INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2006-2007

Proyecto Fin de Carrera

Localización y construcción de mapas en un robot de interiores.

Autor: Ángel Cortés Maya

Tutor: José María Cañas Plaza

Para mi familia y amigos, que siempre están conmigo en lo bueno y en lo malo.

Agradecimientos.

Son muchas las personas a las que quiero dedicar este proyecto y seguro que se me olvida alguien, así que estas líneas van para ellos.

En primer lugar quiero dedicar este proyecto a toda mi familia y en especial a mis padres que siempre se preocupan por mí y me dan todo lo mejor.

Este proyecto también va dedicado a todos mis amigos y en especial a Ismael, Roberto, Carlos, Tati, Rebeca, Pedro, Marcos, Paco, Laura y Leyre. Gracias por vuestra amistad.

Sin duda alguna, parte de este proyecto es para mis colegas de la universidad por todos lo buenos ratos que hemos pasado y todas esas risas nerviosas en los fatídicos exámenes. Ya sabéis quien sois: Jorge, Miguel, Javi, David, Carol, Alexis, Rubén, Gustavo, Sofía, Teresa, Juanlu, Chabir, Tinte, Espinaco y Nano. Sois muchos y si se me olvida alguien daros por aludidos en estas líneas. En especial agradecerle a Miguel esas mañanas en la etapa de aprendizaje. Que días tan largos fueron, pero hubo buenas risas.

También quiero agradecer a José María Cañas por el esfuerzo, conocimientos y apoyo que me ha suministrado para la realización de este proyecto.

El proyecto también es vuestro y espero que os guste.

Resumen.

Una de las característica más deseables de un robot es la *navegación autónoma*, puesto que de esta forma podrá realizar acciones por sí solo. Para poder conseguirlo, el robot debe disponer de sensores, del mapa del lugar por el que se desplaza y a la vez estar localizado en todo momento.

En este proyecto se va aborda la construcción autónoma de mapas y la localización de un robot Pioneer sobre el simulador Player-Stage.

La *contrucción de mapas* se realiza explorando el mundo de forma reactiva, almacenando en una memoria de segmentos la información de obstáculos que nos da el sensor láser. Mientras el robot construye el mapa, sabe en todo momento en qué posición está. Se han realizado pruebas de construcción de mapas de diferentes características para comprobar su buen funcionamiento.

La *localización* del robot se ha realizado conociendo el mapa desde el principio. Conseguir un algoritmo localizador robusto ha sido el principal objetivo de este proyecto. Se han realizado pruebas de localización utilizando dos algoritmos. Se ha usado el *filtro de partículas* realizado en el proyecto de Redouane Kachach [Kachach, 2005], mejorándolo para conseguir más rapidez. También se ha diseñado y programado un *algoritmo evolutivo multimodal* para obtener una localización más robusta. Se han probado los dos algoritmos en mundo de diferentes características para compararlos y comprobar que su funcionamiento es el correcto.

También se han realizado experimentos de construcción de mapas y localización simultáneas, obteniéndose resultados preliminares que nos acercan más a resolver el problema en el robot real.

Índice general

1. Introducción **1**

1.1. La robótica 1

1.2. Construcción de mapas 4

1.3. Localización 7

1.4. SLAM 8

2. Objetivos **10**

2.1. Descripción del problema 10

2.2. Requisitos 11

2.3. Metodología y plan de trabajo 11

3. Entorno y plataforma de desarrollo **13**

3.1. Simulador Player-Stage 13

3.2. Plataforma jde.c 14

3.3. Xforms 16

3.4. Librería Gridslib 16

4. Descripción informática **17**

4.1. Estructura general 17

4.2. Navegación 18

4.2.1. Memoria de puntos 18

4.2.2. Cálculo de fuerzas virtuales 19

4.2.3. Cálculo de velocidades 20

4.3. Construcción de mapas 21

4.3.1. Segmentos láser instantáneos 22

4.3.2. Memoria de segmentos 23

4.4. Localización con filtro de partículas 27

4.4.1. Lectura del mapa 27

4.4.2. Modelo de movimiento 28

4.4.3. Modelo de observación 29
4.4.4. Remuestreo 33
4.4.5. Generación de ruido odométrico sobre el simulador. 34
4.5. Localización con algoritmo multimodal 35
4.5.1. Generación de la siguiente población 37
4.5.2. Interfaz gráfica. 39

5. Experimentos **41**
5.1. Análisis de la navegación 41
5.1.1. Necesidad de la memoria de puntos 42
5.1.2. Balance de fuerzas 42
5.2. Análisis de la construcción de mapas 43
5.2.1. Comportamiento sin ruido 43
5.2.2. Comportamiento con ruido 45
5.3. Análisis de la localización 46
5.3.1. Obtención del láser teórico 46
5.3.2. Obtención de la probabilidad/salud 50
5.3.3. Resultados del filtro de partículas 54
5.3.4. Resultados del algoritmo evolutivo multimodal 57
5.3.5. Comparativa filtro de partículas contra multimodal 60
5.4. Construcción de mapas y localización simultáneas 61

6. Conclusiones y trabajos futuros **65**
6.1. Conclusiones 65
6.2. Trabajos futuros 68

Anexo **69**

Bibliografía **71**

Índice de figuras

1.1. Cadena de montaje de coches de *Toyota Motor Corporation*. 2
1.2. Nanorobot interactuando con células humanas y Robot cirujano Da Vinci de *Intuitive Surgical*. 3
1.3. Volkswagen Touareg "Stanley", ganador de el Darpa Grand Challenger 2005. 4
1.4. Aspiradora roomba (a) y pared virtual para limitar espacios grandes (b). 4
1.5. Modelo topológico (a) y de elementos geométricos (b) representando el mismo mundo. 6
1.6. Sensor GPS (a) y encoders (b). 8

2.1. Modelo en espiral 12

3.1. Mundo y láser simulado (a) y robot Pioneer (b). 14
3.2. Componentes odométricas del robot. 15

4.1. Composición esquemática. 17
4.2. Diagrama de flujo de la construcción de la memoria de puntos. 18
4.3. Reglas de control ad-hoc para el control de la velocidad. 20
4.4. GUI del esquema *navigation* y sus fuerzas virtuales. 21
4.5. Segmentación de lectura láser instantánea. 22
4.6. Diagrama de flujo de la construcción de la memoria de segmentos. . . . 23
4.7. Fragmentación de segmentos. 24
4.8. Segmentos paralelos (a) y segmentos no paralelos (b y c). 25
4.9. Segmento en distancia. 25
4.10. Segmentos en distancia. 26
4.11. GUI del esquema *mapping*. 26
4.12. Diagrama de flujo del localizador con filtro de partículas. 27
4.13. Incorporación del movimiento del robot en la partícula. 28
4.14. Sólo se comprueban las celdillas ocupadas y en distancia. 30
4.15. Forma de recorrer la lista según la orientación del rayo láser. 31

4.16. Búsqueda binaria en la lista ordenada de celdillas ocupadas. Primero buscamos por y y después por x 32
4.17. Láser teórico generado. 32
4.18. Ruleta (a) y generación de 10 nuevas partículas (b). 34
4.19. Diagrama de flujo del algoritmo evolutivo multimodal. 36
4.20. Estimación de posición del robot y actualizaciones durante una iteración. 37
4.21. GUI del esquema *localization*. 40

5.1. Navegación sobre memoria de puntos. 41
5.2. Navegación sobre láser instantáneo. 42
5.3. Diferentes módulos de la fuerza atractiva: Pequeño-Bueno-Grande. . . . 43
5.4. Construcción del departamental II. 44
5.5. Construcción de mundos simulados. 44
5.6. Construcción de mapa en mundo curvilíneo. 45
5.7. Construcción de departamental II sin ruido (a) y con ruido (b). 46
5.8. Haz láser. 47
5.9. Utilización de las proyecciones para encontrar obstáculo. 48
5.10. Rejilla del rayo láser. 48
5.11. Comparativa de la evolución de optimizaciones. 50
5.12. Probabilidad para *diferencia con potencia*(a) y *simple con potencia* (b) modificando x e y. 52
5.13. Probabilidad de las partículas para todos los ángulos, fijando x e y. . . 53
5.14. Probabilidad para *diferencia con potencia* (a) y *correlación con potencia* (b) modificando x e y. 54
5.15. Localización eficiente con el filtro de partículas en un mundo asimétrico. 55
5.16. Localización fallida con el filtro de partículas en un mundo asimétrico. . 55
5.17. Localización eficiente con el filtro de partículas en el departamental II. 56
5.18. Localización fallida con el filtro de partículas en el departamental II. . . 57
5.19. Localización en un mundo asimétrico con algoritmo evolutivo multimodal. 58
5.20. Localización eficiente con algoritmo multimodal en el departamental2. . 58
5.21. Elección de la mejor raza. 59
5.22. Elección de raza incorrecta y corrección posterior. 60
5.23. Comparativa de los algoritmos localizadores. 61
5.24. Mapa construido real (a) - ruidoso (b) - estimado (c). 62
5.25. Mapa construido incompleto en comparación con el real. 64
5.26. Localización en posiciones simétricas ya construidas. 64

6.1. Punto de intersección de dos segmentos. 69
6.2. Mínima distancia punto-recta. 70

Capítulo 1
Introducción

En este primer capítulo vamos a hablar del mundo de la robótica y a introducir los principales aspectos que nos van a interesar en este proyecto.

1.1. La robótica

La robótica desde sus orígenes ha estado ligada a la construcción de "artefactos", los cuales debían ser capaces de descargar de trabajo a los seres humanos. La palabra robot aparece por primera vez en 1920 en la obra de teatro Los Robots Universales de Rossum, escrita por el dramaturgo checo de Karel Capek. ***Robota*** es una palabra checa que significa trabajador, sirviente.

En definitiva, los robot son dispositivos electrónicos y generalmente mecánicos, cuya función es realizar tareas automáticamente, mediante un programa predefinido, supervisión humana directa o siguiendo un conjunto de reglas generales. Están compuestos de sensores, actuadores y procesadores. Los sensores reciben la información que el robot usa para realizar sus funciones. Esta información es suministrada al procesador, el cual la interpreta, la organiza y posteriormente ordena a los actuadores la acción a ejecutar.

La Robótica como tecnología surge alrededor de la década de los sesenta del siglo XX, gracias a avances tecnológicos como el computador eléctrico o el uso de sensores. Desde entonces hasta el día de hoy, el interés que ha despertado y el estudio del que ha sido objeto, han sido infinitamente superiores a cualquier previsión y a la vez su desarrollo ha sido espectacular.

Los robots son usados hoy en día para llevar a cabo tareas sucias, peligrosas, difíciles o repetitivas para los humanos. La Industria automotriz ha tomado gran ventaja de esta nueva tecnología donde los robots son programados para reemplazar el trabajo de

los humanos en muchas tareas repetitivas. Una de las grandes empresas de éste sector es la *Toyota Motor Corporation*. Es una empresa multinacional japonesa, y la mayor fabricante de automóviles (figura 1.1). Toyota vendió 2,34 millones de vehículos en los primeros tres meses de 2007.

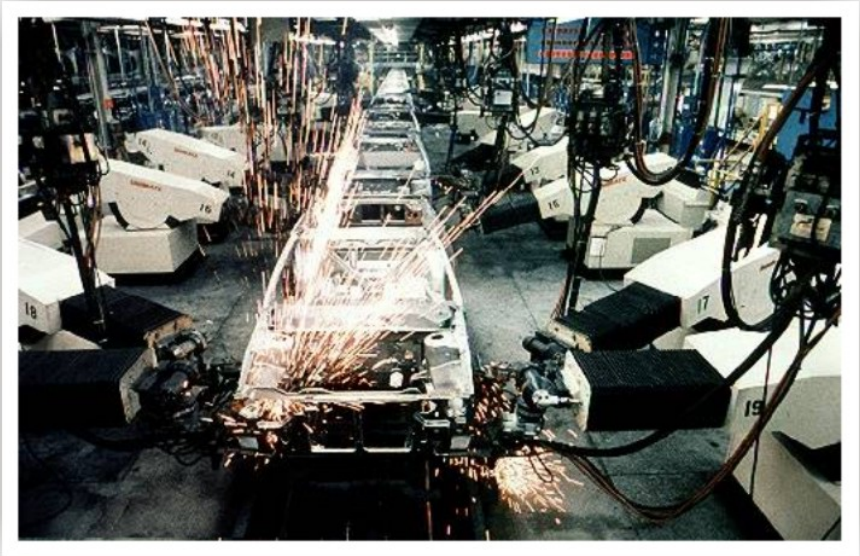

Figura 1.1: Cadena de montaje de coches de *Toyota Motor Corporation*.

En la actualidad, se ha logrado un gran avance en los robots dedicados a la medicina, con dos compañías en particular, *Computer Motion e Intuitive Surgical*, que han recibido la aprobación regulatoria en América del Norte, Europa y Asia para que sus robots sean utilizados en procedimientos de cirugía invasiva mínima. Integrando la tecnología de la robótica con la habilidad y destreza del cirujano, el Sistema de Cirugía Da Vinci de *Intuitive Surgical* (ver figura 1.2) permite a los cirujanos intervenir y operar de una manera jamás experimentada con anterioridad. Muchos procedimientos que hoy en día se ejecutan con la técnica laparoscópica convencional pueden realizarse de manera más rápida y más fácil con el Robot Da Vinci.

La automatización de laboratorios y almacenes también es un área en crecimiento. Aquí, los robots son utilizados para transportar muestras biológicas o químicas entre instrumentos tales como incubadoras, manejadores de líquidos y lectores.

