UNIVERSIDAD REY JUAN CARLOS

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso académico 2004-2005

Proyecto Fin de Carrera

Localización del robot Pioneer basada en láser.

Tutor: José M. Cañas Plaza

Autor: Redouane kachach

A mis Padres y Hermanos

Que estarían muy orgullosos de poder ver esto

A mi novia y mis amigos

Agradecimentos.

Quiero dar las gracias a todo el grupo de robótica de la URJC. De manera especial a Jose María Cañas por su confianza puesta en mi, sus conocimientos facilitados, su apoyo y paciencia, que han sido esencial para llevar a cabo este proyecto.

Dentro del grupo de robótica, agradecer también a Pablo, Paco, Víctor, y Carlos por su apoyo y a mis compañeros Alberto, Antonio y Pedro por todo el tiempo que hemos pasado juntos en el laboratorio.

A mi familia y mis hermanos que estarían encantados de estar conmigo en este momento, por las facilidades y el apoyo recibido durante esta larga trayectoria sin cansarse ni un minuto. A mi novia por su paciencia, apoyo y por estar siempre ahí.

- A mis amigos de siempre: Amine, Bilal, Achraf, Daimossi, Mohammed, Khalid
- A mis amigos de LTMY: Ali, Okba, Chawki, Bakali, Hilal, Tarik, Rwichi...
- A mis amigos de la URJC: Javi, JuanPe, Siro, Vane, Victor, Isma, Raul, Antonio, Ruben, Elena, Manu, Toni, Oscar, Jacob, Ivan...
- A mis amigos de la ENSAT: Abd el Hay, Ahamad, Bakali, Kamal...

Gracias a todos por el apoyo y los buenísimos ratos que hemos pasado juntos.

Índice general

Resumen **1**

1. Introducción **2**

1.1. Robótica 2

1.2. localización 4

1.2.1. Algoritmos de localización 5

1.3. Localización basada en láser 9

2. Objetivos **11**

2.1. Objetivos 11

2.2. Requisitos 12

2.3. Metodología 12

2.3.1. Desarrollo en espiral 13

3. Plataforma de desarrollo **15**

3.1. Robot Pioneer 15

3.1.1. El Sensor láser 16

3.1.2. Los odométros 17

3.2. Arquitectura JDE para aplicaciones robóticas. 17

3.2.1. API jde.c 18

3.2.2. Servidores 18

3.2.3. La plataforma Player/Stage 19

3.3. Librerías auxiliares 20

3.3.1. Gridslib 21

3.4. Xforms 21

4. Localización con mallas de probabilidad **23**

4.1. Fundamentos teóricos 23

4.2. Diseño general del algoritmo 25

4.3. Modelo de observación láser 29

4.4. Modelo de observación odométrica 33
4.5. Acumulación de evidencias con la regla de Bayes 37
4.6. Visualización 41

5. Localcización con el filtro de particulas 44
5.1. Fundamentos teóricos del filtro de Partículas 44
5.2. Diseño general y algoritmo 45
5.3. Modelo de observación láser 47
5.4. Modelo de movimiento 48
5.5. Remuestreo 50
5.6. Visualización 54

6. Experimentos 55
6.1. Resultados con mallas de probabilidad 55
6.1.1. Resolución y tamaño del cubo de probabilidad 55
6.1.2. Número de orientaciones 57
6.1.3. Modelo de observación 58
6.1.4. Simetría 60
6.1.5. Experimentos *sobre el robot real* 62
6.2. Resultados con filtro de partículas 63
6.2.1. Número de partículas 63
6.2.2. Ruido gaussiano 64
6.2.3. Experimentos sobre el robot real 65
6.3. Comparativa entre los algoritmos 66

7. Conclusiones y trabajos futuros 68
7.1. Conclusiones 68
7.2. Trabajos futuros 70

Índice de figuras

1.1. La sonda SPIRIT 4
1.2. Sensores de posición 6
1.3. Localización probabilística con trilateración. 6
1.4. Localización probabilística. 7

2.1. Modelo en espiral 13

3.1. Pioneer 3x 16
3.2. EL sensor láser Sick. 17
3.3. JDE 19

4.1. Plataforma JDE. 26
4.2. pseudocódigo del esquema laserloc-mallas 27
4.3. Estructuras básicas del algoritmo 29
4.4. los ángulos α y θ 30
4.5. observación teórica calculada sobre el mapa de ocupación 32
4.6. incorporación de observación 33
4.7. Double buffering 36
4.8. incorporación del movimiento 37
4.9. $y = ln(\frac{x}{1-x})$ 39
4.10. Mapa de la habitación 39
4.11. Ejecución tipica de localización con mallas de probabilidad 40
4.12. gui_laser_loc 43

5.1. Esquemas. Filtro de partículas 46
5.2. Flujo de control del filtro de partículas 47
5.3. Incorporación de observación láser 48
5.4. Efecto del ruido gaussiano 50
5.5. Incorporación de movimiento 50
5.6. Proceso de remuestreo 52
5.7. Ejecución tipíca del filtro de particulas (simulador) 53

5.8. gui_laser_loc para partículas 54

6.1. posibles orientaciones en el mundo (simulador) 58
6.2. Mapa de la habitación 59
6.3. Modelo lineal de observación láser 59
6.4. Modelo lineal por tramos regulares de observación láser 59
6.5. Modelo por tramos irregulares, de observación láser 59
6.6. Localización en entornos con baja simetría (simulador) 60
6.7. Localización en entronos de simetría media (simulador) 61
6.8. Localización en entornos con alta simetría (simulador) 61
6.9. Localización en simetria media - robot real 62
6.10. 63
6.11. 65
6.12. Ejecución tipíca del filtro de partículas (robot real) 66

Índice de cuadros

6.1. efecto del tamaño de la rejilla en los tiempos (simulador) 56
6.2. efecto del número de orientaciones sobre los tiempos(simulador). 57

Resumen

Para que un robot podrá navegar de forma autónoma dentro de un entorno, necesita saber en cada momento con exactitud su posición dentro de este, de ahí la necesidad de la localización. Es un problema clásico dentro del área de la robótica móvil, se trata de que el robot podrá estimar de forma autónoma su posición dentro del mundo, apoyandose sólo en la información de sus lecturas sensoriales, y información del entorno que en general la recibe en forma de un mapa.

Este proyecto tratará de resolver este problema usando técnicas probabilísticas pensadas para enfrentar este problema y tratarlo de manera eficaz. *La localización probablística* es un conjunto de técnicas que permiten al robot localizarse incluso sin saber su posición inicial, hay un abanico amplio de estas técnicas, aquí se experimentaran dos algoritmos probablísticos distintos, con el fin de hacer una comparativa entre los dos, y sacar conclusiones respecto a las ventajas y desventajas de cada uno. El primer método es el de "mallas de probabilidad", esta técnica se basa en mantener una función de densidad de probabilidad para todas las posiciones posibles dentro del entorno. El segundo es el "filtro de partículas", se basa en muestrear un conjunto de partículas(posiciones) elegidas al azar sobre todo el mundo, esta técnica pertenece a la familia de métodos de Monte Carlo, que es un conjunto de técnicas de muestreo estadístico.

La implementación se ha hecho basandose en JDE[Plaza, 2003], una arquitectura software desarrollada íntegramente en el grupo de robótica de la URJC. Esta plataforma ofrece acceso sencillo a los sensores y actuadores del robot, y pone a disposición del programador una serie de mecanismos para crear aplicaciones robóticas de manera sencilla.

Capítulo 1

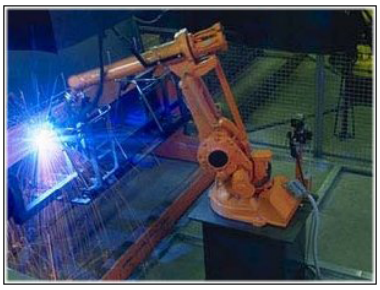
Introducción

Dentro del campo de robótica hay muchos problemas para resolver, pero en general se pueden dividir en tres o cuatro problemas clásicos. Uno de estos problemas es la localización. Para que un robot podrá moverse en un entorno de forma segura, tiene que saber en cada momento su posición dentro de este mundo. La localización consiste en que el robot podrá estimar su posición de forma autónoma, partiendo sólo de su información sensorial, y la del mundo que le rodea. Para resolver este problema Se han ideado varios algoritmos, que se diferencian dependiendo del entorno donde el robot intentará localizarse.

1.1. Robótica

A lo largo de la historia, el hombre siempre se apoyaba en la ciencia para buscar una vida más fácil, para ello fabricaba herramientas o maquinas que le ayudan a descargarse del trabajo, estas maquinas evolucionaban poco a poco, hasta llegar a lo que se conoce hoy como "robots".
El termino "robot" proviene de una obra checoslovaca publicada en 1921 por Karel Capek, denominada Rossum's Universal derivado de la palabra checa robota, que significa trabajo forzado y que se utilizó para nombrar a unas máquinas construidas por el hombre y dotadas de inteligencia que se ocupaban de los trabajos pesados. Sin embargo Los raíces de la robótica descienden a épocas muy antiguas y hay un gran debate sobre cual es el primer robot conocido en la historia. Lo que esta claro es que esta rama comenzó su avance a partir de la revolución industrial. En esta época es cuando la mayoría de las industrias han pasado a utilizar maquinas "automáticas" con el fin de poder desarrollar trabajos cada vez más complejos.
Durante este siglo la robótica ha conocido un gran despliegue, sobre todo con la aparición de nuevas ciencias como la electrónica, informática o la inteligencia artificial, y en vez de ser una disciplina de diversión como era el caso en sus primeras épocas, se con-

virtió en una ciencia que ha demostrado su presencia en todos los campos. Los robots son cada vez más "inteligentes" y autónomos, hoy día las grandes industrias se apoyan en el uso de robots para las tareas complejas que requieren gran precisión, Las fabricas están llenas de brazos manipuladores, para todo tipo de tareas desde el montaje de un coche hasta las soldaduras a nivel microscópico. Campos como la medicina también hacen uso de la robótica, donde brazos móviles son utilizados para llegar a zonas complicadas y actuar con alta precisión, este avance ha hecho que muchas operaciones que antes eran un sueño hoy son realidad gracias a la robótica.



(a) Robot soldando

(b) Robot enfermero creado por Mutsubishi

Los robots sustituyen al hombre para tareas peligrosas donde su vida corre gran riesgo, en este sentido se han ideado robots para manipular sustancias peligrosas, sumergir miles de metros bajo del agua para rescatar un submarino, o incluso para desactivar explosivos.

La investigación en robótica es un campo muy activo, en este sentido se pueden destacar Departamentos como la NASA que actualmente esta desarrollando y utilizando robots para la exploración del espacio, como es el caso de la ultima misión llevada a cabo por los robots SPIRIT Figura 1.1.

Varias empresas están trabajando para construir robots de todos los tipos, incluso androides o humanoides capaces de desarrollar "comportamientos inteligentes" como el QRIO de SONY Figura 1.2(b), otro tipo de robots son los destinados a la diversión como la mascota Aibo 1.2(a) de SONY que ha conocido un gran éxito.

Figura 1.1: La sonda SPIRIT

(a) El mascota Aibo

(b) QRIO

Cada año se disputan varias competiciones internacionales como es el caso de la Robocup[1], en los que participan robots de todos los tipos, con el objetivo de apoyar al desarrollo de esta rama y exponer los logros de este sector que promete mucho en el futuro.

1.2. localización

La localización, es uno de los problemas clásicos dentro del área de robótica en nuestro caso, la robótica móvil. El problema consiste en que un robot se localiza automáticamente dentro de un determinado mundo usando para ello sus sensores, y la información de su entorno que en general se ofrece en forma de un "mapa". Actualmente hay técnicas más avanzadas de localización y creación de mapas simultáneamente, SLAM (Simultaneous Localization and Mapping).

Dentro de la localización, podemos distinguir dos problemas distintos, el primero y el más sencillo es partir de una posición inicial conocida, en este caso para determinar la posición final del robot habrá que estimar los errores acumulados por los odométros.

[1] http://www.robocup.org

El segundo problema, y el más difícil es *localización global*, en este caso se desconoce la posición inicial del robot y los errores son mayores que en el caso anterior, ya que en este caso hay más fuentes de errores.
Finalmente,este problema se puede generalizar y plantear en un grupo de robots, en este caso "el grupo" intentara localizarse, este problema tiene más sentido cuando los robots pueden detectarse entre sí.

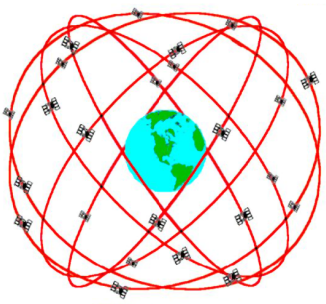
1.2.1. Algoritmos de localización

Para resolver el problema de la localización hay que tener en cuenta varios parámetros que según cambian, cambia el problema y cambia la solución. Dicho esto este problema puede dividirse en "grados de dificultad", partimos del más fácil que puede haber es la localización con "sensores de posición", en este caso la posición nos la dan los sensores de forma directa, como mucho habrá que corregir los errores. Cuando se trata de localización en interiores los mejores candidatos son los encoders Figura 1.2(c), este tipo de sensores se basa en calcular la diferencia entre lecturas sucesivas de las posiciones de las ruedas del robot, sin embargo cuando se trata de localizarse en exteriores, aparecen otros candidatos, pero el más utilizado es el GPS Figura 1.2(d) (Global Positioning System). En pocas palabras este sistema esta basado en satélites girando alrededor de la tierra de manera continua, y lo que mandan es el "reloj". Son necesarios al menos cuatro satélites para obtener las coordenadas de la posición y el tiempo. Se trata de un sistema que permite calcular las coordenadas de cualquier punto de la superficie terrestre, Su principal ventaja es que proporciona la localización absoluta en un área suficientemente grande y sin requerir estructura alguna del entorno. Las fuentes de error del GPS son distintas a las de los encoders, los errores no son acumulativos y pueden deberse a errores del reloj (máximo 1m.), errores en la órbita (máximo 1m.), errores en la modelización de la propagación en la troposfera y la ionosfera (1m. y 10m., respectivamente), errores debidos a rebotes de la señal (0.5m.) y errores de usuario en el receptor o en la configuración del mismo (los más usuales y variables, de 1m. a cientos de km.).

La localización también se puede abordar usando un entorno con "balizas"[2] , en este caso se trata de que el robot se localiza geométricamente. La posición dentro del mundo en este caso se puede calcular de dos manera diferentes: mediante triangulación, en este caso (x,y,θ) se calculan basándose en el ángulo con que se "ven" las balizas, o mediante trilateración done (x,y,θ) se calcula basándose en la distancia a las balizas. La Figura 1.3 muestra un ejemplo de este tipo de localización.

[2]visuales,código de barras,..

(c) Encoders



(d) GPS

Figura 1.2: Sensores de posición

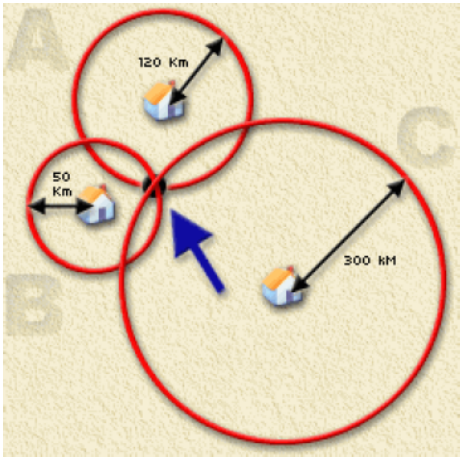


Figura 1.3: Localización probabilística con trilateración.

La versión más difícil y elaborada del problema de la localización, es cuando se enfrenta usando sensores *no específicos de posición* (láser, visión,..), en este caso la información de posición se saca relacionando las lecturas sensoriales (láser,cámara,..) con el "mapa" del entorno, en este sentido hay técnicas como el "scan maching" que se basa en ir "pegando" lecturas del mapa local sobre el global intentado ajustar las lecturas a este ultimo, para ello es necesario corregir la estimación de la posición. El punto débil de esta técnica es que no vale para localización global 1.2. Otra técnica buena para la localización local 1.2 son los filtros de Kalman, se trata de un filtro de Bayes recursivo que estima la distribución a posteriori del estado de un sistema condicionada en función de los datos. En el caso de la localización de un robot móvil, el sistema dinámico es el robot móvil y su entorno, mientras que el estado es su posición, que corresponde a la posición del robot y su ángulo de orientación (x,y,θ) dentro de un sistema de coordenadas cartesianas fijo a su entorno. Su principal limitación es que se trata de una técnica unimodal y exclusivamente gaussiana[Isard y Blake, 1998]. Los entornos dinámicos y el ruido en las lecturas de los sensores tampoco son bien soportados por esta técnica.

Por ultimo la localización probabilística(Objetivo de este proyecto), es una de las me-

jores técnicas para la localización en interiores [Thrun, 2000]. En este sentido permite la incorporación de observaciones y actuaciones. La idea principal de este método se basa en tener un modelo capaz de estimar la probabilidad de "estar en una posición" basandose sólo en la información del entorno, comparando esta información con la que se esta leyendo por el sensor real podemos calcular la probabilidad de esta posición. Esa estimación se actualiza con la incorporación de nuevas observaciones y movimientos. Para fusionar la nueva probabilidad con la anterior se hace uso de la regla de Bayes. Con este planteamiento este método consigue una buena localización incluso desconociendo la posición inicial del robot, también permite representar situaciones ambiguas y resolverlas posteriormente. El inconveniente de esta técnica es que almacena la probabilidad para *todas* las posibles posiciones, esto hace que esta técnica sea lenta y no escalable a grandes entornos donde el número de posiciones posibles es muy grande.
La Figura 1.4 muestra un ejemplo sencillo de esta técnica:

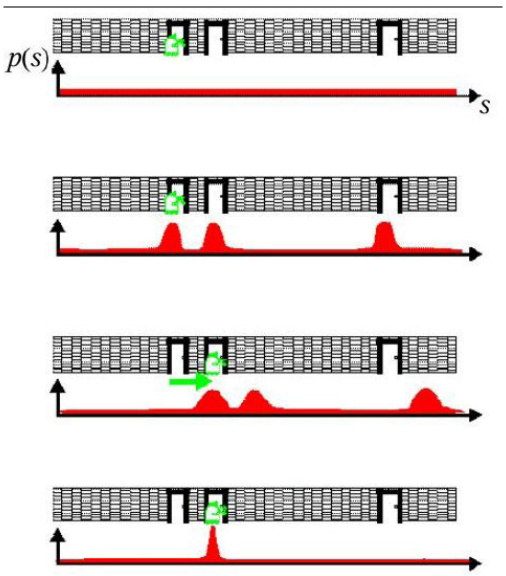


Figura 1.4: Localización probabilística.

En la figura 1.4 el robot está situado en un espacio unidimensional, en el que sólo puede trasladar en el eje horizontal. Inicialmente se desconoce la posición del robot y todas las posiciones son equiprobables. Después de incorporar una observación(lectura sensorial), el robot ha detectado que esta delante de una puerta(por visión, láser, ..), entonces sube la probabilidad de las posiciones cercanas a las puertas, esto da lugar a una ambigüedad ya que hay tres puertas en el escenario. Ahora el robot realiza una traslación hacia delante, incorporamos este movimiento(esto de materializa moviendo la función de probabilidad), y incorporamos de nuevo la observación. Fusionando esta nueva lectura con la anterior el robot rompe la ambigüedad y se localiza en la posición correcta.

Para resolver el problema que padecía la localización probabilística(tener que calcular la probabilidad para todas las posiciones) existen otros algoritmos que han ido evolucionando en los últimos años, se trata de los filtros de partículas. Esta técnica consiste en mantener un conjunto muestral de posibles posiciones, y calcular la probabilidad de cada una, mediante remuestreo estadístico, estas posiciones acaban convergiendo hacia la posición correcta del robot. El número reducido de muestras (comparado con el de localización probablistíca) hace que se minimizan los costes computacionales, y que el algoritmo sea escalable a grandes entornos.
Todas las técnicas presentadas hasta ahora hacen uso de sensores. No hay sensores sin errores, todo sensor tiene sus limitaciones y condiciones de funcionamiento si se perturba el ambiente normal del trabajo (cambio de condiciones climáticas) de forma brutal, su funcionamiento deja de ser normal. Este tipo de detalles son muy importantes, para ello a la hora de elegir el sensor, hay que saber su rango de errores y intentar corregirlos, ya que el resultado final de la localización dependerá de manera directa de estos errores.

Generalmente los errores en los sensores se pueden dividir en dos tipos:

- **Errores sistemáticos:** son errores que vienen desde la fábrica(diferentes diámetros de ruedas, falta de alineamiento de las mismas, resolución limitada de los encoders y velocidad limitada de muestreo de los encoders), son errores acumulativos. Se pueden corregir mediante calibración, ruedas auxiliares, encoders adicionales.

- **Errores no sistemáticos:** son impredecibles, debidos a patinaje de ruedas, la naturaleza del suelo,..etc.
  Pueden corregirse utilizando referencias mutuas (dos robots, uno parado), corrección interna (dos robots en movimiento) y navegación inercial (medir la aceleración en los tres ejes e integrar).

Hasta hora hemos visto las técnicas utilizadas para la localización, otro parámetro importante es el entorno donde nuestro robot tratará de localizarse, aquí surge la pregunta ¿como modelar el entorno en un formato entendible por el robot?.
En general esto se hace mediante “Mapas”, un mapa es una estructura que almacena información del entorno. Hay varios tipos de mapas, pero en general se pueden dividir en categorías depende del criterio con el que se “formaliza” el entorno. En este sentido se puede distinguir entre mapas locales-globales, métricos-topologicos, rejilla-elementos. Depende del entorno y del tipo de sensores disponibles y de la naturaleza del entorno se usará un tipo de mapa o otro.

