de movimiento, se va a estudiar por separado el ruido de la componente angular de la pose, y el ruido de la componente de posición. En primer lugar se considera una varianza constante en la posición del robot, coordenadas (x, y), igual a $\sigma_x^2 = \sigma_y^2 = 0{,}007$, y se varía el ruido angular. De este modo, la

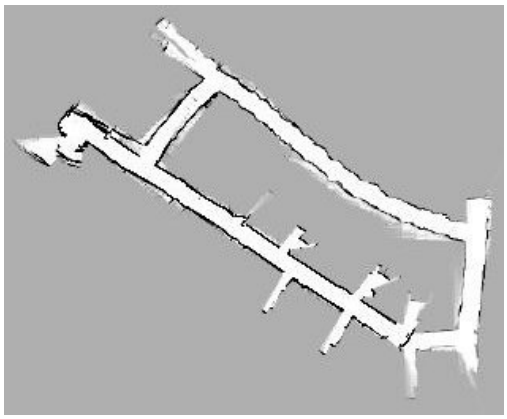
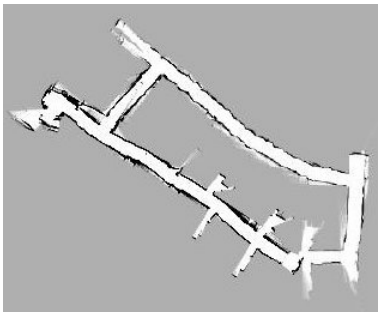


Figura 10.65: Mapa obtenido en base a la odometría.

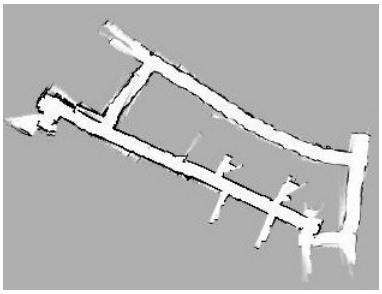
matriz de covarianza del ruido del modelo de movimiento queda,

$$C = \begin{pmatrix} 0{,}007 & 0 & 0 \\ 0 & 0{,}007 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix}$$

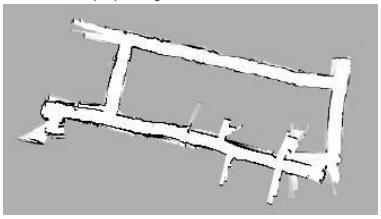
En la figuras 10.66 y 10.67 se muestran los mapas de rejilla obtenidos para distintos ruidos angulares. Partiendo de un ruido muy bajo, se observa una tendencia hacia el mapa



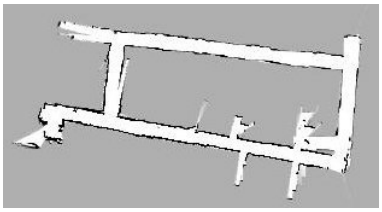
(a) $\sigma_\theta^2 = 0{,}0001$



(b) $\sigma_\theta^2 = 0{,}001$



(c) $\sigma_\theta^2 = 0{,}003$

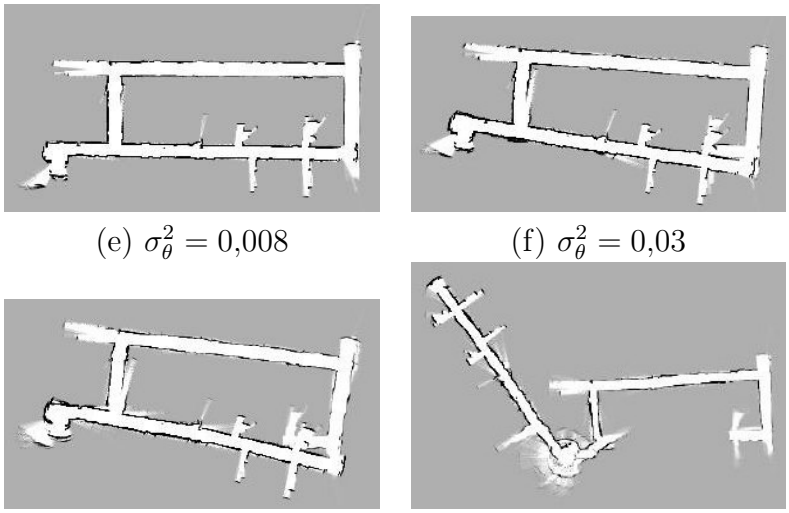


(d) $\sigma_\theta^2 = 0{,}005$

Figura 10.66: Mapa de rejilla obtenido introduciendo distintos ruidos angulares en el modelo.

correcto a medida que aumenta el ruido introducido en el modelo. Sin embargo, si el ruido

introducido es excesivo, el mapa obtenido se aleja de la realidad.



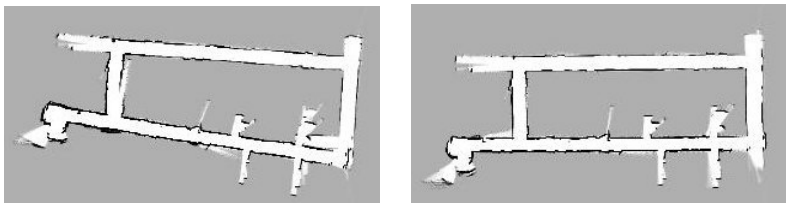
(e) $\sigma_\theta^2 = 0{,}008$ (f) $\sigma_\theta^2 = 0{,}03$

(g) $\sigma_\theta^2 = 0{,}08$ (h) $\sigma_\theta^2 = 0{,}2$

Figura 10.67: Mapa de rejilla obtenido introduciendo distintos ruidos angulares en el modelo.

Cuando el ruido angular considerado es muy inferior al ruido real, como en la figura 10.66 (a) y (b), los mapas obtenidos difieren mucho de la realidad. A medida que el ruido considerado se aproxima al real, como en las figuras 10.66 (c), (d), el mapa se aproxima más al mapa real, mientras que el bucle se cierra correctamente, como se muestra en la figura 10.67 (e), para valores en torno al ruido real. Si aumentamos el ruido excesivamente, como en la figura 10.67 (f), (g) y (h), los mapas obtenidos vuelven a divergir de la realidad.

Si el ruido considerado en el modelo es inferior al ruido presente en la realidad el método funcionará mal. Sin embargo, el considerar un ruido excesivo siempre puede compensarse aumentando el número de partículas. Este efecto puede observarse en la figura 10.68. Al



(a) 50 partículas. (b) 100 partículas.

Figura 10.68: Mapas de rejilla obtenido considerando $\sigma_\theta^2 = 0{,}03$.

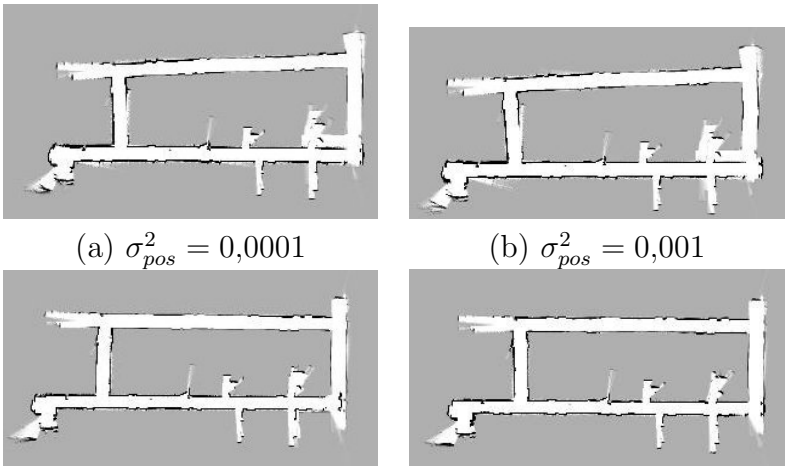
considerar 50 partículas se observa como el mapa difiere de la realidad. Esto es debido a que el excesivo ruido considerado define un *espacio* de posibles orientaciones angulares demasiado grande para ser cubierto por 50 partículas, de modo que no se llega a generar una partícula que se ajuste bien a la realidad. De este modo, aumentando el número de partículas

a 100, es posible ocupar adecuadamente el *espacio* de orientaciones angulares, y por ello el mapa obtenido es correcto. Evidentemente debe existir un compromiso entre el número de partículas y el ruido considerado. Parece aconsejable emplear un error ligeramente superior al real, con un número de partículas que no suponga una excesiva carga computacional.

Para mostrar el efecto del ruido en la posición, coordenadas (x, y), se fija la varianza del ruido $\sigma_\theta^2 = 0{,}008$, correspondiente con el ruido que mejores resultados proporcionó en la experiencia anterior. Se va a considerar el mismo ruido para ambas coordenadas cartesianas, de modo que denominando $\sigma_{pos}^2 = \sigma_x^2 = \sigma_y^2$ a la varianza de cada coordenada, la matriz de covarianza del ruido del modelo de movimiento queda:

$$\begin{pmatrix} \sigma_{pos}^2 & 0 & 0 \\ 0 & \sigma_{pos}^2 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$$

En las figuras 10.69 y 10.70 se muestran los mapas de rejilla obtenidos para distintos ruidos de la componente de posición del modelo de movimiento. Se observa una tendencia hacia el mapa correcto al aumentar el ruido introducido en el modelo. Sin embargo, si el ruido introducido es excesivo, el mapa obtenido se aleja de la realidad.



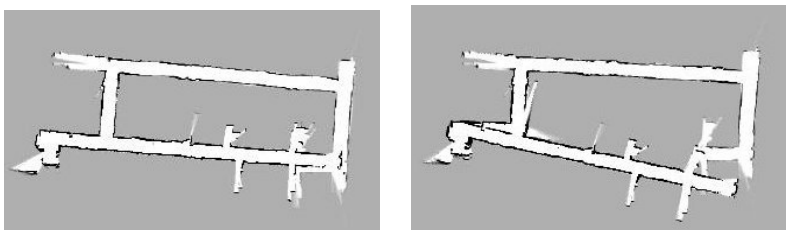
(a) $\sigma_{pos}^2 = 0{,}0001$ (b) $\sigma_{pos}^2 = 0{,}001$

(c) $\sigma_{pos}^2 = 0{,}003$ (d) $\sigma_{pos}^2 = 0{,}005$

Figura 10.69: Mapa de rejilla obtenido introduciendo distintos ruidos de posición en el modelo.

Se observa que, una vez fijado el ruido angular en valores adecuados, el ruido en la posición no tiene una influencia tan notable en el mapa construido. Los mapas de las figuras 10.69 (c) y (d), y 10.70 (e) son aptos para la navegación, mientras que los mapas de las figuras 10.69 (a) y (b), aunque no cerrando correctamente el bucle, muestran la tendencia geométrica de la realidad, aunque distorsionada. El mapa de la figura 10.70 (f) diverge completamente de la realidad por considerar un error demasiado elevado.

Lo que estamos determinando en realidad mediante este experimento es el valor verdadero



(e) $\sigma^2_{pos} = 0{,}008$ (f) $\sigma^2_{pos} = 0{,}02$

Figura 10.70: Mapa de rejilla obtenido introduciendo distintos ruidos de posición en el modelo.

del ruido que afecta a la odometría. Cuando el modelo refleja más fielmente la realidad se minimiza el error.

10.5.2. Ruido de observación

El ruido de observación afecta a las medidas realizadas con el sensor de rango láser. Consecuentemente, el ruido afecta tanto a la incorporación de nuevas características al mapa, como al proceso de asociación de datos o a la aplicación de los filtros de Kalman. Un mayor o menor ruido de observación tiene el efecto de modificar la incertidumbre acerca de las propiedades de un hito (en nuestro caso su posición), de modo que a la hora de realizar la asociación de datos pueden producirse asociaciones incorrectas o ambiguas, es decir, que una misma observación se asocie a varios hitos del mapa (en este caso nuestra implementación ignora la observación). Consecuentemente, para poder apreciar con claridad el efecto del ruido de observación es necesario experimentar con entornos que posean cierto grado de complejidad, de modo que sean susceptibles de producir ambigüedad a la hora de realizar la asociación de datos, por ello mapas sencillos como *Recinto básico* o *Lineamedia* no son adecuados. Por lo tanto para evaluar y poner de manifiesto la influencia del ruido de observación se va a emplear el mapa sintético denominado *Bigloop2* y el mapa real *IUSIANI*.

Comenzamos estudiando el mapa sintético *Bigloop2*. Se va a determinar el error del mapa obtenido frente al mapa de referencia para distintos valores de la matriz de covarianza del ruido del modelo de observación. Para que los resultados resulten más claros, se va a estudiar por una lado la variación de la componente de rango σ^2_ρ y por otro la variación de la componente angular σ^2_θ.

Los parámetros comunes a todas las simulaciones se muestran en la figura 10.12.