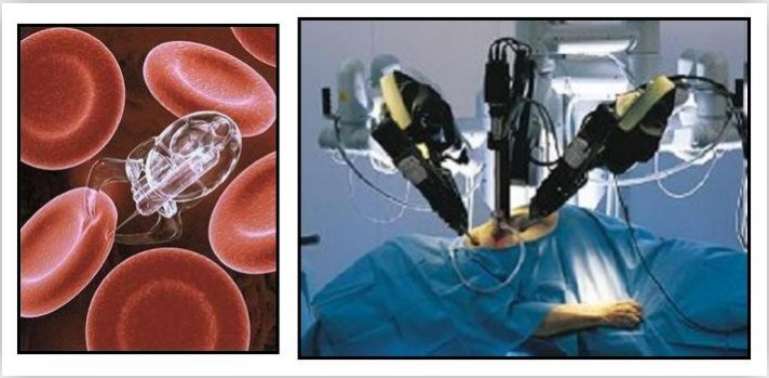

Figura 1.2: Nanorobot interactuando con células humanas y Robot cirujano Da Vinci de *Intuitive Surgical*.

Otro de los aspectos más importantes de la robótica es la navegación autónoma de los robots. La agencia Darpa (Agencia dependiente del Ministerio de Defensa de los EEUU) realiza proyectos tecnológicos que desarrollan este aspecto de la robótica. Por ejemplo, el Darpa Grand Challenge es uno de esos proyectos. Consiste en construir un vehículo completamente autónomo que pueda recorrer grandes distancias (280 km.) a una velocidad relativamente alta (30 km./h.) por sí solo. 5 coches terminaron la carrera en el año 2005. Los ganadores fueron los chicos de la Universidad de Stanford con su Volkswagen Touareg "Stanley" (ver figura 1.3). Su coche completó los 280 kilómetros en 6 horas y 53 minutos, con una velocidad media de 40 km./h.. Se llevaron 2 millones de dólares como premio.

El objetivo en último término de este proyecto es hacer un uso militar de esta tecnología robotizada. Sin embargo, en los pocos meses que esta iniciativa lleva en el candelero, ha quedado claro que hay muchos otros beneficios. El principal es el desarrollo de coches inteligentes capaces de evitar accidentes en carretera.

Además de obtener recursos militares y aplicaciones para aumentar la seguridad, la navegación en la robótica también es usada para facilitar la vida a las personas. Un ejemplo de ello es la aspiradora Roomba, un éxito de ventas (ver figura 1.4). Es una aspiradora automática que barre y aspira suelos sin ningún tipo de supervisión. Para ello debe utilizar la navegación autónoma para poder evitar los diferentes obstáculos que encuentre a la vez de recorrer todo el terreno disponible.

Figura 1.3: Volkswagen Touareg "Stanley", ganador de el Darpa Grand Challenger 2005.

Figura 1.4: Aspiradora roomba (a) y pared virtual para limitar espacios grandes (b).

En conclusión, la robótica móvil es un área de intensa investigación cuyo último objetivo es conseguir la navegación autónoma de los robots para que puedan realizar las tareas por sí solos. La navegación autónoma se puede conseguir por medio de la información de los sensores (navegación reactiva), a partir de mapas (navegación deliberativa) o combinando ambas fuentes de información.

1.2. Construcción de mapas

La obtención del mapa del entorno permite al robot tener una representación del mundo por el que se mueve o actúa. Inicialmente, el ingeniero proporcionaba directamente al robot el mapa de la zona por la que se iba a mover. Posteriormente, es el robot el que se lo construye automáticamente. Para que el robot sea capaz de

construir de forma eficiente el mapa del entorno por sí mismo debe poder navegar reactivamente y a la vez saber en todo momento dónde se encuentra. Además, el robot debe estar continuamente alerta ya que los mundos en los que se mueve el robot suelen ser dinámicos, es decir, están abiertos a cambios en la posición de los obstáculos.

Un mapa o modelo del entorno es una representación de la que se abstraen las cosas superfluas del mundo que rodea al robot. Sólo representa las características que aportan información de interés al robot, es decir, aquella información útil para la navegación. Las características que se suelen descartar del entorno son aquellas que son demasiado variables o no pueden ser detectadas con precisión y fiabilidad por los sensores del robot. La principal utilidad para el robot de poseer el mapa del mundo por el que se mueve consiste en la posibilidad de realizar movimientos inteligentes como una navegación deliberativa. Gracias a esto, el robot puede planificar trayectorias que le permiten alcanzar destinos remotos entre otras utilidades.

Hay dos paradigmas fundamentales de modelado de entornos: modelos métricos y modelos topológicos. Los modelos métricos a su vez se dividen en modelos basados en rejillas y en modelos de elementos métricos. Las principales características de estos modelos son los siguientes:

- *Mapas topológicos*: Los modelos topológicos utilizan grafos para representar las características relevantes del mundo (figura 1.5). Los nodos son utilizados para representar lugares del entorno y los arcos entre los nodos representan caminos entre estos lugares. Los lugares relevantes son llamados *landmarks*, es decir, lugares con datos sensoriales claramente distinguibles a los demás lugares (nodos), o por lo menos a sus vecinos. Cabe destacar que la principal ventaja que suponen los mapas topológicos reside en que se pueden planificar trayectorias gracias a la conectividad existente entre los nodos. Como desventaja estos mapas no pueden inferir distancias.

- *Mapas métricos*: La principal característica que presentan este tipo de modelos es la posibilidad de inferir distancias, a diferencia de los modelos topológicos. No obstante, necesitan tener un sistema de coordenadas para poder conseguir esto. Hay dos tipos diferentes de modelos métricos según sus características:

  1. *Mapas de elementos geométricos*:
     Este tipo de modelos utilizan las características geométricas (esquinas, segmentos, polígonos) para representar el entorno (figura 1.5). Hay diferentes

tipos de modelos geométricos. Un primer enfoque sería utilizar un conjunto de características métricas (segmentos de rectas, esquinas, puntos) y las relaciones entre ellas (distancia, posición, etc.) para definir el entorno. Otro conjunto de modelos geométricos es definir el entorno mediante un mapa CAD del mismo. Un mapa CAD refleja los elementos del entorno que se desea modular, obteniendo sus dimensiones y posiciones.

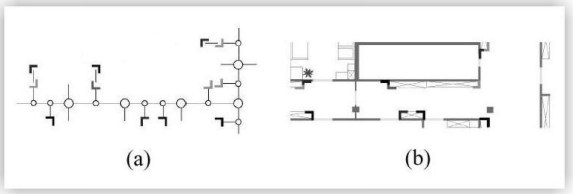

Figura 1.5: Modelo topológico (a) y de elementos geométricos (b) representando el mismo mundo.

2. *Rejillas de ocupación*: Inicialmente fueron propuestas por Moravec y Elfes (Moravec y Elfes 1985). Este tipo de modelado lo que hace es discretizar el entorno en celdillas de igual dimensión. Cada celdilla tiene la probabilidad de que esa zona del entorno esté ocupada o no. Normalmente es el propio robot el que construye la rejilla de ocupación de forma autónoma mediante algún algoritmo de exploración, aunque es posible que sea el usuario el que las defina.

Según sean los entornos, los mapas son dinámicos (abierto a fluctuaciones) o estáticos (no cambia la posición de sus elementos). También cabe distinguir entre mapas globales y locales y entre mapas en dos dimensiones o en tres dimensiones [Cañas Plaza y García, 2002].

Al realizar la construcción automática del mapa, la información de ocupación (existencia de obstáculos) que dan los sensores se vuelca en los mapas para que estos estén continuamente actualizados. Para volcar ésta información hay que tener en cuenta dos aspectos: el modelo sensorial y la regla de actualización. El modelo sensorial se refiere a la forma de interpretar la información proporcionada por los sensores utilizados. Por otro lado, la regla de actualización se refiere a la forma de integrar la observación sensorial actual con las anteriores.

El Volkswagen Touareg “Stanley” del que hemos hablado anteriormente (figura 1.3) usó un mapa topográfico más un mapa local de la zona para lograr ganar la carrera.

Para poder realizar de forma eficiente la construcción de los mapas, el robot debe estar bien localizado sobre éste para saber en todo momento en que posición está. Si no supiera en que posición está, no se sabe la zona en la que incorporar las nuevas informaciones sensoriales. Por este motivo es fundamental la localización del robot para la construcción de mapas.

1.3. Localización

Para un robot móvil una de las habilidades más importantes a desarrollar es la autolocalización, pues permite al robot conocer en cada momento en que posición y orientación del espacio se encuentra. Al estar localizado, el robot puede construir el mapa y usarlo. Si no está localizado no puede usar los mapas. Aunque planifique trayectorias, no sabe dónde está.

Existen dos tipos diferentes de localización: la localización local y la localización global. En la localización local se supone conocida la posición inicial del robot y se realiza un seguimiento *tracking* de la misma para estimar la siguiente posición. El enfoque más usado para solucionar el problema de la localización local es la utilización del filtro de Kalman. La aplicación del filtro de Kalman, para conseguir la localización de los robots móviles estima la posición (x, y, θ) del robot en el entorno mediante una distribución normal. La covarianza de esta distribución representa la incertidumbre local en la posición estimada, es decir, lo que puede diferir de la posición real. Siempre que se mueva el robot, la posición estimada se desplaza según la distancia medida por la odometría del robot. Las observaciones realizadas por los sensores, por lo tanto, se utilizan para actualizar la distribución de probabilidad de la localización y buscar la posición del mundo más parecida a las lecturas obtenidas. El principal problema que supone esta técnica reside en sólo poder representar una única posición posible del robot (es una técnica unimodal).

Por otra parte, en la localización global no se conoce la posición inicial del robot y se intenta estimar la posición por medio de las lecturas de los sensores y su comparación con el modelo del entorno. Para resolver este problema se utilizan enfoques bayesianos que consisten en la estimación de la posición del robot móvil a partir de las observaciones realizadas por el robot y los movimientos de éste.

Uno de los factores que influyen a la hora de conseguir una buena localización es el sensor utilizado para localizarse. La forma más fácil es utilizando sensores que nos dan la posición del robot de forma directa. Estos sensores son llamados "sensores de posición" y nuestro principal esfuerzo consistirá en corregir los posibles errores que den. Diferentes sensores de posición son los encoders y los GPS (figura 1.6).

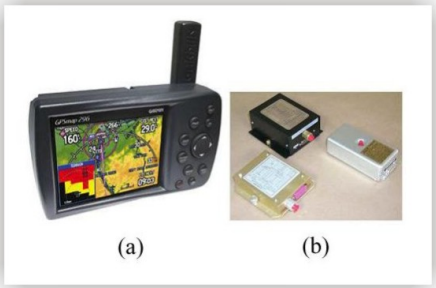

Figura 1.6: Sensor GPS (a) y encoders (b).

La forma más difícil y elaborada del problema de la localización es cuando los sensores que se utilizan no son específicos de posición. La información que dan estos sensores no es la posición del robot y esta habrá que obtenerla. Para conseguirla habrá que realizar transformaciones en los datos sensoriales o utilizar estos datos con otros procedimientos como la comparación con lecturas simuladas. Algunos de estos sensores son el láser, los sonares, la visión por medio de cámaras, etc. (figura 3.1b). En este proyecto se utilizará el sensor láser ya que es un sensor bastante preciso y fiable.

Los chicos de la Universidad de Stanford utilizaron un sensor GPS en su Volkswagen Touareg "Stanley" para localizarse (figura 1.3). Al localizarse pudieron usar los mapas comentados anteriormente para alcanzar la meta con éxito.

Los diferentes algoritmos localizadores comparan las lecturas obtenidas por los sensores con el modelo del entorno para actualizar de esta forma la posición del robot acorde con el resultado de esta comparación. La forma de realizar esta comparación depende por completo del tipo de modelo de entorno. Es más, el problema de la localización necesita disponer del mapa del entorno para ser resuelto de forma satisfactoria.

1.4. SLAM

SLAM son las siglas en inglés de *simultaneous localization and mapping* (localización y construcción de mapas de forma simultánea). Este es uno de los principales problemas actuales para conseguir la autonomía de los robots. En realidad, son dos problemas interconectados: para construir los mapas necesitamos estar localizados y para estar localizados necesitamos tener un mapa.

En el grupo de Robótica de la Universidad Rey Juan Carlos se han realizado varios trabajos sobre estos temas, que conforman el contexto cercano del presente proyecto. En cuanto a navegación, por ejemplo, se han realizado trabajos sobre navegación global utilizando diferentes técnicas. En ellos el robot necesita estar localizado en todo momento [Isado, 2005][López, 2005b]. En estos dos proyectos se asumía conocida la posición del robot en todo momento, es decir, la localización resuelta.

Por otro lado se han realizado trabajos sobre localización utilizando el sensor láser [Kachach, 2005] y visión local [López, 2005a]. En ambos se han utilizado las técnicas de *filtro de partículas* y *mallas de probabilidad*. En estos proyectos se trabajaba la localización del robot asumiendo que se conoce el mapa de antemano, el cual se almacenaba en forma de rejilla.

Hasta ahora se han realizado trabajos de localización asumiendo el mapa, y trabajos de navegación asumiendo la localización. Éste será el primer proyecto dentro del grupo que además de tratar la navegación y la localización de forma separada, se trabajarán de forma conjunta. Este proyecto trata de avanzar en las técnicas de navegación, de construcción de mapas y de localización. En concreto trata de conseguir mayor robustez en estas técnicas, conseguir la localización en entornos grandes y obtener mayor rapidez que los trabajos realizados hasta ahora. Con ello perseguimos estar más cerca de resolver este problema en el robot real.

Después de ésta introducción, la memoria consta en primer lugar de un capítulo de objetivos para explicar las metas que se intentan conseguir y los requisitos que existen. Posteriormente se exponen las plataformas y herramientas utilizadas. En el capítulo 4 se detalla la implementación software desarrollada y sus componentes. Después en el capítulo 5 se explican los experimentos realizados para validar la solución desarrollada. Finalmente en el capítulo de conclusiones se analizan los resultados obtenidos.

Capítulo 2
Objetivos

Después de haber introducido la historia y evolución de la robótica y de haber comentado la importancia en la robótica móvil de aspectos como la localización y la construcción de mapas, vamos a plantear en este capítulo el problema concreto que aborda este proyecto, los requisitos necesarios, así como la metodología y el plan de trabajo seguidos.