1.3. Localización basada en láser

Como hemos mencionado anteriormente, este proyecto intentará resolver el problema de la localización utilizando técnicas probabilisticas, más en concreto trataremos de implementar dos algoritmos distintos, el primero es "localización con mallas de probabilidad", este método se basa en mantener un cubo de probabilidad donde se almacena la función de densidad para todas las posibles posiciones (x, y, θ) dentro del mundo, las probabilidades se calculan comparando las medidas del sensor láser 3.1.2 con medidas "teóricas" efectuadas sobre el mapa 4.3. El segundo algoritmo es "el filtro de partículas", se trata de un filtro bayesiano recursivo, en este caso el conjunto muestral estaría formado por posiciones (x, y, θ) cogidas al azar sobre todo el mundo, de nuevo la probabilidad se calcula a la luz de lecturas del sensor láser comparadas con medidas efectuadas sobre el mapa.
Los dos algoritmos harán uso de un mapa métrico en forma de una rejilla 2D, hablaremos en detalle de este tipo de mapas en la sección 3.3.1. Este mapa servirá para realizar los cálculos "teóricos" del láser, es el punto clave para transformar las medidas del sensor láser en información de posición.

El problema de la localización hasta el momento era una tarea pendiente en el grupo de Robótica de la Universidad Rey Juan Carlos. se trata de un grupo compuesto por profesores y alumnos. Los intereses de este grupo se basan en la generación de comportamientos artificiales en robots. El grupo se dedica a varias lineas de investigación, una de ellas es la de generar comportamientos autónomos en entornos de oficina, de ahí es el objetivo de varios PFCs en los últimos años, en estos podemos destacar navegación global [Isado, 2005] [López, 2005], [Crespo, 2003] [Benítez, 2004], navegación local [Lobato, 2003] , seguimiento de una persona en los pasillos de un edificio [Calvo, 2004].

El grupo también esta presente en las competiciones internacionales como la Robocup uno de los campeonatos más conocidos en robótica móvil. En un principio se usaron los robots EyeBot, pero actualmente se usan los perritos Aibo de Sony, el grupo dispone de un equipo de más de diez de estas mascotas, más una variedad de robots de tamaño medio y pequeño destinados a la investigación. La lista contiene: 30 robots Lego, 6 robots EyeBot, 2 robot Pioneer, 2 cuellos mecánicos y 10 perritos Aibo de Sony.

Este proyecto junto con el de localización por visión [Fernández, 2005] tiene como misión principal resolver el problema de localización en el robot Pioneer 3.x. hasta el momento todos los proyectos de navegación global [Lobato, 2003] [Isado, 2005] para este robot, solo se han implementado en entornos de simulación, porque necesitaban

que el robot está localizado dentro del entorno como prerequisito para la navegación.

Estos comportamientos se programan sobre una plataforma desarrollada íntegramente en el grupo llamada JDE (Jerarquía Dinámica de Esquemas) [Plaza, 2003] . Esta plataforma ofrece acceso sencillo a los sensores y actuadores del robot. Se basa en esquemas que son hebras iterativas que realizan una determinada función, uniendo varios esquemas se consigue generar comportamientos complejos. Veremos esta plataforma más en detalle en el capítulo 3.

Capítulo 2

Objetivos

Después de haber presentado el contexto general, y particular en el que se ha desarrollado este proyecto, en este capítulo vamos a fijar sus objetivos y presentar los requisitos para su realización. Hablaremos también de la metodología que se ha empleado para el desarrollo de las componentes sw, y de las diferentes etapas del desarrollo sw por las que ha pasado el proyecto.

2.1. Objetivos

El objetivo de este proyecto, es el desarrollo de dos algoritmos probablísticos para la localización en interiores del robot piooner, usando para ello como sensor principal el sensor láser 3.1.2.
Los objetivos se pueden dividir en varios subobjetivos, que conjunto dan como fruto el objetivo principal:

1. implementación del algoritmo de "mallas de probabilidad".
2. implementación del algoritmo del "filtro de partículas".
3. probar el funcionamiento de estos algoritmos sobre el simulador Player/Stage, y sobre el robot real pioneer 3.x

La implementación se hará sobre la arquitectura JDE. Queremos optimizar estos algoritmos para que funcionen de manera eficaz sobre la plataforma. Finalmente se hará una comparativa entre los dos algoritmos, describiendo las ventajas y desventajas de cada uno.

2.2. Requisitos

El desarrollo del proyecto estará guiado por los objetivos comentados anteriormente y deberá ajustarse a los requisitos que se comentan a continuación :

1. Los algoritmos son de carácter pasivo, quiere decir que no requieren control del movimiento del robot. De esta forma podrán funcionar en paralelo con cualquier algoritmo de navegación.

2. Los algoritmos estarán disponibles como sw libre, es una política seguida dentro del grupo.

3. La implementación se apoyara sobre la plataforma JDE, que ofrece facilidades para el desarrollo de comportamientos sobre el Pioneer. Esta plataforma está disponible en el grupo, y se explicara en detalle en el capítulo 3.

4. Como objetivo queremos probar el funcionamiento sobre el simulador Player/Stage, se trata de un simulador libre y disponible bajo licencia GPL.

5. Uno de los objetivos es probar el funcionamiento sobre el robot real, en este caso el Pionner 3.x El grupo dispone de dos robots de este tipo, equipados con todos los sensores necesarios.

6. Los algoritmos desarrollados, tienen que resolver el problema consumiendo la menor cantidad de CPU posible, así podrán funcionar con otro algoritmo (de navegación) sin sobrecargar la CPU.

2.3. Metodología

Para el desarrollo de este proyecto, se ha basado en una metodología iterativa donde cada iteración se compone de tres fases: diseño, implementación y experimentos. El desarrollo de este proyecto se basará en el modelo de desarrollo en espiral basado en prototipos. Este modelo se adapta perfectamente a este tipo de proyectos(investigación), ya que permite gran flexibilidad ante cambios en requisitos, cosa que padecen con frecuencia este tipo de proyectos.
Hablando del caso concreto de este proyecto se ha seguido la misma metodología para el diseño y la implementación de los dos algoritmos probabilísticos. Después de pasar un periodo de formación de 4 meses supervisado por profesores del grupo de robótica de la URJC, Se ha pasado a estudiar los algoritmos que se van a implementar en este proyecto en concreto. Terminado el estudio entramos en el ciclo iterativo del espiral.

En cada iteración se fijan los subobjetivos se diseñan las componentes y finalmente pasamos a la implementación de estas. Una vez terminado este proceso es el tutor quien supervisa y valida las pruebas propuestas, en caso de éxito pasamos a la siguiente iteración. Se mantenían reuniones semanales con el tutor con el fin de estudiar el estado del desarrollo y planificar las siguientes etapas. Cada iteración daba fruto a un nuevo prototipo con nueva funcionalidad añadida al algoritmo.

2.3.1. Desarrollo en espiral

En este tipo de desarrollos, los productos son creados gracias al número de iteraciones que se dan en el proceso de vida de software, en cada iteración hay que pasar por cuatro etapas:

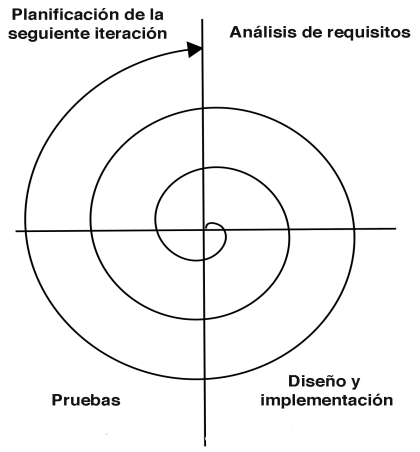


Figura 2.1: Modelo en espiral

1. Requisitos: En este proyecto no ha habido cambios de requisitos, dicho esto los requisitos descritos en la sección 2.2 son comunes para todas la iteraciones.

2. Diseño e implementación: En cada iteración se diseñan y se implementan nuevas componentes sw, que añaden nueva funcionalidad al software, también se estudia el efecto de su integración con el resto de las componentes disponibles de iteraciones anteriores.

3. Pruebas: Se diseñan casos de pruebas con el fin de probar la funcionalidad añadida, y se estudia su comportamiento con el resto de los componentes del algoritmo. Es el tutor quien valida el buen funcionamiento de estas y decide si se saca o no, un nuevo prototipo.

4. Planificar la siguiente etapa: al final de cada etapa se mantienen reuniones con el tutor, para planificar la siguiente etapa, y fijar sus objetivos.

Como hemos mencionado anteriormente, durante el desarrollo de este proyecto se han sacado varios prototipos. Un prototipo es una versión preliminar del producto sw final, es el fruto de una iteración completa del ciclo de desarrollo.
En este proyecto se han realizado varios prototipos, cada uno añade nueva funcionalidad en los algoritmos, cuando se valida un prototipo pasamos a la siguiente iteración:

- **Prototipo 1:** Generación de mundo 2D, y del cubo de probabilidad.
- **Prototipo 2:** Calculo de láser teórico, sobre el mundo 2D.
- **Prototipo 3:** Incorporación de observación y movimiento.
- **Prototipo 4:** Localización con mallas de probabilidad
- **Prototipo 5:** Implementar la infraestructura para el manejo de partículas
- **Prototipo 6:** Modelos de observación y movimiento
- **Prototipo 7:** Remuestreo
- **Prototipo 8:** Localización con Filtro de partículas

Capítulo 3

Plataforma de desarrollo

En este capítulo vamos a hablar de la plataforma tanto hardware como software, sobre la cual se ha desarrollado este Proyecto, empezamos con una descripción del robot Pioneer destacando las propiedades sw/hw de este Robot, a continuación hablaremos de la plataforma software, y de los simuladores que se hayan usado durante el desarrollo del proyecto, también comentaremos gridslib y xforms como bibliotecas auxiliares en los que en las se apoya nuestro software.

3.1. Robot Pioneer

El Robot utilizado para el desarrollo de este proyecto es el Pioneer 3x Figura 3.1 , comercializado por la empresa norteamericana Acrivemedia[1]. Este Robot esta equipado con un conjunto de sensores que le permiten explorar con flexibilidad su entorno, y actuadores que le permiten moverse. Ademas de estos sensores estándar que vienen de fabrica, le hemos incorporado soporte para cámara USB y FireWire, y para un cuello mecánico.

El pioneer lleva incorporado un sensor láser capaz de hacer un barrido de 180 grados, con precisión de 1 grado, este sensor es el que vamos a utilizar para nuestros algoritmos de localización, dispone de una corona de 16 sensores de ultrasonido que rodean al robot y lleva incorporados unos odométros (encoders) asociados a las ruedas para saber cuánto han girado éstas.

Nuestro robot es capaz de llevar a bordo un portátil, que normalmente suele estar conectado a la red exterior mediante un enlace inalámbrico, con una tarjeta de red 802.11 que le proporciona una velocidad de 11Mbps en sus comunicaciones. De esta forma el programa de control puede correr a bordo del portátil o en cualquier otro ordenador mediante la comunicación wi-fi.

[1] http://www.activrobots.com/ROBOTS/index.html#p2dx

Este modelo de la gama Pioneer viene equipado con un procesador de 18 MHz Hitachi H8S/2357 con 32Kb RAM y 128Kb Memoria flash. Tiene una autonomía de 5 horas. El robot Soporta un máximo de 23 Kg de carga.

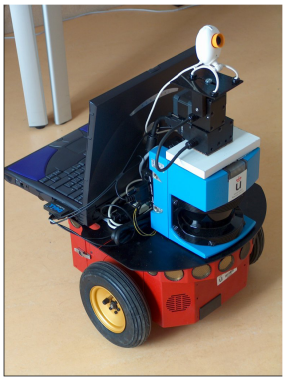


Figura 3.1: Pioneer 3x

3.1.1. El Sensor láser

En la parte delantera del robot se ha instalado un sensor láser Figura 3.2 de marca Sick[2]. El modelo LMS-200 proporciona 10 barridos por segundo, con una precisión de 2 grados y 2cm. Es un un sensor de alta precisión que ofrece perfiles del entorno nítidas, y fiables. Recibe una alimentación continua de 24 Voltios, que se extraen con una placa elevadora de tensión desde los 12 Voltios internos del robot. El sensor entrega sus lecturas a través de un puerto serie convencional. Para comunicar con el dispositivo Se ha reutilizado un driver de ARIA, que configura adecuadamente al dispositivo láser en cuanto a su resolución angular, modo de operación, velocidad de emisión y utiliza el puerto serie a 38400. Este driver entiende el formato de mensajes en el puerto serie que establece el fabricante del sensor, y refresca las variables en las que vuelca las medidas del láser. Además ofrece una interfaz de funciones en C para iniciar al dispositivo (sick_start_laser), lanzar o detener la captura (sick stop continuous mode,sick start continuous mode) y muestrear de modo no bloqueante si hay nuevos mensajes del sensor en el puerto serie.

[2]http://www.sick.es/es/productos/autoident/medicion laser/interior/es.html

(a) El sensor láser (b) Eejemplo de barrido

Figura 3.2: EL sensor láser Sick.

3.1.2. Los odométros

Uno de los sensores vitales para la realización de este proyecto son los odométros [Plaza, 2004a], ya que constituyen la segunda fuente de la información en la que se apoyarán nuestros algoritmos detrás del láser. Su misión principal es la de estimar la posición (x,y,θ) del robot dentro del mundo en cada momento y de ahí estimar la velocidad real del robot. Estos sensores estiman lo que han girado las ruedas izquierda y derecha en un intervalo de tiempo, conociendo las antiguas coordenadas es muy fácil sacar la nueva posición del robot. En un rango diferencial del tiempo el movimiento se puede aproximar a un movimiento circular de radio R y ángulo de giro α. La plataforma encapsula la complejidad que llevan por debajo los odométros, ofreciendo una variable golobal (x,y,θ) la cual usaremos para estimar la posición del robot. Cabe destacar que la estimada por los odométros es *relativa* a la posición inicial del robot, así que los algoritmos implementados en este proyecto utilizan esta información sólo para detectar los movimientos efectuados por el robot.

3.2. Arquitectura JDE para aplicaciones robóticas.

La plataforma JDE [Plaza, 2003] es una arquitectura software que permite generación de aplicaciones robóticas. Ha sido creada íntegramente dentro del grupo de robótica de la URJC, permite acceso sencillo a los sensores y actuadores de robot. Es una jerarquía dinámica de esquemas perceptivos y de actuación que se comunican entre sí a través de variables. Un esquema perceptivo es una hebra iterativa que se dedica a leer la variables, y procesar su información pero no manda comandos de movimiento al robot, por otra parte están los esquema de actuación esto si mandan comandos de movimiento. Las aplicaciones se organizan en una colección de esquemas perceptivos

y de actuación. Los algoritmos implementados en este proyecto están compuestos por dos esquemas perceptivos: laserloc-mallas y laserloc-particulas.

3.2.1. API jde.c

Las variables sensoriales y de actuación pueden tener varias fuentes dentro del jde.c, dicho esto pueden venir de:

1. robot real
2. Simuladores
3. servidores (detrás de estos nuevamente el robot real o simuladores)

Da igual la fuente de estas variables, la implementación encapsula toda la complejidad de la arquitectura y nos presenta una API sencilla de variables, que nos permite controlar el hardware descrito en la sección 3.1, la interfaz esta formada por :

Variables de sólo lectura :

- la posición (x,y,θ) del robot (relativa)
- Láser en forma de un array de 180 medidas
- Sonar en forma de un array de 16 medidas

Variables de lectura/escritura:

- v,w velocidad linea/angular del robot
- pan,tilt posiciones del cuello mecánico

Las aplicaciones se construyen como una jerarquía de esquemas, que usan este API para acceder a las variables, y usan la información proporcionada por los mismos.

3.2.2. Servidores

Generalmente los programas se ejecutan de manera local dentro de los robots en portátiles que estos llevan a bordo, la arquitectura JDE supero este reto y permite alta flexibilidad para ejecutar aplicaciones robóticas de manera remota, gracias a su arquitectura cliente-servidor que se muestra en la Figura 3.3.

Tal y como se muestra en la Figura 3.3, JDE se basa en dos servidores, donde cada uno se encarga de funciones distintas del otro. El servidor *otos* se encarga de servir

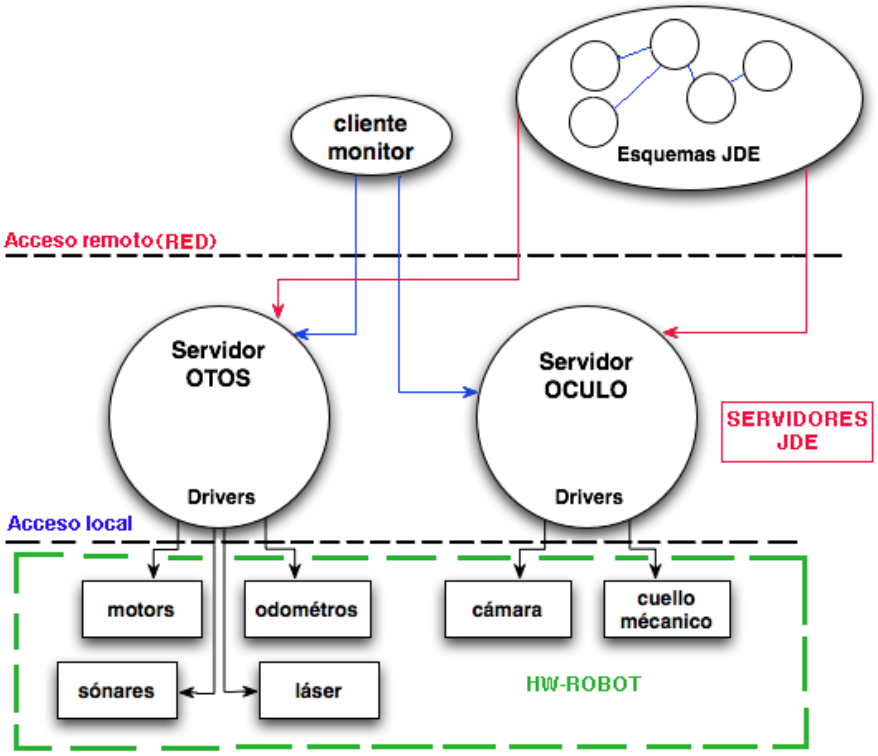


Figura 3.3: JDE

las lecturas de los sonars, láser, táctiles, y la información ofrecida por los odométros. También permite mandar comandos a la base motora en forma de velocidades (v,w) y estimar estas velocidades basandose en la información de la odometría. Permite también saber el nivel da la batería a partir del sensor de voltaje.
Para recibir las lecturas sensoriales de un determinado sensor, hay que estar suscrito en este sensor, cuando hay nuevos datos el servidor se encarga de entregarlos a todos los clientes suscritos en este sensor. Este servicio se llama *suscripción directa*, permite a los clientes suscribir y dessuscribir en cualquier sensor, en cualquier momento. De este modo los clientes se pueden suscribir sólo en los sensores que les interesan, esto da más flexibilidad al servicio.

El servidor *oculo* se encarga de dos dispositivos distintos, son la cámara y el cuello mecánico, en este PFC no se hace uso de este servidor.

3.2.3. La plataforma Player/Stage

Player/Stage[3] [Brian P. Gerkey, 2003] es un simulador para aplicaciones robóticas. que permite el desarrollo de aplicaciones multi-robot. El entorno proporciona un servidor de simulación de dispositivos robóticos Stage, y una interfaz para acceder a este

[3]http://playerstage.sourceforge.net

simulador Player. Este simulador es recientemente acoplado con la plataforma JDE [Isado, 2005].
Para recibir lecturas sensoriales, los clientes tienen que conectarse al servidor Player y pedirselas a través de mensajes sobre un socket TCP. De esta forma el programa cliente puede ejecutarse en cualquiera maquina con acceso remoto. Player sigue el modelo UNIX de tratar los dispositivos como ficheros. Así para recibir observaciones sensoriales, basta con abrir el dispositivo apropiado en modo lectura y leer las medidas requeridas, y como cabe esperar, para mandar ordenes hay que esciribrlas en el dispositivo después de abrirlo en modo escritura. Player/Stage recibe el mundo en forma de una imagen PNM, que representa los espacios "ocupados"/"vacios" dentro del mundo. La estructura del archivo imagen tiene el siguiente formato:

```
P5
# CREATOR: The GIMP's PNM Filter Version 1.0
271 316
255
\uffff\uffff\uffff\uffff\uffff\uffff\uffff\uffff\uffff\uffff\uffff\uffff
```

Donde (271,316) es el número de pixeles en ancho, alto respectivamente 255 son los niveles de grises que tiene el archivo, y a continuación el choro de bytes que contiene la información de la imagen. Plyer/Stage recibe un archivo de texto en el cual se le indica, el archivo a cargar, a qué escala y cuántos robots se van a utilizar(En este proyecto sólo se usa un robot), a continuación vemos la sintaxis de este fichero:

```
include "usc_pioneer.inc"
bitmap
(
file "stage_dep2_puertas.pnm"
resolution 0.1425
)

usc_pioneer (name "robot1" port 6666 pose [5.0  5.0 90.000 ])
```

El fichero indica que el mundo imagen a cargar es el del fichero "stage dep2 puertas.pnm" y se tiene una escala de 0.1425 m/pixel. El robot esta en la posición $(x, y, \theta) = (5, 5, 90)$ respecto al eje cuyo origen es la esquina inferior izquierda de la imagen.

3.3. Librerías auxiliares

Como se ha comentado al principio de este capítulo, más de apoyarse en la arquitectura JDE, nuestro software hace uso de otras bibliotecas auxiliares que le facilitan algunas tareas muy concretas. La implementación se apoya en dos bibliotecas: gridslib y Xforms utilizadas para el manejo de rejillas, y crear interfaces gráficas respectivamente.