En primer lugar se considera la siguiente matriz de covarianza del ruido del modelo de observación:

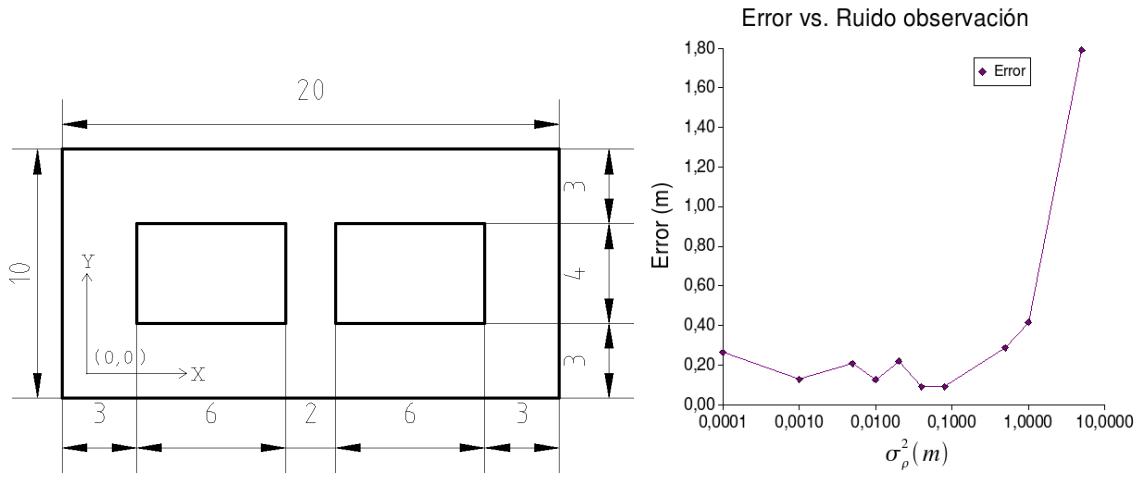
$$\begin{pmatrix} \sigma^2_\rho & 0 \\ 0 & 0{,}001224 \end{pmatrix}$$

Los resultados de las simulaciones en el mapa *Bigloop2* se muestran en la figura 10.71. Valores bajos de la covarianza significan una mayor "credibilidad" de los datos recibidos por el sensor. En este sentido, a la hora de realizar la asociación de datos el algoritmo tiende

Filtro			Split & Merge			Robot
N_{eff}	P0	N_p	P_{min}	D_{max}	L_{min}	Covarianza
∞	0,005	100	9	0,02	0,4	$\begin{pmatrix} 0{,}007 & 0 & 0 \\ 0 & 0{,}007 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

Tabla 10.12: Parámetros de las simulaciones para el estudio del ruido de observación en el mapa *Bigloop2*.

a considerar que la mayoría de las observaciones se corresponden con hitos previamente no observados, ya que tanto los hitos detectados, como los presentes en el mapa se consideran muy fiables, y por ello pequeñas diferencias hacen *creer* al algoritmo que se trata de hitos distintos. Consecuentemente, el error que se registra es mayor. Los mejores resultados se obtienen para valores en el rango comprendido entre 0.001 y 0.08, aunque concretamente los valores 0.04 y 0.08 proporcionan el mínimo error. A partir de estos valores, aumentar la covarianza aumenta las posibilidades de realizar asociaciones erróneas y por ello el error crece.

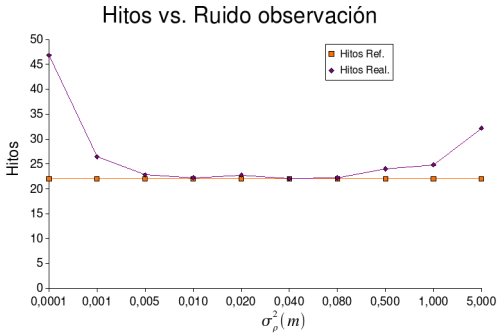


σ_ρ^2	0,0001	0,001	0,005	0,01	0,02	0,04	0,08	0,5	1	5
Error	0,26	0,13	0,21	0,13	0,22	0,09	0,09	0,29	0,42	1,79
σ_{error}^2	0,0073	0,0014	0,0032	0,0009	0,0041	0,0021	0,0002	0,0123	0,0095	1,1602

Figura 10.71: Evolución del error frente a la Covarianza del ruido de observación, modificando la varianza de la componente de rango.

Para observar más en detalle la evolución del número de hitos detectados en función del ruido de observación se incluye la figura 10.72. En ella se aprecia que para valores inferiores a 0,01 o superiores a 0,08, el número de características detectadas es superior al número

de características reales, que para este entorno son 22. Para valores pequeños el fenómeno se explica por la excesiva credibilidad que se concede a cada característica, de modo que pequeñas diferencias hacen considerar al algoritmo que se trata de hitos distintos. Para el caso de valores de σ_ρ^2 elevados, la detección de hitos extra se debe a fallos en la asociación que distorsionan el mapa percibido.



σ_ρ^2	0,0001	0,001	0,005	0,01	0,02	0,04	0,08	0,5	1	5
Hitos detectados	46,8	26,4	22,8	22,3	22,8	22,0	22,2	24,0	24,8	32,2

Figura 10.72: Número de hitos detectados frente a σ_ρ^2 para el mapa *Bigloop2*.

Un segundo conjunto de simulaciones se realiza considerando la siguiente matriz de covarianza del ruido del modelo de observación:

$$\begin{pmatrix} 0{,}08 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix}$$

Los resultados de las simulaciones se muestran en la figura 10.73. El comportamiento es análogo al observado al modificar la varianza del ruido de la componente de rango, mostrado en la figura 10.71, siendo en este caso valores adecuados los comprendidos entre 0,0008 y 0,001224.

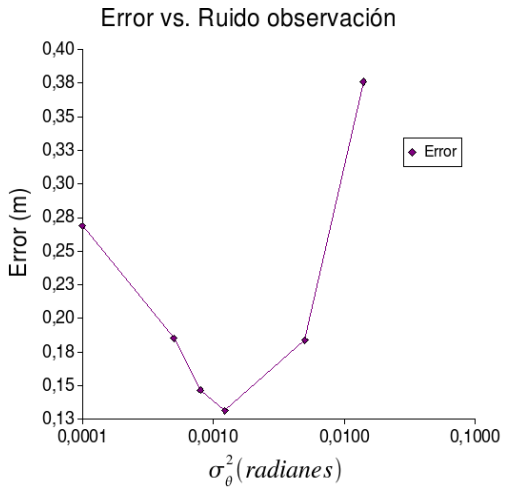
En la figura 10.74 se muestra el número de hitos detectados frente a la varianza de la componente angular del ruido de observación. Cuando el ruido angular considerado es pequeño se detectan más hitos que los realmente presentes en el mapa. A continuación, a partir de un valor de 0,0005 se detecta correctamente el número de hitos.

Por último se muestra en la figura 10.75 el mapa rejilla y el recorrido realizado por el robot considerando la siguiente covarianza del error del ruido de observación:

$$\begin{pmatrix} 0{,}01 & 0 \\ 0 & 0{,}0008 \end{pmatrix}$$

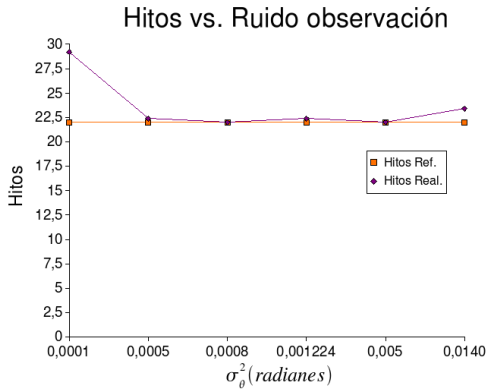
Las componentes de la matriz de covarianza son aquellas que mejores resultados han ofrecido en cada caso.

Tras haber estudiado el comportamiento del algoritmo frente al ruido de movimiento en un mapa sintético, pasamos seguidamente a estudiar el comportamiento en un mapa y



σ_θ^2	0,0001	0,0005	0,0008	0,001224	0,0050	0,0140
Error	0,27	0,19	0,15	0,13	0,18	0,38
σ_{error}^2	0,0246	0,0034	0,0022	0,0009	0,0059	0,0711

Figura 10.73: Evolución del error frente a la Covarianza del ruido de observación, modificando la varianza de la componente angular.



σ_θ^2	0,0001	0,0005	0,0008	0,001224	0,0050	0,0140
Hitos detectados	29,2	22,4	22,0	22,4	22,0	23,4

Figura 10.74: Número de hitos detectados frente a σ_ρ^2 para el mapa *Bigloop2*.

recorrido reales. Como ya se ha mencionado en esta ocasión empleamos un recorrido real en el laboratorio del IUSIANI (ver figura 10.8). Los parámetros empleados en todas la simulaciones se muestran en la figura 10.13.

En primer lugar se varía la covarianza de la componente de rango del error del observación

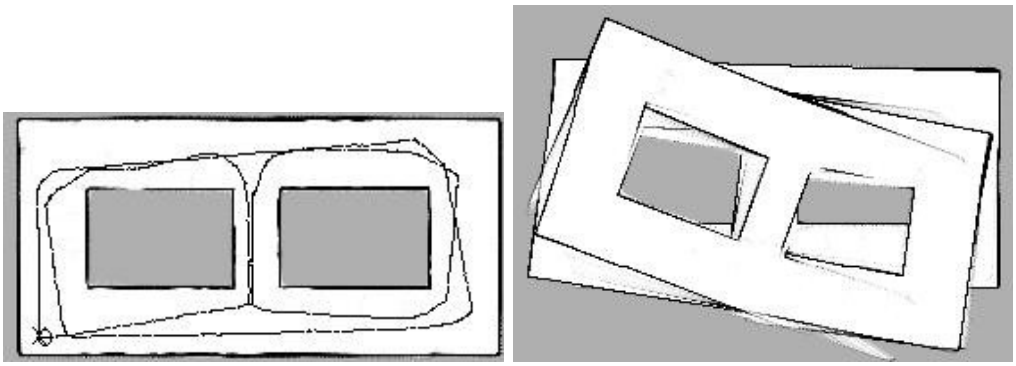


Figura 10.75: A la izquierda mapa de rejilla y recorrido para $\sigma_\rho^2 = 0{,}01$ y $\sigma_\theta^2 = 0{,}0008$, X punto inicial O punto final. A la derecha, mapa de rejilla basado en la odometría.

Filtro			Split & Merge			Robot
N_{eff}	P0	N_p	P_{min}	D_{max}	L_{min}	Covarianza
∞	0,005	100	15	0,02	0,6	$\begin{pmatrix} 0{,}005 & 0 & 0 \\ 0 & 0{,}005 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

Tabla 10.13: Parámetros de las simulaciones para el estudio del ruido de observación en el mapa *IUSIANI*.

σ_ρ^2, de modo que la matriz de covarianza empleada es la siguiente:

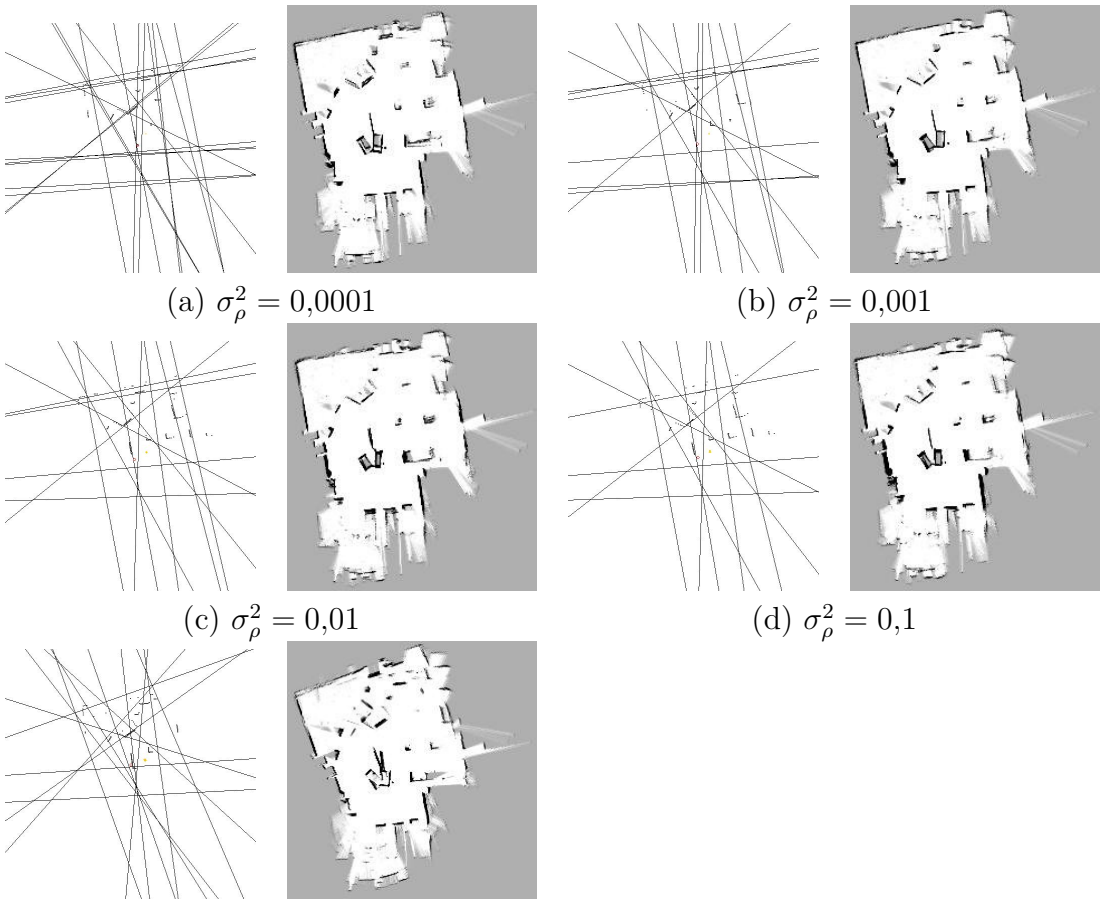
$$\begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & 0{,}001224 \end{pmatrix}$$

En la figura 10.76 se muestra una serie de mapas de rejilla, y de hitos detectados para distintos valores de σ_ρ^2. Prácticamente los mapas de la figura 10.76 (a), (b), (c) y (d) son correctos, sin embargo en (a) y (b) puede observarse una detección excesiva de hitos, mientras que (c) y (d) muestran una detección adecuada, si bien en (d) algunas rectas cercanas se confunden y se consideran una sola. En (f) el mapa obtenido comienza a ser defectuoso, debido a las asociaciones de datos incorrectas. No obstante la componente de rango de la covarianza del ruido de observación no tiene un efecto muy acusado sobre el mapa obtenido, ya que las rectas presentes en el mapa no están muy cerca unas de otras.