2.1. Descripción del problema

En este proyecto se pretende conseguir la localización de un robot de forma robusta sobre simulador, así como obtener una construcción de mapas eficiente, también sobre el simulador. Este objetivo lo hemos articulado en tres subobjetivos:

1. Conseguir una navegación reactiva segura del robot para que sea capaz de evitar tanto obstáculos estáticos como dinámicos. La función de ésta navegación es explorar el mundo para construir el mapa y probar la localización.

2. Realizar un algoritmo de construcción de mapas para tener la representación interna del entorno que rodea al robot. De esta forma se obtendrá un alcance mayor a la información sensorial instantánea. Gracias a este mapa se podrá realizar una navegación deliberativa planificando posibles caminos.

   La construcción del mapa se tendrá que realizar de tres formas distintas:

   a) La primera forma consistirá en utilizar una localización perfecta, es decir construir el mapa utilizando la odometría ideal.

   b) En la segunda forma habrá que usar odometría ruidosa para construir el mapa.

   c) La tercera forma tendrá que emplear autolocalización y utilizar la posición estimada por nuestro algoritmo localizador para construir el mapa.

3. Lograr la localización del robot en ambientes dinámicos mediante el sensor láser. A partir de la información sensorial se utilizarán los mapas para localizar al robot con dos algoritmos distintos: mediante el filtro de partículas y mediante un algoritmo evolutivo multimodal.

   La localización se tendrá que conseguir en escenarios grandes utilizando mapas perfectos.

   También se realizarán pruebas de SLAM para intentar localizarse utilizando los mapas construidos por el algoritmo constructor de mapas.

2.2. Requisitos

Los requisitos necesarios para la realización de este proyecto son los siguientes:

1. Utilización de la plataforma jde.c como entorno para programar nuestras aplicaciones [Plaza, 2003]. El software de nuestro proyecto hereda la forma modular de jde.c. La utilización de jde.c impone la utilización del lenguaje c al ser el entorno en el que está implementada.

2. Necesidad de procesamiento de datos en tiempo real. Conseguir una localización en orden de segundos y una navegación y construcción de mapas vivaz.

3. Precisión centimétrica en la construcción de mapas y en la localización. Para aumentar ésta robustez se deberá generar ruido artificial para digerir bien el ruido realista producido en las condiciones reales.

4. Utilizar como mapa simulado el departamental II de la ESCET-URJC para acercar el problema a la aplicación real.

2.3. Metodología y plan de trabajo

El plan de trabajo que se ha seguido para la realización de este proyecto sigue el modelo de desarrollo en espiral basado en prototipos. La ventaja de utilizar este modelo de desarrollo es la flexibilidad que aporta ante un posible cambio de requisitos. Este modelo llega a la obtención del objetivo final mediante la realización de subtareas que se reparten en un número determinado de ciclos. Cada uno de estos ciclos a su vez se subdivide en cuatro etapas: análisis de requisitos, diseño e implementación, pruebas (mediante el simulador Player-Stage) y planificación del próximo ciclo.

Figura 2.1: Modelo en espiral

Para la elaboración de este proyecto se han realizado los siguientes ciclos:

1. Etapa de aprendizaje de la *infraestructura y software* utilizado. Tareas tales como utilizar el simulador Player-Stage, dar órdenes al robot Pioneer y saber acceder a la información suministrada por éste

2. Análisis, diseño y pruebas del algoritmo de navegación local para evitar obstáculos y explorar el entorno para la construcción del mapa y probar la localización.

3. Análisis, diseño y pruebas del esquema *constructor de mapas* estando localizado.

4. Análisis, diseño y pruebas del esquema *localizador* teniendo el mapa del entorno.

5. Experimentos de la integración de la construcción de mapas y la localización.

6. Escritura de la memoria.

La metodología ha incluido reuniones semanales con el tutor comentando los resultados y decidiendo los pasos a seguir en cada punto.

Capítulo 3
Entorno y plataforma de desarrollo

En este tercer capítulo vamos a hablar de la plataforma, bibliotecas y aplicaciones que se han utilizado para poder llevar a cabo este proyecto.

3.1. Simulador Player-Stage

El simulador utilizado para realizar el proyecto es Player-Stage[1], el cual se encuadra dentro del proyecto de software libre Player/Stage/Gazebo.

Player-Stage es capaz de simular diferentes tipos de robots. Entre ellos nosotros nos hemos apoyado en el robot Pioneer. El Pioneer 3DXE (figura 3.1b) es un robot construido por ActivMedia [2] de tamaño mediano. En cuanto a los sensores de los que dispone, tiene una corona de 16 ultrasonidos, un cinturón de sensores táctiles y un encoder en cada rueda para contar las vueltas del eje. Tiene tres ruedas: dos motrices, con sendos motores, y una rueda loca.

De los sensores simulados del Pioneer se han utilizado el sensor láser y la odometría. De los actuadores utilizados hemos utilizado los motores. El sensor láser (figura 3.2a) es un sensor muy fiable y con lo cual se pueden simular de una manera bastante real sus lecturas, como se muestra en la figura 3.1a, la cual muestra la simulación de éste sensor en el simulador Player-Stage. Por este motivo es el sensor elegido para realizar este proyecto. El sensor láser tiene 180 rayos. Según su configuración, el láser ofrece 10 medidas por segundo con una precisión de un grado, es decir el rango angular es de 180 grados. Cada rayo alcanza una distancia máxima de 8 metros. De este modo detecta los obstáculos situados a menos de 8 metros del robot.

El otro sensor utilizado en este proyecto son los sensores de odometría. Su misión principal es la de estimar la posición (x,y,θ) del robot respecto el marco absoluto de

[1] http://playerstage.sourceforge.net/
[2] http://www.mobilerobots.com/

referencia. Estos sensores determinan estos valores según lo que hayan girado las ruedas. Las coordenadas del sistema odométrico están referenciadas a un sistema absoluto externo y fijo. La posición se actualiza 10 veces por segundo.

Por último, se han utilizado los motores, los cuales controlan la velocidad lineal y angular del robot. Los valores máximos permitidos son ±1000 mm/seg para la velocidad lineal y ±180 grados/seg para la velocidad angular.

Mediante la utilización del láser, la odometría y los motores, el robot navegará en mundos simulados como el representado en la figura 3.1a. El robot se moverá a la velocidad que le ordenen los motores, registrando siempre los valores medidos por sus sensores de odometría y el láser

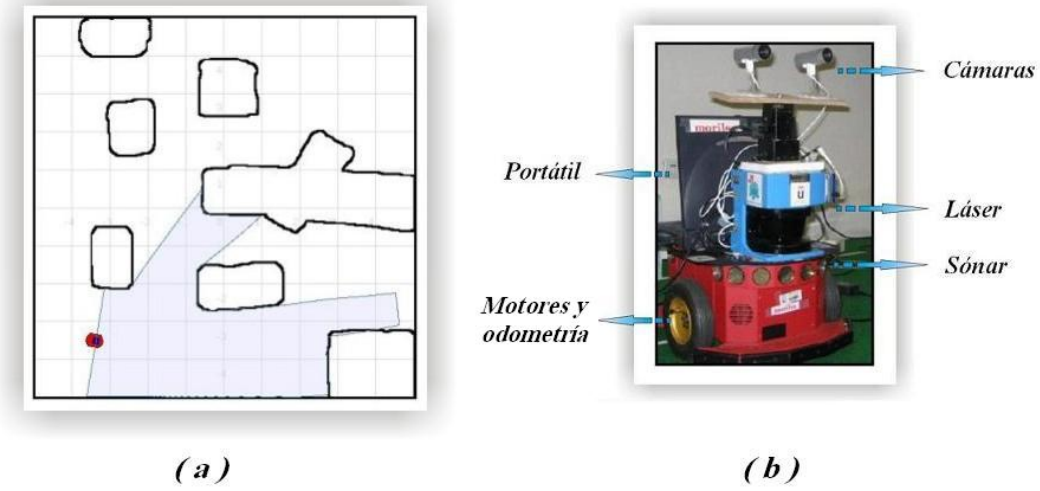

Figura 3.1: Mundo y láser simulado (a) y robot Pioneer (b).

3.2. Plataforma jde.c

Jde.c[3] [Cañas Plaza *et al.*, 2006] es una plataforma software para la programación de comportamientos autónomos en robots móviles. Desde que surge en 1997 ha ido evolucionando e incorporando nuevas funcionalidades. *Jde.c* ofrece un conjunto de variables que hace posible que el acceso a los sensores y actuadores del robot se realice de forma cómoda, fácil y eficiente. Estas variables se pueden clasificar en perceptivas y de actuación. La plataforma recoge las últimas observaciones sensoriales de las variables perceptivas (las cuales permanecen constantemente actualizadas) y

[3]http://svn.robotica-urjc.es/jde/jdec/

manda las órdenes al robot escribiendo en las variables de actuación. Las variables perceptivas existentes son *laser*, *us*, *robot*, *colorA*, *colorB*, *pan* y *tilt* (respectivamente láser, sonar, odometría, imágenes y cuello mecánico). Las variables de actuación son *v*, *w*, *latitude* y *longitude* (velocidad lineal, velocidad angular, ángulo vertical del cuello mecánico y ángulo horizontal de cuello mecánico). Estas variables deben de conectarse a los dispositivos del robot (reales o simulados). Esta plataforma soporta el simulador Player-Stage utilizando las mismas variables que si se trabajase con el robot real.

En referencia a los sensores que hemos utilizado, las medidas que suministra el sensor láser se almacenan en un *array* de 180 enteros, *int laser[180]*. En este *array* se guarda la distancia en milímetros para cada uno de los ángulos de barrido del sensor.

En cuanto a los encoders, la plataforma *jde.c* utiliza un *array* de 5 posiciones: *robot[0]* es la x del robot, *robot[1]* es la y, *robot[2]* es el ángulo, *robot[3]* es el $\cos(\theta_{robot})$ y *robot[4]* es el $\text{sen}(\theta_{robot})$.

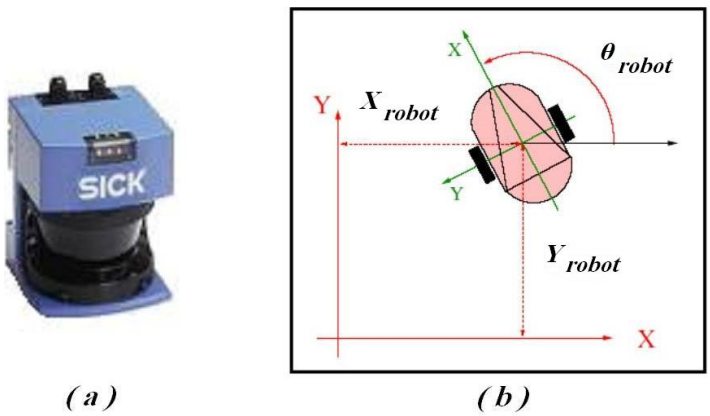

Figura 3.2: Componentes odométricas del robot.

Los motores son controlados por *jde.c* a través de las variables de actuación *float v* y *float w*, que son la velocidad lineal (en mm/seg) y la velocidad de giro (en grados/seg). Los valores positivos representan avances hacia delante y los negativos, retrocesos (mirando al frente). Los giros positivos representan giros en sentido horario y los negativos, giros en sentido antihorario.

Existe un fichero de configuración que permite usar el robot real, conectando directamente al portátil los sensores y actuadores del robot; o el simulador, utilizando los sensores y actuadores simulados.

Dentro de *jde.c* las aplicaciones se crean en forma de esquemas concurrentes. Si la aplicación es relativamente sencilla se utiliza un esquema único. Si aumenta la dificultad se suelen usar varios esquemas que se ejecutan en paralelo y si es un problema aun mayor suele haber una jerarquía de esquemas que se activan y modelan unos a otros. Este proyecto se compone de tres esquemas que se ejecutan de forma paralela.

3.3. Xforms

En la mayoría de las aplicaciones robóticas, uno de los aspectos más importantes a la hora de depurarlas y usarlas es poseer una buena interfaz gráfica, que aporte gran funcionalidad y a su vez sea sencilla de usar. Nosotros hemos utilizado las interfaces gráficas para depurar los algoritmos y mostrar los resultados de la aplicación. Para realizar las interfaces gráficas de nuestro proyecto se utilizara la biblioteca XForms[4] porque *jde.c* la usa. XForms es una biblioteca de libre uso construida en C que permite crear y usar interfaces gráficas sobre el sistema X-Window de Linux.

Esta biblioteca ofrece una herramienta llamada *fdesign* que se usa para construir gráficamente las interfaces y generar automáticamente su código en lenguaje C. Cada interfaz consta de un formulario sobre el cual podemos poner diferentes objetos gráficos como cajas, textos, relojes, diales, deslizadores, menús, botones, etc.). Estos elementos se pueden usar para controlar la activación o desactivación de parte del código, modificar el valor de variables o parámetros, visualizar valores de variables, crear grupo de objetos y multitud de funcionalidades más.

3.4. Librería Gridslib

En algunas ocasiones los mapas utilizados para la navegación se materializan en forma de rejilla. Una rejilla es un *array* bidimensional de celdillas donde se pueden guardar dos valores: ocupado ó vacío. De esta forma se puede almacenar la información de ocupación de los mapas.

Nosotros hemos utilizado la librería *Gridslib* para almacenar el mapa utilizado en la localización del robot. Con esta biblioteca podemos manejar la rejilla que tendrá la información de este mapa además de ofrecernos funciones para crear, iniciar, y reubicar la rejilla.

[4]http://www.xforms.org/

Capítulo 4
Descripción informática

Presentado el contexto y fijado el objetivo concreto vamos a detallar en este capítulo la solución desarrollada.

4.1. Estructura general

Nuestro proyecto consta de 3 esquemas claramente diferenciados, los cuales se encuentran al mismo nivel. Estos esquemas son: *navigation*, *mapping* y *localization* (figura 4.1).