3.3.1. Gridslib

Una rejilla es un array bidimensional de celdillas donde sólo se puede almacenar dos valores: ocupado o vacío, es uno de los elementos clave en la implementación de nuestros algoritmos, ya que es el que permite representar almacenar el mapa de ocupación, y el cubo de probabilidad.

Una celdilla tiene varios parámetros, pero los más importantes para nosotros son su centro y su resolución, este ultimo parámetro es el mismo para todas las celdillas de hecho es un parámetro de la rejilla en sí, y puede afectar de manera directa a nuestro algoritmo en términos de velocidad como veremos más adelante en el desarrollo de este proyecto. Dicho esto que hay que elegir un valor de compromiso para satisfacer los dos requisitos: rapidez y precisión de manera razonable.

para el manejo de la rejilla nuestro programa se apoya en *gridslib* una biblioteca auxiliar, que le ofrece funciones para crear, iniciar, y reúbicar la rejilla. En realidad esta librería Implementa varias técnicas de construcción de mapas como la regla de Bayes, la regla Dempster-Shafer (teoría de evidencia), rejillas borrosas o rejillas histógramicas, pero en nuestro caso la funcionalidad de esta biblioteca se limite a generar una rejilla cuadrada con unas celdas de tamaño regular predeterminado, reubicarla dentro del mundo. La biblioteca saca toda la información necesaria para crear y procesar la rejilla de un archivo de texto que se le pasa como parámetro de entrada.

3.4. Xforms

Uno de los elementos más importantes de nuestra aplicación es la interfaz gráfica, ya que nos permite ajustar los distintos parámetros de los algoritmos, visualizar las estructuras utilizadas en los dos algoritmos y depurar su funcionamiento.

Xforms[4] es una biblioteca de libre uso escrita en C. Su misión principal es facilitar

[4]

la creación y uso de interfaces gráficas sobre el sistema X-Window de linux ocultando la complejidad de este al programador. Para ello ofrece un extenso repertorio de elementos gráficos(botones,dials,canvas..) sencillos que junto permiten crear interfaces complejas. La biblioteca ofrece una herramienta *fdesign* de uso visual que nos permite crear y personalizar la interfaz de manera muy sencilla.

En nuestro caso la biblioteca será utilizada para crear la interfaz "gui_laser" que será mantenido por el esquema gui_laser_loc, la biblioteca permite manejar estructuras gráficas básicas que serán utilizadas para representar las diferentes estructuras utilizadas en nuestros algoritmos.

Capítulo 4

Localización con mallas de probabilidad

Como hemos mencionado anteriormente, esta técnica pertenece a la familia de algoritmos de *localización probabilística*. la localización "con mallas de probabilidad" Se apoya en dos modelos fundamentales: modelo de observación láser y modelo de observación odométrica, gracias a estos modelos, consigue traducir las lecturas sensoriales ofrecidas por el sensor láser en probabilidades calculadas para todas las posibles posiciones dentro del mundo. Para llevar un "historial probabilístico" de cada posición, esta técnica usa la regla de Bayes para fusionar las nuevas probabilidades con las anteriores. la probabilidad acaba convergiendo a una posición dentro del mundo, tras una sucesión de lecturas proporcionadas por los sensores (el sensor Láser en este caso), y movimientos efectuados por el robot, que el algoritmo tratará de incorporar usando el modelo de observación odométrica.

En este capítulo veremos los fundamentos teóricos en los que se basa esta técnica, su diagrama de bloques, veremos también los distintos modelos que representan el núcleo de nuestro algoritmo localizador, hablaremos del modelo de observación láser, modelo de observación odométrica, y acabamos describiendo la acumulación de evidencias. Presentaremos también los mecanismos implementados para "visualizar" las diferentes estructuras usadas por el algoritmo.

4.1. Fundamentos teóricos

A lo largo de la historia, los algoritmos probabilísticos han sido siempre los que mejores soluciones han dado para el problema con el que nos enfrentamos, así vamos a optar por el uso de una de estas técnicas.

La localización con mallas de probabilidad consiste en mantener un cubo tridimensional de densidades de probabilidad, en el cual para cada posición (x,y,θ) se guarda la probabilidad asociada a esta posición. Esta probabilidad refleja la verosimilitud de

que el robot se encuentre en esta posición, este valor irá evolucionando en el tiempo ganando o perdiendo probabilidad, según las observaciones incorporadas y los movimientos que nuestro robot efectuará a lo largo del tiempo.

Nuestro algoritmo recibe como entrada, por una parte la información de su entorno que viene dada en formato de un mapa que representa "los espacios ocupados" dentro del mundo, y por la otra la información recibida de sus sensores, en este caso el sensor Láser y la odometría. Esta información será utilizada para alimentar a los dos principales modelos de nuestro algoritmo: observación láser 4.3 y observación odométrica 4.4 respectivamente, los dos modelos tienen como misión "sacar información de posición a partir de las observaciones sensoriales láser y odometría". El primer modelo usa la observación láser para calcular la probabilidad de cada una de las posibles posiciones dentro del mundo. Esta probabilidad no viene de forma directa de la lectura láser, sino se obtiene comparando esta observación con las medidas "teóricas" realizadas sobre el mapa Figura 4.5. De esta manera las posiciones que dan medidas "compatibles" con la real serán más probables que el resto de las posiciones. El segundo Modelo usa la observación odométrica para detectar los movimientos del robot, y cuando haye que el robot se ha movido, calcula los desplazamientos correspondientes y *desplaza* las densidades de probabilidad dentro del cubo tridimensional, sin que se cambia la forma de la nube de probabilidad, ya que lo único que hace este modelo es desplazarla, en la Figura Figura 4.8 se muestra un ejemplo de este desplazamiento.

A la hora de incorporar la primera observación habrá varias posiciones "compatibles" con esta lectura(generalmente por razones de simetría) y por lo tanto equiprobables. Esto da lugar a una ambigüedad en la posición del robot. Para ello el modelo de observación láser además de la función antes descrita, se encarga también de "acumular las evidencias" este proceso hace que las posiciones que mantienen "un historial compatible" con varias lecturas láser siguen en la nube de probabilidad, mientras que las que dejan de ser compatibles(tras un movimiento por ejemplo) desaparecen de la nube poco a poco.

De esta manera conseguimos que tras varias iteraciones del algoritmo, donde en cada iteración se incorpora observación láser y/o odométrica, la nube converge hacia la posición correcta del robot, que es la única que ha sido compatible con todas las observaciones láser realizadas hasta el momento.

Nuestro algoritmo siempre da como resultado la posición que tiene la máxima probabilidad dentro del cubo , la cual llega un momento en el que siempre va ser la misma.

La localización con mallas de probabilidad se realiza en *dos pasos*:

- *Modelo de observación odométrica*, actúa de forma *pasiva* capturando la información odométrica. Cuando detecta un desplazamiento (Δx,Δy,$\Delta\theta$) se encarga de *desplazar* las densidades de probabilidad dentro del cubo tridimensional Figura 4.8. Este modelo sólo desplaza la nube, y no afecta a su forma(valores de probabilidad). De esta manera se consigue que las posiciones que eran compatibles antes del movimiento y lo siguen siendo después, acumulan más probabilidad que el resto y la nube de probabilidad va convergiendo dentro del cubo, hacia la posición correcta.

- *Modelo de observación láser*, es el "núcleo" de este algoritmo, dada la lectura del sensor láser calcula la probabilidad "de estar" en cada una de las posiciones (x,y,θ) del cubo de probabilidad. Esto se consigue comparando la lectura sensorial láser con medidas "teóricas" realizadas sobre el mapa Figura 4.5. Este modelo se encarga también de refrescar la probabilidad acumulada hasta el momento para una posición, fusionando la nueva probabilidad (proporcionada por el mismo modelo) con la almacenada en esta posición. Para ello se utiliza *la regla de Bayes* asumiendo un entorno con rejilla regular markoviana (este paso se abordara en profundidad mas adelante).

Los dos pasos se ejecutan de forma iterativa, y cada iteración puede dar lugar a una incorporación láser/odométrica pero los dos son totalmente independientes.

4.2. Diseño general del algoritmo

Hemos diseñado el algoritmo dentro de jde.c, como dos esquemas *laserloc-mallas* y *gui_laser_loc*. En realidad lasaer_loc es el que se encarga de la localización, para ello inicializa el mapa, leyendo el archivo del mundo (en este caso una imagen PNM), además se encarga de crear el cubo de probabilidades. Es el que implementa los dos pasos de la localización con mallas de probabilidad : observación odométrica, observación láser y acumulación de evidencias. En pocas palabras este esquema lleva toda la complejidad de este algoritmo. El segundo esquema es *gui_laser_loc*, se trata de un esquema de servicio que se encarga de crear y refrescar la interfaz gráfica, esto nos permite depurar el primer esquema visualizando sus estructuras y permitiendo modificar sus parámetros como viremos en la sección 4.6 de este capítulo. La Figura 4.1 muestra la interacción entre los diferentes esquemas.

En la figura se ve la plataforma JDE, en este caso está compuesta por nuestros esquemas. Vemos el esquema *laserloc-mallas*, que recibe el mundo en un forma de una

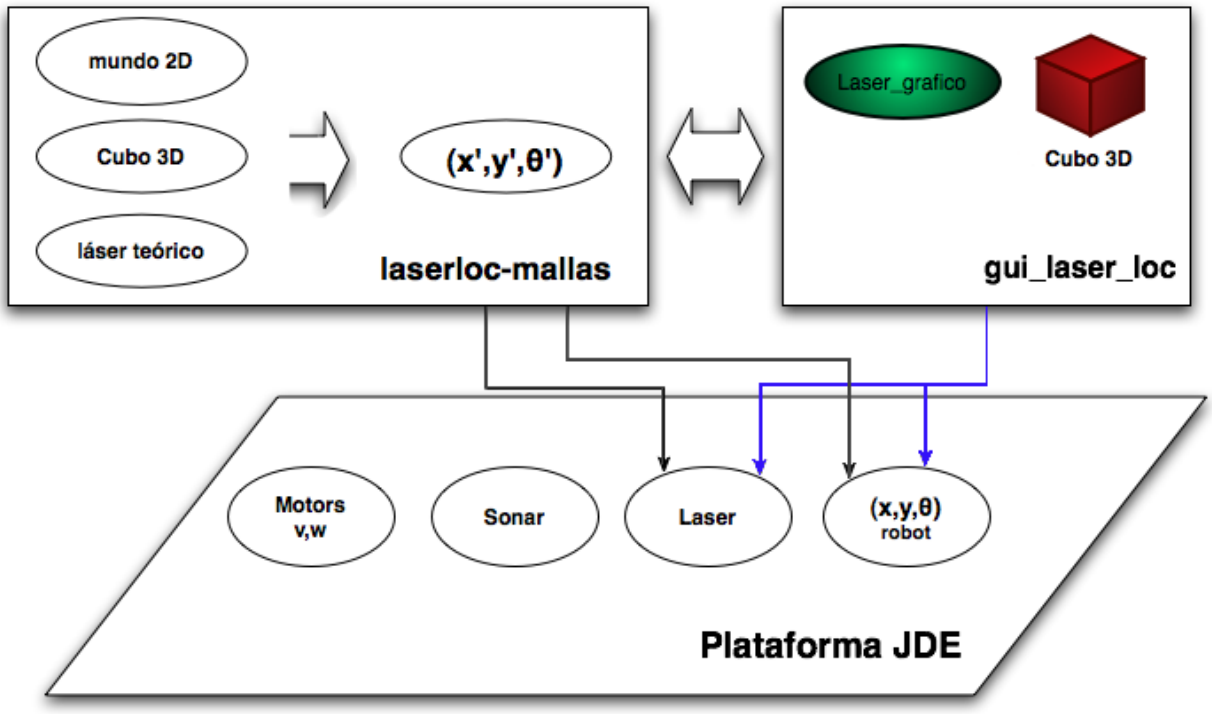


Figura 4.1: Plataforma JDE.

imagen PNM y lo transforma en un mapa bidimensional de ocupación. Junto con este mantiene otra estructura que es el cubo donde se almacenan las probabilidades para todas las posiciones (x,y,θ) dentro del mundo. Además mantiene otra estructura que permite guardar el láser calculado sobre el mapa, de ahí el nombre de "láser teórico". Esto por un lado por el otro tenemos el esquema *gui_laser_loc* que se encarga de mantener en paralelo otras estructuras "gráficas", que nos permiten visualizar las estructuras del primer esquema, y perseguir los pasos del algoritmo.

Como hemos mencionado anteriormente, laserloc-mallas es el esquema que lleva toda la complejidad de este algoritmo sus funciones se dividen principalmente en :

1. Leer el archivo del mundo, y transformarlo en una mapa bidimensional.
2. Crear y inicializar el cubo de probabilidades.
3. Localización.
   a) Modelo de observación láser y acumulación de evidencias.
   b) Modelo de observación odométrica.

El diagrama 4.2 muestra el flujo de control de este esquema:

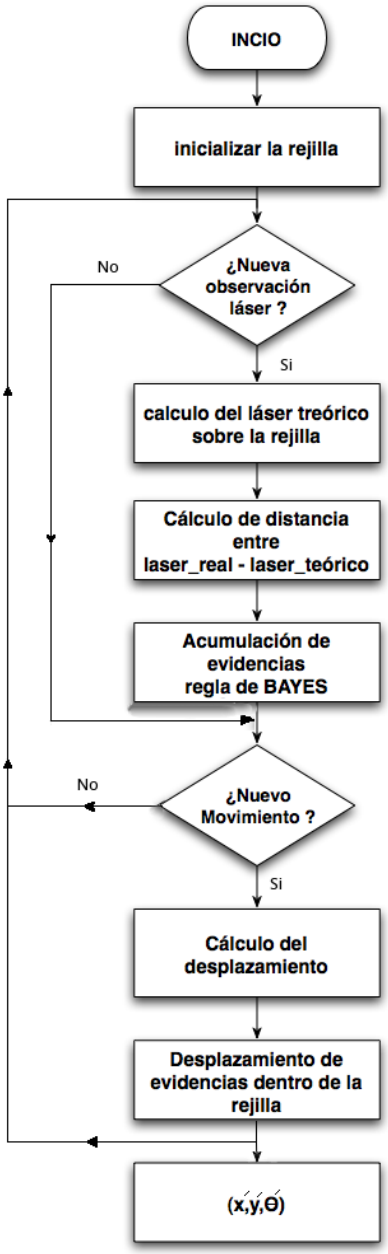


Figura 4.2: pseudocódigo del esquema laserloc-mallas

En el diagrama 4.2 podemos ver a los dos modelos: observación láser y observación odométrica, como dos pasos dentro del flujo del algoritmo. Cada uno tiene sus condiciones de activación, y cuando se satisfacen se activa el modelo y incorporamos la observación láser/odométrica, pero la activación de uno u otro es *independiente*. Quiere decir que las condiciones de activación de un modelo no afectan a las del otro y tampoco están relacionadas. Si se activan los dos a la vez, se ejecuta uno u otro al azar.

Este esquema, se lanza por el usuario a través de la interfaz GUI, una vez lanzado empieza sus funciones preparando todo el entorno que haga falta para ejecutar el algoritmo de localización, como información de entrada recibe dos archivos:

1. Archivo de configuración del mundo : es un archivo de texto que contiene la ruta

del archivo del mundo, la resolución de este ultimo, la posición inicial (x,y,θ) del robot dentro del mundo(sirve en el caso de trabajar con un simulador), y el puerto donde se lanzará el simulador.

2. Archivo del mundo: es una imagen PNM, en las primeras lineas viene el ancho y alto que ocupa la imagen en pixeles, a continuación viene el choro de byte que constituye la información de la imagen.

El esquema se encarga de leer el fichero de la configuración del mundo 3.2.3. Realizando un análisis sintáctico sobre este, saca el nombre del archivo imagen que contiene el mundo, y el resto de los parámetros como la resolución de la rejilla.

Una vez leídos estos datos, el esquema creará un archivo de texto con la información recogida y hará las llamadas necesarias a la biblioteca *gridslib* para crear y inicializar el mapa. A priori este mundo(la imagen PNM) no es continuo ya que los pixeles de la imagen tienen cierta resolución, y como se ha explicado anteriormente cada pixel sólo puede tener dos valores que representan "ocupado" o "vacío". La única información de posición que sabemos sobre la imagen es que el pixel inferior izquierda es la coordenada (0,0) y la resolución de los pixeles. El objetivo final de este proceso es construir un mapa Figura 4.3(b) en forma de una rejilla 2D formado por celdillas que contienen información de estado "ocupada" o "vacía" y de posición, en este caso las coordenadas del centro de cada cedilla.
El paso de pixeles a celdillas se hace con las siguientes ecuaciones:

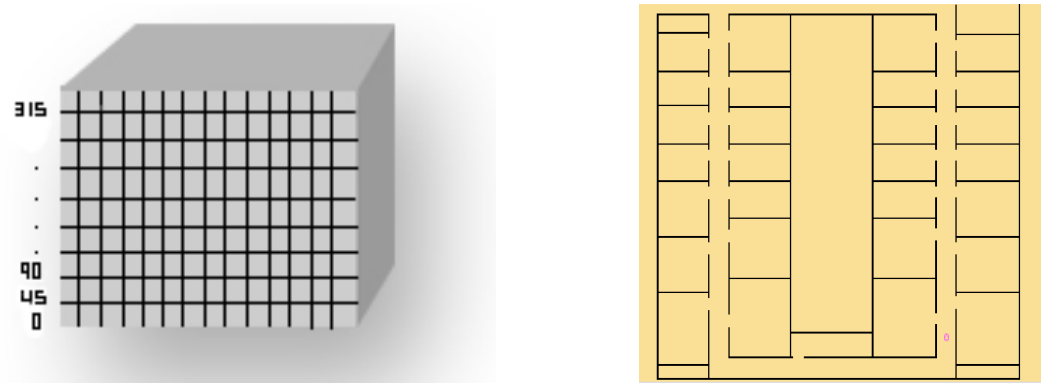
$$Pos_y = \frac{(celda/Grid.size) \cdot Grid.Resolucion}{Mundo.resolucion} \tag{4.1}$$

$$Pos_x = \frac{(celda \bmod Grid.size) \cdot Grid.resolucion}{Mundo.resolucion} \tag{4.2}$$

$$Pixel = (Pos_y \cdot Mundo.columnas) + Pos_x \tag{4.3}$$

Donde *Grid.size* es la dimensión de la rejilla.

Una vez construido, este mapa servirá para realizar los cálculos "teóricos" de las medidas láser. Junto con este el esquema construye un cubo de posiciones (x,y,θ) en forma de una rejilla 3D Figura 4.3(a) que nos sirve para almacenar las probabilidades para cada una de las celdillas de nuestro mapa. En cada posición (x,y,θ) la (x,y) es el centro de la correspondiente celdilla de nuestro mapa, mientras que θ puede tomar 8 posibles valores que representan las 8 posibles orientaciones dentro de esta celdilla. Esta discretización determina la precisión angular del resultado final de la localización, en este caso tenemos una precisión de 45º. En principio el mapa y el cubo están anclados en el espacio bidimensional pero son dos estructuras independientes, y cada una tiene un "rol" distinto en el algoritmo localizador.



(a) Cubo 3D (b) Mapa de ocupación

Figura 4.3: Estructuras básicas del algoritmo

En la Figura 4.3 se presentan las dos estructuras clave de este algoritmo: El mapa de ocupación y el cubo de probabilidad respectivamente. Cada celdilla del mundo tiene un centro (x,y) y permite 8 posibles orientaciones dependiendo del valor de θ, esta información se mapea dentro del cubo como posiciones (x,y,θ) de esta forma podemos almacenar las probabilidades para las 8 posibles orientaciones dentro de cada celdilla de nuestro mapa. Recordaremos que el mapa sirve para calcular el láser teórico 4.3 del cual se saca la probabilidad de una celdilla, este valor se guarda en la posición correspondiente dentro del cubo de probabilidades. Todos los modelos de los que hablaremos en los siguientes apartados se apoyan en estas dos estructuras.

4.3. Modelo de observación láser

En el escenario real, nuestro robot tiene tres cosas un mapa que modeliza los espacios ocupados dentro del mundo, el propio mundo real que le rodea y el sensor láser que representa el "ojo" del robot en este caso. En principio la información capturada por el sensor láser no tiene ninguna información de posición, en este punto interviene este modelo para sacar esta información a partir de las lecturas láser. Para ello se apoya en el mapa Figura 4.3 para calcular la observación láser teórica para cada posición dentro del mundo, y compara esta lectura con la lectura del sensor real. Las posiciones que tienen una lectura teórica "perecida" a la real tendrán más probabilidad , mientras que las que no son compatibles con esta lectura tendrán menos probabilidad. La comparación entre la lectura láser teórica y la real, se hace midiendo "la distancia" entre estas dos lecturas, este valor representa "la semejanza" entre las dos lecturas, del cual sacamos la prob(posición/lectura_laser) este valor es el que usamos para las formulas que entran en juego más adelante.

Este paso sin duda es la columna vertebral de nuestro algoritmo. relacionando el

proceso que hemos descrito antes con las estructuras básicas del algoritmo Figura 4.3, dada una posición (x,y,θ) dentro de nuestro cubo de probabilidades, calcula el *láser teórico* para esta posición, y estima su *distancia* respecto a la lectura que esta dando el sensor láser del robot real (o simulador) en este momento. A partir de esta distancia se usa otra función para calcular la probabilidad de "estar en esta posición".