A continuación se varía la covarianza de la componente angular del error del observación σ_θ^2, de modo que la matriz de covarianza empleada es la siguiente:

$$\begin{pmatrix} 0{,}001 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix}$$

En la figura 10.77 se muestra una serie de mapas de rejilla, y de hitos detectados para distintos valores de σ_θ^2. Los mapas mostrados en (a) y (b) son incorrectos, así como son



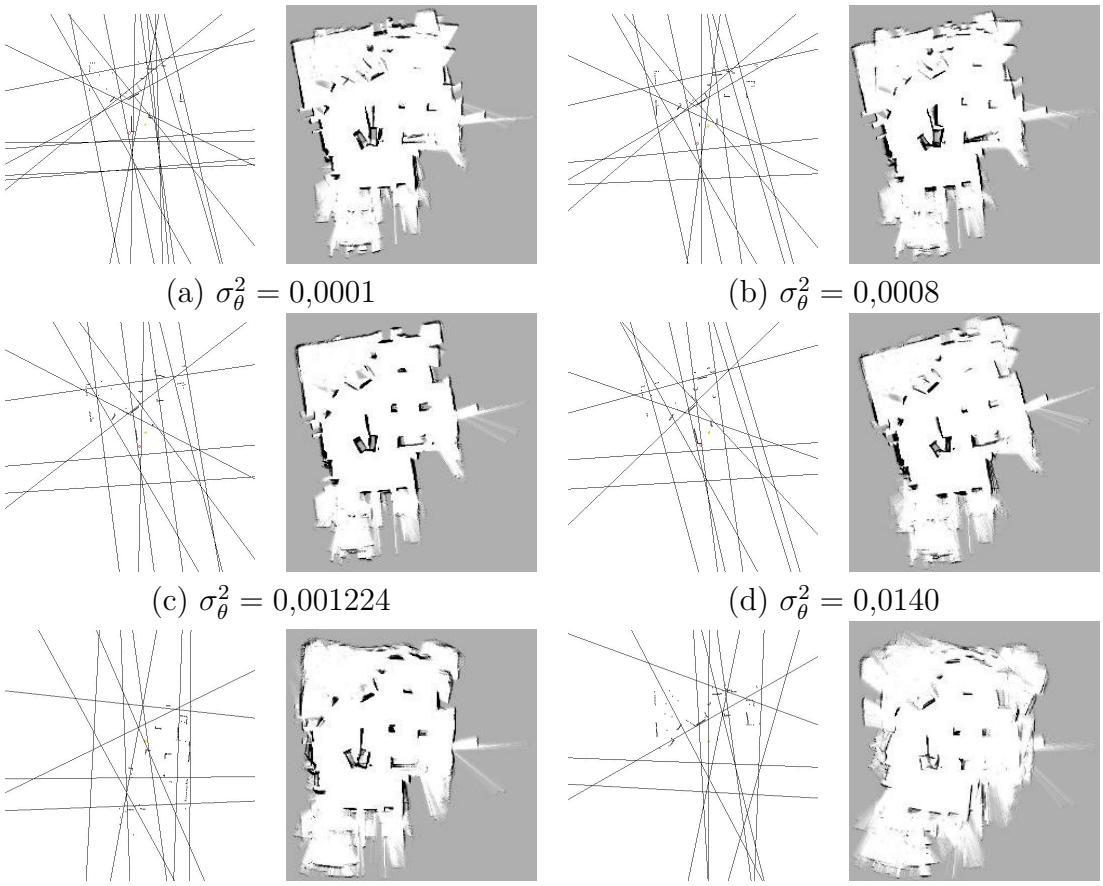
(a) $\sigma_\rho^2 = 0{,}0001$ (b) $\sigma_\rho^2 = 0{,}001$

(c) $\sigma_\rho^2 = 0{,}01$ (d) $\sigma_\rho^2 = 0{,}1$

(f) $\sigma_\rho^2 = 1$

Figura 10.76: Mapa de rejilla e hitos detectados para distintos valores de la componente de rango de la covarianza del ruido de observación, en un recorrido por el laboratiro del IUSIANI.

excesivos el número de hitos detectados. Los mapas de (c) y (d) son ambos correctos, si bien es mejor, menos ruidoso, el mapa (c). En los casos mostrados en (e) y (f) los errores de asociación conducen a mapas incorrectos. El efecto de la componente angular de la covarianza del ruido de observación es más acusado. El motivo se encuentra en el modelo de movimiento. A la hora de generar las poses de las nuevas partículas se introduce una cierta variabilidad en las coordenadas (x, y, θ). Un σ_θ^2 excesivamente pequeño provocará que la reobservación con mínimas diferencias de una recta previamente vista sea considerada como una nueva, de modo que prácticamente todas las partículas, salvo la que se ajuste a la verdadera pose del robot, añadirán un hitos a sus mapas. De este modo la sensibilidad ante posibles picos de ruido en los controles es extrema. Por otro lado para valores elevados de σ_θ^2 el efecto que se produce es el de no proporcionar una puntuación mucho más alta a las partículas que más se ajusten a la observación, frente a las que difieren un poco, dado que la tolerancia angular disminuye estas diferencias.



(a) $\sigma_\theta^2 = 0{,}0001$ (b) $\sigma_\theta^2 = 0{,}0008$

(c) $\sigma_\theta^2 = 0{,}001224$ (d) $\sigma_\theta^2 = 0{,}0140$

(e) $\sigma_\theta^2 = 0{,}2$ (f) $\sigma_\theta^2 = 0{,}5$

Figura 10.77: Mapa de rejilla e hitos detectados para distintos valores de la componente angular de la covarianza del ruido de observación, en un recorrido por el laboratorio del IUSIANI.

Por último se muestra en la figura 10.78 el mapa de rejilla construido en base únicamente a la odometría, y el mapa de rejilla aplicando SLAM con los siguientes valores para la matriz de covarianza del ruido de observación:

$$\begin{pmatrix} 0{,}01 & 0 \\ 0 & 0{,}003 \end{pmatrix}$$

En este caso se aprecia perfectamente el efecto de aplicar un algoritmo de SLAM sobre unos datos aparentemente aleatorios.

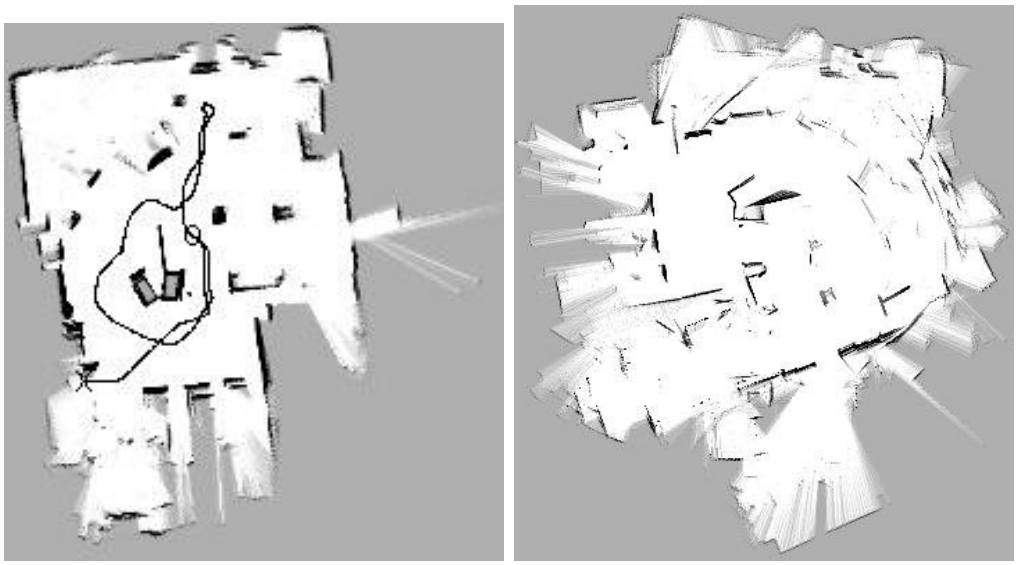


Figura 10.78: A la izquierda mapa de rejilla y recorrido para $\sigma_\rho^2 = 0{,}01$ y $\sigma_\theta^2 = 0{,}003$, X punto inicial O punto final. A la derecha, mapa de rejilla basado en la odometría.

10.6. Estudio de la influencia de los Parámetros del filtro de partículas

Los distintos factores que afectan a los filtros de partículas son fundamentales para el funcionamiento del algoritmo FastSLAM, ya que éste se basa precisamente en el empleo de este tipo de filtros. Nuestra implementación permite controlar tres aspectos de los filtros de partículas. Por un lado es posible especificar el número de partículas del filtro. Por otro, se pueden establecer las condiciones de realización del proceso de remuestreo, en forma de un umbral para el número de partículas efectivas, o N_{eff}, por debajo del cual se remuestrea. Por último es posible especificar la probabilidad P_0 que se asigna a la consideración de una observación de un hito como nueva.

En la presente sección se estudia el efecto de los diversos factores parametrizables en nuestra implementación de un filtro de partículas, ofreciendo para cada uno de ellos unos valores recomendados. De este modo, en la sección 10.6.1 se estudia la influencia de diversos umbrales para el N_{eff}. En la sección 10.6.2 se estudia el efecto de la probabilidad de una nueva observación P_0. Por último, en la sección 10.6.3 se estudia la repercusión del número de partículas.

10.6.1. Estudio de la influencia del umbral de partículas efectivas en el remuestreo

En cuanto al proceso de remuestreo, es posible especificar cuándo ha de aplicarse. El remuestreo puede realizarse en todas y cada una de las iteraciones del algoritmo, o bien

especificar un umbral para el número de partículas efectivas o N_{eff}, por debajo del cual se realiza el remuestreo. Remuestrear demasiado a menudo (N_{eff} excesivamente alto) puede provocar un empobrecimiento en la población de partículas, es decir se reduce el número de hipótesis consideradas. Remuestrear poco frecuentemente (N_{eff} excesivamente bajo) puede provocar el mantenimiento de muchas hipótesis incorrectas que en un momento dado pueden recibir mayor puntuación que las hipótesis verdaderas, provocando la divergencia del filtro [40]. Sin embargo el remuestreo no puede observarse como un factor aislado, ya que otros factores, como los ruidos de movimiento y observación, o la cantidad de hitos detectables en el entorno, pueden determinar distintas necesidades de remuestreo. Además, dependiendo del algoritmo empleado para remuestrear, la frecuencia de remuestreo puede no ser un factor decisivo. Para el algoritmo de remuestreo sistemático [67] empleado en nuestro simulador, la frecuencia de remuestreo puede mantenerse en su valor máximo, sin un gran riesgo de empobrecimiento del conjunto de partículas. Por otro lado se ha observado que es muy raro obtener valores del número de partículas efectivas inferiores a 1, por lo que establecer un $N_{eff} = 1$, equivale a no remuestrear nunca.

En la presente sección se va a estudiar la influencia del umbral de partículas efectivas N_{eff} para dos recorridos, uno real y otro simulado en un mapa sintético.

En general, y antes de analizar los resultados, puede decirse que en entornos en los que se dispone de mucha información, es decir, es frecuente detectar hitos en los barridos láser, es mejor remuestrear muy frecuentemente, y de este modo aprovechar la información disponible para enriquecer el conjunto de partículas. Por el contrario, en entornos con un nivel de información bajo, es necesario emplear menores frecuencias de remuestreo, de modo que se mantengan varias hipótesis *abiertas*, hasta que una serie de evidencias nos permita determinar cuales son mejores.

Como recorrido real vamos a emplear el mapa denominado *USC SAL building*, mostrado en la figura 10.79. Los datos de este recorrido han sido obtenidos del repositorio de datos para SLAM radish (`http://radish.sourceforge.net`). Los parámetros comunes a todas las simulaciones se muestran en la figura 10.14.

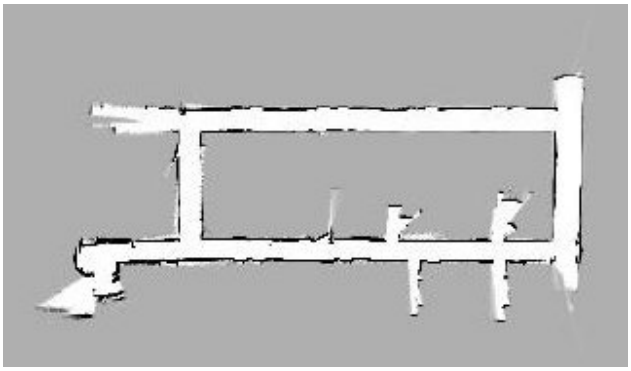


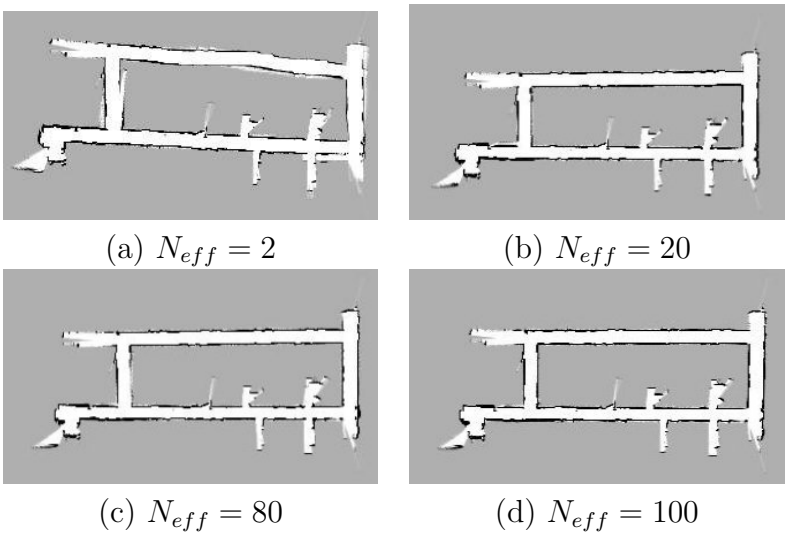
Figura 10.79: Mapa de rejilla del *USC SAL building*

En la figura 10.80 se observa como no existen diferencias significativas entre los mapas

Filtro		Split & Merge			Láser	Robot
P_0	N_p	P_{min}	D_{max}	L_{min}	Covarianza	Covarianza
0.005	100	15	0.02	0.6	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$	$\begin{pmatrix} 0{,}007 & 0 & 0 \\ 0 & 0{,}007 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

Tabla 10.14: Parámetros de las simulaciones para el estudio del N_{eff} en el mapa *USC SAL building*.

obtenidos con umbrales de N_{eff} que van desde 2 hasta 100 partículas efectivas, es decir, desde remuestrear relativamente poco hasta remuestrear siempre. La observación de la simulación en modo interactivo revela que en realidad, incluso para valores bajos del umbral de N_{eff}, el remuestreo se lleva a cabo cada 3 o 4 iteraciones. Esto se debe a que la frecuente observación de hitos hace aumentar mucho la puntuación de las partículas buenas, y esto provoca un descenso en el número de partículas efectivas, con lo que en pocas iteraciones se supera el umbral, y se activa el proceso de remuestreo. Por otro lado se constata como la aplicación frecuente del remuestreo sistemático no es peligrosa de cara al empobrecimiento del conjunto de partículas. Para observar efectos negativos hay que emplear valores del umbral N_{eff} extremadamente bajos, del orden de las unidades.