- ***navigation***: El esquema *navigation* es el encargado de la navegación reactiva segura del robot. Su función principal es explorar autónomamente el mundo mientras se construye el mapa y deambular para comprobar si la localización funciona bien.

- ***mapping***: La función de este esquema es la construcción del mapa del entorno por el que se mueve el robot recogiendo la información de obstáculos que proporciona el sensor láser y la posición del robot.

- ***localization***: Este esquema se encarga de estimar la posición real del robot a partir de la odometría, el láser y el mapa del entorno. La estimación de posición del robot que obtiene es utilizada por *mapping* para construir el mapa.

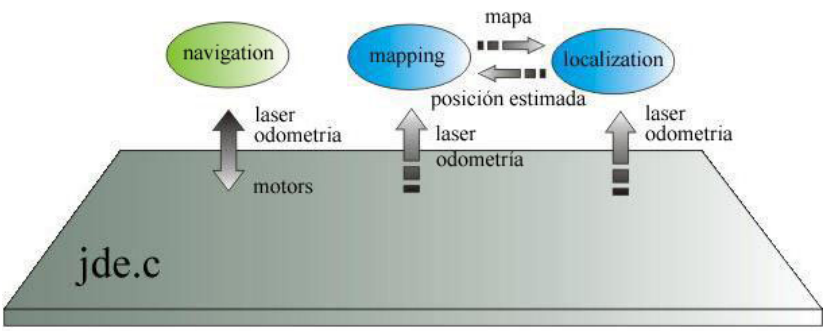

Figura 4.1: Composición esquemática.

4.2. Navegación

El principal objetivo de la navegación en este contexto es conseguir que el robot explore el mundo en el que se encuentra. Para ello el robot necesita un algoritmo de navegación reactiva segura y vivaz.

El algoritmo que se ha decidido implementar es VFF (virtual force field)[Borenstein, 1989]. En él los obstáculos generan fuerzas repulsivas que hacen que el robot se aleje de ellos y a su vez el objetivo genera una fuerza atractiva que hace que éste se acerque al objetivo. Con estas fuerzas se genera una fuerza resultante que determina la dirección que debe seguir el robot y sus velocidades. Esta fuerza resultante debe conseguir que el robot evite los obstáculos a la vez que avanza hacia el objetivo.

La navegación del robot la hemos obtenido de tres formas distintas: teleoperando el robot manualmente, mediante VFF semiautónomo (el destino se obtiene pinchando en la interfaz gráfica) y utilizando VFF autónomo (el destino lo genera el propio algoritmo).

4.2.1. Memoria de puntos

En la figura 4.2 se muestra el diagrama de flujo de la construcción de la memoria.

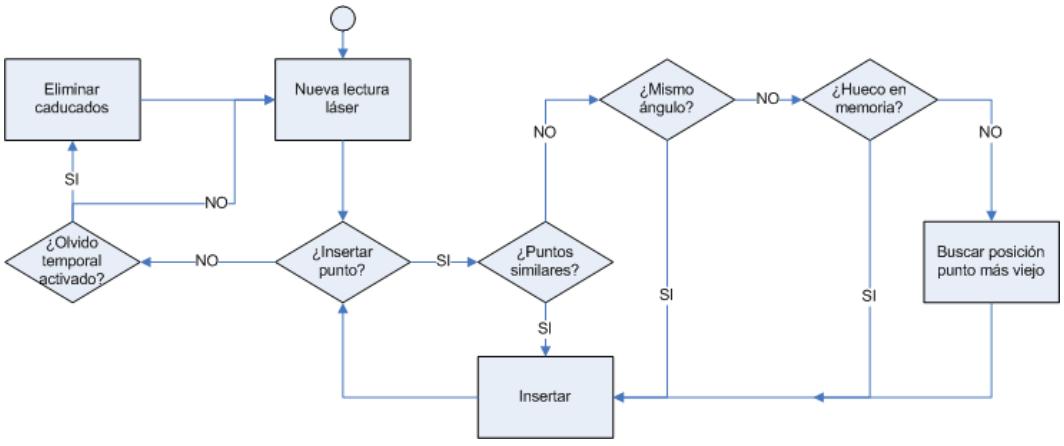

Figura 4.2: Diagrama de flujo de la construcción de la memoria de puntos.

La información de obstáculos se obtiene del sensor *laser(t)*. Un obstáculo se representa por un conjunto de puntos medidos en *laser(t)*. Como en *laser(t+1)* ya no se recuerdan los obstáculos de *laser(t)* se ha realizado una memoria de puntos para

almacenar la información de obstáculos a lo largo del tiempo. En la figura 4.4 se puede ver como la memoria de puntos abarca más espacio que *laser(t)*. La necesidad de la memoria se justifica detalladamente en la sección 5.1.

En cada iteración del algoritmo se incorporan los nuevos puntos medidos con *laser(t)* y se reemplazan los similares o situados en el mismo ángulo respecto del robot para mantener actualizada la memoria. Se ha añadido la posibilidad de sacar de la memoria los puntos con una vida superior a 30 segundos.

4.2.2. Cálculo de fuerzas virtuales

Para calcular la fuerza atractiva hay que obtener el punto destino. Este punto lo obtenemos pinchando en el *canvas* de la GUI (modo semiautónomo) o lo genera el propio algoritmo en una posición al frente del robot alejada entre 1 y 3 metros (modo autónomo). El módulo de esta fuerza se fija experimentalmente (5.1.2) y el ángulo hace que la fuerza apunte a la posición destino.

La memoria de puntos se utiliza para calcular las fuerzas repulsivas sobre el robot generadas por los obstáculos contenidos en ella. Sólo se calcula la fuerza repulsiva de los puntos de la memoria cuya distancia respecto al robot no supera cierto umbral. Éste umbral se utiliza para que los obstáculos lejanos no influyan en la navegación. Es de 5 metros para obstáculos al frente y detrás del robot y para los obstáculos laterales 2 metros. El umbral lateral es menor para que el robot no oscile en el pasillo.

El ángulo de la fuerza de un punto es el opuesto al ángulo que tiene el punto respecto el robot. Esto es así para alejar al robot del punto donde está el obstáculo. El módulo se obtiene siguiendo las ecuaciones 4.1 y 4.2, de modo que los obstáculos más cercanos repelan más.

$$SI\ distancia \geq 2000 \qquad modulo = \frac{2000}{distancia} \tag{4.1}$$

$$SI\ distancia < 2000 \qquad modulo = \frac{3000}{distancia} \tag{4.2}$$

Para cada punto de la memoria obtenemos el ángulo y el módulo para calcular las componentes x e y, las cuales las sumamos a las obtenidas en los otros puntos de la memoria. Así ya tenemos la fuerza repulsiva global.

Una vez que ya tenemos la fuerza atractiva y la fuerza repulsiva, ya podemos obtener

la fuerza resultante por medio de suma vectorial de sus componentes cartesianas. Esta fuerza hace al robot alcanzar el objetivo sin chocarse con los obstáculos.

4.2.3. Cálculo de velocidades

El cálculo de la velocidad de traslación (v) y de rotación (w) que se ordena a los motores del robot se obtiene a partir de la fuerza resultante. Para calcular las velocidades se han utilizado las reglas de control ad-hoc reflejadas en la figura 4.3.

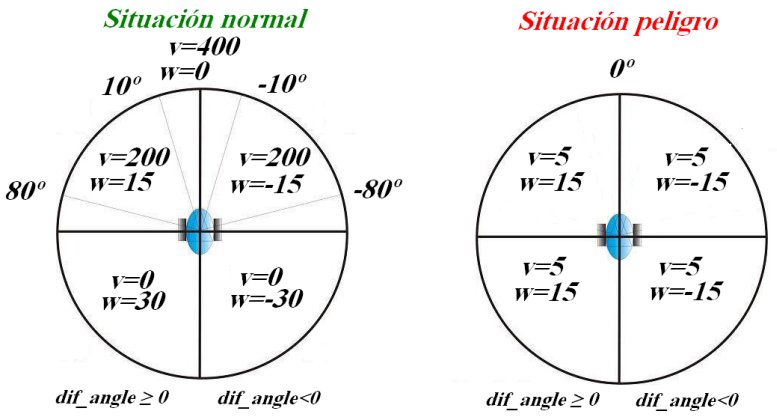

Figura 4.3: Reglas de control ad-hoc para el control de la velocidad.

Por ejemplo, cuando el robot tiene a un objeto muy próximo entonces el robot entra en *situación peligro*. Si el robot está en *peligro* y la diferencia entre el ángulo del robot y el de la fuerza resultante es menor que 0, el valor de v es 5 y el de w es -15.

Si no hay obstáculos muy próximos al robot, está en *situación normal*. Si el robot esta en *situación normal* y la diferencia de ángulo es 75 el valor de v es 200 y el de w es 15. Como último ejemplo, si la diferencia de ángulo no es superior a 10 ni inferior a -10, el valor de v es 400 y el de w es 0.

La figura 4.4 muestra la interfaz gráfica del esquema de navegación. Podemos ver la memoria de puntos generada para calcular la fuerza repulsiva (línea negra de la figura 4.4). También vemos la fuerza atractiva (línea azul) y la fuerza resultante (línea roja). El punto destino está conectado con el robot mediante una recta amarilla para comprobar que el ángulo de la fuerza atractiva es el correcto. En cuanto al ángulo de la fuerza repulsiva, vemos cómo aleja al robot de la pared superior para que éste no se choque con la pared. Como el ángulo de la fuerza resultante se obtiene mediante suma vectorial de la fuerza atractiva y la repulsiva, según vaya girando hacia la izquierda

las fuerzas generadas por los obstáculos de la pared superior al robot irán influyendo menos en éste, haciendo que el robot se dirija al objetivo.

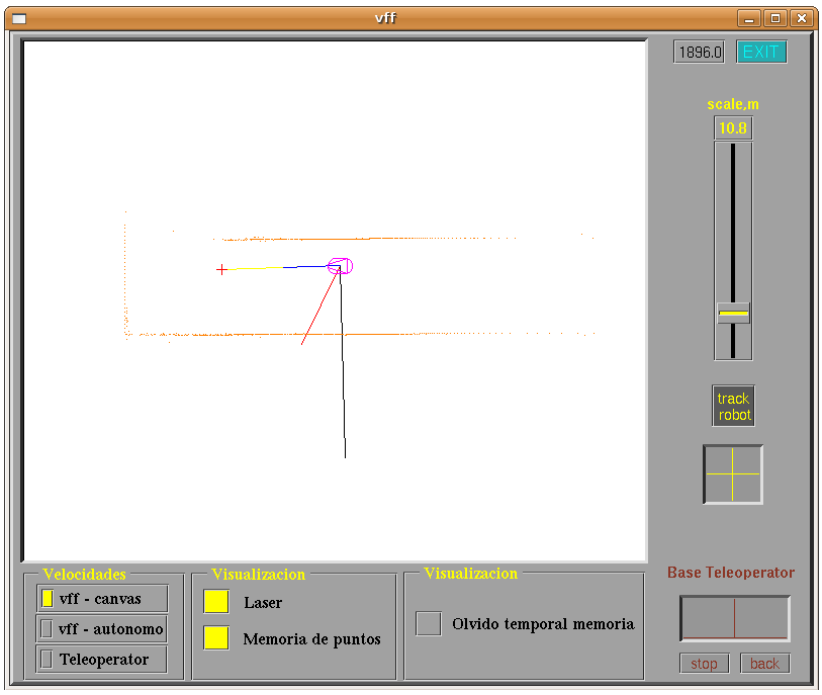

Figura 4.4: GUI del esquema *navigation* y sus fuerzas virtuales.

4.3. Construcción de mapas

Este esquema se encarga de construir el mapa del entorno por el que se mueve el robot. Como se trata de un esquema pasivo, no nos tenemos que preocupar de controlar el movimiento del robot y tan solo generamos el mapa a partir de las observaciones láser recogidas y la posición del robot.

A la hora de construir el mapa y almacenarlo, en vez de utilizar puntos, hemos utilizado segmentos por ser una representación más compacta. Para ello hemos reutilizado el algoritmo que segmenta una instantánea láser realizado en el proyecto fin de carrera de Ricardo Palacios [Palacios, 2006] para detectar las paredes (figura 4.5). Posteriormente, hemos creado una memoria de segmentos para guardar estos segmentos en el tiempo.

Para guardar los segmentos se ha creado un nuevo tipo de dato, *TipoSegment*, el cual tiene dos campos en los que se guardan: el punto de inicio y el punto final del segmento. A la vez se ha creado una estructura *TipoSegmentos* para almacenar en

un *array* de segmentos todos los segmentos que se van construyendo. La estructura *TipoSegmentos* tiene también un campo entero que representa el número de segmentos existentes.

4.3.1. Segmentos láser instantáneos

Para generar los segmentos (figura 4.5) se recorren las medidas láser instantáneas agrupando los puntos que pertenecen al mismo segmento. Para ello se van recorriendo las medidas láser y mirando que los puntos están próximos entre sí y pertenecen a la misma recta. Cuando una de las dos condiciones no se cumple se genera el segmento y se empieza otro a partir de la posición láser donde nos hemos quedado comprobando. La distancia mínima a la que se tienen que encontrar los puntos consecutivos para que se considere que pertenecen al mismo segmento es de 50 centímetros. Por otro lado para determinar si dos puntos pertenecen a la misma recta se utilizan mínimos cuadrados.

Con las ecuaciones 4.3, 4.4 y 4.5 se comprueba si un punto pertenece al segmento que se esta construyendo. a y b determinan los parámetros del segmento y se definen a partir de los puntos que pertenecen al segmento. El punto a determinar si pertenece a la recta es $P(x_1, y_1)$, DIST_SEGM es una constante creada cuyo valor es 5 centímetros y N es el número de puntos que pertenecen al segmento en cada momento.