Para calcular el láser teórico, se usa una técnica ideada de la manera del funcionamiento normal de un sensor láser. Dada una posición (x,y,θ), simulamos "el rayo láser" con una recta en el espacio bidimensional euclidiano. La dirección viene determinada por α y θ Figura 4.4 que representan el desplazamiento angular del láser respecto al eje Horizontal del robot y la orientación de este último respectivamente. Para cada ángulo se calcula un "punto destino", donde las coordenadas vienen determinadas por ecuaciones 4.4 y 4.5:

$$B_x = A_x + RADIO\_LASER \cdot cos(\theta - \alpha - 90) \tag{4.4}$$

$$B_y = A_y + RADIO\_LASER \cdot sin(\theta - \alpha - 90) \tag{4.5}$$

- `A: punto origen`
- `B: punto destino`
- `RADIO_LASER: distancia máxima medida por el sensor láser (8 metros)`
- θ`: orientación del robot`
- α`: constante que representa el desplazamiento del láser respecto al eje horizontal del robot`

La Figura 4.4 muestra los diferentes ángulos que entran en juego. En nuestro caso $\alpha = 0$ porque el sensor láser es paralelo al eje horizontal del robot.

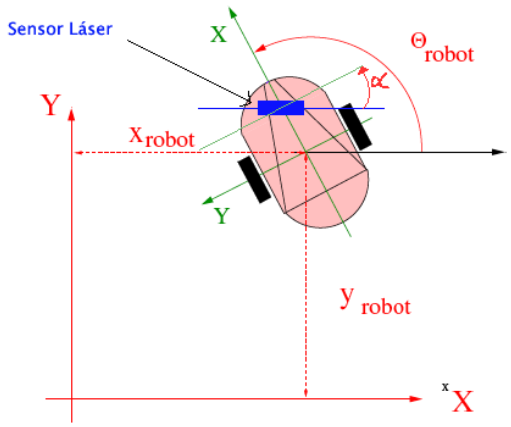


Figura 4.4: los ángulos α y θ

El láser real abarca un ángulo de 180º dando lugar a una precisión de 1º y 180 medidas láser. Queremos que nuestro *láser teórico* tenga las mismas especificaciones así que vamos a tirar 180 rayos donde la trayectoria de cada uno viene determinada por la recta que pasa por el punto A donde queremos calcular el "láser teórico", y el punto destino correspondiente al ángulo de este rayo, con estos dos puntos podemos determinar la trayectoria de cada uno los rayos teóricos.
Para calcular la distancia teórica de cada rayo, tenemos que encontrar el primer obstáculo partiendo del punto origen hacia el punto destino, para ello fijamos un paso λ que representa la distancia entre punto y punto, esta distancia se tiene que elegir de tal forma, que no se "escapan" obstáculos saltando de un punto a otro, pero no tiene que ser demasiado pequeña para no perder eficiencia a la hora de calcular el láser teórico.

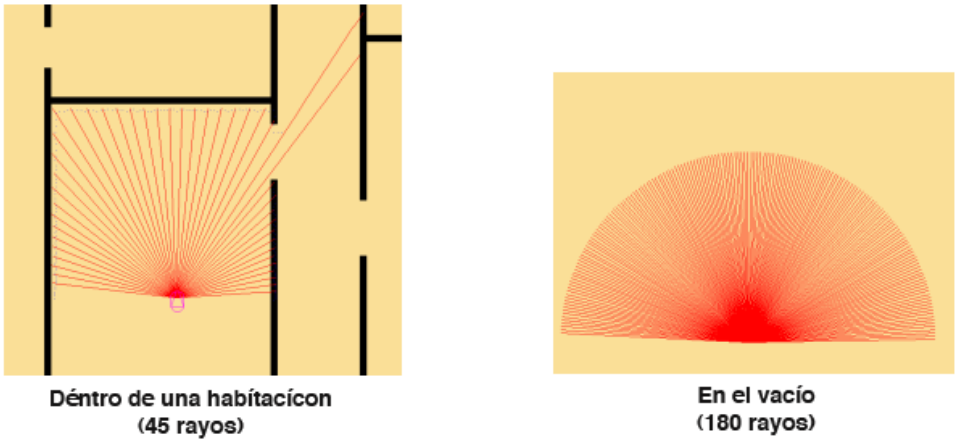
A continuación se muestra al pseudocodiogo de este proceso :

```
A = punto_origen;
angulo = theta - alpha - 90;
for (i=0 ; i<LASER_NUM ; i++)
    {
     B = calcular_punto_destino();
     angulo++;

     for(lambda=0 ; lambda<=1 ; lambda+=salto){
         siguiente_punto.x = A.x + lambda*(B.x-A.x);
         siguiente_punto.y = A.y + lambda*(B.x-A.x);
         celda = calcular_celdilla(siguiente_punto);
         if (celda = ocupada){
              distancia = lambda*RADIO_LASER;
              pasar_al_siguiente_rayo;
         }else{
              pasar_a_la_siguiente_iteracion;
         }
     }
}
```

En Figura 4.5 se muestra un ejemplo de este calculo. Los puntos azules representan el láser real, mientras que los rayos rojos son las *rectas de búsqueda* utilizadas para el

calculo, del láser teórico. En este proceso sólo se han utilizado 45 rayos en vez de 180 por fines ilustrativos.



(a) láser teórico

Figura 4.5: observación teórica calculada sobre el mapa de ocupación

Una vez tenemos la observación teórica, calculamos la distancia entre esta lectura y la lectura dada por el sensor real. Esto se hace con la siguiente función:

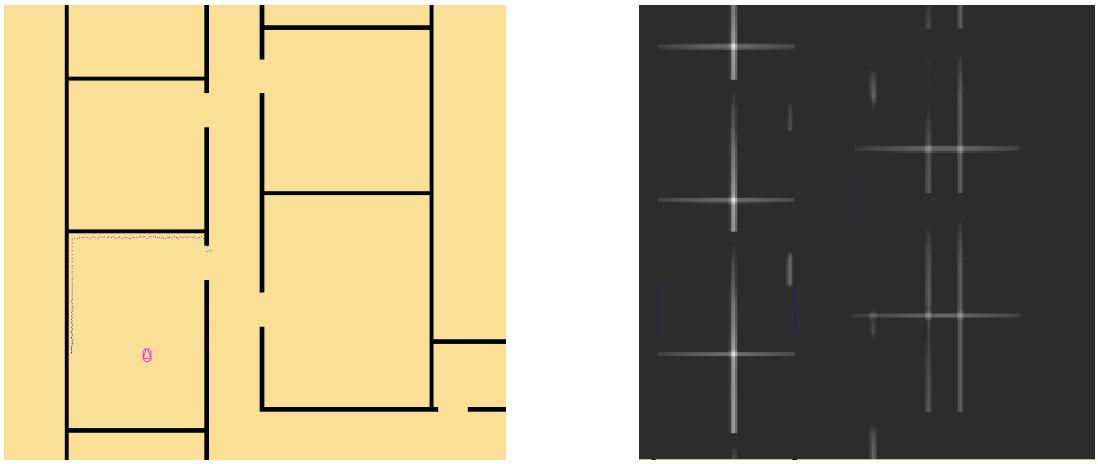
```
int distancia(Tipo_laser laser_real,laser_teorico){
    pos_no_compatibles = 0;
    for(i=0 ; i<LASER_NUM ; i++){
      if (valor_absoluto(laser_real[i]-laser_teorico[i]) > rango_compatibilidad)
      pos_no_compatibles++;
    }
    return pos_no_compatibles;
}
```

Donde *rango_compatibilidad = 20 cm*, es un umbral a partir del cual consideremos que las medidas láser son parecidas. Esto da lugar a un rango de ±10 cm para que las lecturas se consideran parecidas.
La función devuelve el número de *posiciones no compatibles* entre el láser teórico y el real. A partir de esta distancia calculamos la probabilidad de "estar en esta posición" con la formula 4.6:

$$P(_{X_t}) \quad = \quad \frac{LASER\_NUM - distancia(laser\_real, laser\_teorico(_{X_t}))}{LASER\_NUM} \qquad (4.6)$$

Donde X_t es la posición $(x, y, \theta)_t$ del robot y LASER_NUM es una constante que representa el número de rayos láser en este caso es igual a 180.

En la Figura 4.6 se muestra un ejemplo de incorporación de la observación láser. La probabilidad se mapea en niveles de gris donde los valores próximos a "1" se representan con colores claros, y los próximos a "0" se presentan con colores oscuros. Se puede observar que hay varias posiciones con alta probabilidad debido a la simetría del entorno.



(a) mapa (b) observación láser incorporada

Figura 4.6: incorporación de observación

Como ya hemos mencionado anteriormente, El modelo de observación sirve para incorporar nuevas observaciones, aquí surge una pregunta ¿cuando se incorpora una observación?

La incorporación de nuevas observaciones tienen que ser independientes. una observación es independiente de la anterior cuando la distancia entre las dos supera un cierto umbral, que nosotros consideramos suficiente para que sean distintas, cabe aclarar que estamos hablando de observaciones reales, y no de láser teórico. La idea detrás de esto es incorporar la observación cuando *aporta nueva información*, y por otros motivos que veremos más adelante en este capítulo. También se consideran independientes dos observaciones cuando la distancia entre las posiciones desde las que fueron obtenidas supera un cierto umbral fijado en termino de distancias.

4.4. Modelo de observación odométrica

El objetivo principal de este modelo es desplazar la probabilidad, siguiendo los pasos del robot, más en concreto queremos desplazar la nube de probabilidad de la

misma manera que la odometría ha *obsrevado* que se ha movido el robot. Quiere decir que es un movimiento observado por los encoders(sensores de odometría). Estas carácteristicas hacen que este modelo sea más bien de observación odometrica, que un modelo de movimiento. Generalmente, en los algoritmos de localización se realiza un planteamiento clásico de este modelo para el cálculo de nuevas observaciones, donde la probabilidad viene determinada por el estado anterior, la observación en el instante actual y las ordenes que se mandan a los actuadores del robot. Como se ha dicho antes, nuestro modelo tiene un planteamiento diferente y tratará de incorporar el movimiento sin mandar ordenes al robot, sino apoyándose sólo en la información de la odometría, para incorporar los desplazamientos que el robot va efectuando a lo largo del tiempo. De esta manera este modelo puede funcionar en paralelo con cualquier programa capaz de mover el robot.

Como hemos mencionado en el párrafo anterior, nuestro modelo actúa de forma *pasiva* capturando la información ofrecida por los encoders, que permiten saber la posición del robot dentro del mundo en cada momento. Después de procesar esta información este modelo determina si el robot se ha movido o no, y en su caso calcula los desplazamientos (Δx,Δy,$\Delta\theta$) que se han efectuado, usando las formulas 4.8 4.9 4.10.

$$distancia = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{4.7}$$

$$\Delta g_x = \frac{distancia \cdot cos(\theta)}{Grid.resolucion} \tag{4.8}$$

$$\Delta g_y = \frac{distancia \cdot sin(\theta)}{Grid.resolucion} \tag{4.9}$$

$$\Delta g_\Theta = \frac{\Delta m_\Theta}{disc\_theta} \tag{4.10}$$

- ditancia: desplazamiento recorrido por el robot desde que se hizo la ultima incorporación de la observación odométrica
- Δg_x: número de celdillas que serán desplazadas en el eje X
- Δg_x: número de celdillas que serán desplazadas en el eje Y
- Δg_θ: desplazamiento angular

Donde *disc_theta*, es el valor que hemos fijado para discretizar la orientación dentro de una celdilla 4.2. En la formulas 4.8 y 4.9 se ve que en el calculo del desplazamiento interviene la orientación θ del robot, esto es así porque a cada celdilla hay que desplazarla *según su orientación*. El siguiente pseudocódigo muestra el algoritmo de incorporación de la observación odométrica:

```
for(theta=0;theta<360;theta+=disc_theta){

 calcular_dsp_theta(theta);
 dspz_x = calcular_dspx(distancia_recorrida,theta);
 dspz_y = calcular_dspx(distancia_recorrida,theta);

 for(i=0;i<grid.size);i++)
   for(j=0;j<grid.size);j++)
    {
       if (celda_valida(i,j,dspz_x,dspz_y,grid.size))
        {
          //la celda viene desde dentro de la rejilla.
          celda_actual = i+j*grid.size;
          nueva_celda = aplica_dsp_inverso(celda_actual,dspx,dspy);
          intercambia_probabilidades(celda_actual,nueva_celda);
        }else{
          //la celda viene desde fuera de la rejilla.
          asignar_prob_nuetral(celda_actual);
        }
   }
}
```

Se hace un calculo inverso, y en vez de calcular en que celda cae la celda actual si le aplicamos el desplazamiento, se calcula la celdilla a la que aplicando el desplazamiento cae en la celda actual. Si esta celda es valida(viene desde dentro del cubo de probabilidades) hacemos el intercambio, sino le asignamos un valor neutral de la probabilidad Figura 4.8. Este método funciona porque se dispone de dos cubos y no de uno como se ha visto hasta ahora. Aplicar este método usando un solo cubo es imposible ya que se sobrescibirán celdillas dentro del cubo a la hora de desplazar su contenido. Necesitamos otro cubo que actúa como una variable auxiliar de intercambio.

Come hemos mencionado, en vez de un cubo se usan dos, este método se llama: *Doble buffering* Figura 4.7. Es una técnica usada en el mundo de tratamiento de gráficos y imágenes, consiste en mantener dos buffers en paralelo, uno de ellos siempre tendrá el resultado de la operación anterior y es el que ve el usuario mientras el contenido del otro se esta procesando para sacar el siguiente resultado. De esta forma el usuario siempre verá el contenido refrescado, y nunca ve estados intermedios del buffer mientras se esta procesando su contenido. Una vez terminado el proceso sobre el segundo buffer se intercambian los papeles y el se visualiza ahora es el segundo, mientras que las

operaciones pasan a efectuase sobre el primero.
A bajo nivel se mantiene un puntero que siempre apunta al cubo con el contenido refrescado, cuando se incorpora el movimiento este puntero pasa a apuntar al segundo cubo(se considera que el intercambio de punteros es atómico). Este proceso lo lleva a cabo el esquema *laserloc-mallas*, de esta forma el esquema gui_laser_loc usa este puntero para visualizar el contenido del cubo de probabilidad, pero no sabe que se esta haciendo un "doble buffering" en el otro extremo. En la Figura 4.7 se ve una descripcion de este proceso.

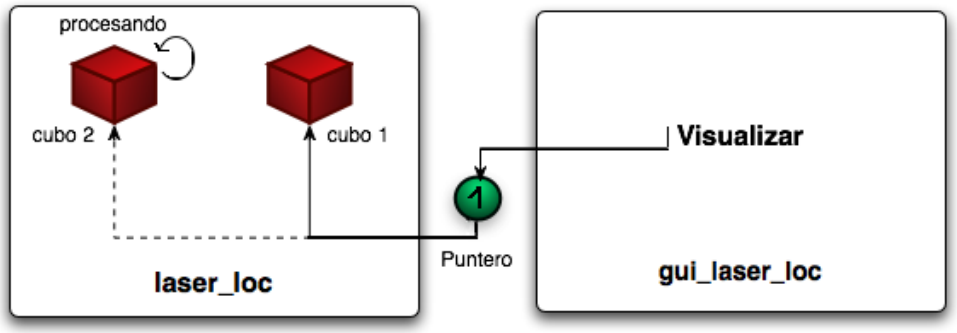


Figura 4.7: Double buffering

Igual que en el modelo de observación surge la pregunta: ¿cuando incorporamos el movimiento?.
Como se ha visto en la sección 4.2 de este capítulo de este capítulo, nuestro mundo esta discretizado en celdillas, y la orientación dentro de cada celdilla esta discretizada en 8 posibles orientaciones (los incrementos en el ángulo son de 45º). Visto la naturaleza de nuestro mundo, la incorporación del movimiento también se va discretizar. Dicho esto sólo se incorporan movimientos cuando el desplazamiento observado supere unos umbrales U_θ o (U_x,U_y) que equivalen a una rotación y una distancia respectivamente, de hecho el umbral que está fijado es en termino de distancias y no de desplazamientos en coordenadas. Cuando se detecta un movimiento, este modelo calcula el desplazamiento (Δx,Δy,$\Delta\theta$) que ha efectuado el robot desde que se incorporó el ultimo movimiento hasta instante actual, y *desplaza* la nube de probabilidades dentro del cubo según los desplazamientos calculados, sin afectar a su forma(valores de probabilidad).
En la figura 4.8, se muestra un ejemplo de un desplazamiento en el eje Y.

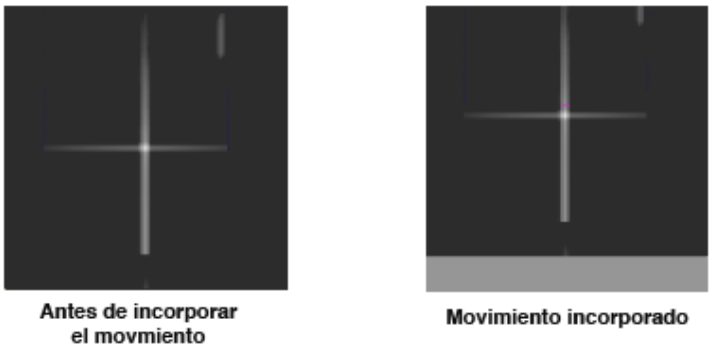


Figura 4.8: incorporación del movimiento

Se puede observar que la probabilidad, se ha desplazado en el mismo sentido que el del movimiento(desplazamiento en el eje Y) efectuado por le robot, también se ve claramente que las posiciones con alta probabilidad(zonas blancas), se han desplazado y siguen rodeando al robot. Se puede observar que en la parte baja aparece una zona gris. Esta zona corresponde a las celdillas recién incorporadas al cubo, a estas se les asigna un valor neutral de probabilidad 0.5 en este caso.

4.5. Acumulación de evidencias con la regla de Bayes

La probabilidad de las posiciones a la luz exclusivamente de la ultima observación no es suficientemente discriminante, ya que dentro del cubo habrá varias posiciones que tienen la misma probabilidad Figura 4.6. Habrá que ir acumulando las evidencias a la luz de todas las observaciones sensoriales recogidas a partir de varias posiciones distintas dentro del mundo. Esta función es tarea del modelo de observación láser, es el que "fusiona" la nueva probabilidad calculada para una determinada celdilla, con la anterior almacenada en la misma.

Cuando se incorpora una observación por primera vez, aparecen en el cubo varias posiciones compatibles con alta probabilidad (generalmente por razones de simetría) el objetivo es ir eliminando ambigüedad hasta llegar a la posición correcta del robot. Usando la regla de Bayes logramos que a lo largo del tiempo y mientras que el robot vaya moviendo por su entorno, incorporando láser/odometría unas posiciones acumulan más probabilidad que el resto, y la nube de probabilidad acaba convergiendo a una posición concreta dentro del cubo de probabilidad.

En el modelo de observación láser 4.3 se ha comentado que la incorporación de una nueva observación tiene que ser independiente de la anterior, en su momento dijimos

que era para que "aporte nueva información". En realidad no es este el único motivo. Que las observaciones sean independientes es un requisito para poder aplicar la *regla de Bayes*, que es la base teórica en la que se apoya este modelo para acumular las evidencias. Esta regla permite fusionar las probabilidades que se van calculando a lo largo del tiempo. Inicialmente todas las celdillas son equiprobables, quiere decir que todas tienen una probabilidad de 0,5.

Siguiendo un desarrollo matemático completo de la regla de Bayes [Thrun, 1997], llegamos a la forma incremental de esta regla donde en vez de manejar productos de probabilidades pasamos a "sumar evidencias" :

$$P(x, y, \Theta/obs_1, obs_2...obs_N) = P(x, y, \Theta/obs_1, obs_2...obs_{N-1}) \cdot P(x, y, \Theta/obs_N) \quad (4.11)$$

$$Pac(x_t) \sim Pac_{t-1} \cdot Pobs_t \quad (4.12)$$

$$\rho ac(x_n) = \rho ac(x_{n-1}) \cdot \rho obs(x_n) \quad (4.13)$$

$$\ln \rho ac(x_n) = \ln \rho ac(x_{n-1}) + \ln \rho obs(x_n) \quad (4.14)$$

Desarrollando la ultima formula recursiva, llegamos la formula general acumulativa :

$$\ln \rho ac(x_n) = \sum_{i=0}^{n} \ln \rho obs(x_i) + \ln \rho ac(x_0) \quad (4.15)$$

Donde:

- $P(x_t)$ es la probabilidad de estar en la posición x en el instante t.
- $obs(x_i)$ observación de la posición x_i, en la iteración i del algoritmo.
- $\rho ac(x_i)$ ratio de probabilidad de la posición x_i, en la iteración i del algoritmo.

En palabras, la ecuación 4.14 calcula "la probabilidad acumulada"(logaritmo del ratio de la probabilidad) en la iteración n de nuestro algoritmo, sumando "la probabilidad acumulada" en la iteración n-1 con el logaritmo del ratio de la probabilidad observada en este momento. El comportamiento de la función $\ln \frac{x}{1-x}$ en el intervalo [0..1] hace que probabilidades mayores que 0.5 suman más probabilidad a la que ya tenemos acumulada, mientras que las menores que 0.5 restan probabilidad y como cabe esperar los iguales a 0.5 no tienen efecto sobre la probabilidad almacenada.