(a) $N_{eff} = 2$ (b) $N_{eff} = 20$

(c) $N_{eff} = 80$ (d) $N_{eff} = 100$

Figura 10.80: Mapa de rejilla obtenido introduciendo distintos umbrales para el número de partículas efectivas.

Como recorrido en un mapa sintético vamos a emplear el mapa denominado *Simulación IUSIANI*, mostrado en la figura 10.81. Este mapa se corresponde con datos de un recorrido sobre un mapa del laboratorio del IUSIANI. Los parámetros comunes a todas las simulaciones se muestran en la figura 10.15

A la vista de los resultados mostrados en la figura 10.82, se constata que el umbral de

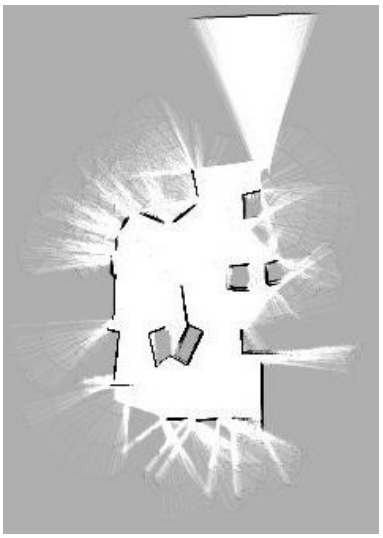
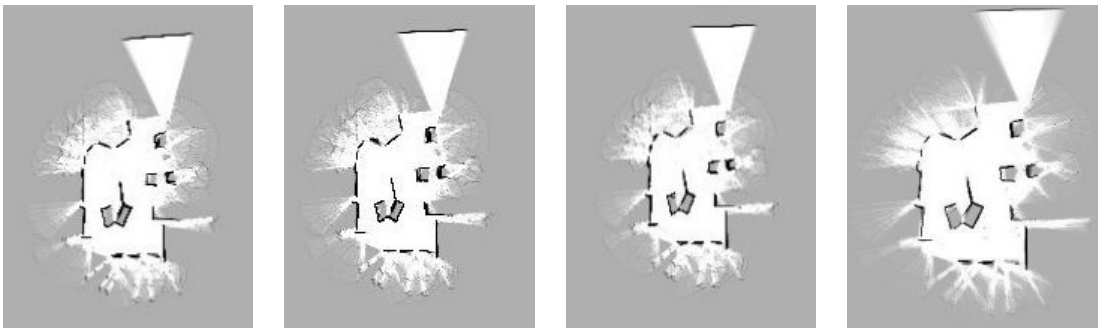


Figura 10.81: Mapa de rejilla del laboratorio del IUSIANI

Filtro		Split & Merge			Láser	Robot
P_0	N_p	P_{min}	D_{max}	L_{min}	Covarianza	Covarianza
0.005	100	30	0.02	1,8	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$	$\begin{pmatrix} 0{,}001 & 0 & 0 \\ 0 & 0{,}001 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$

Tabla 10.15: Parámetros de las simulaciones para el estudio de N_{eff} en un recorrido simulado del laboratorio del IUSIANI.



(a) $N_{eff} = 100$ (b) $N_{eff} = 80$ (c) $N_{eff} = 40$ (d) $N_{eff} = 2$

Figura 10.82: Mapa de rejilla obtenido introduciendo distintos umbrales para el número de partículas efectivas.

N_{eff} no tiene una influencia notable sobre el mapa obtenido, al menos cuando se usa un proceso de remuestreo sistemático. Se observa que incluso para valores del N_{eff} tan bajos

como 2, el mapa obtenido es correcto. La observación de la simulación en modo interactivo revela que el remuestreo se produce frecuentemente, incluso para valores del umbral de N_{eff} bajos. El mapa analizado anteriormente, el mapa *USC SAL building*, empeora al emplear un $N_{eff} = 2$, esto no ocurre para el mapa presente debido a que al tratarse de un mapa simulado, está sujeto a menos errores y por ello el remuestreo tiene aun menos importancia.

10.6.2. Estudio de la influencia de la probabilidad asignada a una nueva observación P_0

P_0 es la similitud que se asocia con la observación de un hito por primera vez. Durante el proceso de asociación de datos se considera que un hito es una nueva observación, si el parámetro (evitamos explícitamente emplear el término probabilidad, ya que los valores de la similitud no se encuentran estrictamente restringidos al intervalo $[0, 1]$) P_0 es mayor que la similitud (likelihood en la literatura inglesa) de que el hito provenga de la observación de cualquiera de los hitos presentes en el mapa. Para obtener más información acerca del proceso de asociación de datos, consultar la sección 5.

A priori puede pensarse que valores elevados de P_0 conducirán a la adición al mapa como nuevos hitos, de lo que únicamente son observaciones repetidas de un mismo elemento del entorno. En este sentido el esfuerzo computacional para realizar el cálculo del vector de asociación de datos se incrementará notablemente, al verse incrementado el número de hitos presentes en el mapa. Valores muy pequeños de P_0 deberían conducir a la incapacidad de incorporar correctamente nuevos hitos al mapa, es decir, hitos observados por primera vez serían asociados con hitos presentes en el mapa (los únicos hitos que se incorporan correctamente al mapa en cualquier situación son la primera o primeras rectas y esquinas observadas). Para observar este efecto hay que considerar valores de P_0 del orden de 10^{-270}. Cualquiera de los mapas es apto para estudiar este parámetro, por lo que se emplea el mapa *Bigloop2* representativo de entornos interiores de oficina. Los parámetros comunes a todas la simulaciones se muestran en la figura 10.16

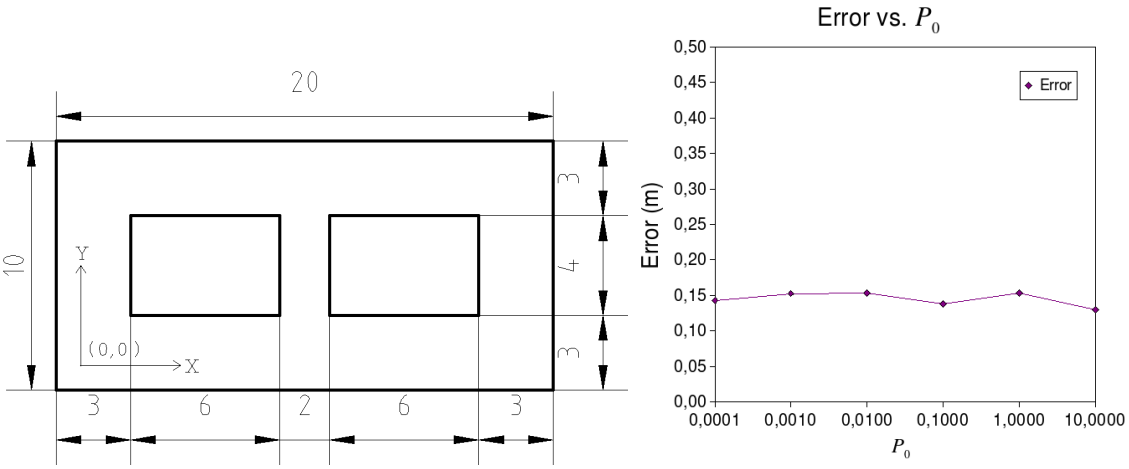
Filtro		Split & Merge			Robot	Láser
N_{eff}	N_p	P_{min}	D_{max}	L_{min}	Covarianza	Covarianza
∞	100	9	0.02	0.4	$\begin{pmatrix} 0{,}007 & 0 & 0 \\ 0 & 0{,}007 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$

Tabla 10.16: Parámetros de las simulaciones para el estudio de P_0 en el mapa *Bigloop2*.

Tanto el mapa como la evolución del error se muestran en la figura 10.83. Valores de P_0 entre 0,0001 y 10 no tienen una influencia directa sobre el error. Esto se debe a que no todas las observaciones se incluyen en el mapa como nuevos hitos, sino que sigue habiendo asociación de datos y por tanto remuestreo, que está permitiendo evolucionar sólo a aquellas partículas que mejor explican la realidad. Lo que ocurre es que algunos de los hitos muy similares a los presentes en el mapa se consideran como nuevos, pero no se está permitiendo

la supervivencia de partículas con hitos erróneos, gracias a que sigue habiendo asociación de datos y remuestreo. La forma de computar el error como el error total divido por el número de características, enmascara la detección de múltiples hitos muy similares.

Se ha iniciado una prueba con $P_0 = 100$ observándose en este caso que todos y cada uno de los hitos observados se incorporan al mapa como nuevas observaciones. Ante esta situación todas las partículas tienen el mismo peso, por lo que el remuestreo no tienen ningún efecto y no se puede construir un mapa. No se ha podido completar el experimento debido a las que las enormes dimensiones del mapa desbordan las capacidades computacionales disponibles.



P_0	0,0001	0,001	0,01	0,1	1	10
Error	0,14	0,15	0,15	0,14	0,15	0,13
σ^2_{error}	0,0017	0,0025	0,0010	0,0014	0,0024	0,0002

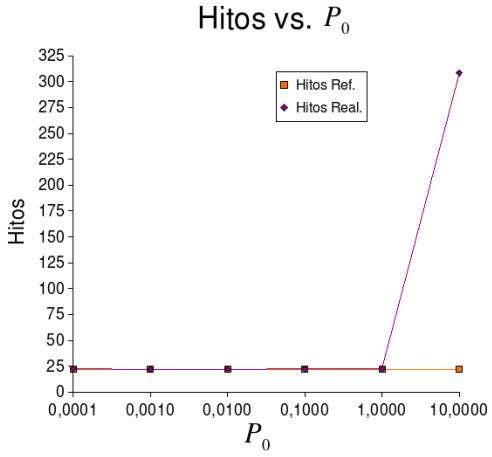
Figura 10.83: Mapa real, a la izquierda, y evolución del error frente P_0

En cuanto al número de hitos detectados, cuya evolución frente a P_0 puede observarse en la figura 10.84, se aprecia que funciona correctamente hasta cierto punto entre 1 y 10 en el que el número se dispara, lo cual está en consonancia con la influencia esperada de P_0.

En general cualquier valor de P_0 en el rango $[10^{-3}, 1]$ da buenos resultados, no apareciendo ni detecciones incorrectas de hitos, ni asociaciones de datos erróneas.

10.6.3. Estudio de la influencia del número de partículas (Par)

El número de partículas suele ser el elemento al que más atención se presta a la hora de hablar de filtros de partículas, ya que es uno de los factores que intervienen en las necesidades computacionales de su cálculo, y puede ser un factor limitante para su aplicación. Por ello para estudiar la influencia del número de partículas en el error de los mapas obtenidos, se van a emplear los mapas denominados *Recinto* (Figura 10.1), *Lineamedia* (Figura 10.2), *Bigloop* (Figura 10.3), *Bigloop2* (Figura 10.4) y *Complex* (Figura 10.1), es decir, todos los mapas sintéticos para los que podemos calcular el error, por disponer de información precisa sobre



P_0	0,0001	0,001	0,01	0,1	1	10
Hitos detectados	22,4	22,2	22,2	22,4	22,6	308,4

Figura 10.84: Número de hitos detectados frente a P_0 para el mapa *Bigloop2*.

las características que los integran. De este modo podremos encontrar un límite superior para el número de partículas, a partir del cual no se reduce el error del mapa obtenido, o la reducción es tan pequeña que no compensa el aumento en los requisitos computacionales. Los números de partículas a simular en cada caso son 5, 10, 20, 50, 100, 200 y 500.

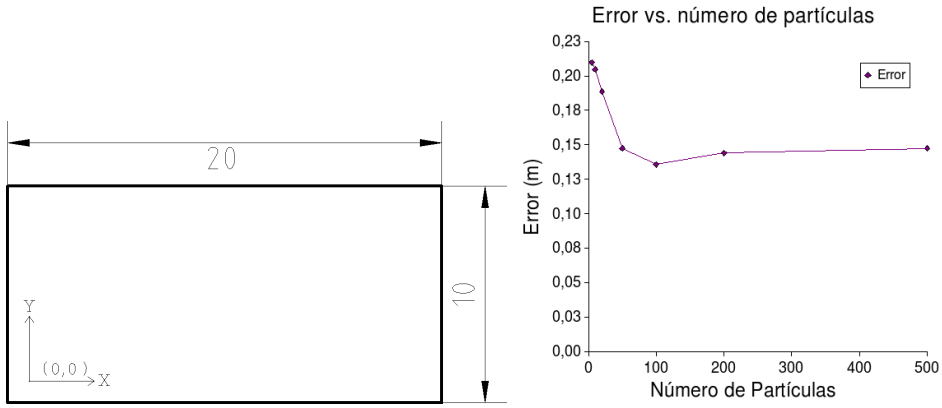
Tras realizar múltiples simulaciones con distintas configuraciones, se ha determinado una configuración media que permite ejecutar todos los mapas, obteniendo resultados coherentes y reproducibles. Esta configuración se muestra en la figura 10.17.

Filtro		Split & Merge			Robot	Láser
N_{eff}	$P0$	P_{min}	D_{max}	L_{min}	Covarianza	Covarianza
∞	0.005	9	0.02	0.4	$\begin{pmatrix} 0{,}01 & 0 & 0 \\ 0 & 0{,}01 & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$

Tabla 10.17: Parámetros de las simulaciones para el estudio de *Par* en mapas sintéticos.

En primer lugar, en la figura 10.85 puede observarse el mapa de un recinto rectangular vacío, denominado *Recinto Básico*. Además en la misma figura se muestra la evolución del error frente al número de partículas empleadas en la simulación.

En este caso la simplicidad del entorno provoca que la influencia del número de partículas no sea muy pronunciada. Aun así se observa la tendencia típica que muestra una reducción rápida del error, en este caso de 0 a 50 partículas, y una estabilización del mismo cuando se emplean 50 o más partículas. En cuanto al número de hitos detectados, a partir de 50 partículas coincide con el número de hitos reales, en este caso 8 (ver figura 10.1). Es destacable el hecho de que los distintos errores calculados se mueven entre 21 y 14 cm, lo que demuestra



Par	5	10	20	50	100	200	500
Error	0,21	0,20	0,19	0,15	0,14	0,14	0,15
σ^2_{error}	0,0032	0,0039	0,0114	0,0014	0,0007	0,0024	0,0029
Hitos detectados	9,0	8,3	8,2	8,0	8,0	8,0	8,0
Hitos reales	8						

Figura 10.85: Mapa real, a la izquierda, unidades en metros. Error frente al número de partículas, a la derecha.

la escasa influencia del número de partículas.