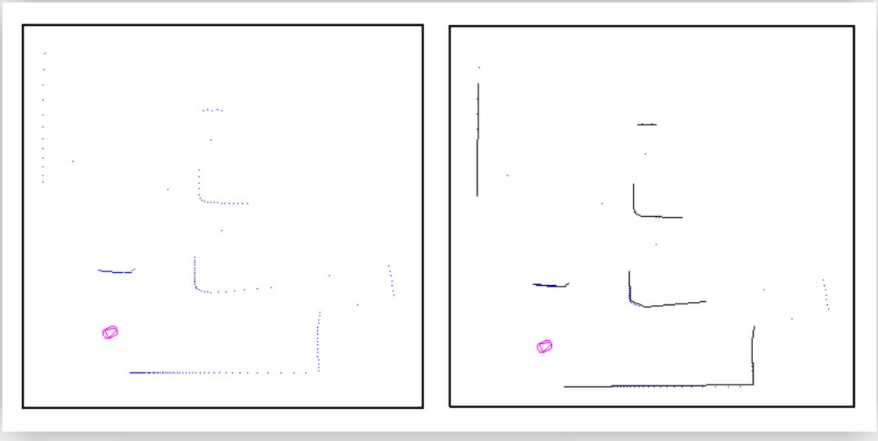

Figura 4.5: Segmentación de lectura láser instantánea.

$$a = \frac{\frac{\sum xy}{N} * \sum y * \sum x}{\frac{\sum x^2}{N} * (\sum x)^2} \tag{4.3}$$

$$b = \frac{\sum y * \sum x^2 - \sum x * \sum xy}{N * \sum x^2 - (\sum x)^2} \tag{4.4}$$

$$Pertenece\ Si \rightarrow \frac{\mid a * x_1 - y_1 + b \mid}{\sqrt{a^2 + 1}} \leq \frac{DIST\_SEGM}{\sqrt{N}} \tag{4.5}$$

4.3.2. Memoria de segmentos

En la figura 4.6 se muestra el diagrama de flujo de la construcción de la memoria de segmentos.

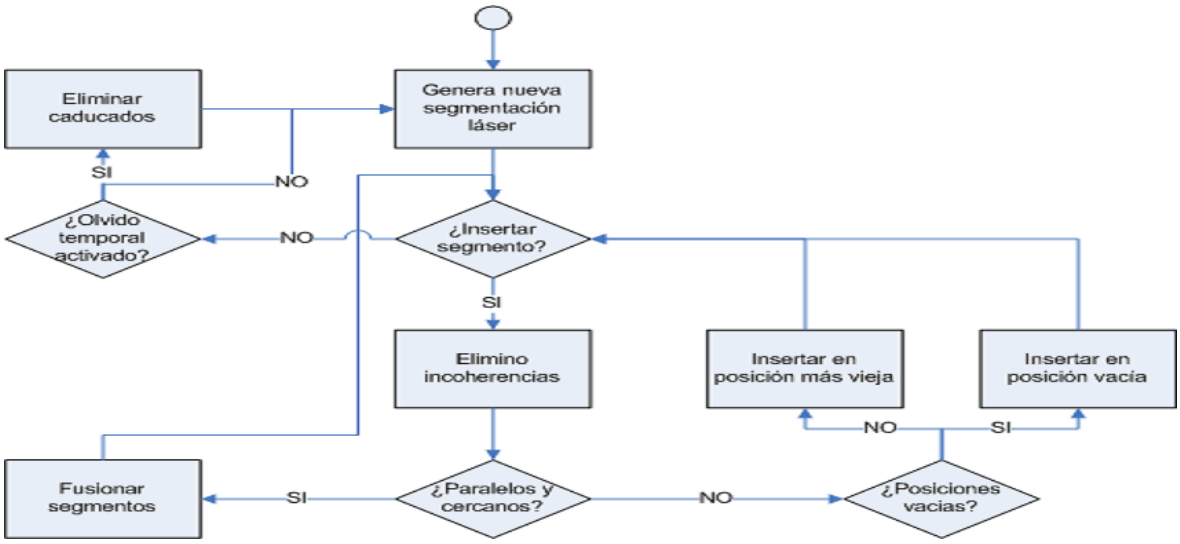

Figura 4.6: Diagrama de flujo de la construcción de la memoria de segmentos.

A la hora de construir la representación del mundo por el que se mueve el robot vamos a necesitar más información que la observación del láser en un instante. Tenemos que ser capaces de tener una memoria de segmentos para calcular rutas, navegar deliberativamente sobre espacios amplios, etc. A la hora de realizar esta incorporación hay que tener en cuenta estos aspectos:

- Los segmentos están representados en coordenadas absolutas.
- El sensor láser es muy fiable. Por este motivo se deben eliminar de la memoria segmentos incoherentes con la observación instantánea, es decir, los segmentos que es imposible que sigan existiendo según los segmentos de la última instantánea láser.
- Se deben fusionar los segmentos que sean continuación uno del otro o estén integrados uno dentro de otro.

- Se debe insertar un nuevo segmento si no es parte o continuación de otro. Si no hay espacio en la memoria se inserta en la posición del más viejo.
- Eliminar los segmentos viejos si la opción de olvido temporal está activada.

En cada iteración de este esquema, primero se eliminan los segmentos incoherentes existentes en la memoria. Para ello se genera un rayo uniendo cada punto del láser instantáneo con el centro del robot y se mira si intersecta con los segmentos de la memoria. Aquellos segmentos que intersecten se eliminan.

Para programar esto se ha utilizado geometría proyectiva, según se explica en el anexo (6.2). Se decidió ésta implementación por dos motivos:

1. La geometría proyectiva, a diferencia de la euclídea, es totalmente genérica, es decir, los mismos cálculos sirven para todo tipo de rectas sin hacer excepciones con rectas paralelas a los ejes de coordenadas.
2. Al usar geometría proyectiva conseguimos hacer todos los cálculos mediante sumas, restas o productos y con lo cual evitamos tener que controlar que no se produzcan divisiones por cero y es muy rápido.

Una vez que se determina que el segmento se debe de eliminar hay que saber si eliminarlo entero o sólo una parte, ya que si el segmento es muy grande puede que se pierda mucha información y sólo se necesite eliminar una parte de éste. Estas son las reglas que se siguen a la hora de depurar/recortar un segmento:

1. Aquellos segmentos menores de 3 metros se eliminan enteros.
2. Aquellos segmentos que tienen una longitud superior a 3 metros se fragmentan y sólo se elimina la parte que intersecta con el rayo láser que es la parte que produce la incoherencia. En la figura 4.7 se muestran los casos que se pueden dar.

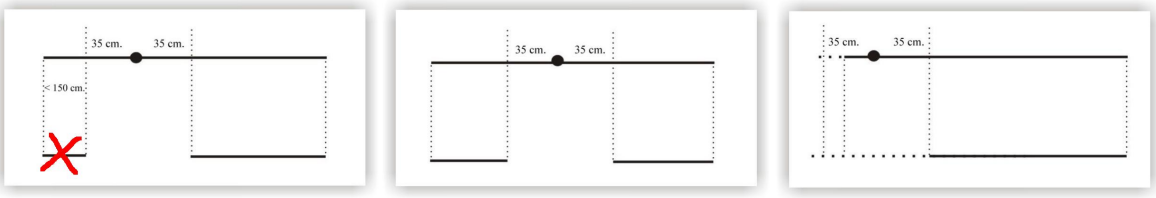

Figura 4.7: Fragmentación de segmentos.

Después de eliminar los segmentos incoherentes comprobamos si el segmento a insertar es prolongación de alguno ya existente y en caso afirmativo realizamos la fusión de ambos segmentos. La fusión se hace sobre el segmento nuevo puesto que es probable que sea más válido que el viejo. Para determinar si un segmento es prolongación de otro exigiremos:

1. Ambos segmentos deben ser *paralelos*. Para que sean paralelos las distancias de los extremos del segmento guardado en memoria a la recta que contiene el nuevo segmento deben ser inferiores a 3 centímetros (figura 4.8).

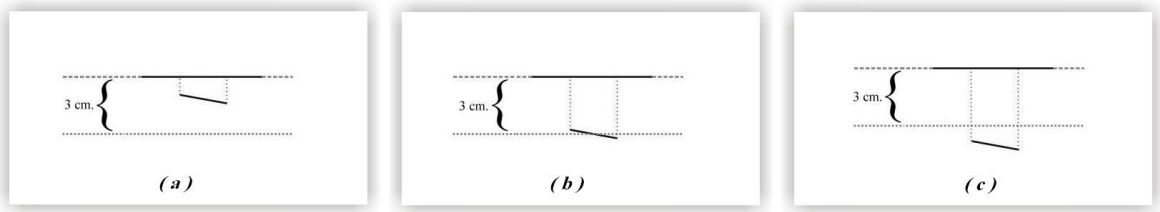

Figura 4.8: Segmentos paralelos (a) y segmentos no paralelos (b y c).

2. Los segmentos deben *estar en distancia*. Para averiguar si los segmentos están cerca calculamos las proyecciones del segmento de la memoria sobre la recta del segmento nuevo. Así obtenemos dos puntos. Según la posición de estos puntos se dan dos casos:

   a) Los dos puntos están fuera del segmento nuevo. Si alguno de estos dos puntos están a menos de 20 centímetros de alguno de los extremos del nuevo segmento se consideran a los segmentos cerca entre sí y por tanto aptos para fusionarse (figura 4.9).

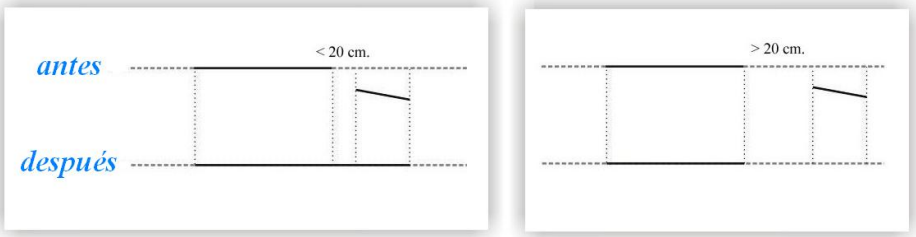

Figura 4.9: Segmento en distancia.

   b) Al menos un punto está dentro. Evidentemente se determina que están cerca (figura 4.10).

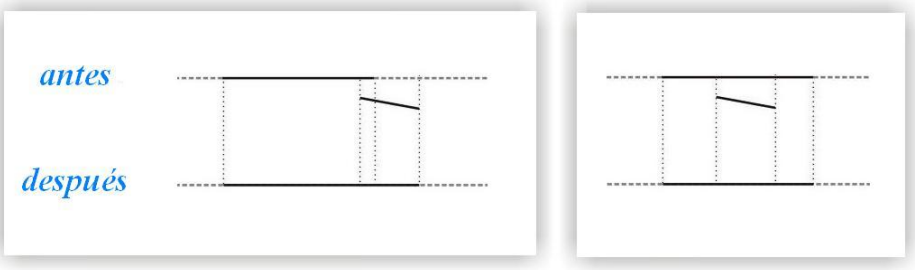

Figura 4.10: Segmentos en distancia.

Si no encontramos ningún segmento existente del que sea prolongación, insertamos el nuevo en la memoria, colocándolo en una posición vacía. De no existir ninguna posición vacía lo colocamos en la posición del segmento más viejo, renovando así el contenido de la memoria.

La posición absoluta de los segmentos se calcula sumando a la posición del robot la posición relativa del segmento medida con el sensor láser. Como la posición del robot puede acumular ruido, puede convenir olvidar segmentos antiguos, que estarán mal colocados. Por ese motivo existe en la GUI la posibilidad de activar un olvido temporal con el cual los segmentos con una vida superior a 30 segundos son eliminados de la memoria.

La figura 4.11 muestra la interfaz gráfica del esquema constructor de mapas mostrando la construcción del departamental II.

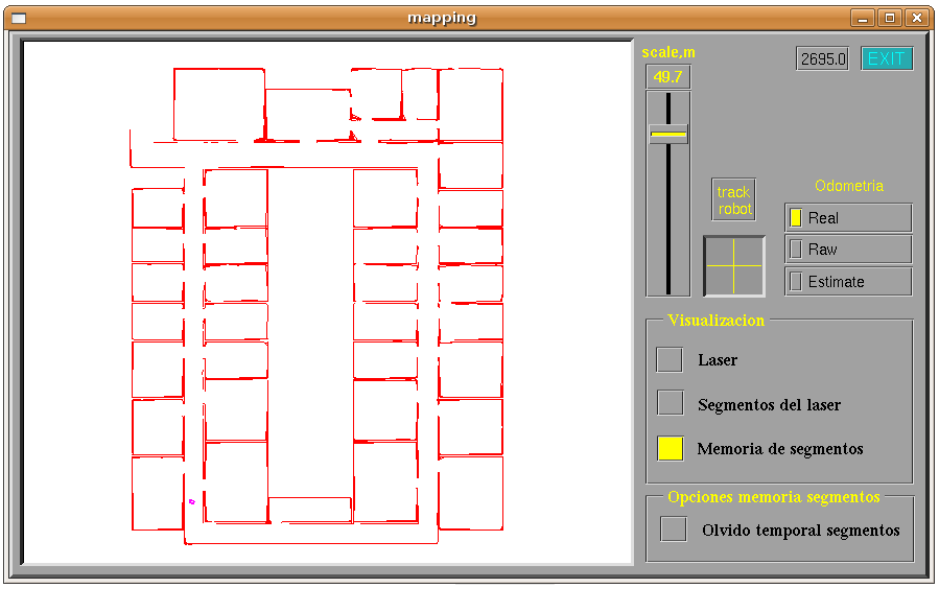

Figura 4.11: GUI del esquema *mapping*.

4.4. Localización con filtro de partículas

La función de este esquema consiste en estimar la posición real del robot. Para ello se han implementado dos técnicas distintas, el filtro de partículas y el algoritmo evolutivo multimodal. En esta sección vamos a explicar el filtro de partículas. Se utilizará el filtro de partículas realizado en el proyecto de Redouane Kachach [Kachach, 2005], haciendo modificaciones para conseguir un mejor funcionamiento.