De esta manera la probabilidad en las posiciones compatibles sube, mientras que de las posiciones menos compatibles baja. Queda por aclarar que la probabilidad acumulada no es un valor en el intervalo [0..1] sino es una valor que varia en el intervalo $[-\infty, +\infty]$. Para evitar el manejo de valores infinitos en la función log se establecen dos

umbrales inferior y superior al valor de esta probabilidad, y todo valor que los supere se recorta a estos valores. Para representar estos valores en niveles de gris se hace un mapeo de este intervalo al [0..1]. En la figura 4.9 se ve la curva de esta función y su comportamiento en el intervalo [0..1] :

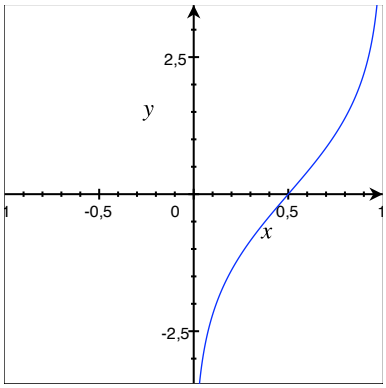
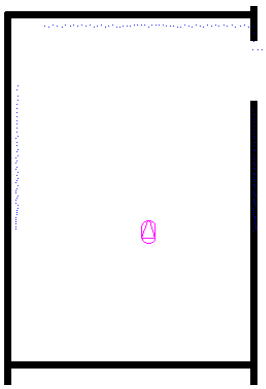


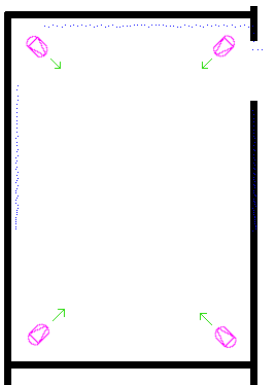
Figura 4.9: $y = ln(\frac{x}{1-x})$

A continuación veremos una secuencia de ejecución típica de nuestro algoritmo, donde intervienen todos los modelos que hemos visto hasta ahora. El robot empieza con una incorporación de observación láser a continuación realiza una serie de movimientos y incorporaciones láser/odometría.

En figura 4.10 se muestra el mapa de la habitación, donde nuestro robot tratará de localizarse.



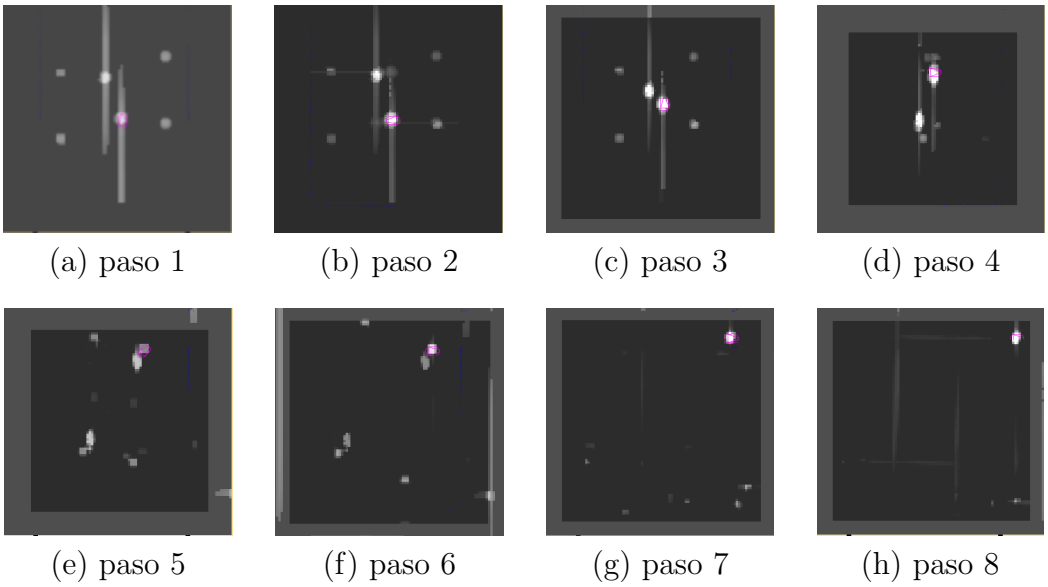
(a) habitación simétrica



(b) posiciones que dan lecturas láser muy parecidas

Figura 4.10: Mapa de la habitación

La figura 4.11 muestra Los diferentes estados de lal cubo durante la localización, a continuación explicaremos cada uno de los pasos.



(a) paso 1 (b) paso 2 (c) paso 3 (d) paso 4

(e) paso 5 (f) paso 6 (g) paso 7 (h) paso 8

Figura 4.11: Ejecución tipica de localización con mallas de probabilidad

Las figuras representan una vista superior del cubo de probabilidad, quiere decir que para cada celdilla se muestra el valor máximo que se ha alcanzado en todas las orientaciones. El valor de la probabilidad esta mapeado en niveles de grises, los colores claros representan valores próximos a 1, mientras que los oscuros son valores próximos a 0.

- Paso 1: corresponde a la primera incorporación de la observación láser, podemos ver que hay cuatro puntos con alta probabilidad, aunque algunos con más probabilidad que otros. También se puede observar que hay dos tramos paralelos de probabilidad que recorren el pasillo central de la habitación pero con menos probabilidad.

- Paso 2: se ha incorporado un giro de +90º y después la nueva observación láser, se ve claramente que el blanco de los puntos probables ahora es más intenso, y que siguen apareciendo los cuatro puntos más probables. Las celdillas que siguen siendo no compatibles con la nueva lectura láser son cada vez más oscuras.

- Paso 3: se ha efectuado un giro de -90º y una translación en Y, después de incorporar este movimiento el robot incorpora la nueva observación láser. En la figura se ve claramente que los puntos más probables se van acercando y cada vez el color blanco es más intenso.

- Paso 4: se han hecho dos desplazamientos siempre en el eje Y, y un giro de -90º. Después se incorporó la nueva observación láser. En esta figura sólo hay dos puntos con alta probabilidad, el resto ha perdido la probabilidad porque llevan varias iteraciones siendo incompatibles, se puede observar que el tramo de

probabilidad que se acumulaba en el pasillo ahora es más fino y esta perdiendo probabilidad. Los dos puntos simétricos siguen apareciendo.

- Paso 5: se ha aplicado un giro de +45º y dos desplazamientos, después se incorporó la observación láser. No hay novedades, siguen apareciendo los dos puntos simétricos. Aparecen nuevas zonas probables en el cubo de probabilidad porque son compatibles con la nueva observación láser.

- Paso 6: después de efectuar una rotación de -45º y una traslación, el robot esta al lado de la puerta, ahí incorporamos la nueva observación láser, como cabía esperar el punto simétrico empieza a perder probabilidad porque la puerta ha roto la simetría que se mantenía hasta el momento. Esta perdida de probabilidad se debe a la Regla de Bayes 4.14. Esto por un lado por otro la posición correcta acumula más probabilidad porque sigue siendo compatible con la nueva lectura láser.

- Paso 7: el robot esta alrededor de la puerta después de aplicar un giro de 45º y una traslación. Incorporando la nueva observación láser, el punto simétrico desaparece difinitivamente del cubo de probabilidad porque lleva dos iteraciones siendo incompatible y se ha bajado su probabilidad. En el cubo aparecen nuevos puntos pero son poco probables.

- Paso 8: después de aplicar una traslación, rotación de -45º y otra traslación, nuestro robot está situado en la puerta. En el cubo sólo existe un punto probable, podemos decir que el algoritmo ha terminado, y el robot está localizado correctamente después de 8 iteraciones de nuestro algoritmo.

En el ejemplo anterior, hemos necesitado 8 iteraciones para localizarse dentro de una habitación(7x7 m^2). Una iteración dura un tiempo considerable 6.1, esto hace que este algoritmo sea muy lento y impracticable para entornos grandes(el departamental por ejemplo), para ello se ha pasado a probar otro algoritmo: el filtro de partículas 5.

4.6. Visualización

para seguir la evolución de nuestro algoritmo y visualizar las distintas estructuras que hemos creado, necesitamos una interfaz gráfica donde se puede ver todo esto y que nos permite variar algunos parámetros de nuestro algoritmo. Afortunadamente la arquitectura JDE permite crear interfaces usando la librería XForms 3.4. sobre esta librería se pueden crear interfaces mantenidos por esquemas JDE, en este caso el esquema *gui_laser_loc* se va encargar de mantener y refrescar las diferentes estructuras gráficas.

gui_laser_loc es un esquema de servicio, quiere decir que siempre esta ejecutando y que el usuario no puede pararlo, este esquema se encarga de varias tareas, a continuación describimos las más importantes :

- leer los datos sensoriales y visualizarlos.
- Activar/desactivar los motores del robot.
- Joystick para teleoperar el robot dentro del mundo.
- Activar/desactivar el esquema laserloc-mallas.
- Visualización del mapa de ocupación.
- Visualización del cubo de probabilidades - dos modos -
- Visualización del láser teórico

La interfaz ofrece un conjunto de botones Figura 4.12 que permiten realizar las funciones antes descritas. Se dispone de un botón para visualizar el mapa de ocupación y otro para visualizar el cubo de probabilidades que permite dos modos distintos de visualización: el primero es *con cortes de ángulos* Figura 4.8 y el segundo es una *vista superior del cubo* Figura 4.11 y 4.12.

El contenido del cubo se pinta en distintos niveles de grises, que mapean el valor de probabilidad almacenado en cada celdilla en niveles de grises. Los colores claros representan valores próximos a 1, mientras que los oscuros son valores próximos a 0.

El esquema permite también visualizar el láser teórico Figura 4.5 dibujando las rectas de búsqueda que hemos descrito en la sección 4.3 para ello basta con activar el botón “laser_teórico” y pinchar en la posición donde queremos obtener la observación teórica. Esto sirve también para hacer trazas y comprobar el buen funcionamiento del modelo de observación láser.
La figura 4.12 muestra esta interfaz con todos sus elementos.

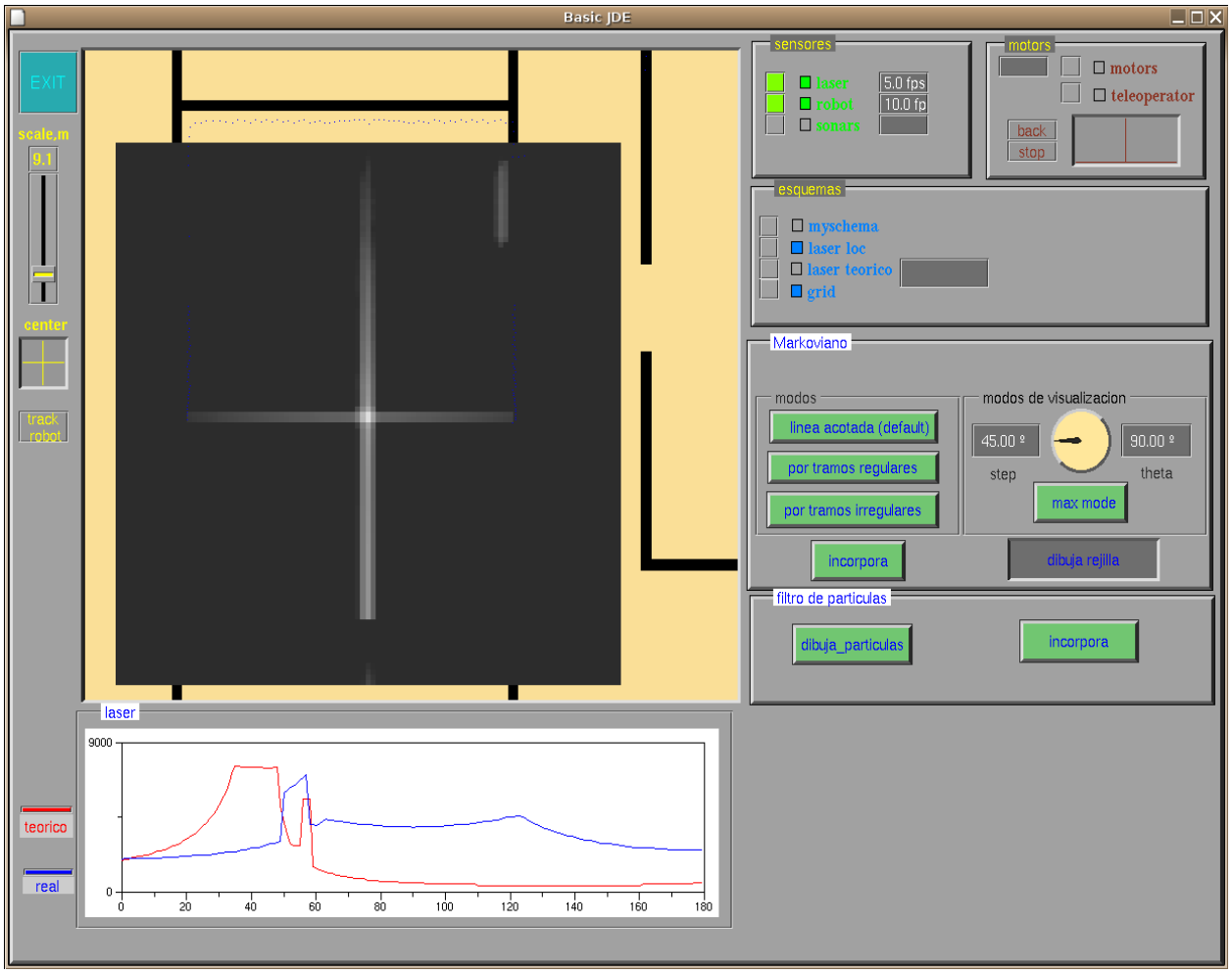


Figura 4.12: gui_laser_loc

Capítulo 5

Localcización con el filtro de particulas

Como vimos en el capítulo 2.1 el objetivo es afrontar el problema de la localización con dos algoritmos distintos. Hasta ahora hemos visto la técnica de mallas de probabilidad que se basa en mantener las probabilidades de todas las posibles posiciones dentro del mundo discreto. Este algoritmo se comporta bastante bien para entornos donde el número de posiciones es reducido, pero el tiempo crece exponencialmente con el espacio abarcado. Para resolver estos problemas hemos optado por el uso de la segunda técnica probabilística objeto de este proyecto: *el filtro de partículas*.

La idea de esta técnica se basa en mantener una población de partículas aleatorias sobre todo el mundo [Fox *et al.*, 1999] [Mackay, 1999] . El objetivo es que la población converja hacia la posición correcta del robot, aplicando de nuevo modelos de observación , movimiento y un nuevo paso: el remuestreo, que se encarga de redistribuir la nube de partículas alrededor de las posiciones más probables de la población. Esta técnica reduce enormemente los costes computacionales y permite llevar la tarea de localización al tiempo real, algo que era imposible con la otra técnica.

5.1. Fundamentos teóricos del filtro de Partículas

El filtro de partículas pertenece a la familia de técnicas de Monte Carlo. Es un conjunto de métodos probabilisticos de remuestreo que han sido desarrollados entre los años 1945 y 1955 para el campo de la física. Una de estás técnicas es el filtro de partículas que actualmente es la base de varios algoritmos de localización, dentro del área de la robótica móvil. La idea principal de este algoritmo se basa en mantener un conjunto finito de Partículas $\{x_i$,P$(x_i)\}$ $i = 1..n$ donde x_i representa la ubicación (x,y,θ) del robot y P(x_i) su probabilidad asociada, calculada mediante un modelo de observación.
El objetivo de este algoritmo es que la nube de partículas distribuidas de forma aleatoria sobre todo el mundo, vaya convergiendo hacia la posición correcta a medida que el robot

va moviendo por el entorno adquiriendo nuevas observaciones sensoriales.

Inicialmente la población de partículas se distribuye uniformemente en (x,y,θ) sobre todo el mundo. En cada iteración del algoritmo se genera una nueva población a partir de la anterior, la cual esta formada por hijos de partículas que tenían alta probabilidad en la población anterior. El algoritmo se realiza en *tres pasos* que se ejecutan *siempre* y de forma iterativa:

- **Modelo de movimiento:** se desplazan las muestras (Δr,$\Delta\theta$) que corresponde al desplazamiento efectuado por el robot desde la ultima vez que se incorporó el movimiento. Se trata de un modelo probabilístico donde a cada partícula se le aplica un ruido gaussiano a la hora de desplazarla.

- **Modelo de observación:** se calculan las nuevas probabilidades de todas las muestras a partir de la lectura del sensor láser. Es el que se encarga de subir la probabilidad de las zonas compatibles y bajar de la de las zonas incompatibles. En principio es el mismo utilizado con el algoritmo de mallas de probabilidad, pero esta vez en vez de calcular la P(celdilla/láser) calcula P(partícula/láser).

- **Remuestreo:** se genera una nueva población a partir de la anterior siguiendo el algoritmo de la ruleta. Su efecto hace que la nube de partículas se concentra alrededor de las posiciones más probables.

Cabe destacar que en este algoritmo no hay "acumulación de evidencias" que era el tercer paso en la técnica de mallas de probabilidad , y por la tanto no se dispone de una historia explícita para cada partícula. Podemos decir que hay una historia implícita, vinculada a la nueva distribución de las muestras que se concentran alrededor de las partículas que eran probables en la población anterior. Esto hace que la nube de partículas acaba convergiendo a las zonas determinadas por las posiciones más probables dentro de la población que es el objetivo principal de este algoritmo.

5.2. Diseño general y algoritmo

La arquitectura en bloques de la implementación que hemos realizado del filtro de partículas es muy parecida a la de mallas de probabilidad. De nuevo el diseño esta basado en dos esquemas JDE, *laserloc-particulas* que se encarga de todo el proceso de la localización dejando el manejo de la interfaz gráfica para *gui_laser_loc*, que ahora tiene más funciones y más estructuras por visualizar. Este último no se ha visto afectado, su interacción con la plataforma no ha cambiado, lo único es que ahora tiene una nueva estructura por visualizar en este caso "las partículas gráficas".

Lo único que ha cambiado es el algoritmo localizador, la interacción entre los esquemas, y la distribución de estas dentro de la plataforma, el API de variables ofrecidas por esta siguen siendo los mismos. En la Figura 5.1 se muestra la nueva jerarquía.

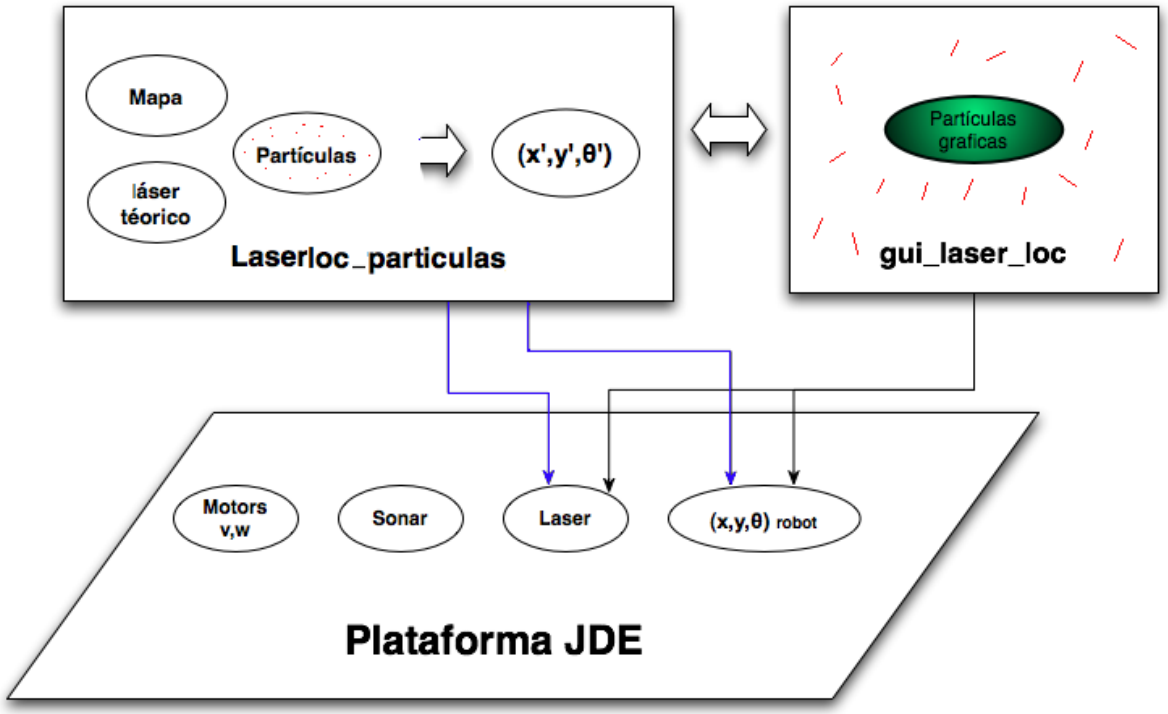


Figura 5.1: Esquemas. Filtro de partículas

Mientras que en la técnica de mallas de probabilidad *laserloc-mallas* generaba un cubo 3D sobre el mundo para almacenar las probabilidades, en este seguimos teniendo el mismo mundo que nos servirá para calcular el láser teórico, pero en vez de un cubo tenemos una población de partículas.

De nuevo *laserloc-particulas* vuelve a albergar toda la complejidad de este algoritmo. Las funciones de este esquema no se han cambiado mucho respecto a las que tenía en el método de mallas, ahora se dividen en:

1. Inicialización del esquema laserloc-particulas.
2. Localización.
   - *a*) Modelo de observación láser.
   - *b*) Incorporación del movimiento.
   - *c*) Remuestreo.

En la fase de inicialización se prepara el mapa del entorno de la misma manera que se hace en con el de mallas de probabilidad 4.2 y se distribuyen las partículas uniformemente en (x,y,θ) sobre todo el mundo. A partir de este momento el algoritmo entra en

un bucle en el que *siempre* se ejecutan *tres pasos*: incorporación de láser, incorporación de odometría y remuestreo ya que no hay "condiciones de incorporación" como era el caso en la técnica de mallas de probabilidad.

En la Figura 5.2 se muestra el flujo de este algoritmo.