En la figura 10.86 se muestra el mapa de rejilla y el recorrido del robot calculados con 50 partículas, y el mapa de rejilla basado únicamente en información odométrica. en estos mapas el color blanco indica espacio para el que se dispone de evidencias de que se encuentra vacío. El espacio negro identifica posiciones ocupadas, mientras que el color gris identifica zonas para las que no se dispone información. Se observa como la aplicación del algoritmo de SLAM corrige los defectos introducidos en el mapa por una odometría ruidosa.

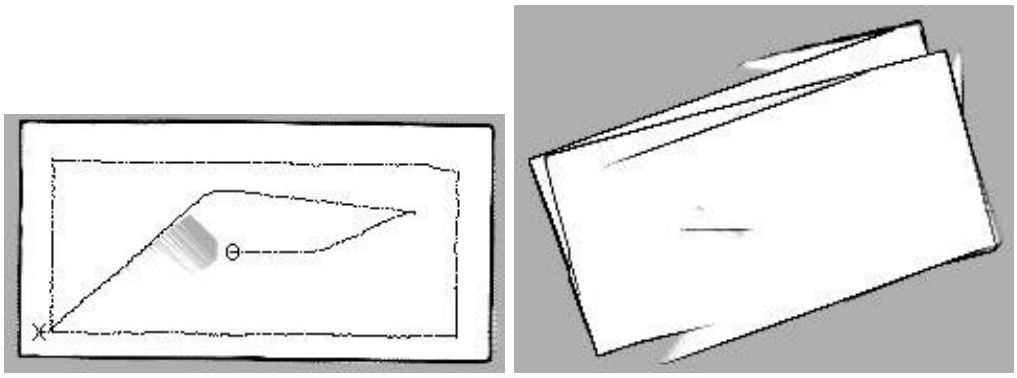
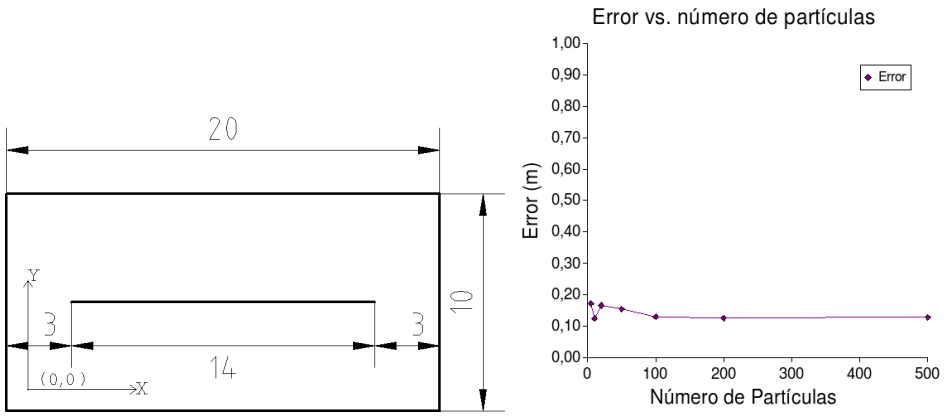


Figura 10.86: A la izquierda mapa de rejilla y recorrido para $Par = 50$ partículas, X punto inicial O punto final. A la derecha, mapa de rejilla basado en la odometría.

En la figura 10.87 puede observarse el mapa de un recinto básico con una pared central, denominado *Lineamedia*. En la misma figura se muestra la evolución del error frente al número de partículas empleadas en la simulación.



Par	5	10	20	50	100	200	500
Error	0,17	0,13	0,17	0,16	0,13	0,13	0,13
σ^2_{error}	0,0007	0,0048	0,0012	0,0061	0,0028	0,0036	0,0040
Hitos detectados	9,0	9,6	9,0	9,0	9,0	9,0	9,0
Hitos reales	9						

Figura 10.87: Mapa real, a la izquierda, unidades en metros. Error frente al número de partículas, a la derecha

Este mapa es algo más complejo que el mapa de un recinto básico. Aún así la simplicidad del entorno provoca que el número de partículas empleado en el filtro no tenga prácticamente influencia sobre el error ni sobre el número de hitos detectados. No obstante se observa una zona de reducción del error entre 0 y 100 partículas. A partir de 100 partículas el error se estabiliza en unos 13 cm. La introducción de la pared central en el entorno aporta información sin incrementar la complejidad del mismo, por lo que en este caso el proceso de SLAM funciona mejor, y la diferencia de error entre el mejor y el peor mapa es de tan sólo 4 cm. El número de hitos presentes en el mapa (figura 10.2) real es 9, 5 rectas y 4 esquinas, número detectado correctamente a partir de 20 partículas.

En la figura 10.88 se muestra el mapa de rejilla y el recorrido del robot calculados con 100 partículas, y el mapa basado únicamente en información odométrica. Se observa como tras dar dos vueltas al bucle el mapa obtenido corrige los defectos de la odometría y es muy similar al mapa real.

En la figura 10.89 puede observarse el mapa de un recinto con bucle de grandes dimensiones, denominado *Bigloop* y la evolución del error frente al número de partículas empleadas en la simulación.

Este es el primer mapa cuyo grado de complejidad permite observar claramente el efecto del número de partículas. Se observa una primera zona en la que el error disminuye drásticamente, hasta llegar a 50 partículas. A partir de ahí el error se estabiliza alrededor de 10

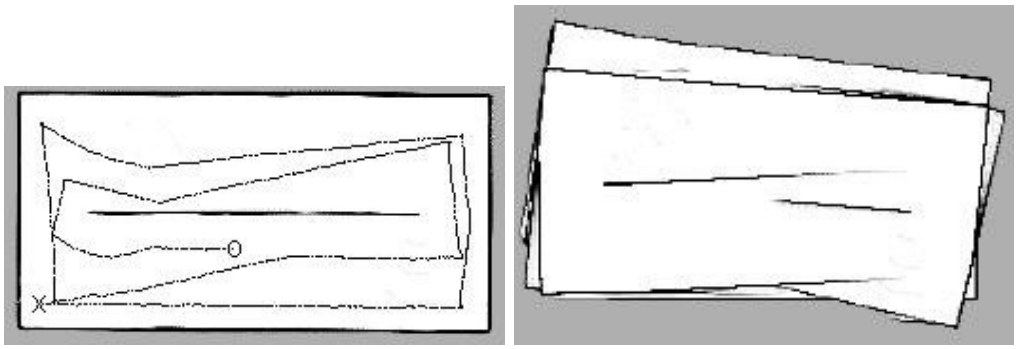
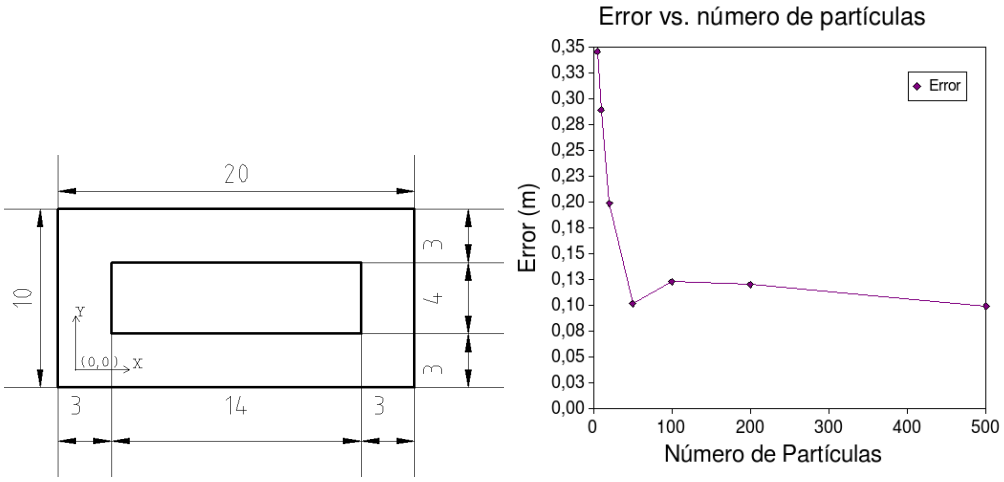


Figura 10.88: A la izquierda mapa de rejilla y recorrido para $Par = 100$ partículas, X punto inicial O punto final. A la derecha, mapa de rejilla basado en la odometría.



Par	5	10	20	50	100	200	500
Error	0,35	0,29	0,20	0,10	0,12	0,12	0,10
σ^2_{error}	0,0257	0,0474	0,0293	0,0013	0,0014	0,0002	0,0006
Hitos detectados	21,2	18,0	16,8	16,2	16,0	16,0	16,0
Hitos reales	16						

Figura 10.89: Mapa real, a la izquierda, unidades en metros. Error frente al número de partículas, a la derecha.

cm. En cuanto al número de hitos detectado cabe destacar que a partir de 100 partículas se detecta el número correcto, es decir 16 hitos (ver figura 10.3), detectándose para un número inferior de partículas más hitos de los realmente presentes.

En la figura 10.90 se muestra para el mapa *Bigloop* el mapa de rejilla y el recorrido del robot calculados con 100 partículas, y el mapa basado únicamente en información odométrica. De nuevo se observa que tras recorrer dos veces el bucle, el mapa obtenido mediante SLAM elimina los efectos de la odometría ruidosa, y el mapa obtenido es fiel a la realidad.

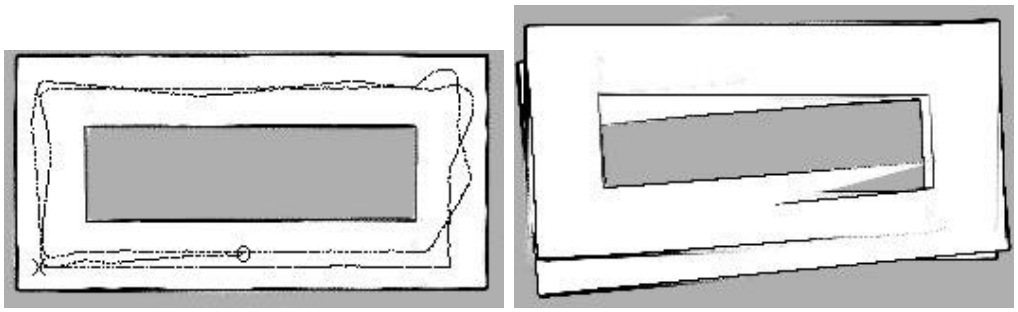
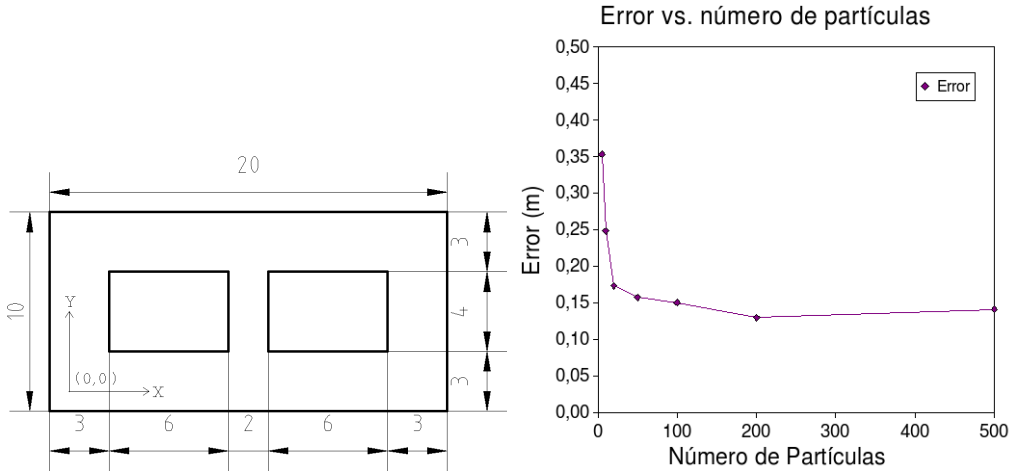


Figura 10.90: A la izquierda mapa de rejilla y recorrido para $Par = 100$ partículas, X punto inicial O punto final. A la derecha, mapa de rejilla basado en la odometría.

En la figura 10.91 puede observarse el mapa de un recinto con dos bucles de tamaño medio, denominado *Bigloop2*. Este entorno pretende ser un reflejo de entornos de oficina, en los que se encuentran rectas de diversos tamaños, y un cierto orden estructural. En la misma figura puede observarse la evolución del error frente al número de partículas empleadas en la simulación.



Par	5	10	20	50	100	200	500
Error	0,47	0,40	0,25	0,17	0,12	0,13	0,12
σ^2_{error}	0,0100	0,0059	0,0115	0,0048	0,0017	0,0040	0,0031
Hitos detectados	31,0	26,4	23,8	22,0	22,6	22,0	22,2
Hitos reales	22						

Figura 10.91: Mapa real, a la izquierda, unidades en metros. Error frente al número de partículas, a la derecha

Se observa una primera zona en la que el error disminuye drásticamente, hasta llegar a 20 partículas. A continuación el error sigue disminuyendo, aunque a un ritmo mucho menor

hasta llegar a 100 partículas. A partír de ahí el error se estabiliza. En cuanto al número de hitos se observa que a partir de 50 partículas el número de hitos detectados es correcto, es decir 22 hitos (ver figura 10.4) siendo excesivo cuando se usa un número menor de partículas.

En la figura 10.92 se muestra el mapa de rejilla y el recorrido del robot calculados con 100 partículas, y el mapa basado únicamente en información odométrica. A medida que se complica el mapa, la cantidad de giros y el tamaño del recorrido que hay que realizar para poder cartografiarlo aumentan, con lo que el error inducido por la odometría es tambien mayor. Sin embargo, el agoritmo FastSLAM sigue siendo capaz de construir un mapa fiel a la realidad.