La finalidad del filtro de partículas es la localización del robot y para ello se genera una población de partículas que se distribuyen uniformemente a lo largo del mundo. Una partícula se define mediante su posición (x, y, θ) y una probabilidad asociada. Así una cierta partícula en un determinado instante se representa de la forma (x_t, y_t, θ_t). La población de partículas, si el algoritmo funciona bien, convergerá a la posición estimada del robot mediante la aplicación iterativa de las tres fases de las que consta el algoritmo: modelo de movimiento, modelo de observación y remuestreo.

La necesidad de realizar este esquema se debe a intentar corregir el error producido por el ruido odométrico presente en condiciones reales. Como vamos a trabajar en el simulador Player-Stage, y éste no incorpora ruido en la odometría, lo hemos generado nosotros (explicado en la sección 4.4.5). Esto lo hemos hecho para acercarnos más a la situación real.

En la figura 4.12 se muestra el diagrama de flujo correspondiente a la ejecución típica del algoritmo de filtro de partículas.

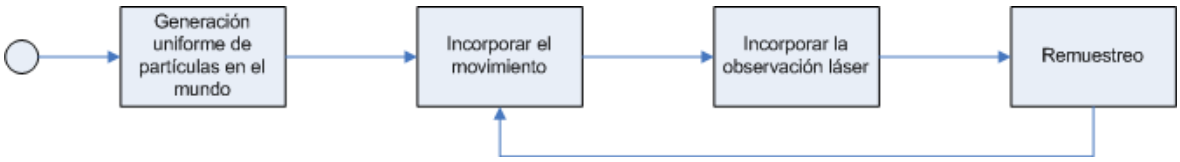

Figura 4.12: Diagrama de flujo del localizador con filtro de partículas.

4.4.1. Lectura del mapa

Para localizarnos necesitamos tener un mapa y las medidas láser (para saber en que lugar está el robot). Para manejar este mapa hemos utilizado una rejilla generada con la librería gridslib (sección 3.4). Para rellenar ésta rejilla utilizamos el mismo fichero que emplea el simulador Player-Stage, del cual obtenemos los siguientes datos:

- La posición inicial del robot (x,y,theta).
- La dimensión del mundo, es decir, su altura y su anchura. La obtenemos en metros y la pasamos a milímetros para trabajar con ella.
- El nombre del archivo con la imagen del mapa, cuyo formato es *pgm* (escala de grises).
- La resolución del mundo. Si el número de filas de la imagen es mayor que el número de columnas, la resolución es la dimensión dividido por las filas. Si por el contrario hay más columnas que filas la resolución es la dimensión partido por el número de columnas.

La información obtenida en el archivo *pgm* la utilizamos para rellenar internamente la rejilla de ocupación. Para saber si una celdilla está ocupada se mira el píxel correspondiente de la imagen. Para ello se utilizan las siguientes ecuaciones, las cuales pasan de celdilla a píxel y viceversa.

$$Y = \frac{(celda/Grid.size) * Grid.resolucion}{mi\_mundo.resolucion} \tag{4.6}$$

$$X = \frac{(celda\ mod\ Grid.size) * Grid.resolucion}{mi\_mundo.resolucion} \tag{4.7}$$

$$Pixel = Y * mi\_mundo.columnas + X \tag{4.8}$$

4.4.2. Modelo de movimiento

Este modelo incorpora el movimiento efectuado por el robot. De esta forma conseguimos que las partículas generadas se desplacen según el movimiento realizado por el robot y así las partículas que caen en la posición del robot mantienen la posición de éste. En la figura 4.13 se muestra un ejemplo de la incorporación a una partícula del movimiento efectuado por el robot.

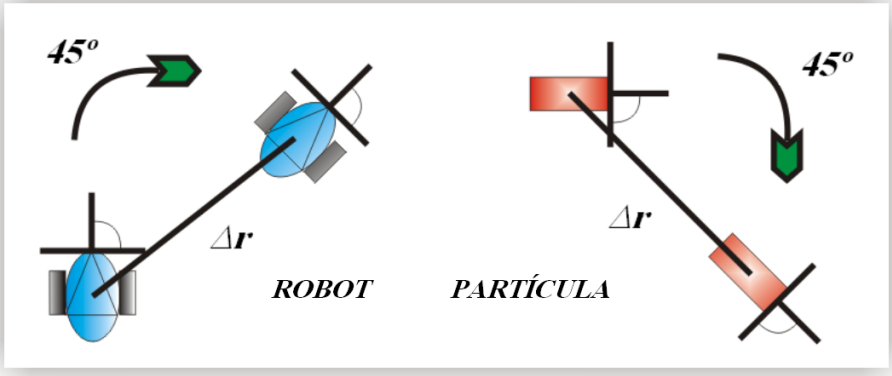

Figura 4.13: Incorporación del movimiento del robot en la partícula.

En primer lugar obtenemos el desplazamiento efectuado por el robot en coordenadas polares para trabajar con un desplazamiento en radio y en ángulo $(\Delta r, \Delta\theta)$. A la hora de incorporar este movimiento a las partículas añadimos un ruido gaussiano $(ruido_r, ruido_\theta)$. De esta forma las partículas que en un primer lugar son copias idénticas de su padre (ver remuestreo en 4.4.4), se redistribuyen alrededor de este.

De esta forma para determinar las nuevas coordenadas de una partícula $(x_{t+1}, y_{t+1}, \theta_{t+1})$ se realizan los siguientes cálculos:

$$x_{t+1} = x_t + (\Delta r + ruido_r) * cos(\theta_t) \quad (4.9)$$

$$y_{t+1} = y_t + (\Delta r + ruido_r) * sin(\theta_t) \quad (4.10)$$

$$\theta_{t+1} = (\theta_t + \Delta\theta + ruido_\theta)\,\%360 \quad (4.11)$$

4.4.3. Modelo de observación

Después de haber incorporado el movimiento del robot a todas las partículas hay que incorporar la observación. La fase de incorporación de la observación consiste en asignar a cada partícula de la población la probabilidad que tiene de que el robot esté realmente situado en la posición (x, y, θ) representada por la partícula.

Para este modelo hemos construido un algoritmo que genera el láser teórico para la posición de cada partícula de la población. Si éste láser es parecido al real la posición es verosímil y su probabilidad es alta. Ésta generación del láser teórico se realiza en cada iteración del algoritmo, 90 veces para cada partícula (1 vez cada 2 grados para cubrir los 180 grados de apertura en cada barrido del sensor láser). Por esto, la generación del láser teórico es fundamental para la realización de este proyecto y debe ser lo suficientemente rápida para que el robot pueda localizarse en tiempo real, un requisito muy fuerte.

La forma de generar el láser teórico será totalmente distinta a la forma utilizada en el proyecto de Redouane. Esto se debe a que los tiempos obtenidos con la forma en que lo hacía Redouane no son válidos para utilizarlo en tiempo real (ver experimentos 5.3.5). Para ello usaremos dos listas (listaXY, listaYX), que contienen las celdillas ocupadas del mapa del mundo ordenadas doblemente.

A la hora de generar el láser teórico hay que tener en cuenta:

- Cómo saber si una celdilla ocupada de la lista intersecta con el rayo láser.

- Cómo crear la lista de ocupadas y cómo recorrerla para que la generación del rayo láser sea lo más rápida posible.

En primer lugar, para saber si una celdilla intersecta con el rayo láser hemos utilizado nuevamente geometría proyectiva (anexo 6.2). Para todas las posibles celdillas se comprueba si su centro está a una distancia menor o igual de la mitad de la diagonal de una celdilla (figura 4.14). Si se encuentra a mayor distancia eso quiere decir que ese obstáculo no se interpone en el recorrido del rayo láser y por tanto no es un obstáculo para ese rayo. Si por el contrario se encuentra a menor distancia hay que determinar la distancia que hay desde esa celdilla hasta el robot. Para ello generamos la proyección (P) del centro de la celdilla (C) sobre el rayo láser de la misma forma que se generaban las proyecciones para realizar la fusión de segmentos en el esquema constructor de mapas. Gracias a esta proyección representada mediante λ, obtenemos el punto sobre la recta donde proyecta el centro de la celdilla y para calcular la distancia al robot tan solo hay que calcular la distancia entre este punto y el robot. En la figura 4.14 se muestra este comportamiento.

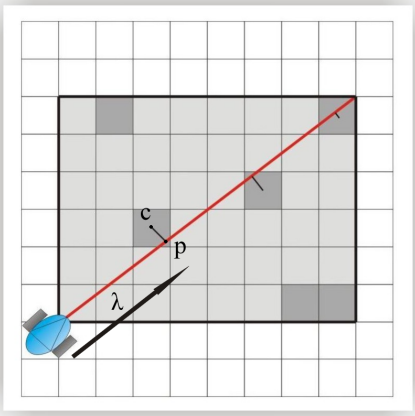

Figura 4.14: Sólo se comprueban las celdillas ocupadas y en distancia.

En segundo lugar, las celdillas ocupadas se han insertado ordenando las celdillas según las componentes x e y:

- Para *listaXY* ordenando por x en un primer lugar y después en cada entrada de x ordenar por y.
- Para *listaYX* ordenando por y para posteriormente ordenar por x en cada entrada de y.

Para calcular el láser teórico se recorre la lista en el orden establecido en la figura 4.15 hasta encontrar una celdilla ocupada que intersecte con el rayo virtual. Esa celdilla representa el obstáculo y si no se encuentra ninguna no existe obstáculo para el alcance de ese rayo (8 metros). Primero hay que decidir qué lista usar en cada caso y en qué orden recorrerla. Realizar un recorrido en orden posibilita que al encontrar la primera celdilla que intersecta con el rayo se pare de buscar porque sabemos que en ese lugar se encuentra el obstáculo más cercano en la dirección del rayo láser.

El ángulo del rayo láser determina si se debe usar *listaXY* o *listaYX*. Si la ventana que hay que comprobar es más alta que ancha en muchos casos habrá menos celdillas que comprobar buscando por las x, y por tanto, es mejor usar *listaXY*. Si por el contrario, la ventana es más ancha, es mejor usar listaYX. Como ejemplo, si el ángulo vale 65, usamos *listaXY* y la primera componente que se comprueba es la x (figura 4.15). El acceso a ambas listas se realiza con búsqueda binaria (figura 4.16).

El ángulo del rayo láser también determina cómo recorrer la lista a usar, si recorrer de mayor a menor las componentes o viceversa. Así conseguimos ganar velocidad significativamente a la hora de generar el láser teórico. En la figura 4.15 se muestra que recorrido usar en cada caso. Por ejemplo, si el ángulo vale 330, las x las comprobamos de menor a mayor. Las y por el contrario, se comprueban de mayor a menor.

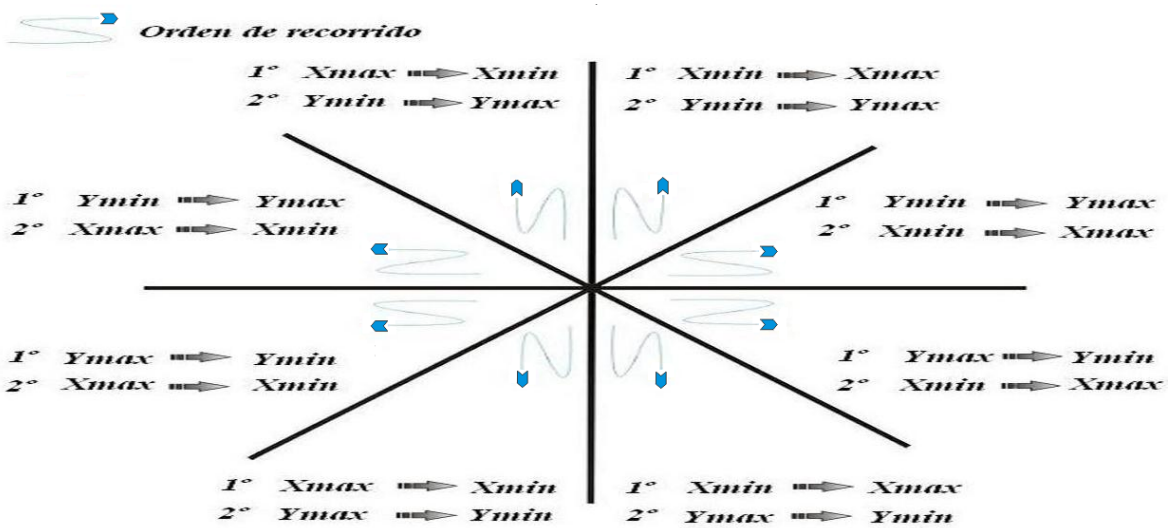

Figura 4.15: Forma de recorrer la lista según la orientación del rayo láser.

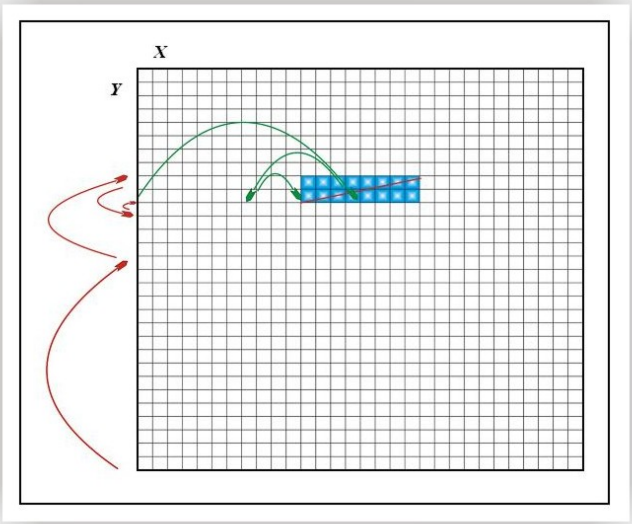

Figura 4.16: Búsqueda binaria en la lista ordenada de celdillas ocupadas. Primero buscamos por y y después por x

A continuación se muestra la generación de los 180 rayos teóricos para diferentes posiciones. Para los experimentos éste número se ha reducido a 90 para ganar velocidad. En los experimentos se demuestra que ésta reducción no produce problemas a la hora de la localización (5.3).