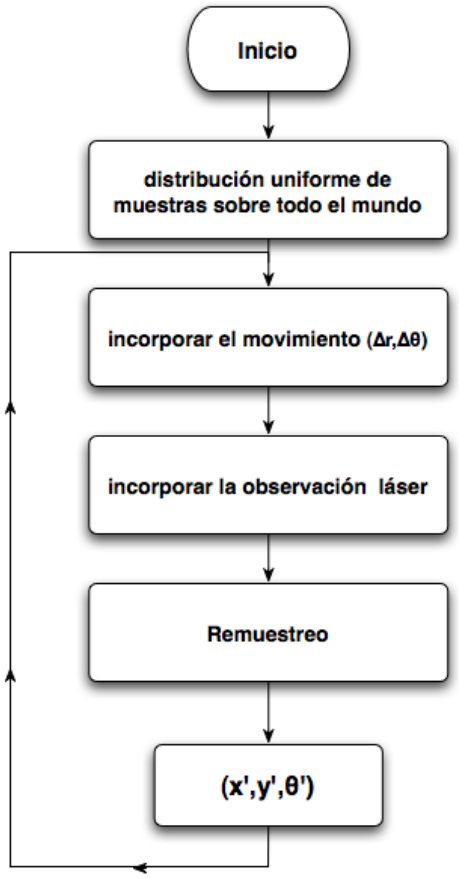


Figura 5.2: Flujo de control del filtro de partículas

5.3. Modelo de observación láser

De nuevo hay que tener un modelo capaz de calcular las medidas láser de forma teórica. En la técnica de mallas de probabilidad se disponía de un cubo tridimensional, y la función de este modelo era estimar la probabilidad de cada una de sus celdillas p(celdilla/laser). En este algoritmo cambia el escenario y en vez de un cubo, tenemos una población de partículas que son tuplas de la forma $\{x_i$,P$(x_i)\}$. Este modelo se encarga de calcular P(x_i) para cada x_i dentro de la población.

La Figura 5.3 muestra un ejemplo de esta incorporación. Las partículas cercanas al robot (o compatibles) tienen un color rojo más intenso ya que son más probables que

las lejanas donde vemos que la mayoría tienen color rosa o blanco.

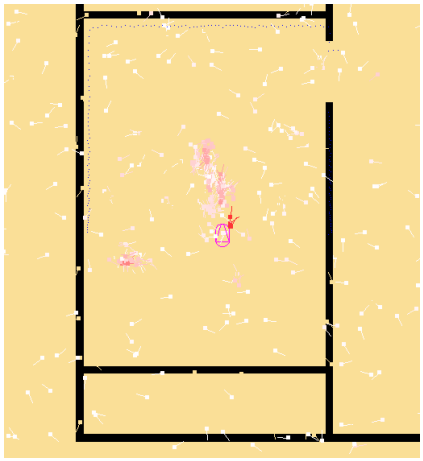


Figura 5.3: Incorporación de observación láser

El calculo del "láser teórico" se hace de la misma manera que en el método de mallas de probabilidad 4.3 en este caso no hay condiciones de incorporación, este paso se ejecuta de manera continua. Aquí podemos notar la diferencia con el de mallas de probabilidad, donde la observación láser sólo se incorporaba cuando se activan las condiciones. En este algoritmo cambian las cosas y las observaciones se incorporan de manera continua, de tal manera que la influencia de una observación láser en la población de partículas puede durar varias iteraciones, tantas como iteraciones del esquema da tiempo a incorporar entre dos observaciones láser consecutivas.
Esto es posible sin perder el tiempo real, debido a que el número de muestras dentro de una población comparado con el número de celdillas dentro del cubo de probabilidad es un valor muy pequeño.

5.4. Modelo de movimiento

Este modelo es el equivalente al de "observación odométrica" de mallas de probabilidad, de nuevo se trata de un modelo pasivo que no necesita mandar órdenes al robot, su funcionamiento se apoya en las lecturas ofrecidas por los odométros para detectar los movimientos realizados por el robot.

La incorporación del movimiento consiste en calcular el desplazamiento (Δx,Δy,$\Delta\theta$) con las ecuaciones 5.15.2 y 5.3, transformarlo a un desplazamiento en coordenadas polares (Δr,$\Delta\theta$) y aplicarlo a todas las muestras de la población. la incorporación en este caso es más fácil que en el otro modelo (el de mallas de probabilidad) ya que lo único que hacemos es aplicar un desplazamiento (Δr,$\Delta\theta$) a todas las partículas.
Los desplazamientos vienen determinados en coordenadas polares(en el otro modelo se

expresaban en coordenadas cartesianas). Para ello usamos Las formulas 5.35.4:

$$\Delta_x = x_{t+1} - x_t \tag{5.1}$$

$$\Delta_y = y_{t+1} - y_t \tag{5.2}$$

$$\Delta\Theta = \Theta_t - \Theta_{t-1} \tag{5.3}$$

$$\Delta r = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{5.4}$$

A diferencia del modelo de "observación odométrica" del algoritmo de mallas de probabilidad, donde los movimientos estaban discretizados, aquí son "continuos" y se incorporan de manera continua por muy pequeños que sean. Quiere decir que la incorporación se hace en tiempo real.

Se trata de un modelo probabilístico en el que a los movimientos se les aplica un ruido gaussiano, cosa que no se hacia en el modelo equivalente de mallas de probabilidad, donde los movimientos se aplicaban sobre las celdillas sin ningún tipo de error. Cada partícula almacena una ubicación (x,y,θ) junto con su probabilidad, la θ representa la orientación de la partícula dentro del mundo. A cada partícula se le aplica un ruido gaussiano, $(\Delta_{gr}, \Delta_{g\theta})$ a la hora de calcular sus nuevas coordenadas. Dicho esto, una muestra P situada en (x_t,y_t,θ_t), sus nuevas coordenadas (x_{t+1},y_{t+1},θ_{t+1}) en el espacio tridimensional vienen determinadas por las ecuaciones 5.55.65.7.

$$x_{t+1} = x_t + ((\Delta_r + ruido\_radio) \cdot cos(\theta_t)) \tag{5.5}$$

$$y_{t+1} = y_t + ((\Delta_r + ruido\_radio) \cdot sin(\theta_t)) \tag{5.6}$$

$$\theta_{t+1} = (\theta_t + \Delta_\theta + ruido\_angulo)\ mod\ 360 \tag{5.7}$$

Donde *ruido_radio* y *ruido_angulo* son el ruido gaussiano aplicado en el radio y el ángulo respectivamente.

De esta manera conseguimos que los hijos que al principio son "copias idénticas" de la partícula generadora Figura 6.2.2(a), se "redistribuyen" alrededor de esta formando una campana de guass Figura 6.2.2(b). Así podemos "explorar" el entorno de esta posición buscando nuevas posiciones compatibles.

(a) Sin ruido gaussiano(varias superpuestas)

(b) Ruido incorporado

Figura 5.4: Efecto del ruido gaussiano

El orden no es muy importante cuando se trata de desplazamientos incrementales (como en nuestro caso), dicho esto es lo mismo aplicar un giro antes que una traslación o viceversa. Cada partícula se desplaza en la dirección indicada por su orientación θ, ya que representa una posible ubicación donde el robot "piensa" estar. En la Figura 5.5 se muestra un ejemplo de esta incorporación. Inicialmente la partícula tiene un ángulo θ=90º mientras que el robot tiene una orientación de 0º. El robot realiza un desplazamiento en (x,y) y un giro de 90º, se ve que la partícula hace el mismo desplazamiento pero en el sentido de su orientación.

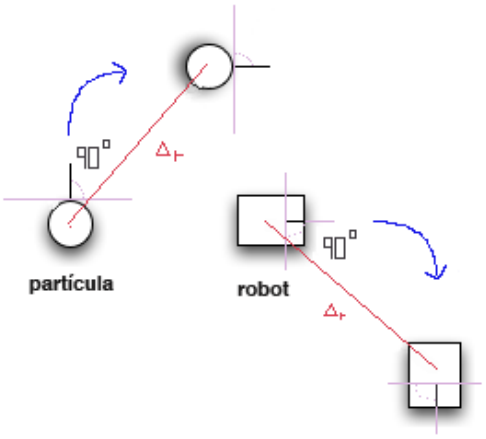


Figura 5.5: Incorporación de movimiento

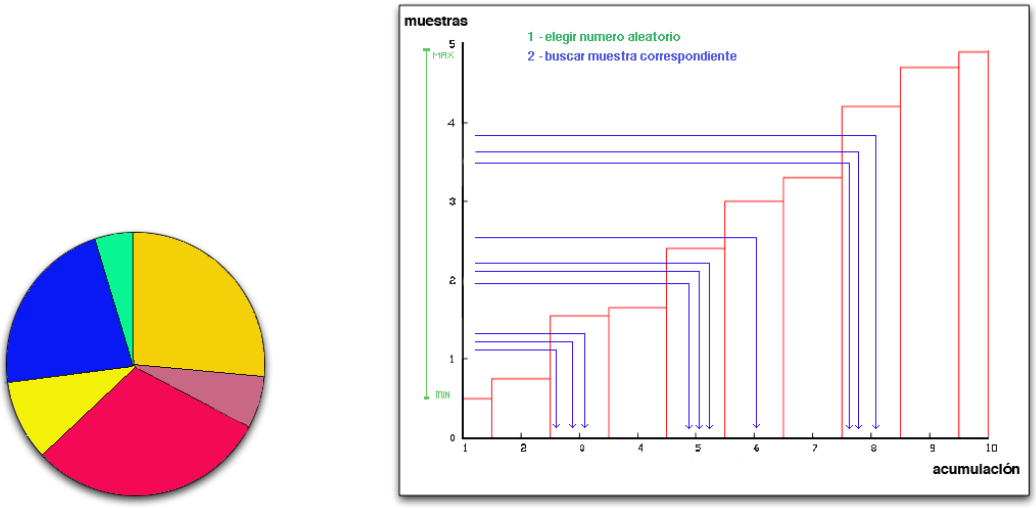
5.5. Remuestreo

El tercer paso del algoritmo del filtro de partículas es el remuestreo. Este paso consiste en generar una nueva población a partir de la anterior de tal forma que la nueva está formada por *las partículas más probables*, para hacer que la nube se concentra en el entorno cercano de las posiciones con alta probabilidad.

Esto se ha implementado con el algoritmo de la ruleta Figura 5.6(a), que consiste en girar la ruleta tantas veces como hijos queremos generar. Cada partícula tiene un sector proporcional a su probabilidad, si sale en la ruleta la partícula genera un hijo "idéntico" en la siguiente población.

Cuanto más ancho será el sector, más opciones tendrá la partícula de ser elegida para generar un hijo en la siguiente población, de esta manera la nube se concentra alrededor de las muestras con alta probabilidad. En principio los hijos son copias idénticas de la partícula generadora pero una vez aplicado el ruido gaussiano Figura 5.4 se "redistribuyen" alrededor de la posición generadora permitiendo de esta manera "explorar" su entorno.
El algoritmo de la ruleta lo hemos implementando acumulando la probabilidad para cada muestra, donde $Pac(x_i) = P(x_i) + Pac(x_{i-1})$. Para determinar las partículas que pasarán a la siguiente población, se realiza un muestreo aleatorio uniforme en el eje que representa la acumulación de probabilidad(que representa la ruleta circular) y seleccionamos la partícula correspondiente en el eje de las muestras Figura 5.6. La idea de este proceso es que las partículas que tienen alta probabilidad producen "picos" en la probabilidad acumulado, y por lo tanto tendrán más opciones de ser elegidas a la hora de muestrear. Una partícula con alta probabilidad puede generar muchos hijos, que antes de dispersarlos en su entorno cercano aplicando el modelo de movimiento 5.4 son copias "idénticas" de la generadora.

En la Figura 5.6(b) podemos ver este proceso(remuestreo). El eje X representa las muestras, mientras que el eje Y representa el valor de la probabilidad acumulada por cada una de estas. En este caso por fines ilustrativos solamente disponemos de 10 muestras, el máximo valor acumulado es de 5, mientras que el mínimo es 0.
El proceso consiste en elegir un número aleatorio(girar la ruleta) entre 0 y 5, a continuación buscamos la partícula que acumula el valor más cercano al número elegido(donde para la ruleta). Podemos ver a partículas que producen saltos en la probabilidad acumulada(como es el caso de las partículas 8 y 3 de la figura 5.6), estas tendrán más probabilidad de que el número aleatorio elegido de manera uniforme cae en su rango, y de hecho ser elegidas para generar un hijo en la siguiente población. La ruleta se gira tantas veces como número de muestras queremos en la siguiente población.



(a) Ruleta (b) Acumulación de probabilidad

Figura 5.6: Proceso de remuestreo

En términos de código, las partículas están representadas por un array de posiciones (x,y,θ), entramos en un bucle en el cual para cada partícula se elige un valor aleatorio entre el mínimo y el máximo valor acumulado por todas las partículas(la suma de todas sus probabilidades), ahora mediante una búsqueda binaria(para acelerar el proceso) buscamos la partícula que tiene la probabilidad acumulada más cercana a este valor, y que sustituyera a la partícula actual.

Una vez que todas las partículas convergen alrededor de la misma posición, el resultado final de este algoritmo es la posición media aritmética de las partículas(posiciones). El siguiente pseudocodigo muestra este proceso:

```
Mientras hay_particulas {
   particula(t)_N = particula_actual;
   Rand = obtener_numero_aleatorio(min_acumulacion,max_acumulaion);
   particula(t+1)_N = obtener_particula_con_acumulacion(Rand);
   posicion_del_robot = Estimar_posicion(particulas);
}
```

Donde N va de 0..N-1 es el número de partículas.

En la figura 5.7 se puede ver la secuencia de la ejecución típica de este algoritmo. En este caso el robot tratará de localizarse *sobre toda la planta*, y no sobre una habitación

como era el caso con el de mallas de probabilidad.

Inicialmente se parte de una distribución uniforme sobre todo el mundo, a continuación el algoritmo entra en un bucle en el cual siempre se ejecutan los tres pasos vistos en este capituló: observación láser, movimiento y remuestreo. En la Figura 5.7(b), podemos ver que después de incorporar la observación láser y muestrear la población, la nube se concentra alrededor de varias posiciones compatibles. Para quitar la ambigüedad realizamos una rotación de 180 grados y incorporamos la observación odométrica y a continuación la observación láser.

Gracias al proceso de remuestreo visto en la sección 5.5 las posiciones "ambiguas" pierden probabilidad y la nube de partículas se concentra alrededor de la posición correcta del robot 5.7(d) tras dos observaciones y un movimiento.

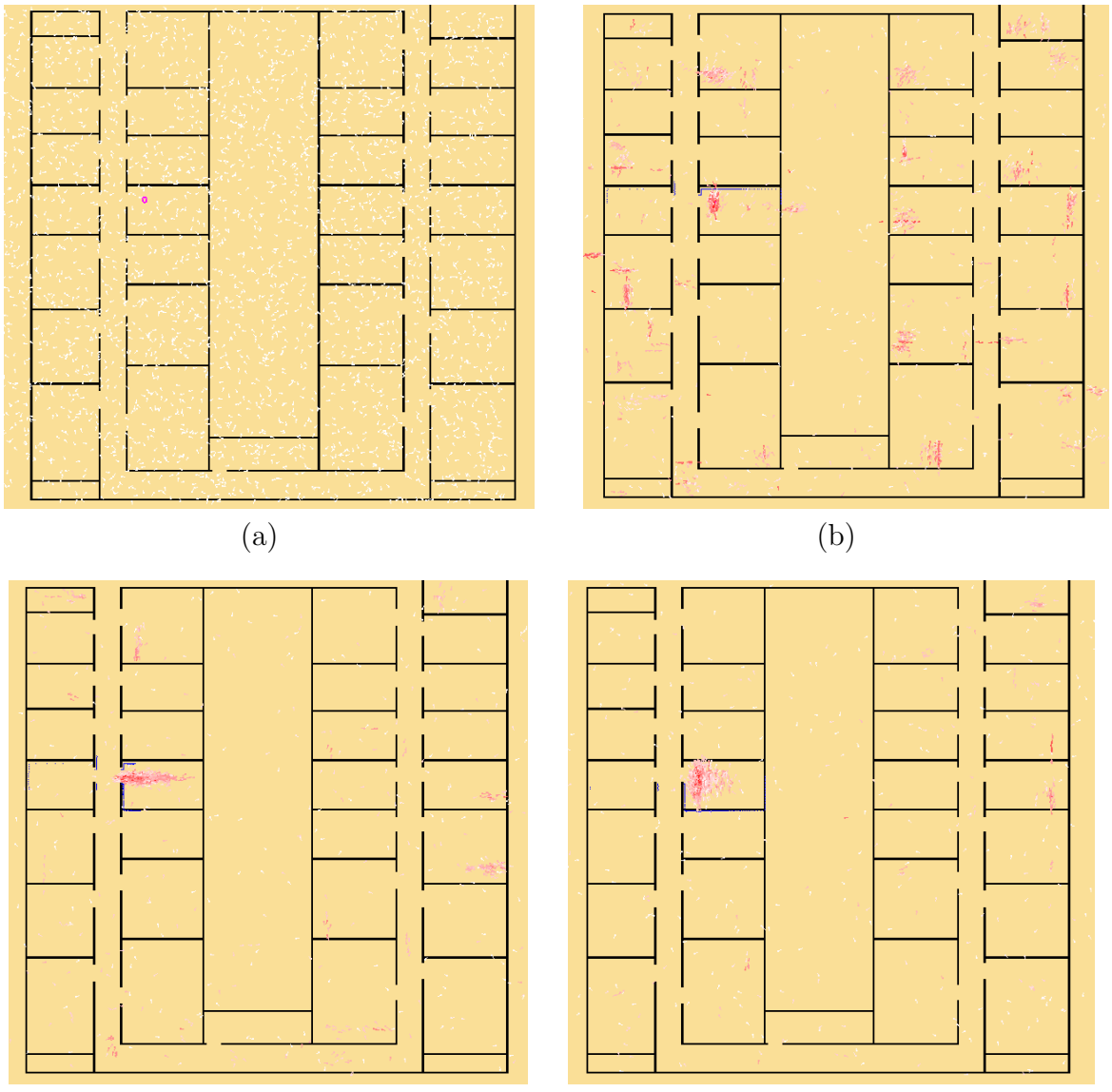


Figura 5.7: Ejecución tipíca del filtro de particulas (simulador)

5.6. Visualización

De nuevo necesitamos un esquema que nos permite visualizar las estructuras en las que se apoya el filtro de partículas, para seguir la evolución del algoritmo, y depurar el funcionamiento de los distintos modelos en los que se apoya.

El esquema se ejecuta todo el tiempo de forma iterativa captando nuevos eventos, fruto de la interacción entre el usuario y la interfaz. En función de estos hace una función o otra. En principio es el mismo esquema utilizado con el método de mallas de probabilidad, y sigue manteniendo las funciones que ofrecía para llevar a cabo esta técnica. Al mismo le hemos añadido nueva funcionalidad para soportar las estructuras gráficas del filtro de partículas. La figura 5.8 muestra la interfaz, con las nuevas estructuras gráficas(partículas).

La nueva interfaz Figura 5.8 incorpora más elementos, para activar y desactivar las funciones relacionadas con el filtro de partículas. Entre estos elementos se puede destacar :

- Botón para pintar las partículas
- Botón para redistribuir las partículas
- Botón para Activar/desactivar el algoritmo

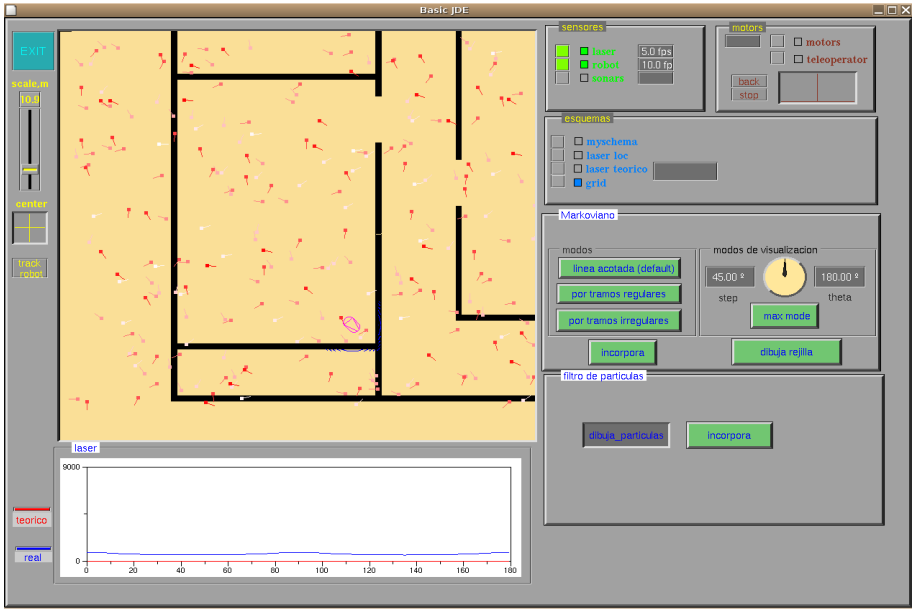


Figura 5.8: gui_laser_loc para partículas

Capítulo 6

Experimentos

Después de detallar la implementación de los dos algoritmos presentados en este proyecto en los capítulos 4 y 5 en este capítulo vamos a describir los experimentos que se han llevado a cabo para probar su comportamiento, explicarlo, ajustar sus parámetros y hacer una comparativa entre los dos.

A continuación veremos los diferentes experimentos que se han llevado a cabo, describiendo en cada caso el efecto en cuestión que se esta estudiando, y los resultados obtenidos.

6.1. Resultados con mallas de probabilidad

Como se ha visto en la sección 4.2 los elementos básicos de este algoritmo son el mapa bidimensional de ocupación y el cubo de probabilidad Figura 4.3. Se han realizado varios experimentos para estudiar el efecto que tiene cada uno de sus parámetros sobre el comportamiento del algoritmo. A continuación describimos los resultados que se han obtenido en cada experimento.