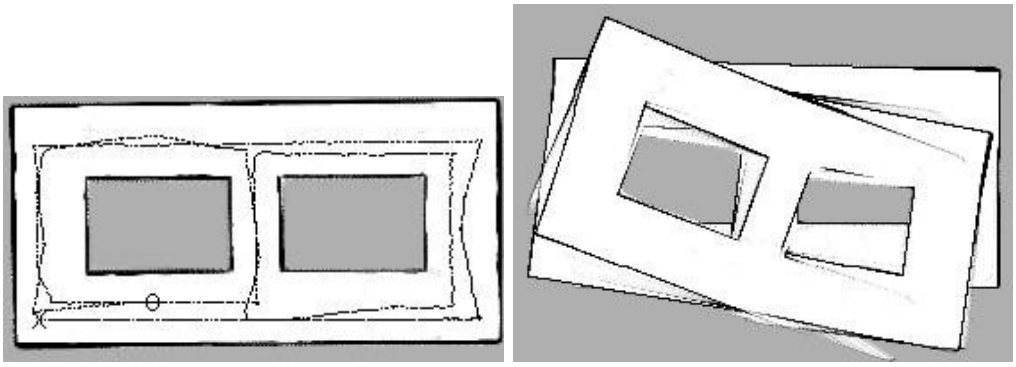


Figura 10.92: A la izquierda mapa de rejilla y recorrido para $Par = 100$ partículas, X punto inicial O punto final. A la derecha, mapa de rejilla basado en la odometría.

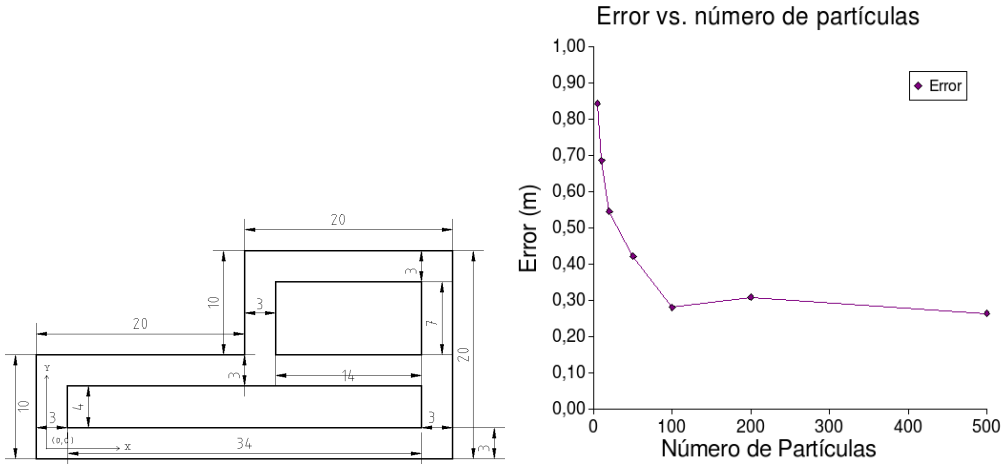
En la figura 10.93 puede observarse el mapa de un recinto con bucles complejos de grandes dimensiones, denominado *Complex*. Además en la misma figura puede observarse la evolución del error frente al número de partículas empleadas en la simulación.

Se observa una primera zona en la que el error disminuye drásticamente, hasta llegar a 100 partículas. A partir de ahí el error se estabiliza, por lo que no tiene sentido aumentar el número de partículas con el consiguiente aumento de esfuerzo computacional. En cuanto al número hitos detectados, en ningún caso se detecta el número exacto, es decir 26 hitos (ver figura 10.1), sin embargo a partir de 50 partículas el número de hitos detectados se aproxima.

En la figura 10.94 se muestra el mapa de rejilla y el recorrido del robot calculados con 150 partículas, y el mapa basado únicamente en información odométrica.

En general los mapas más complejos, es decir, *Bigloop2* y *Complex* necesitan como mínimo 100 partículas para ser cartografiados correctamente. Emplear un número mayor no aporta ventajas significativas. Estos entornos son los más próximos a situaciones reales, luego como punto de partida aconsejamos establecer el número de partículas a 100.

Otro aspecto interesante es analizar el error estacionario, es decir, el error para el que un aumento en el número de partículas no supone una disminución del mismo. En nuestro caso vamos a considerar que este error se produce cuando se emplean en torno a 100 partículas. Como se observa en la figura 10.95 el error es prácticamente igual para los mapas *Recinto Básico*, *Lineamedia*, *Bigloop*, y *Bigloop2*. Esto nos indica que el método es capaz de realizar



Par	5	10	20	50	100	200	500
Error	0,84	0,69	0,55	0,42	0,28	0,31	0,26
σ^2_{error}	0,0149	0,0413	0,1196	0,0091	0,0017	0,0107	0,0053
Hitos detectados	37,0	33,6	29,6	26,4	25,2	25,0	25,6
Hitos reales	26						

Figura 10.93: Mapa real, a la izquierda, unidades en metros. Error frente al número de partículas, a la derecha

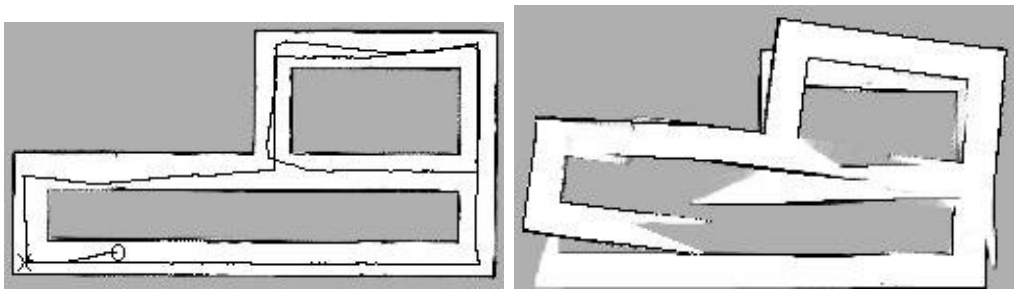
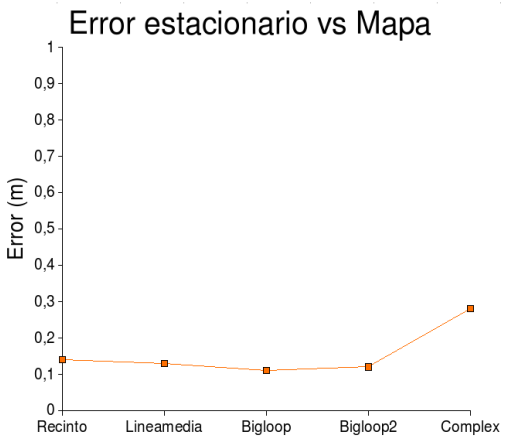


Figura 10.94: A la izquierda mapa de rejilla y recorrido para $Par = 20$ partículas, X punto inicial O punto final. A la derecha, mapa de rejilla basado en la odometría.

mapas con la misma precisión, ya sea de entornos simples o complejos. El caso del mapa *Complex* es un caso especial, diseñado para inducir situaciones más ambiguas. Fundamentalmente el gran pasillo de 40 metros, en el que no hay nigún otro hito salvo las rectas que lo delimitan, induce error en la localización en el eje X. Por esto este mapa presenta un mayor error estacionario. sin embargo este tipo de situaciones no es común en entornos reales, en los que los pasillos suelen estar jalonados de puertas, macetas, armarios, etc, hitos éstos que ayudan a la localización del robot y la disminución del error.

Para ilustrar el efecto del número de partículas en un entorno real vamos a emplear el



Mapa	Recinto	Lineamedia	Bigloop	Bigloop2	Complex
Error	0,14	0,13	0,11	0,12	0,28

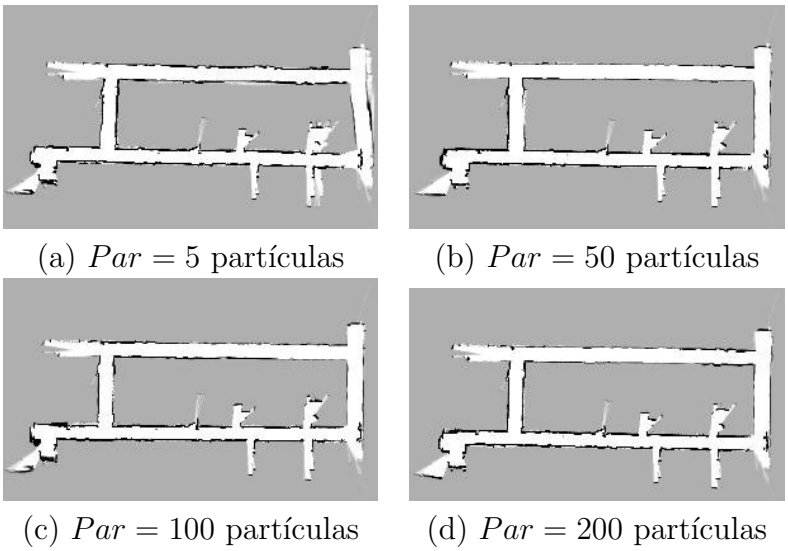
Figura 10.95: Error estacionario para distintos mapas.

mapa *USC SAL building*. Este mapa representa un entorno y recorrido reales fuera de nuestro control. La configuración empleada en todas las simulaciones se muestran en la figura 10.18.

Filtro		Split & Merge			Robot	Láser
N_{eff}	P0	P_{min}	D_{max}	L_{min}	Covarianza	Covarianza
∞	0.005	15	0.02	0.6	$\begin{pmatrix} 0{,}001 & 0 & 0 \\ 0 & 0{,}001 & 0 \\ 0 & 0 & 0{,}0008 \end{pmatrix}$	$\begin{pmatrix} 0{,}04 & 0 \\ 0 & 0{,}00122 \end{pmatrix}$

Tabla 10.18: Parámetros de las simulaciones para el estudio de *Par* en el mapa *USC SAL building*.

En la figura 10.96 se muestran los mapas de rejilla obtenidos con distinto número de partículas. El hecho más destacado es que, con los parámetros empleados, el número de partículas empleado no necesita ser muy elevado, ya que el mapa obtenido con 5 partículas se asemeja mucho a la realidad, mientras que con 50 partículas ya se obtiene el mapa correcto, no observándose mejoría al aumentar el número de partículas a 100 o 200. Esto indica que un ajuste cuidadoso del resto de parámetros que intervienen en el algoritmo FastSLAM, permite mantener el número de partículas relativamente bajo, con el ahorro computacional que esto implica. En cualquier caso este mapa está compuesto por rectas grandes y fácilmente distinguibles lo que facilita la aplicación del método.



(a) $Par = 5$ partículas (b) $Par = 50$ partículas

(c) $Par = 100$ partículas (d) $Par = 200$ partículas

Figura 10.96: Mapa de rejilla obtenido con distinto número de partículas.

10.7. Ajustes recomendados del algoritmo

Como resumen de todo lo visto en este capítulo, se muestra en la tabla 10.19 una configuración media de todos los parámetros que permite nuestra implementación del método FastSLAM. Esta configuración debe ser tenida en cuenta como un punto de partida a la hora de realizar un ajuste fino del algoritmo.

Parámetros	Valor	Observaciones
Split & Merge		Extracción de segmentos rectilíneos.
D_{max}	2 - 3 cm	Distancia máxima de un punto a la recta de la que forma parte.
L_{min}	0,5 - 2 m	Longitud mínima de un segmento válido.
P_{min}	15	Número de puntos mínimo de un segmento válido.
Modelo de moviento		
C	$\begin{pmatrix} [0{,}003 - 0{,}008] & 0 & 0 \\ 0 & [0{,}003 - 0{,}008] & 0 \\ 0 & 0 & 0{,}008 \end{pmatrix}$	Covarianza del error
Modelo de observación		
C	$\begin{pmatrix} [0{,}01 - 0{,}08] & 0 \\ 0 & [0{,}0008 - 0{,}003] \end{pmatrix}$	Covarianza del error
Filtro de partículas		
N_p	100-200	Número de partículas
P_0	0.005	Probabilidad de una nueva observación

N_{eff}	∞	Umbral del número de partículas efectivas (∞ = remuestrear siempre)

Tabla 10.19: Configuración recomendada para los parámetros del algoritmo FastSLAM.

Capítulo 11

Conclusiones

11.1. Introducción

La construcción de mapas viene siendo un campo de investigación muy activo en el ámbito de la robótica móvil durante las dos últimas décadas. El problema de la construcción de mapas se considera uno de los hitos más importantes en el camino hacia robots que sean auténticamente autónomos. Hoy en día se dispone de métodos robustos para la construcción de mapas en entornos estáticos, estructurados y de un tamaño limitado, sin embargo, construir mapas de entornos no estructurados, dinámicos y de grandes dimensiones continúa siendo un problema abierto a la investigación. Prácticamente todos los algoritmos actuales para la construcción automática de mapas mediante robots móviles son probabilísticos, y entre ellos destaca la familia de algoritmos estudiada en el presente proyecto, los algoritmos FastSLAM.

A lo largo del presente documento se ha estudiado en profundidad el algoritmo para la construcción automática de mapas FastSLAM. En este último capítulo se exponen las conclusiones obtenidas y se proponen líneas para el desarrollo futuro. Así en la sección 11.2 se reúnen las conclusiones alcanzadas a lo largo del proyecto, mientras que en la sección 11.3 se proponen una serie de líneas de trabajo futuro para enriquecer el funcionamiento del algoritmo.

11.2. Problemas versus solución

En el momento de comenzar el proyecto nos planteamos como primer objetivo el desarrollo de una versión funcional del algoritmo FastSLAM 1.0, para así poder probarlo, evaluarlo y comprobar los parámetros de su funcionamiento. Este primer objetivo has sido completado con éxito, y gracias a la implementación del método desarrollada ha sido posible estudiarlo, comprenderlo, y obtener una serie de conclusiones.

11.2.1. Generalidades

En primer lugar el método de FastSLAM es un método de SLAM que ajustado convenientemente funciona, incluso en entornos densamente ocupados y con robots cuya información odométrica es muy ruidosa. Como ejemplo se muestran en la figura 11.1 el mapa de rejilla obtenido empleado FastSLAM y el mapa de rejilla basado únicamente en la odometría, para un entorno y recorrido reales del laboratorio del IUSIANI.