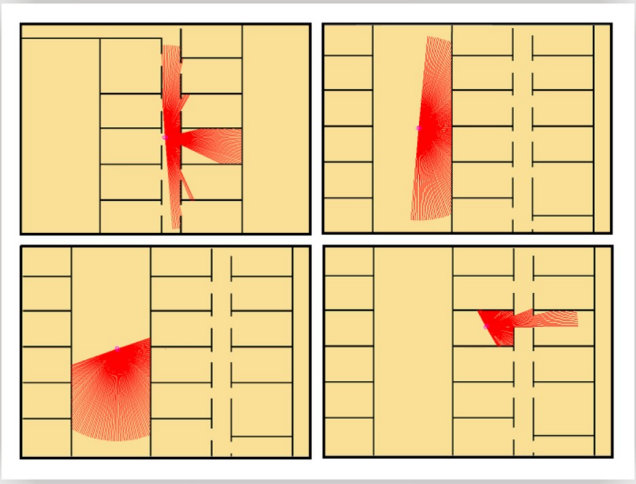

Figura 4.17: Láser teórico generado.

Como hemos dicho anteriormente, la misión del modelo de observación es calcular la probabilidad de que la posición de una partícula sea la posición real del robot. La forma de determinar la probabilidad de una partícula consta de dos funciones separadas e

independientes. En primer lugar, una función distancia determina la diferencia existente entre el láser real y el láser teórico que se obtendría si el robot estuviera en la posición de esa partícula. En segundo lugar, una función traduce esta distancia a la probabilidad que tiene esa partícula de ser la posición real del robot. A menor distancia, mayor probabilidad. Se han probado varias funciones para cada propósito para finalmente elegir las mejores (como veremos en los experimentos de la sección 5.3.2)

La función de distancia que hemos utilizado se basa en el sumatorio de las diferencias de los pares de medidas láser (ecuación 4.12). La forma de pasar la distancia obtenida anteriormente a probabilidad se muestra en la ecuación (ecuación 4.13).

$$distancia = \sum_{i}^{n} \mid l_t(i) - l_r(i) \mid \tag{4.12}$$

$$prob = (\frac{RES * 8000 - distancia}{RES * 8000})^K \tag{4.13}$$

El láser real se representa como l_r y el láser teórico como l_t. RES es una constante cuyo valor es 90, número de rayos láser teóricos calculados. El valor de k es 2 y se ha obtenido haciendo pruebas con diferentes valores para comprobar cual mostraba mejor comportamiento (sección 5.3.2).

4.4.4. Remuestreo

En esta fase se genera la nueva población de partículas. Se ha utilizado el mecanismo de la ruleta que se ha implementado en el proyecto de Redouane [Kachach, 2005](figura4.18(a)). Este mecanismo asigna un sector a cada partícula proporcional a la probabilidad de esta. Así cuanta más probabilidad tenga una partícula, más grande será su sector y más probabilidad tendrá esa partícula de pasar a la siguiente población. Se girará la ruleta tantas veces como partículas se quiera generar y se creará la partícula correspondiente al sector que salga. De esta forma las partículas con más probabilidad tienen más opciones de salir y por tanto la nueva población estará formada a partir de las partículas más probables.

Para la implementación del mecanismo de la ruleta guardamos el acumulado de probabilidad de cada partícula. Por ejemplo para la primera partícula el acumulado es su propia probabilidad. Para la segunda partícula el acumulado es su probabilidad más la probabilidad de la primera partícula. De esta forma las partículas con una probabilidad alta producen saltos mayores en el acumulado y tienen más opciones de salir. Para determinar qué partícula copiar se genera un número aleatorio entre 0 y 1

y se multiplica al acumulado de la última partícula de la población (acumulado total). Cuando una partícula es elegida se inserta una copia en la siguiente población.

En la figura 4.18(b) vemos un ejemplo en el que se crean 10 partículas. El eje x representa las muestras y e el eje y representa el valor de la probabilidad acumulada por cada una de estas. El máximo valor acumulado es de 5. Calculamos un número aleatorio (girar la ruleta) entre 0 y 5, y buscamos la partícula que acumula el valor más próximo al número que ha salido en la ruleta. Podemos ver que las partículas que producen saltos en la probabilidad acumulada (como la partícula 8) tienen más probabilidades de salir en la tirada de la ruleta.

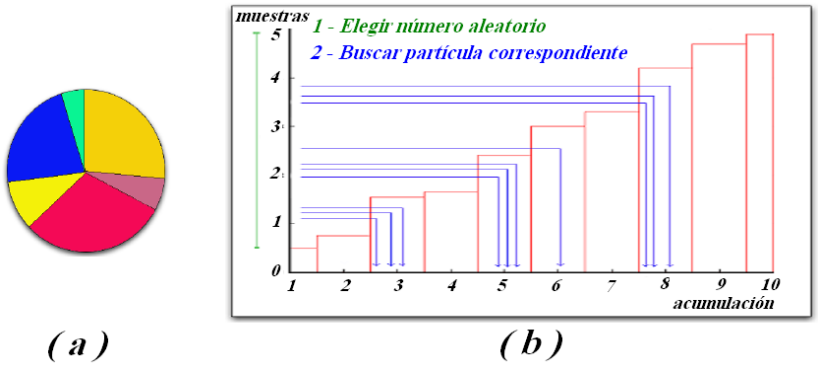

Figura 4.18: Ruleta (a) y generación de 10 nuevas partículas (b).

4.4.5. Generación de ruido odométrico sobre el simulador.

Uno de los objetivos del proyecto es resolver el problema de la localización con las condiciones que se va a encontrar el robot en un entorno real. Un problema que presenta la realidad es el ruido que se produce en los sensores de odometría. Este ruido implica que el robot piense que está en una posición cuando en realidad se encuentra en otra.

El simulador Player-Stage no incorpora ruido de posición. Por ese motivo hemos creado un ruido artificial para poder simular en él este ruido en odometría y así poder demostrar que la construcción de mapas se realiza mejor con nuestro algoritmo de localización, al cual le llega la odometría ruidosa y estima la buena. La implementación de este ruido lo generamos de forma gaussiana (distribución normal).

El ruido que se incorpora a las partículas en el modelo de movimiento explicado en la sección 4.4.2 se genera también de forma gaussiana, pero utilizando otros valores

distintos de los parámetros.

4.5. Localización con algoritmo multimodal

La segunda técnica que se ha desarrollado para llevar acabo la localización del robot se trata de un algoritmo genético o evolutivo [Martínez, 2007]. Ésta técnica tiene cierto paralelismo con el filtro de partículas hablando de individuos en vez de partículas y salud en vez de probabilidad.

La técnica evolutiva multimodal desarrollada se basa principalmente en poblaciones de dos tipos diferentes: exploradores y explotadores. Los individuos de ambas poblaciones se definen de la misma forma que las partículas: (x, y, θ) y se les asocia la salud. Las diferentes poblaciones de explotadores son denominadas razas. Cada raza está compuesta por 50 individuos. El individuo de más salud de una raza será el representante de la raza y las características de ese individuo serán las de la raza que lo contiene.

La función principal de los exploradores es rastrear el mundo para localizar posiciones con alta probabilidad de que el robot se encuentre en ella. Por el otro lado se encuentran los explotadores. Cada vez que los exploradores encuentran una posición muy probable se crea una raza de explotadores. Cada raza explotadora consta de un número fijo de individuos, los cuales se redistribuyen alrededor de la posición a partir de la cual se ha creado la raza. La función primordial de los explotadores es rebuscar al máximo una posición probable encontrada con los exploradores y sus alrededores para determinar si efectivamente en esa posición se sitúa el robot.

El algoritmo evolutivo multimodal consta de dos fases, las cuales se realizan tanto para los exploradores como para los explotadores:

- *Generación de la siguiente población.* En esta fase se generan los individuos de la siguiente población aplicando operadores genéticos. Gracias a estos operadores se puede generar la siguiente población con las características de la población anterior que nosotros deseemos. Los operadores genéticos comunes y que se pueden adaptar a cada problema son el operador de elitismo, el de cruce, el ruido térmico y la mutación. En nuestro problema añadiremos el operador de movimiento.

- *Obtención de la salud.* La obtención de la salud determina la "probabilidad" que tiene cada individuo de estar en la posición real del robot. La forma de calcular esta salud es idéntica a la forma de calcular la probabilidad de las partículas en el filtro de partículas.

En la figura 4.19 se muestra el diagrama de flujo de la iteración del algoritmo evolutivo multimodal.

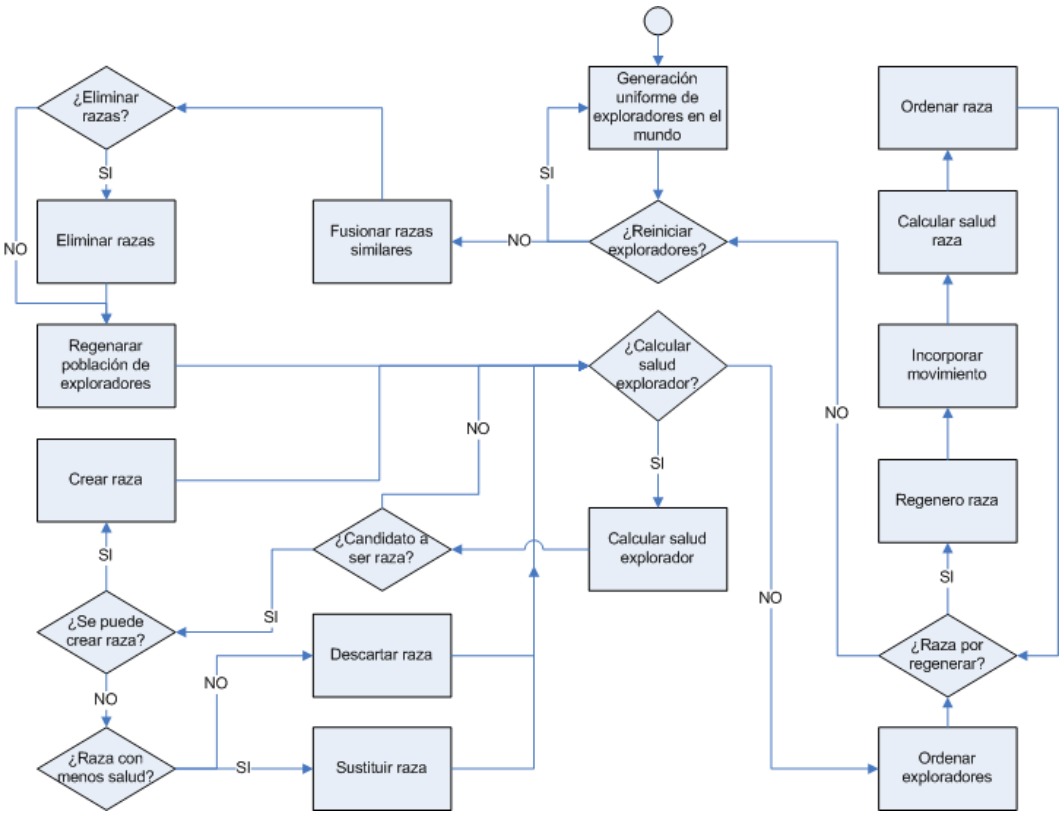

Figura 4.19: Diagrama de flujo del algoritmo evolutivo multimodal.

Lo primero que se hace en una iteración del algoritmo evolutivo multimodal (figura 4.19) es comprobar si se ha activado en la GUI el reinicio de las razas. Si es así se borran todas las razas para crear más nuevas. Si no se ha activado el reinicio se comprueba si alguna de las razas existentes no es válida. Si la salud de alguna raza está por debajo de un cierto umbral esta raza se elimina. Posteriormente se intentan fusionar las razas. De esta forma sólo se van manteniendo las razas más saludables.

Se considera que dos razas se deben fusionar si están a menos de 50 centímetros sus representantes y la diferencia de ángulo de éstos es menor de 60 grados. Para fusionar dos razas se coge el representante con mayor salud y se regenera toda la raza con ese

explotador para que la próxima población se cree a partir de él.

Posteriormente se generan las nuevas poblaciones y se calcula la salud para los exploradores y todas las razas explotadoras. Finalmente después de calcular la salud de cada población, sus individuos se reordenan según la salud creciente.

Después de éste proceso se debe calcular la estimación de posición del robot real que se suministra al constructor de mapas. Será la posición del representante de la raza más saludable. Como la construcción del mapa se realiza de forma continua y la iteración del algoritmo evolutivo lleva un tiempo no despreciable, además de estimar la posición del robot al final de cada iteración, hay que actualizar la posición de éste de forma continua durante la ejecución de la iteración. En la figura 4.20 podemos ver como en cada iteración se estima la posición al robot. Además durante la iteración se realizan actualizaciones de forma continua. Estas actualizaciones llevan incorporado ruido, el cual se corrige cuando se vuelve a dar una estimación.

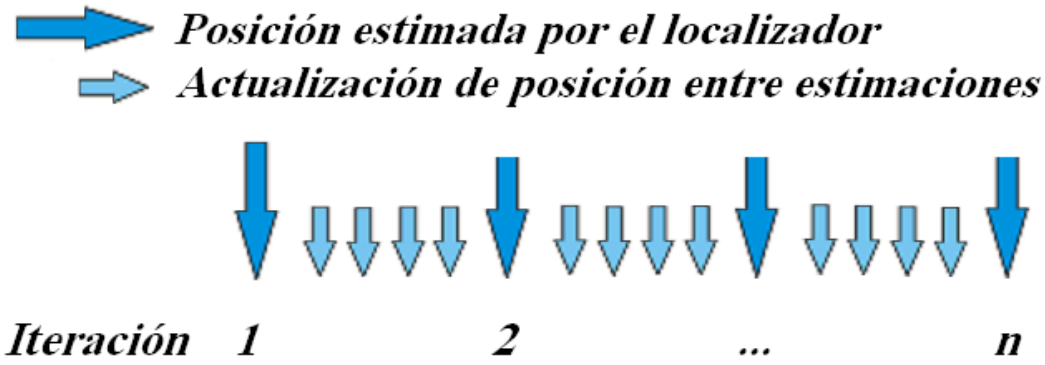

Figura 4.20: Estimación de posición del robot y actualizaciones durante una iteración.