Todas las capturas que se muestran a continuación, representan una vista "superior" del cubo de probabilidades, donde para cada celdillas se muestra el valor máximo de probabilidad que se ha alcanzado para todas las orientaciones.

6.1.1. Resolución y tamaño del cubo de probabilidad

Como comentamos en la sección 4.2 el mundo se divide en celdillas de un tamaño uniforme. Se trata de un parámetro clave en este algoritmo ya que la velocidad dependerá de él de manera directa. Cuanto más pequeño sea este tamaño habrá más celdillas en el mundo, y por lo tanto necesitamos más memoria para almacenar el cubo de probabilidad. Esto hace que el algoritmo será muy lento puesto que hay muchas

celdillas por procesar, sin embargo el resultado final de la localización será muy preciso.

Por el otro lado si este tamaño es grande, habrá menos celdillas dentro del mundo y por lo tanto se reducen los costes en memoria y en tiempo de ejecución del algoritmo que irá más rápido, pero el resultado final tendrá mucho margen de error. Visto esto hay que establecer un valor de compromiso para satisfacer los criterios deseados de nuestro algoritmo: rapidez, menor coste en memoria, buena precisión en el resultado final.
La resolución de las celdillas afectará de manera directa a los tiempos del algoritmo. Se han realizado pruebas con distintas resoluciones. Usando un cubo que alberga una superficie de 7x7 m^2, con celdillas de 70x70 mm^2 se tarda 104s para incorporar una observación láser, mientras que con celdillas de 200x200 mm^2 se tarda sólo 19s. Al final hemos optado por usar 70x70 mm^2, en este caso tendremos un rango de error de ± 35 mm en el resultado final.

El segundo parámetro de vital importancia en este algoritmo es el tamaño del cubo de probabilidad(el espacio albergado). Normalmente el cubo tiene que cubrir todo el mundo, pero esto aveces es imposible porque el algoritmo acaba siendo muy lento. Se han realizado experimentos con cubos que albergan distintas porciones del mundo la tabla 6.1.1 muestra un resumen de los tiempos conseguidos:

tamaño del cubo(m^2)	Nº de orientaciones	tiempo de una observación(s)
7x7	8	104
	16	212
15x15	8	621
	16	1204

Cuadro 6.1: efecto del tamaño de la rejilla en los tiempos (simulador)

Como se puede ver en la tabla, usando un cubo que cubre una superficie de 15x15 m^2 y con 8 orientaciones, tardamos casi 10 min para incorporar una sola observación. Visto esto para 6 o 7 iteraciones que es la media de iteraciones de nuestro algoritmo, el robot tardará más de 1 hora para localizarse. Para un cubo que abarca todo el mundo (más de 800 m^2), el tiempo estimado supera 10 horas. Esto hace que la localización sea casi imposible con este algoritmo (en tiempo real).

Una medida para paliar el problema del tiempo es almacenar la observación teórica para cada celdilla. Esto se hará solo una vez en el inicio del proceso, y los cálculos posteriores se limitan a comparar la observación real con la teórica, en vez de tener que calcular esta ultima para cada celdilla. Esta solución funciona bien si el entorno es pequeño, pero los gastos en memoria se disparan cuando aumenta en el número

de celdillas. Para dar una idea de este proceso suponemos una rejilla de 100x100x8 (100x100 celdillas con 8 posibles orientaciones) que son 80000 posiciones dentro del cubo de probabilidad. La memoria necesaria para almacenar esta estructura es de: 80000*tamaño_celdilla , suponemos que cada celdilla ocupa 200 B (suele ocupar más) vamos a necesitar 15.25 MB para almacenar el cubo de probabilidad. Haciendo los mismos cálculos para un cubo que alberga todo el mundo (680x680x8 posiciones usando celdillas de 70 mm^2) necesitaremos 0.6 GB de memoria RAM para almacenar el cubo de probabilidad.

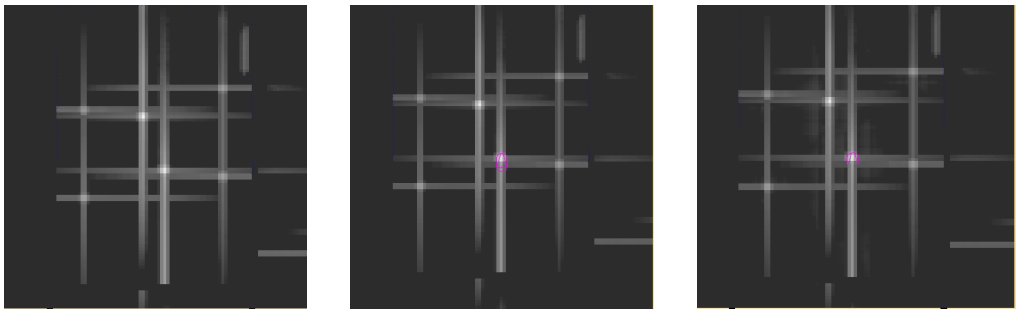
Visto la gran cantidad de gastos en memoria, hemos optado por calcular la observación teórica sobre marcha pagando el coste temporal en vez del coste en memoria.

6.1.2. Número de orientaciones

Aunque nos gustaría tener una celdilla por cada grado en al ángulo de orientación, esto en la práctica es casi imposible puesto que dispararía los costes computacionales(tiempo y memoria) de manera enorme. Visto esto, tal y como se vio en la sección 4.2 el mundo se va discretizar en orientaciones para aliviar costes.
Se han hecho pruebas con 4, 8 y 16 orientaciones. Con 16(precisión de 11.75ª) el coste es alto y el número de celdillas es muy grande, con 4(precisión de 45º) los costes son razonables, pero el margen de error cometido es bastante grande. Visto esto la elección final ha sido de 8 orientaciones como valor intermedio, permitiendo una precisión de 22.5º. Este número es suficiente para localizarse dada la naturaleza de nuestro mundo (Planta de la URJC Figura 4.3(b)) porque el número de direcciones "privilegiadas" se reduce a 4 direcciones ya que el mundo esta "cuadriculado" y no contiene muchas curvas.
En la tabla 6.1.2 se muestra un resumen de los tiempos obtenidos:

Número de orientaciones	tiempo que tarda una incorporación (s)
4	59
8	118
16	237

Cuadro 6.2: efecto del número de orientaciones sobre los tiempos(simulador).



(a) 4 (b) 8 (c) 16

Figura 6.1: posibles orientaciones en el mundo (simulador)

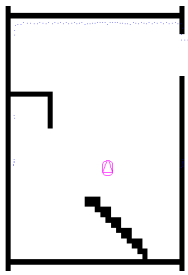
En la figura 6.1 podemos ver que no hay gran diferencia entre las figuras, podemos notar un poco de diferencia entre la figura de 8 y la de 16 orientaciones que tiene un poco de probabilidad alrededor de los tramos que acumulan mucha probabilidad. Esto se debe a que la función de distancia 4.3 descarta muy bien las posiciones poco probables.

6.1.3. Modelo de observación

Como se mencionó en la sección 4.3 es el modelo de observación que se encarga de calcular las probabilidades para todas las celdillas del cubo. Para ello se basa en una función que transforma la distancia entre el láser teórico y el láser real en una probabilidad.
Se han hecho experimentos con varias funciones distintas a continuación veremos los resultados obtenidos con cada una realizando la misma secuencia de pasos y incorporaciones. La secuencia común para los tres experimentos es la siguiente:

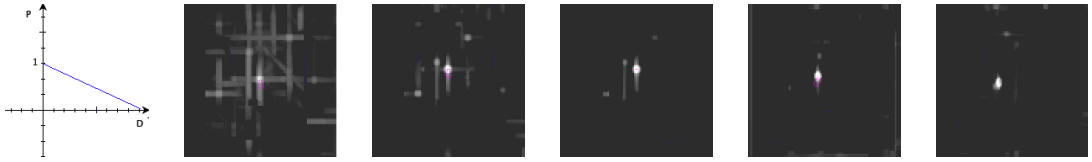
1. observación
2. traslación de 50 cm y observación
3. rotación de 180º en sentido trigonométrico, y observación
4. rotación de 45º en sentido horario, traslación de 50 cm, y observación
5. traslación de 50 cm, rotación de 45º en sentido horario, y observación

Los experimentos se han realizado sobre el simulador, utilizando el mapa de la Figura 6.2:



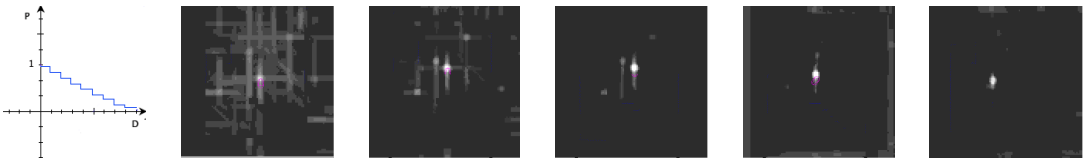
(a)

Figura 6.2: Mapa de la habitación



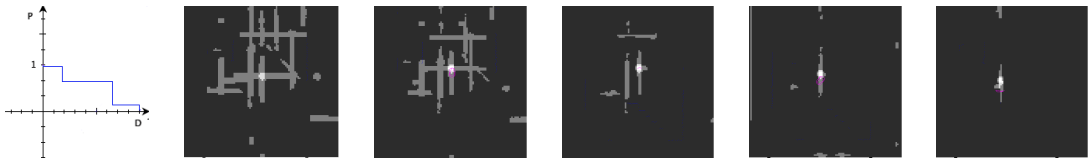
(a) (b) (c) (d) (e) (f)

Figura 6.3: Modelo lineal de observación láser



(a) (b) (c) (d) (e) (f)

Figura 6.4: Modelo lineal por tramos regulares de observación láser



(a) (b) (c) (d) (e) (f)

Figura 6.5: Modelo por tramos irregulares, de observación láser

Analizando el comportamiento de la tres funciones, podemos observar que las de "por tramos" *tardan* más en olvidar la simetría, mientras que la "lineal" olvida de manera *demasiado rápida*. Otra propiedad destacable de la función lineal es su comportamiento "discriminante" a la hora de asignar probabilidades a las celdillas del cubo, en la figura 6.3(b) se ve que a partir la primera incorporación láser, la posición correcta ya tiene la probabilidad más alta en todo el cubo, mientras que con las otras no destacan tanto en probabilidad la posición real del robot

Dicho esto queremos una función que coja lo mejor las tres las anteriores, quiere decir que no sea "muy discriminante" (porque el mundo real contiene mucho ruido), y que olvida de manera razonable. La función elegida al final es una combinación de las tres anteriores. Se trata de una función definida por tramos sobre el intervalo [0..180]. A continuación describimos el pseudocódigo de esta función.

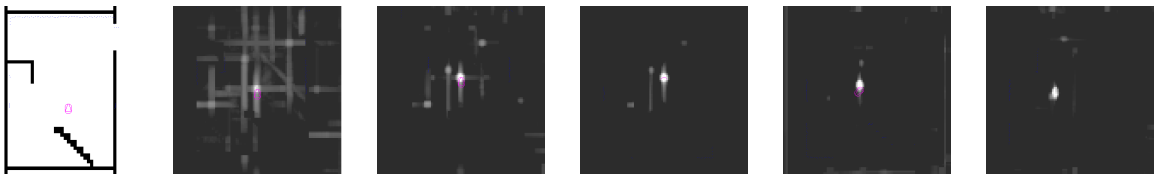
```
probabilidad(int distancia){
float aux_prob=0;
 if (LASER_NUM-distancia<70) {
      aux_prob = 0.2;
    }else if (LASER_NUM-distancia>135) {
      aux_prob = 0.9;
    }else {
      aux_prob = (LASER_NUM-distancia)/(LASER_NUM);
   }

return aux_prob;
}
```

6.1.4. Simetría

Uno de los problemas más importantes, que nuestro algoritmo tiene que enfrentar y resolver es la simetría en el entorno, por lo que "despista" al algoritmo localizador. Se han realizado varios experimentos con distintos niveles de simetría en el entorno, con el fin de probar el comportamiento de nuestro algoritmo ante estas situaciones. A continuación veremos capturas de estos experimentos, donde en general el robot se dirige a la puerta para romper la simetría.

- *Baja simetría*.



(a) mapa (b) (c) (d) (e) (f)

Figura 6.6: Localización en entornos con baja simetría (simulador)

- *Simetría media* .

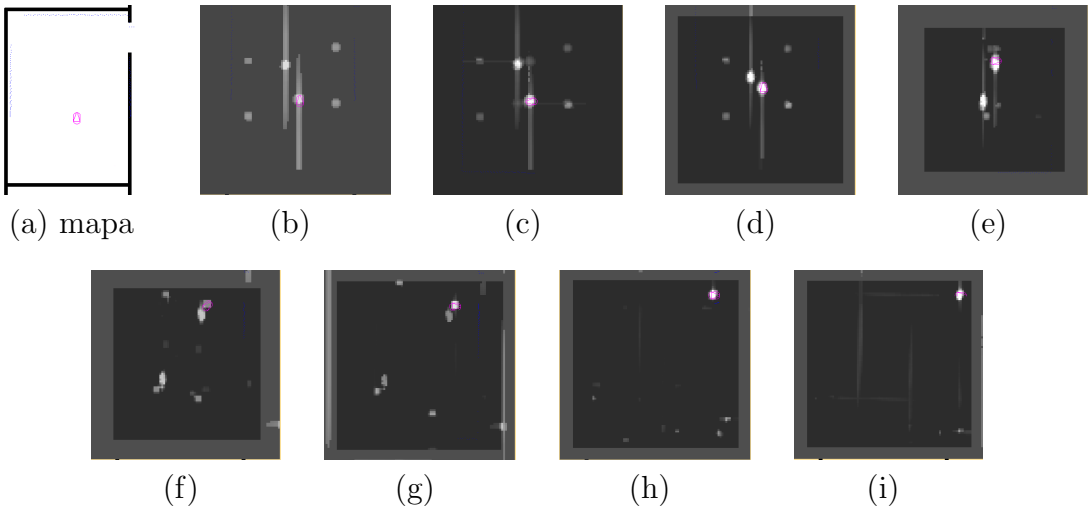


Figura 6.7: Localización en entronos de simetría media (simulador)

- *Alta simetría*.

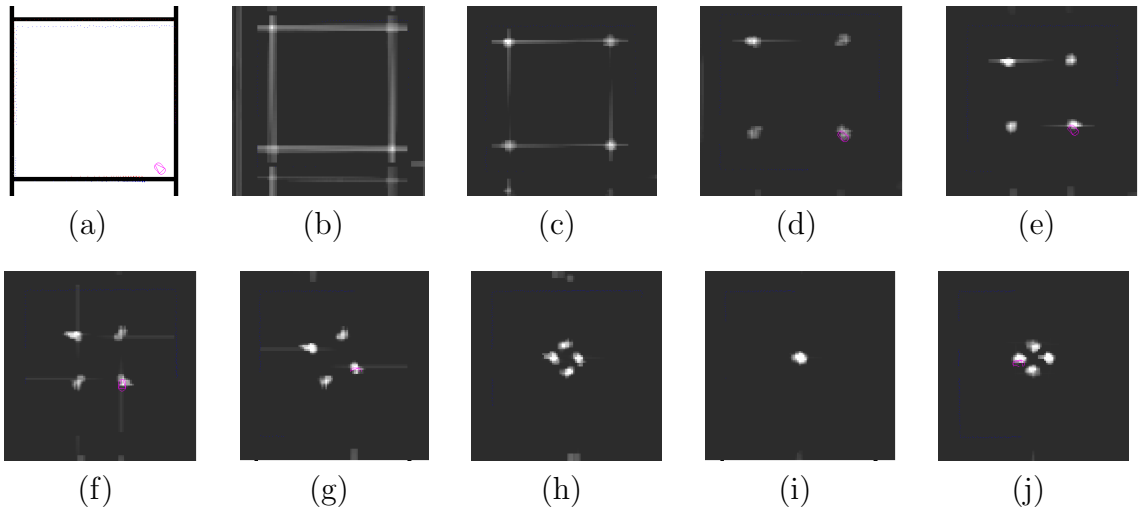


Figura 6.8: Localización en entornos con alta simetría (simulador)

Podemos ver que depende de la simetría el algoritmo tarda más o menos iteraciones antes de llegar a la posición correcta. En la figura 6.6 podemos ver que en un entorno con poca simetría se han necesitado 4 iteraciones antes de encontrar la posición real correcta del robot, mientras que en un entorno con simetría media como es el caso de la Figura 6.7 ha necesitado 7 iteraciones, para localizarse. Esto de una parte por la otra podemos ver que en la figura 6.8 no ha podido localizarse ya que la simetría es perfecta (una habitación cuadrada cerrada), y el flujo de observaciones no es totalmente discreminativo para poder desambiguar la situación. aún así mantiene 4 posiciones con alta probabilidad.

6.1.5. Experimentos *sobre el robot real*

El objetivo principal de este proyecto era probar los dos algoritmos sobre el robot real, en este caso el pioneer 3.x. Se han realizado varios experimentos para probar el comportamiento del algoritmo sobre este robot. A continuación se muestra una ejecución típica sobre el robot real en un entorno con simetría media:

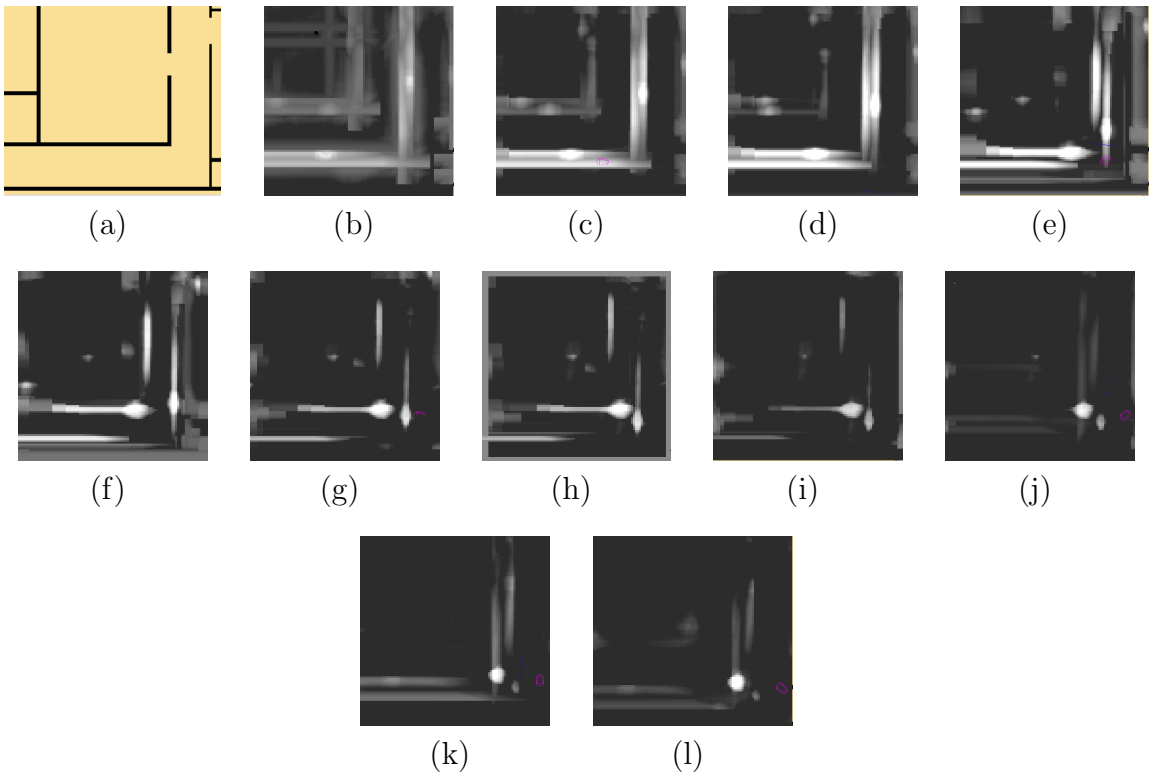


Figura 6.9: Localización en simetria media - robot real

En la figura 6.9 podemos ver una secuencia de localización utilizando el robot real en un entorno con simetría media. La figura 6.9(a) muestra el entorno real, donde el robot tratará de localizarse. En la primera captura 6.9(b) vemos que la probabilidad se concentra en el centro de los pasillos tras la primera observación. A medida de que el robot avanza hacia la esquina incorporando observación y movimiento la ambigüedad se va quitando y la probabilidad se concentra en las dos zonas compatibles. Esta situación permanece hasta el instante 6.9(f). Para desambiguar hacemos una rotación hacia el rincón del pasillo, se ve que después de este movimiento el punto simétrico desaparece y la probabilidad se concentra en la posición real correcta del robot.

El algoritmo funciona muy bien sobre el robot real, el único problema que padece es el tiempo excesivo necesario para localizarse. Para la localización anterior el robot ha necesitado más de media hora.