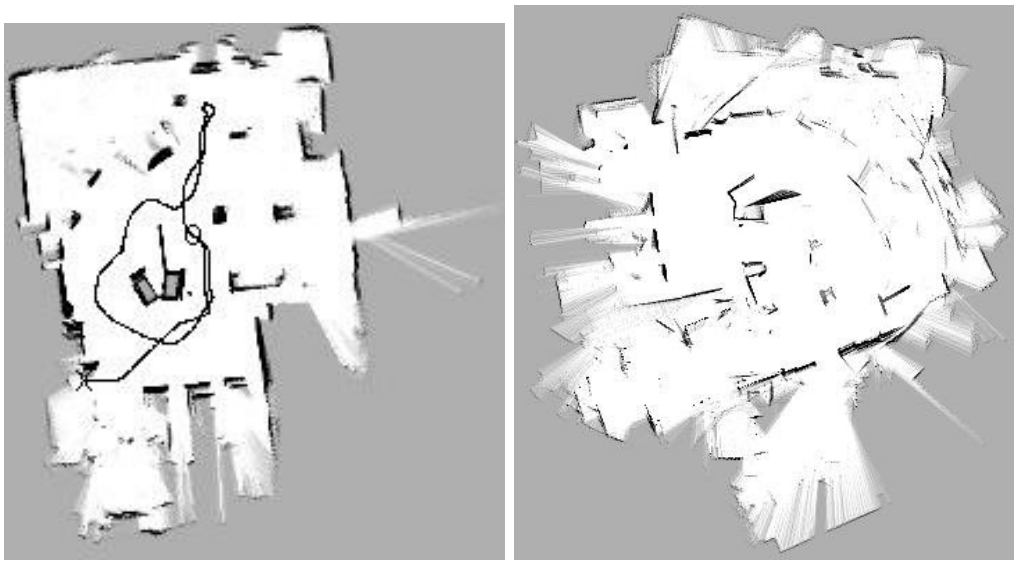


Figura 11.1: A la izquierda mapa de rejilla empleando FastSLAM. A la derecha, mapa de rejilla basado en la odometría. Obsérvese la enorme distorsión angular del mapa basado en odometría, en el que los elementos que es posible distinguir se encuentran rotados casi 90 grados con respecto al mapa real.

En todos los casos estudiados, con una adecuada configuración del algoritmo, el mapa de rejilla de la mejor partícula en el momento de la detención del proceso de cartografía es apto para la navegación.

11.2.2. Tipos de hitos referenciales

En cuanto a la detección de hitos referenciales, estudiada en el capítulo 4:

- Las rectas infinitas y las esquinas han resultado ser adecuadas y suficientes para la cartografía de entornos interiores de tipo oficina.
- El empleo de rectas como hitos referenciales es imprescindible para el funcionamiento del método, son la evidencia que va a soportar el proceso. Además son suficientes, es decir, no es necesario emplear la información que aporta la detección de esquinas para obtener mapas correctos.

- La información extra que aportan las esquinas produce una convergencia más rápida del método. Aun no siendo imprescindibles, las esquinas son mucho menos ambiguas que las rectas, siendo poco común encontrar dos esquinas muy próximas que puedan provocar una asociación de datos errónea, y en este sentido aportan mucha información. No obstante, por ser las esquinas puntos, su observación se produce esporádicamente, cuando el robot se encuentra en las proximidades de la esquina, mientras que las rectas, gracias a su extensión, pueden ser observadas mucho más frecuentemente.

11.2.3. Split & Merge

El método de extracción de segmentos rectilíneos de un barrido láser Split & Merge es un método fiable, robusto, y sencillo, tanto conceptualmente como a nivel de implementación. No obstante, el método es muy sensible a alguno de sus parámetros:

- D_{max}. Valores muy pequeños conducen a la detección escasa de segmentos, al exigir que éstos se ajusten mucho a una recta teórica. Valores excesivamente grandes producen la detección de rectas falsas, al relajarse excesivamente la exigencia de alineación. Un valor adecuado dependerá del ruido de medida y el tipo de rectas presentes en el entorno.

- L_{min}. Valores muy pequeños conducen a la detección de rectas falsas, ya que cualquier segmento rectilíneo, por pequeño que sea, será detectado, mientras que valores muy grandes producen la no detección de rectas, al exigir que éstas sean muy grandes. En general el valor dependerá del tipo de rectas presentes en el entorno, y nunca debe superar los límites impuestos por el propio rango de percepción del sensor empleado para medirlo.

- P_{min}. Valores pequeños conducen a la detección de rectas falsas, ya que no se exige que estén soportadas por un número suficiente de evidencias. Por el contrario valores excesivamente grandes provocarán la no detección de rectas, al ser excesiva la exigencia.

- $(\rho_{max}, \theta_{max})$. El método es poco sensible a los parámetros de colinealidad. Para valores muy pequeños no se encuentran segmentos colineales prácticamente nunca, por lo que no se produce fusión de segmentos a nivel de barrido láser. No obstante, no se produce un incremento de las rectas presentes en el mapa final. Esto se debe a que el algoritmo de asociación de datos realiza lo que podríamos llamar una fusión de datos entre las observaciones y el mapa, de modo que todas aquellas rectas muy similares que se han detectado en el barrido láser terminan siendo asociadas con el mismo hito en el mapa, y por ello no hay una proliferación de rectas en el mismo. Si se produce un incremento notable en la detección de esquinas, ya que al no fusionarse segmentos, se detectan muchas intersecciones, y estas no son tan fácilmente *absorbidas* por la asociación de datos. Para valores excesivamente grandes se fusionan segmentos que en realidad son rectas completamente distintas, con lo que las rectas resultantes son erróneas y el proceso de SLAM no funciona adecuadamente. Existe un amplio rango de valores para los que se obtienen buenos resultados. Realmente para que estos parámetros influyan negativamente, ambos deben tomar valores elevados, y poco racionales.

11.2.4. Detección de esquinas

La detección de esquinas realizando la intersección de segmentos rectilíneos y empleando un umbral de distancia máxima a las rectas intersectadas, según se explica en la sección 4.3, es un método fiable, robusto, y fácil de implementar. El hecho de emplear un único parámetro también es una ventaja. A lo largo de las pruebas mostradas en la sección 10.4.5 se ha constatado que un valor del umbral de distancia máxima muy pequeño con lleva la detección pocas esquinas, mientras que para un valor excesivamente grande se detectan esquinas falsas y se puede llegar a producir una incorrecta cartografía.

11.2.5. Ruidos en el modelo de movimiento

Los ruidos considerados en el modelo de movimiento tienen una influencia grande sobre el proceso de SLAM.

- Los ruidos considerados en el modelo de movimiento deben ser como mínimo iguales a los ruidos máximos reales presentes en la odometría, para así poder generar en cada paso una población de partículas cuyas poses cubran todo el espacio de posibilidades.
- El encontrar el ruido que produce buenos resultados puede considerarse como una calibración de la odometría.
- En general, un incremento en el ruido considerado en el modelo debe venir acompañado por un incremento en el número de partículas del filtro, de modo que se disponga de más partículas para cubrir un mayor espacio de posibilidades en la pose del robot.
- Como comentan los autores del método FastSLAM [40, 66], el ruido en la orientación del robot tiene más influencia que el ruido en la posición.

11.2.6. Ruidos en el modelo de observación

Los medidores de rango láser suelen ser instrumentos precisos, por lo que conviene obtener información de los fabricantes acerca de los ruidos que les afectan. A partir de esta información hay que realizar un ajuste fino.

- Los ruidos considerados en el modelo de observación no pueden ser compensados por ningún otro parámetro. Como se ha comentado en la sección anterior, el considerar en el modelo de movimiento un ruido mayor que el real puede llegar a provocar el incorrecto funcionamiento del proceso de SLAM. Sin embargo, un aumento en el número de partículas del filtro puede compensar este efecto (los resultados experimentales se muestran en la sección 10.5.1). Esto no ocurre para el modelo de observación, lo que obliga a que el ajuste de los ruidos considerados en él deba ser más cuidadoso.
- Una vez ajustada la componente de rango, la componente angular del ruido influye notablemente en el proceso, ya que no es extraño encontrar rectas próximas cuyos

ángulos difieren ligeramente, y, según el error considerado, pueden ser asociadas incorrectamente provocando la divergencia del método.

- La componente de rango, una vez ajustada la componente angular, influye menos ya que no es tan común encontrar rectas paralelas y próximas, con lo que disminuye la probabilidad de realizar una asociación de datos incorrecta.

11.2.7. Asociación de datos

El esquema de asociación de datos empleado es un esquema sencillo y conservador. Tal como se explica en la sección 5.4.2, si en algún momento se detecta que un hito observado podría asociarse plausiblemente a varios hitos del mapa, se ignora dicha observación. Además frente a la observación de múltiples hitos en el mismo barrido láser, no se están teniendo en cuenta restricciones geométricas (ángulos relativos, distancias, etc, $\cdots$).

- Aun con este esquema sencillo, una vez ajustados convenientemente los ruidos de los modelos, la asociación de datos empleando el método de *máxima similitud* funciona como cabría esperar, asociando correctamente las observaciones con sus hitos correspondientes en el mapa.

- El funcionamiento de la asociación de datos depende de los ruidos de los modelos de movimiento y observación, que son los que determinan la incertidumbre asociada un hito observado, y a uno presente en el mapa.

- La *similitud* que se calcula para cada partícula, y que dirige el proceso de remuestreo, efectivamente es mayor para aquellas partículas que mejor explican la observación, con lo que en general sobreviven las partículas más cercanas a la realidad.

11.2.8. Filtro de partículas

En cuanto al filtro de partículas existen tres aspectos sobre los que es posible extraer conclusiones:

- A pesar de la importancia que se da al proceso de remuestreo en la literatura acerca de los filtros de partículas, hemos encontrado que el umbral para el N_{eff}, que determina cuándo debe realizarse el remuestreo, no tiene prácticamente ninguna influencia, por lo que es posible remuestrear siempre. Esto se debe a que el tipo de remuestreo empleado, denominado *remuestreo secuencial o de baja varianza*, tiene la propiedad de mantener siempre la variedad en el conjunto de partículas [67].

- En cuanto a la probabilidad asignada a la observación de un nuevo hito P_0, se ha comprobado que mientras se mantenga por debajo de un cierto umbral su influencia en el proceso es nula. El rango de valores que toma la similitud de la observación de un hito, va desde valores superiores a 1 para un hito que se corresponde con un hito del mapa, hasta valores tan pequeños como 10^{-270} para un hito que no ha sido previamente

observado. Por lo tanto cualquier valor de P_0 dentro de este rango no interfiere en el proceso de asociación de datos.

- Por último, se ha corroborado que el incremento del número partículas produce una disminución en el error de los mapas obtenidos, si bien el empleo de más de 200 partículas no ha sido necesario para conseguir cartografiar adecuadamente ninguno de los entornos estudiados, ni sintéticos, ni reales.

11.3. Trabajo Futuro

A lo largo del desarrollo del presente proyecto, y gracias a los concimientos adquiridos sobre el proceso de SLAM en general, y del método FastSLAM en partícular, se han identificado una serie de posibles líneas de trabajo futuro. En los siguientes apartados se desarrollan las que hemos considerado más prometedoras.

11.3.1. Extracción de hitos referenciales

- A través de la experiencia obtenida a lo largo del desarrollo del presente proyecto, el desarrollo que consideramos tiene más potencial para mejorar el algoritmo es la inclusión de detectores de otro tipo de hitos, fundamentalmente la sustitución de rectas infinitas por rectas cuya extensión espacial esté delimitada. Esto eliminaría mucha ambigüedad a la hora de realizar la asociación de datos.

- En cuanto al tipo de hitos detectados y la información que se registra sobre ellos las posibilidades son infinitas. Si se añadieran otro tipo de sensores como una cámara, sería posible *anotar* los hitos con propiedades como el color o la textura que los acompaña, lo que aportaría una información interesante a la hora de evitar la ambigüedad en la asociación de datos.

11.3.2. Asociación de datos

En nuestra implementación ante asociaciones de datos ambiguas, es decir, cuando una observación puede asociarse plausiblemente con varios hitos del mapa, simplemente ignoramos dicha observación. Frente a esta situación en [45] se propone:

- Crear nuevas partículas, cada una de ellas con una de las diferentes hipótesis posibles. De este modo la propia evolución del filtro mediante remuestreo permitirá la supervivencia futura de aquellas partículas que hayan recibido asociaciones correctas, y descartará aquellas con asociaciones incorrectas.

11.3.3. FastSLAM 2.0

Mejoras como las comentadas anteriormente podrían hacer uso de casi la totalidad del software desarrollado en el presente proyecto.

Planteamientos más radicales, pero a los que también se podrían aplicar los dos desarrollos futuros mencionados anteriormente, serían la implementación de la versión 2.0 propuesta por los autores [40]. Esta versión soluciona principalmente el problema de la degeneración de las partículas del filtro, requiriendo, en las mismas condiciones, el empleo de un menor número de partículas.

La versión 1.0 únicamente considera los controles ejecutados por el robot a la hora de predecir la pose de las distintas partículas, de modo que, si la odometría es muy ruidosa, un gran número de partículas recibirán poses alejadas de la realidad. La versión 2.0 incorpora, además de los controles ejecutados, la información de la observación realizada, a la hora de predecir la pose de las distintas partículas, y de este modo consigue que más partículas reciban poses realmente plausibles, con lo que disminuye el peligro de degeneración del filtro de partículas. La incorporación de la información de la observación realizada eleva la complejidad, tanto desde el punto de vista matemático, como de implementación.

11.3.4. Integración en un sistema robótico real

Una vez alcanzado un método robusto, el paso lógico consiste en la integración del método en un sistema robótico real. Ésta no es una cuestión trivial, entre otras cosas hay que decidir:

- En qué condiciones se considera que el mapa ya es apto para su uso en navegación.
- Qué recorrido seguir para obtener el mapa.
- Qué hacer si se detecta que el mapa es incorrecto.
- Cómo integrar el mapa en un sistema de consecución de objetivos.

La última cuestión nos abre otra vía de investigación futura interesante, la obtención de mapas que contengan abstracciones de más alto nivel, como puertas, fuentes de energía para el robot, destinos de entrega de mercancías, etc. Combinando esta información junto con la información para navegación, sería posible planificar estrategias para el logro de objetivos.

11.3.5. Entornos dinámicos

Por último otra gran conjunto de mejoras lo encontramos en el estudio de entornos dinámicos. Los entornos en los que un robot móvil es potencialmente útil son dinámicos por naturaleza. Puertas que se abren y se cierran, humanos u otros robots que se mueven dentro del rango de percepción del robot, cajas o sillas que cambian de posición, todos son elementos comunes de cualquier entorno en el que se esté llevando a cabo una actividad. En este sentido sería interesante estudiar:

- La implantación de un sistema que permitiera clasificar los hitos detectados en el entorno entre estructurales y potencialmente dinámicos, posiblemente con grados intermedios.