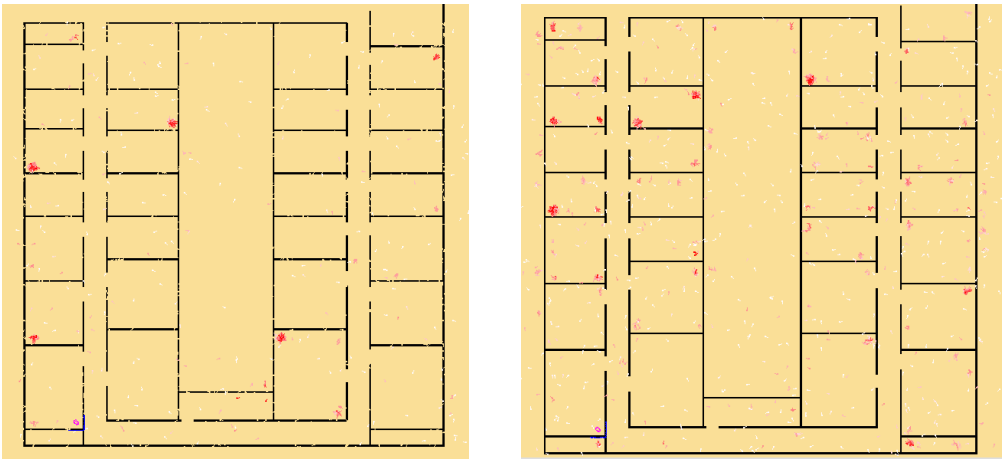
6.2. Resultados con filtro de partículas

Se han realizado varias pruebas, para experimentar el comportamiento de este algoritmo. Todas están orientadas a analizar el funcionamiento de los distintos modelos en los que se basa, y ajustar los distintos parámetros para optimizar su rendimiento. A continuación, veremos las distintas pruebas que se han llevado a cabo, y comentaremos el efecto que se esta estudiando en cada caso, y los resultados obtenidos.

6.2.1. Número de partículas

Uno de los parámetros clave de este algoritmo es el número de muestras. Un número grande de partículas asegura la convergencia del algoritmo al lugar concreto, ya que en este caso las partículas "abarcan" casi todas las zonas del mundo, y hay más probabilidad de que caigan cerca de la posición correcta, sin embargo los costes computacionales aumentan, y algoritmo acaba siendo lento.
En el otro extremo si este número es pequeño la ejecución será más rápida, pero no se puede asegurar su convergencia, y hay menos esperanza de que las partículas caigan cerca de la posición correcta del robot.

Para analizar el efecto de este parámetro, Se han realizado experimentos con 2000 y 5000 partículas. En la figura 6.10 podemos ver que con 5000, las partículas se han concentrado alrededor de varias posiciones posibles dentro del mundo, aumentando de esta manera la probabilidad de que una de estas sea la posición correcta del robot, mientras que con 2000 muestras sólo han abarcado unas cuantas posiciones, en este caso hay menos esperanza de que la nube converja hacia la posición correcta del robot, su convergencia dependerá del ruido gaussiano 5.4, del cual hablaremos en el siguiente punto de este capítulo.



(a) 2000 muestras (b) 5000 muestars

Figura 6.10:

Finalmente se ha establecido un valor de compromiso de 3000 partículas visto que ha dado buenos resultados, con gastos de tiempo-memoria bastante razonables.

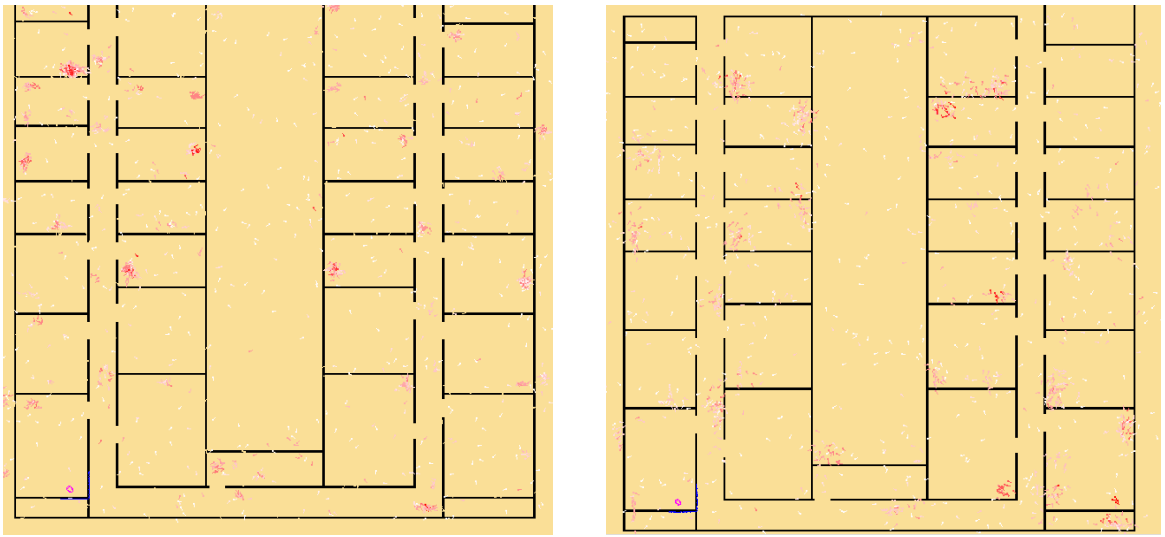
6.2.2. Ruido gaussiano

El ruido gaussiano es esencial en este algoritmo, ya que es el que da el "poder exploratorio" a las partículas introduciendo una variedad en la población de estas. Si lo quitamos perderemos estas características. Como hemos mencionado en el modelo del movimiento 5.4, a todas las partículas recién generadas se les aplica un ruido gaussiando a la hora de calcular sus nuevas coordenadas. De nuevo hay que establecer un compromiso a la hora de elegir este valor.

Partimos de un ruido "nulo" (sin ruido), en este caso las partículas serán *copias idénticas* de la generadora sin moverlas Figura , y no podrán "explorar" el entorno de esta(que era uno de los objetivos para meter este ruido). La ventaja en este caso es que si caen en la posición correcta la precisión va ser muy alta.
En el otro extremo, un valor grande de este ruido permite que las partículas se mueven alrededor de la generadora buscando nuevas posiciones compatibles Figura , sin embargo siempre habrá una cierta desviación típica en el resultado final, ya que las partículas "no paran", están moviendose continuamente alrededor de una posición en concreto.

Dicho esto hay que coger valores razonables, permitiendo que las partículas puedan "explorar" el entorno, pero tomando en cuenta la precisión obtenida en el resultado final. Después de probar el comportamiento con varios valores, finalmente se ha escogido un valor de 400 mm, para *ruido_radio* y 35º para *ruido_ángulo*.

En la figura 6.11 podemos ver el efecto de este parámetro. Se ve claramente que en la figura (a) las nubes están más concentradas alrededor de las posiciones con alta probabilidad, mientras que en la (b) se quedan dispersas, este parámetro influirá en la precisión del resultado final de la localización.



(a) 100 mm (b) 500 mm

Figura 6.11:

6.2.3. Experimentos sobre el robot real

Después de probar el funcionamiento de este algoritmo con el simulador Player/Stage, se ha pasado a las pruebas sobre el robot real. Los resultados han sido muy parecidos a los obtenidos en el mundo simulado, a diferencia de que ahora las lecturas presentan más desviaciones respecto del mapa, debido errores de los sensores 1.2.1. En general la simulación del sensor láser es muy próxima al comportamiento real, debido a el sensor láser se comporta muy bien en la realidad(pocos errores).
En la figura 6.12 se muestra una ejecución del algoritmo en el pioneer 3.x 3.1.
En la prueba se ha utilizado un ruido gaussiando de 600 mm en el *ruido_radio* y 40º en el *ruido_angulo*, con un conjunto de 5000 partículas. El robot ha necesitado unos 40 segundos para localizarse correctamente dentro del mundo.

El robot esta situado en el pasillo de la esquina inferior derecha del mundo, a la hora de incorporar la primera observación aparecen varias zonas compatibles (en los pasillos), pero a medida de que el robot incorpora más observaciones láser avanzando hacia la esquina, la ambigüedad va desapareciendo y la nube acaba convergiendo hacia la posición real correcta del robot. En la ultima figura podemos observar una desviación en el láser(azul), esto se debe a que el láser se dibuja en la posición estimada por la odometría, que acumula error de posición.

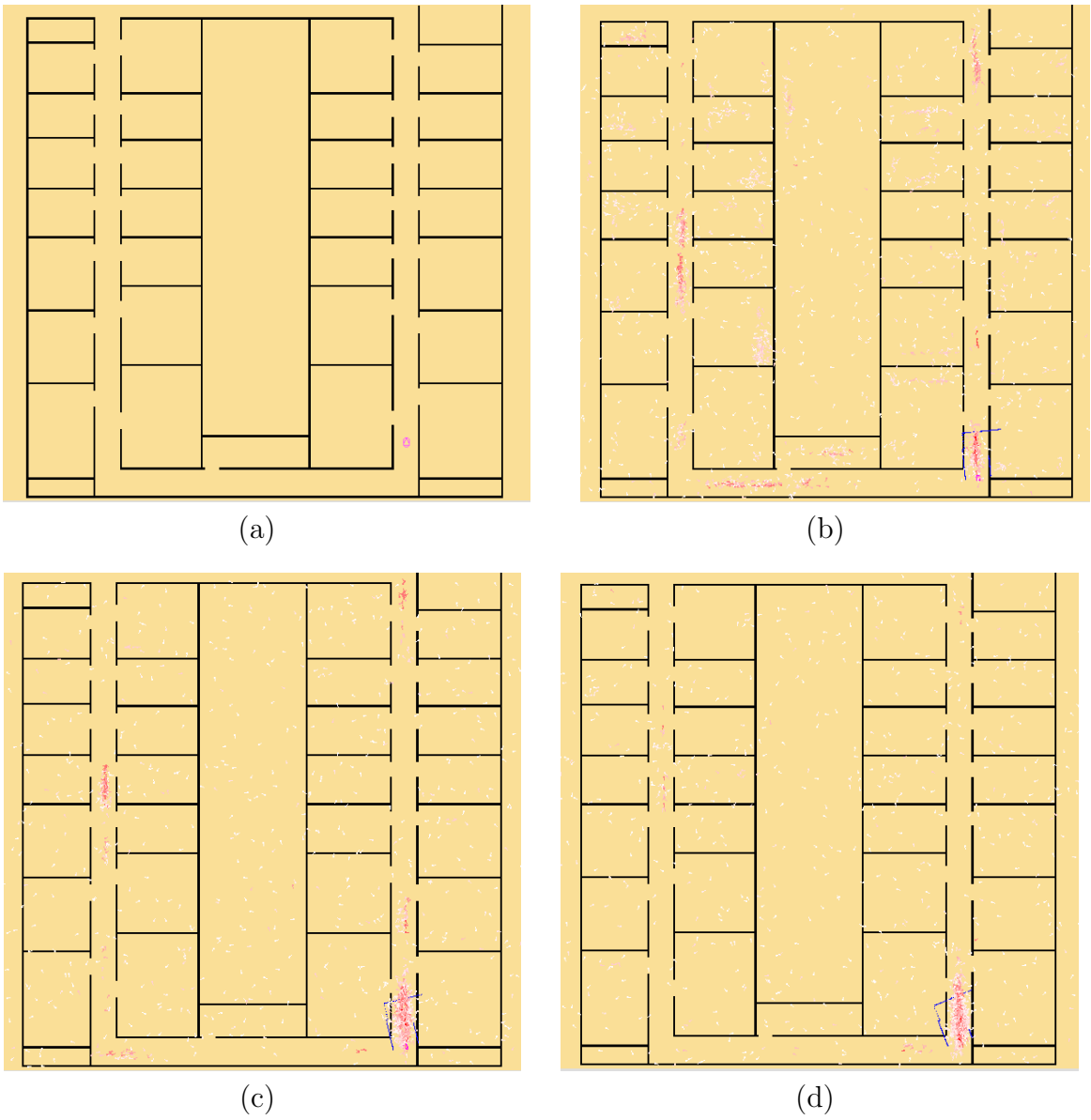


Figura 6.12: Ejecución tipíca del filtro de partículas (robot real)

6.3. Comparativa entre los algoritmos

Después de experimentar los dos algoritmos tanto sobre el simulador como en el robot real, y tomando en cuenta los requisitos descritos en la sección 2.2, podemos decir que el "filtro de partículas" es el que responde a estos requisitos de la manera eficaz posible.

Este algoritmo permite localizarse sobre toda la planta del departamental URJC (más de 800 m^2) en tiempo real (menos de 1 minuto), consumiendo poca memoria(las partículas son estructuras "ligeras") y tiempo de proceso, mientras que el de "mallas de probabilidad" se limita a entornos pequeños(habitación $15m^2$) consumiendo mucha memoria (para almacenar el mapa de ocupación y el cubo de probabilidad) y tiempo

de proceso. Para localizarse sobre toda la planta necesitará más de 10 horas, esto contradice con el 5(localización en tiempo real) considerado como uno de los requisitos principales de este proyecto 2.2.

Sin embargo la técnica de "mallas de probabilidad" puede afrontar situaciones con alta simetría y resolverlas de manera eficaz, cosa que el filtro de partículas no pudo tratar con robustez debido al problema al problema de "convergencia prematura". Este problema consiste se debe a que el algoritmo no puede mantener para mucho tiempo subpoblaciones de muestras en zonas compatibles/ambiguas. Visto esto el algoritmo aveces converge a posiciones simétricas y no a la posición correcta. Se trata de un problema clásico de este algoritmo, resolverlo no es intuitivo ya que es debido a la propia dinámica del filtro de partículas, y de más cosas coma la función de distancia en el modelo de observación láser 4.3, ya que es la que da "la inteligencia" a las observaciones.

Resumiendo lo anterior el filtro de partículas es el mejor candidato para acompañar a otro algoritmo de navegación, ya que permite localización en tiempo real consumiendo pocos recursos del sistema.

Capítulo 7

Conclusiones y trabajos futuros

Después de plantear los objetivos de este proyecto en el capítulo 2.1, y detallar la implementación y el diseño de los dos algoritmos presentados en los siguientes capítulos, llega el momento de resumir las conclusiones generales sobre este proyecto, y ver hasta que punto se han logrado los objetivos. Hablaremos también de las posibles lineas futuras que puede seguir este proyecto.

7.1. Conclusiones

Teniendo en cuenta los objetivos que hemos planteado en el capítulo 2.1, que principalmente se dividen en tres partes: implementar el algoritmo de "mallas de probabilidad", implementar "el filtro de partículas", y experimentar los dos algoritmos sobre el robot real y simuladores. En esta sección vamos a repasar cada uno de estos objetivos comentando hasta que punto se ha logrado , y como se ha logrado.

El objetivo principal estaba dividido en 3 subobjetivos, para cumplir el objetivo 1 que se articulaba en la implementación del algoritmo de "mallas de probabilidad", se han utilizado dos esquemas JDE 3.2.2, laserloc-mallas y gui_laser_loc. El primero se encarga de todo el proceso de la localización, basandose en dos modelos: modelo de observación láser 4.3, y modelo de observación odométrica 4.4 el primero se encarga de incorporar las observación láser y fusionar el historial de estas, mientras que el segundo incorpora los movimientos del robot. Las probabilidades se almacenan en el cubo que junto al mapa 2D constituyen las principales estructuras en las que se apoya este algoritmo Figura 4.3.
A la luz de los experimentos realizados en la sección 6.1 podemos decir que este objetivo se ha satisfecho con éxito. El algoritmo dio buenos resultados tanto sobre el simulador, como sobre el robot real, cumpliendo todos los requisitos descritos en la sección 2.2, menos el 5, ya que con este algoritmo no se puede localizar en tiempo real.

El segundo subobjetivo 2 se ha cumplido implementando el algoritmo del "filtro de partículas", esta técnica se basa en mantener una población de partículas sobre todo el mundo, y hacer que converja hacia la posición correcta del robot. La implementación se ha hecho con dos esquemas JDE, laserloc-particulas y gui_laser_loc, el primero es el encargado de la localización, para ello usa tres modelos: observación láser, movimiento y remuestreo descrito detalladamente en el capítulo 5. El efecto del tercer paso, es el que hace converger a la nube, su implementación se ha hecho con el algoritmo de "la ruleta" 5.5.
Viendo los resultados de los experimentos realizados en el capítulo 6.2 podemos decir que los objetivos se han logrado con éxito. ya que este algoritmo respondió a todos los requisitos planteados en la sección 2.2 incluso el de llevar la tarea al tiempo real, requisito que el de "mallas de probabilidad" no pudo satisfacer.

Repasando los experimentos del capítulo 6, se puede ver fácilmente que el primer algoritmo (mallas de probabilidad) es muy costoso computacionalmente, y que su eficiencia depende de varios parámetros como la resolución de las celdillas 6.1.1 y otros más. En la sección 6.1.1 se ha visto que este algoritmo se limita a la localización en pequeños entornos(habitación) y no es "escalable" a grandes superficies. Esto hace que sea imposible su funcionamiento en áreas grandes, ya que producirá una sobrecarga en la CPU. Quiere decir que va ser difícil que se ejecute en paralelo con un algoritmo de navegación.

Por el otro lado el segundo algoritmo (filtro de partículas) viene para resolver los problemas del primero, llevando la tarea de la localización al tiempo real, con unos costes muy reducidos en comparación con el de mallas de probabilidad. Dicho esto el "filtro de partículas" es el mejor candidato para acompañar a un algoritmo de navegación, ya que la carga extra que produce en la CPU apenas se nota.

En general los dos algoritmos se comportan bastante bien, aún con problemas como la "convergencia prematura" del filtro de partículas, o "la no escalabilidad" del de mallas de probabilidad. Cabe destacar que los resultados obtenidos sobre el robot real, han sido muy parecidos a los obtenidos sobre el simulador, esto se debe a que el sensor láser es muy preciso y que el simulador lo simula bastante bien.

Repasando el indice de la memoria, la segunda parte esta concentrada en el capítulo 4, donde se presentan los detalles teóricos y de implementación del algoritmo de "mallas de probabilidad". También se habla de los distintos modelos que constituyen el núcleo de este algoritmo, y su interacción con la plataforma del desarrollo.

La tercera parte se concentra en el capítulo 5, en este capítulo se habla en detalle del segundo algoritmo "filtro de partículas", esta técnica se se basa en tres modelos, observación odométrica, observación láser y remuestreo.

Finalmente en la cuarta parte concentrada en el capítulo 6, se presentan los experimentos realizados con los dos algoritmos destacando las ventajas y desventajas de cada uno, y señalando cada vez los parámetros más importantes de cada uno y como se han ajustado para optimizar el rendimiento de los dos algoritmos.

7.2. Trabajos futuros

Una vez resumido los puntos fundamentales de este proyecto, y presentadas las conclusiones sobre cada uno de los algoritmos. En esta sección vamos a presentar algunas lineas, por las que se podría extender este proyecto en el futuro.
Hasta el momento todos los proyectos de navegación llevados acabo dentro del grupo de robótica como el de navegación global [Isado, 2005] [Lobato, 2003] se realizaban sobre simuladores, porque requieren como prerequisito que el robot sepa en cada momento su posición dentro del entorno. Tras las realización de este proyecto, estos algoritmos pueden probarse sobre el robot real.
Otra linea es fusionar este algoritmo con otro por ejemplo "localización visual" [Fernández, 2005], para obtener un algoritmo combinado de los dos y más eficaz.
Este proyecto resuelve el problema de la localización global, sin tener en cuenta los obstáculos dinámicos, una mejora es afinar los algoritmos para que se comportan de forma robusta ante la presencia de estos.

Bibliografía

[ActivMedia, 2002] ActivMedia. Aria reference manual. *Technical Report version 1.1.10, ActiveMedia Robotics*, 2002.

[Benítez, 2004] Raúl Benítez. Sistema de localización basado en la detección de segmentos para robot móviles. *Proyecto fin de carrera, URJC*, 2004.

[Brian P. Gerkey, 2003] Andrew Howard Brian P. Gerkey, Richard T. Vaughan. The player/stage project: Tools for multi-robot and distributed sensor systems. *Proceedings of the international conference on Advanced Robotics.*, pages 317–323, 2003.

[Calvo, 2004] Roberto Calvo. Comportamiento sigue persona con visión direccional. *Proyecto fin de carrera, URJC*, 2004.

[Crespo, 2003] María Angeles Crespo. Localización probabilística en un robot con visión local. *Proyecto fin de carrera, Universidad Politécnica de Madrid*, 2003.

[Fernández, 2005] José Alberto López Fernández. Localización visual en el robot pioner. *Proyecto fin de carrera, URJC*, 2005.

[Fox *et al.*, 1999] Dieter Fox, Wolfram Burgard, Frank Dellaert, y Sebastian Thrun. Monte Carlo localization: efficient position estimation for mobile robots. In *Proceedings of the 16th AAAI National Conference on Artificial Intelligence*, pages 343–349, Orlando (Florida, USA), July 1999.

[Isado, 2005] Jose Raul Isado. Navegación global por el método del gradiente. *Proyecto Fin de Carrera, URJC*, 2005.

[Isard y Blake, 1998] M. Isard y A. Blake. Condensation-conditional density propagation for visual tracking. *Int. J. Computer Vision, in press*, 1998.

[Lobato, 2003] David Lobato. Evitación de obstáculos basada en ventana dinámica. *Proyecto fin de carrera, URJC*, 2003.

[López, 2005] Alejandro López. Navegación global utilizando grafo de visibilidad. *Proyecto fin de carrera, URJC*, 2005.

[Mackay, 1999] D.J.C. Mackay. Introduction to Monte Carlo methods. In M. Jordan, editor, *Learning in Graphical Models*, pages 175–204. MIT Press, Cambridge (MA, USA), 1999.

[Plaza, 2003] José María Cañas Plaza. Jerarquía dinámica de esquemas para la generación de comportamiento autónomo. *Tesis doctoral, Universidad Politécnica de Madrid*, 2003.

[Plaza, 2004a] José María Cañas Plaza. Manual de programación de robots con jde. *URJC*, pages 1–36, 2004.

[Plaza, 2004b] José María Cañas Plaza. Manual de programación de robots con jde. *URJC*, pages 1–36, 2004.

[Thrun, 1997] S. Thrun. Bayesian landmark learning for mobile robot localization. In *To appear in Machine Learning*, April 1997.

[Thrun, 2000] S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 2000.