- La integración de esta clasificación con la asociación de datos de modo que permita al robot cartografiar y localizarse en entornos dinámicos.
- Elaboración de mecanismos de eliminación o modificación de hitos del mapa, que tuvieran en cuenta los posibles elementos dinámicos.

11.3.6. Intercambio de información

Hemos comprobado prácticamente la inexistencia de unos estándares mínimos en la representación de plataformas robóticas, sensores o datos registrados durante recorridos. En este sentido consideramos verdaderamente útil promover el desarrollo de unos estándares XML para este tipo de representaciones, que permitan una comunicación más fluida entre los distintos grupos investigadores, y unas herramientas más fáciles de configurar y manejar.

11.3.7. Propuesta de línea de trabajo

Personalmente considero que la secuencia de mejoras a realizar comienza con la consecución de un sistema real que sea capaz de desenvolverse en entornos estáticos, explorando, persiguiendo unos objetivos, etc. Una vez conseguido un sistema de este tipo, y con la experiencia adquirida, se daría el paso a entornos dinámicos.

Apéndice A

Formato de archivos de datos

Como información añadida al proyecto, en el presénte apéndice se van a mostrar los distintos formatos de archivos de datos utilizados. Como archivo de datos nos referimos a los archivos de registro que alimentan la ejecución del algoritmo y a los archivos que contienen los hitos del mapa de la mejor partícula en un momento dado. Los archivos de registro pueden contener diversas informaciones, a nosotros nos va a interesar únicamente la información odométrica y la información sobre barridos láser.

En primer lugar en el apéndice A.1 se expone el formato de los archivos generados con la herramienta Player/Stage, seguidamente, en el apéndice A.2, se muestra el formato de los archivos de datos del mapa *USC SAL Building*. Además en el apéndice A.3 se presenta un formato más sencillo, y por lo tanto cuyo procesamiento es más rápido, que hemos denominado *formato normalizado*. Por último en el apéndice A.4 se expone el formato de los archivos que contienen los mapas exportados por la aplicación.

A.1. Formato Player/Stage

Los archivos de registro Player tienen un formato de texto sencillo. Cada mensaje ocupa una linea. Las líneas que comienzan por # son ignoradas. Cada nuevo mensaje comienza con una serie de metadatos comunes:

```
time host robot interface index type subtype
```

Los campos son:

- time(double): Marca temporal en segundos.
- host(uint): La parte del host de la dirección del dispositivo Player.
- robot(uint): La parte del robot de la dirección del dispositivo Player.
- interface(string): La parte de la interface de la dirección del dispositivo Player.
- index(string): La parte del índice de la dirección del dispositivo Player.

- type(uint): El tipo de mensaje.
- subtype(uint): El subtipo de mensaje.

Siguiendo a esta meta-información viene la verdadera información del mensaje, siguiendo un formato específico para cada tipo y subtipo de mensaje:

- Para la odometría el tipo y subtipo deben valer ambos 1. El formato de estos mensajes es:

  `[metainformación] px py pa vx vy va stall`

  - px(float): Coordenada X en metros de la pose del robot.
  - py(float): Coordenada Y en metros de la pose del robot.
  - pa(float): Orientación del robot en radianes.
  - vx(float): Componente X de la velocidad del robot en m/s.
  - vy(float): Componente Y de la velocidad del robot en m/s.
  - va(float): Velocidad angular del robot en rad/s.
  - stall(int): Bandera (flag) para indicar el estado del motor.

- Para los barridos láser el tipo y el subtipo deben valer ambos 1. El formato de estos mensajes es:

  `[metainformacion] scan_id min_angle max_angle resolution max_range count readings`

  - scan_id(int): Identificador único del barrido láser.
  - min_angle(float): Ángulo mínimo del barrido en radianes.
  - max_angle(float): Ángulo máximo del barrido en radianes.
  - resolution(float): Resolución angular en radianes.
  - max_range(float): Rango máximo en metros.
  - count(int): Número de lecturas.
  - readings: Lista de n lecturas con el siguiente formato: `range intensity`
    - range(float): Rango en metros.
    - intensity(int): Intensidad.

A.2. Formato Radish

La información se encuentra dividia en líneas de texto plano. Cada línea es una unidad completa de información, ya sea odométrica, láser, o de otro tipo:

- Línea de información odométrica:

  `position 0 time px py pa vx vy va`

  - time(float): Marca de tiempo.
  - px(float): Coordenada X en metros de la pose del robot.
  - py(float): Coordenada Y en metros de la pose del robot.
  - pa(float): Orientación del robot en radianes.
  - vx(float): Componente X de la velocidad del robot en m/s.
  - vy(float): Componente Y de la velocidad del robot en m/s.
  - va(float): Velocidad angular del robot en rad/s.

- Línea de información láser:

  `laser 0 time readings`

  - time(float): Marca de tiempo.
  - readings: Lista de n lecturas con el siguiente formato: `range bearing intensity`
    - range(float): Rango en metros.
    - bearing(float): Ángulo en radianes.
    - itensity(int): Intensidad.

A.3. Formato Normalizado

La información se encuentra dividia en líneas de texto plano. Cada línea es una unidad completa de información, ya sea odométrica, láser, o de otro tipo:

- Línea de información odométrica:

  `ODOM time V W`

  - time(float): Marca temporal.
  - V(float): Velocidad lineal en m/s.
  - W(float): Velocidad angular en rad/s.

- Línea de información láser:

  `LASER time n readings`

  - time(float): Marca temporal.
  - n(int): Número de lecturas.
  - readings: Lista de n lecturas con el siguiente formato: `x y`
    - x(float): Coordenada x del punto detectado en metros.
    - y(float): Coordenada y del punto detectado en metros.

A.4. Formato de los mapas

El software desarrollado y que permite simular la ejecución del algoritmo FastSLAM 1.0 en diversas condiciones, ofrece la posibilidad de almacenar las características que componen el mapa de la mejor partícula en un archivo de texto plano. Este es el mismo formato que se emplea en los archivos que definen los hitos presentes en los mapas diseñados por nosotros, y que permiten calcular el error entre los mapas elaborados por FastSLAM y los mencionados mapas diseñados. Este archivo almacena información sobre una característica en cada línea, de modo que encontramos dos tipos de líneas:

- Datos sobre rectas:

  `rho: rango theta: angulo`

  - rango(float): Rango en coordenadas polares, en metros.
  - angulo(float): Ángulo en coordenadas polares, en grados.

- Datos sobre esquinas:

  `x: cx y: cy`

  - cx(float): Componente x en coordenadas rectangulares, en metros.
  - cy(float): Componente y en coordenadas rectangulares, en metros.

Bibliografía

[1] *Java 2, edición 2000*. Ediciones Anaya Multimedia (Grupo Anaya, S.A.), 2000.

[2] *PHP 4*. Pearson Educación, S.A., 2000.

[3] K. Arras and R. Siegwart. Feature extraction and scene interpretation for map-based navigation and map building, 1997.

[4] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.

[5] G. Booch. *Object-Oriented Analysis and Design with Applications, Second Edition*. Benjamin/Cummings, 1993.

[6] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[7] G. A. Borges. A split-and-merge segmentation algorithm for line extraction in 2-d range images. In *ICPR '00: Proceedings of the International Conference on Pattern Recognition*, page 1441, Washington, DC, USA, 2000. IEEE Computer Society.

[8] Premebida C. and Nunes U. Segmentation and geometric primitives extraction from 2d laser range data for mobile robot applications. In *Robótica 2005: Scientific meeting of the 5th National robotics festival*.

[9] R. Chatila and J. P. Laumond. Position referencing and consistent world modelling for mobile robots. In *1985 IEEE International Conferencen on Robotics and Automation*, 1985.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm.

[11] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[12] A. Elfes. Sonar-based real-world mapping and navigation. pages 233–249, 1990.

[13] Alberto Elfes. *Occupancy grids: a probabilistic framework for robot perception and navigation*. PhD thesis, 1989.

[14] Free Software Foundation. The gnu operating system. http://www.gnu.org/.

[15] Manuel García. Cartografía. http://www.mgar.net/var/cartogra.htm, 2007.

[16] RibbonSoft GmbH. The cad system for everyone. http://www.ribbonsoft.com/qcad.html.

[17] J. Gosling and H. McGilton. The java language environment. http://java.sun.com/docs/white/langenv/, 1997.

[18] Neil Gray and Neil A. Gray. *Web Server Programming*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[19] Object Management Group. Unified modeling language (uml), v1.4.2. http://www.omg.org/cgi-bin/apps/doc?formal/05-04-01.pdf.

[20] Object Management Group. Mof 2.0 / xmi mapping specification v2.1. http://www.omg.org/technology/documents/formal/xmi.htm, 2007.

[21] TeX Users Group. Just what is tex? http://www.tug.org/whatis.html.

[22] I. Hwang, H. Balakrishnan, K. Roy, J. Shin, L. Guibas, and C. Tomlin. Multiple-target tracking and identity management. Dept. of Aeronautics and Astronautics, Stanford University.

[23] MobileRobots Inc. Pioneer p3-dx. http://www.activrobots.com/ROBOTS/p2dx.html, 2006.

[24] Sun Microsystems Inc. and CollabNet Inc. Netbeans ide. http://www.netbeans.org/index.html.

[25] Wikipedia Foundation Inc. Cartography. http://en.wikipedia.org/wiki/Cartography, 2007.

[26] Wikipedia Foundation Inc. History of cartography. http://en.wikipedia.org/wiki/History_of_cartography, 2007.

[27] Hammersley J.M. and Handscomb D.C. *Monte Carlo methods*. Methuen's monographs on applied probability and statistics. Methuen & Co., 1967.

[28] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[29] H. Kopka. *A guide to LaTEX2e : document preparation for beginners and advanced users*. Addison-Wesley, Wokingham,England, 2nd. ed. edition, 1995.

[30] Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy ofSpatial representations. Technical Report AI90-120, 1, 1990.

[31] Csorba M. *Simultaneous Localization and Map Building*. PhD thesis, Department of Engineering Science, University of Oxford, 1997.

[32] William G. Madow. On the theory of systematic sampling, ii. *The Annals of Mathematical Statistics*, 20:333-354, 1949.

[33] William G. Madow. On the theory of systematic sampling, iii. *The Annals of Mathematical Statistics*, 24:101-106, 1953.

[34] William G. Madow and Lillian H. Madow. On the theory of systematic sampling, i. *The Annals of Mathematical Statistics*, 14:1-24, 1944.

[35] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Technical report, Cambridge, MA, USA, 1990.

[36] Sun Microsystems. Free office suite. http://www.openoffice.org/.

[37] Scott Mitchell. *Designing Active Server Pages*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000.

[38] M. Montemerlo and S. Thrun. Simultaneous localization and maping with unknown data association using fastslam. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*.

[39] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[40] Michael Montemerlo and Sebastian Thrun. *FastSLAM. A scalable method for the simultaneous localization and mapping problem in robotics.* Springer Tracts in Advanced Robotics 27. Berlin: Springer. xv, 119 p. , 2007.

[41] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. pages 253–276, 1989.

[42] K. Murphy. Bayesian map learning in dynamic environments. In MIT press, editor, *Advances in neural information processing systems (NIPS)*, 1999.

[43] J. Neira and J.D. Tardós. Data association in stochastic mapping using the joint compatibitity test. *IEEE Transactions on robotics and automation*, 2001.

[44] Viet Nguyen, Agostino Martinelli, Nicola Tomatis, and Roland Siegwart. A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics. In *Proceedings of the IEEE/RSJ Intenational Conference on Intelligent Robots and Systems, IROS, Edmonton, Canada*. IEEE, 2005.

[45] J. Nieto, J. Guivant, E. Nebot, and S. Thrun. Real time data association for fastslam, 2003.

[46] National Institute of Standards and Technology. Jama: A java matrix package. http://math.nist.gov/javanumerics/jama/, 2005.

[47] Mourad Oussalah. Towards probabilistic jpdaf. K.U. of Lueven. Belgium.

[48] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. Mc-Graw Hill, 1984.

[49] T. Pavlidis and S. L. Horowitz. Segmentation of plane curves. *IEEE Trans. on Computers*, (C-23):860–870, 1974.

[50] Havoc Pennington. *GTK+/Gnome application development*. New Riders Publishing, 1999.

[51] The Player Project. Free software tools for robot and sensor applications. http://playerstage.sourceforge.net/.

[52] D. Reid. An algorithm for tracking multiple targets. In *IEEE Transactions on Automatic Control*, 1979.

[53] SICK Optic-Electronic S.A. Laser measurement sensors, lms-200. http://mysick.com/partnerPortal/eCat.aspx?go=DataSheet&ProductID=9168.

[54] D. Sack and W. Burgard. A comparison of methods for line extraction from range data. In *Proc. of the 5th IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*, 2003.

[55] H. Schildt. *Java 2: The Complete Reference, fifth edition*. McGraw-Hill/Osborne, 2600 Tenth Street, Berkeley, California 94710, USA, 2002.

[56] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Book, 2004.

[57] Open SLAM. Open slam repository. http://www.openslam.org/.

[58] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. pages 167–193, 1990.

[59] Randall Smith, Matthew Self, and Peter Cheeseman. A stochastic map for uncertain spatial relationships. In *on The fourth international symposium*, pages 467–474, Cambridge, MA, USA, 1988. MIT Press.

[60] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainly. *Int. J. Rob. Res.*, 5(4):56–68, 1987.

[61] A. Sokal. Monte carlo methods in statistical mechanics: Foundations and new algorithms.

[62] Ian Sommerville. *Software engineering*. Addison-Wesley, Boston, 7th ed. edition, 2004.

[63] Clemens Szyperski. *Component software*. Addison-Wesley, Harlow [u.a.], 1998.

[64] The Carmen Team. Carmen robot navigation toolkit. http://carmen.sourceforge.net/.

[65] The Gimp Team. The gnu image manipulation program. http://www.gimp.org.

[66] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002. to appear.

[67] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, Cambridge, Massachusetts, London, England, 2005.

[68] Tigris.org Open Source Software Engineering Tools. Argouml features. http://argouml.tigris.org/features.html, 2006.

[69] World Wide Web Consortium W3C. Extensible markup language (xml). http://www.w3.org/XML/.

[70] G. Welch and G. Bishop. An introduction to the kalman filter, Mayo 2006.

Índice alfabético