4.5.1. Generación de la siguiente población

La generación de la nueva población se calcula por medio de los operadores genéticos. Según el tipo de población se usan unos operadores u otros:

1. ***EXPLORADORES***:

   Para la población de exploradores solamente se utiliza el operador de mutación, es decir, se crean exploradores aleatorios por todo el mundo.

   - *MUTACIÓN*. Se crea un individuo en una posición dentro del mundo posible totalmente aleatoria.

Cuando calculamos la salud de los exploradores, si ésta probabilidad supera un cierto umbral consideramos que esa posición representada por el explorador es apta para ser explotada. Entonces el objetivo es crear una raza en esa posición. Una raza esta formada en un primer momento por copias idénticas de ese explorador para posteriormente ir explotando esa posición.

Hay un número máximo de razas que se pueden mantener y los pasos al intentar crear una raza son los siguientes:

- Si hay alguna raza existente similar a la raza nueva a crear se deja sólo la raza cuya salud es mayor.
- Si no hay ninguna raza existente similar se comprueba si se pueden crear más razas. En caso afirmativo se crea la raza nueva. En caso negativo se comprueba si alguna raza existente tiene menos salud que la nueva raza a crear y si así fuera se sustituye la antigua raza por la nueva. Si por el contrario todas las razas existentes tienen una salud mayor no se crea la nueva raza.

2. ***EXPLOTADORES***:

Para regenerar la población siguiente de explotadores de cada raza lo que nos interesa es que la nueva población rebusque la posición representada por esa raza para así determinar la posible posición que tiene realmente el robot. El 20 % de los explotadores se generan mediante el operador de elitismo. Otro 20 % se generan mediante el operador de cruce. Por último el 60 % restante de explotadores se generan a partir de ruido térmico.

- *ELITISMO*. Los individuos creados a partir del operador de elitismo son copias idénticas de los individuos más saludables de la generación anterior. Como necesitamos coger los más saludables la población de individuos debe estar ordenada en función a su salud.
- *CRUCE*. Mediante el operador de cruce se realiza la fusión de dos individuos saludables de la población anterior para generar el nuevo individuo. Éste operador también necesita la ordenación de la población. El valor de x e y se obtiene como el punto medio de las x e y de los padres. En el caso del ángulo se ha optado por utilizar el ángulo del padre más saludable.
- *RUIDO TÉRMICO*. Se genera un individuo con los valores del padre aplicando un cierto ruido uniforme. De esta forma el nuevo individuo se

crea en cualquier posición dentro de la circunferencia de centro el padre y radio 25 centímetros y con un ángulo similar.

- *MOVIMIENTO*. Para todos los individuos de la población de explotadores se aplica el operador de movimiento para incorporar el movimiento efectuado por el robot a los individuos. Se realiza de la misma forma que en el modelo de movimiento del filtro de partículas (sección 4.4.2).

Existe la posibilidad de activar en la GUI el uso de repulsión para todas las poblaciones. Si está activado este modo se evita la creación de individuos próximos entre sí. Si el individuo a crear está cerca de alguno se descarta y se crea otro nuevo.

Este algoritmo tiene cierto paralelismo con el filtro de partículas: partículas frente a individuos, modelo de movimiento y remuestreo frente a operadores genéticos, modelo de observación frente a obtención de la salud. Sin embargo, la diferencia que presentan y que supone la principal ventaja de este algoritmo respecto al filtro de partículas es que es multimodal y por tanto es capaz de mantener de manera estable y consistente varios modos, o en nuestro caso, varias posiciones del mundo candidatas de ser la real del robot, como veremos en el capítulo 5.

4.5.2. Interfaz gráfica.

En la figura 4.21 se muestra el esquema *localization*. En él se han implementado muchos *widgets* para depurar bien los algoritmos y probar con multitud de parámetros. Se han detallado por bloques los elementos principales de la GUI:

1. ***BLOQUE 1***: Con estas opciones se pinta el láser real, el teórico generado y el mapa. También se elige entre el filtro de partículas o el evolutivo multimodal. Con *localization* se tiene el mapa desde el principio y con *slam* se obtiene el mapa que se va construyendo.

2. ***BLOQUE 2***: Opciones para elegir la función de probabilidad y el valor de k de la ecuación 4.13. Gracias a estos deslizadores se han probado las distintas funciones generadas para calcular la probabilidad (salud) y así elegir la que mejores resultados ha dado.

3. ***BLOQUE 3***: Con esto se pintan las partículas, se reinician, y con incorpora se arranca el algoritmo del filtro de partículas.

4. ***BLOQUE 4***: Con estas opciones se pintan las trayectorias del robot con ruido (camino rojo de la figura 4.21) y sin ruido (camino azul). Además se puede elegir los parámetros de ruido radial y angular para este ruido.

5. ***BLOQUE 5***: Opciones del algoritmo evolutivo multimodal como pintar poblaciones (tanto exploradores como explotadores), arrancar el algoritmo y deslizadores para elegir los valores para crear, eliminar y visualizar razas.

6. ***BLOQUE 6***: Gráfica para mostrar el valor del láser teórico generado, del láser real y la probabilidad de las partículas o individuos.

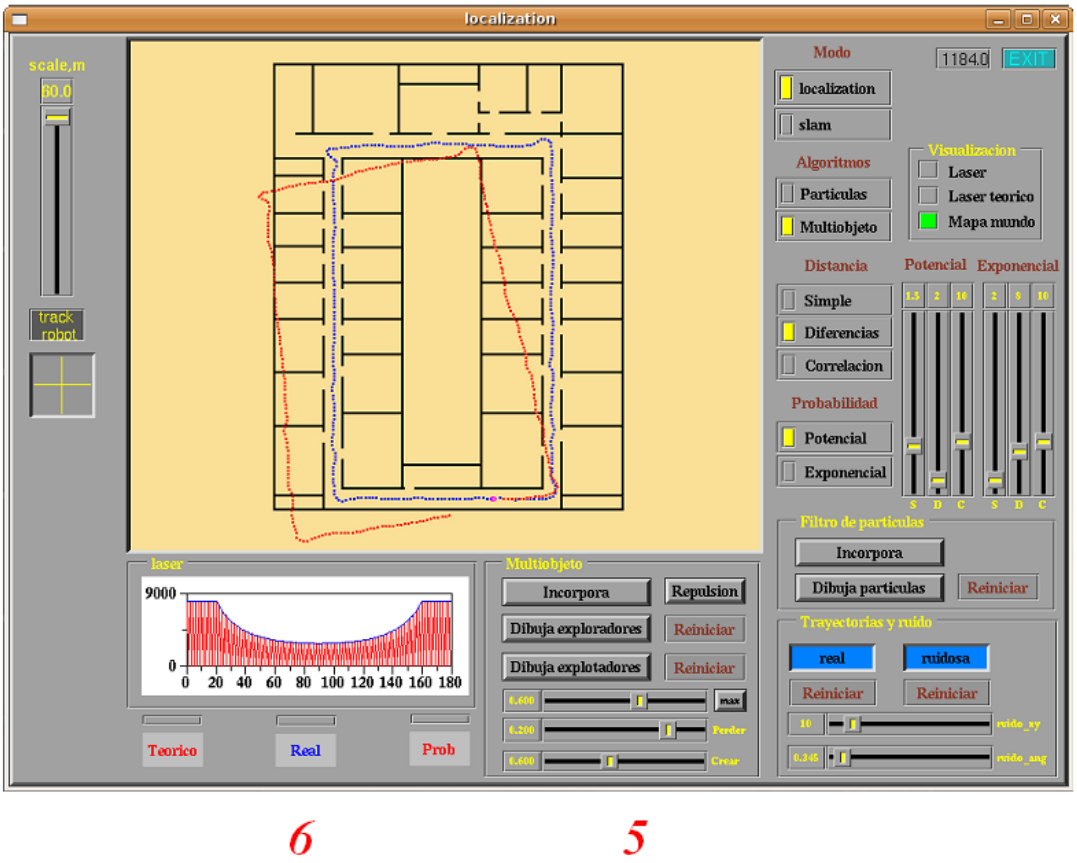

Figura 4.21: GUI del esquema *localization*.

Capítulo 5
Experimentos

Una vez explicada la solución desarrollada vamos a comentar las pruebas realizadas para validar experimentalmente las decisiones tomadas en cada uno de los esquemas realizados, haciendo énfasis en los dos algoritmos de localización.

Además de validar experimentalmente la solución elegida mencionaremos algunas implementaciones alternativas significativas y analizaremos su rendimiento.

5.1. Análisis de la navegación

La navegación conseguida con el esquema *navigation* es vivaz y segura. Para comprobar su buen funcionamiento se ha probado exhaustivamente en mundos simulados en Player-Stage verificando que el robot no se choca.

El mecanismo VFF implementado con la memoria de puntos descrito en la sección 4.2.1 alcanza bien los objetivos cercanos y esquiva obstáculos cuando aparecen en su camino. Por ejemplo en la figura 5.1 se muestra como el robot es capaz de sortear un obstáculo para dirigirse hacia el objetivo.

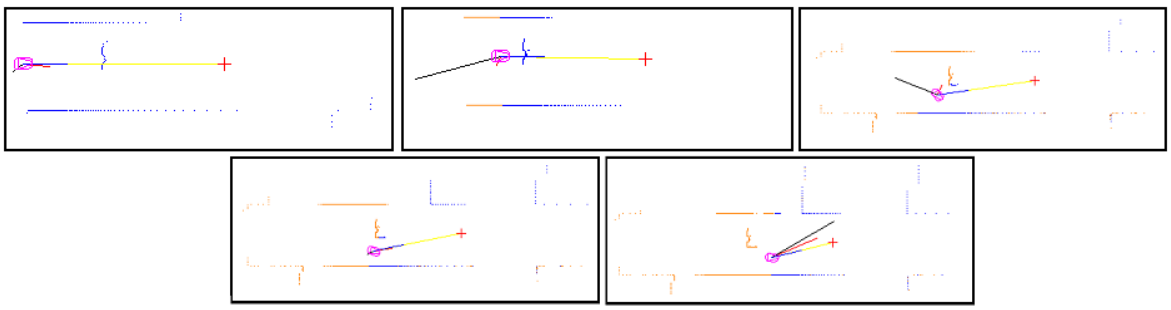

Figura 5.1: Navegación sobre memoria de puntos.

5.1.1. Necesidad de la memoria de puntos

El principal problema que se presenta a la hora de construir el algoritmo de navegación reactiva son las oscilaciones. Cuando el robot ve obstáculos a la izquierda con el sensor láser instantáneo éstos le hacen girar a la derecha. Entonces al girar a la derecha no ve los obstáculos de la izquierda y si los de la derecha que le dicen que gire a la izquierda. Entonces el robot gira a la izquierda y se vuelve al punto de partida.

Para resolver este problema se ha construido una memoria de puntos para generar una estructura capaz de almacenar los obstáculos en el tiempo. Así, cuando se gira, a la hora de calcular la fuerza resultante en la siguiente iteración, además de tener en cuenta los obstáculos que se ven en ese momento tenemos los obstáculos vistos anteriormente.

En la figura 5.2 vemos el comportamiento que tiene el robot si computa las fuerzas virtuales sobre el laser instantáneo y no sobre la memoria, calculándose exactamente del mismo modo. En ambos casos el escenario y el punto de destino son los mismos. Mientras que con memoria evita el obstáculo correctamente (figura 5.1), sin memoria oscila y no es capaz de avanzar.

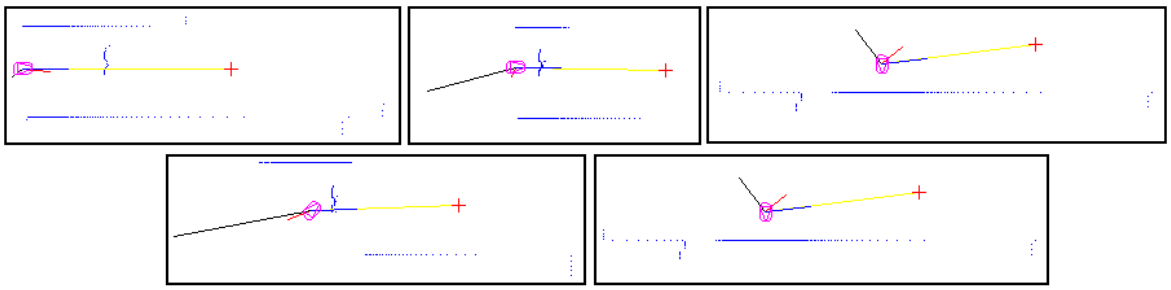

Figura 5.2: Navegación sobre láser instantáneo.

5.1.2. Balance de fuerzas

Otro aspecto interesante en la navegación autónoma es el balance entre fuerzas repulsivas y fuerzas atractivas. Elegir correctamente el módulo de las fuerzas determina que el robot no se choque y llegue al objetivo de forma eficiente.

Para conseguir que el robot llegue al destino evitando los obstáculos que se encuentra en el camino, hay que buscar un equilibrio entre los módulos de la fuerza atractiva y la repulsiva. Podemos ver en la figura 5.3 que si la fuerza atractiva es muy grande, el módulo de las fuerzas repulsivas no influyen lo suficiente para evitar los

obstáculos y el robot en su afán por alcanzar su objetivo se chocaría (figura 5.3c). Si por el contrario el módulo es muy pequeño el robot no se acercaría al objetivo (figura 5.3a). Gracias a los experimentos realizados se ha obtenido el valor del módulo de la fuerza atractiva para que el robot evite los obstáculos acercándose a su vez al objetivo (figura 5.3b).

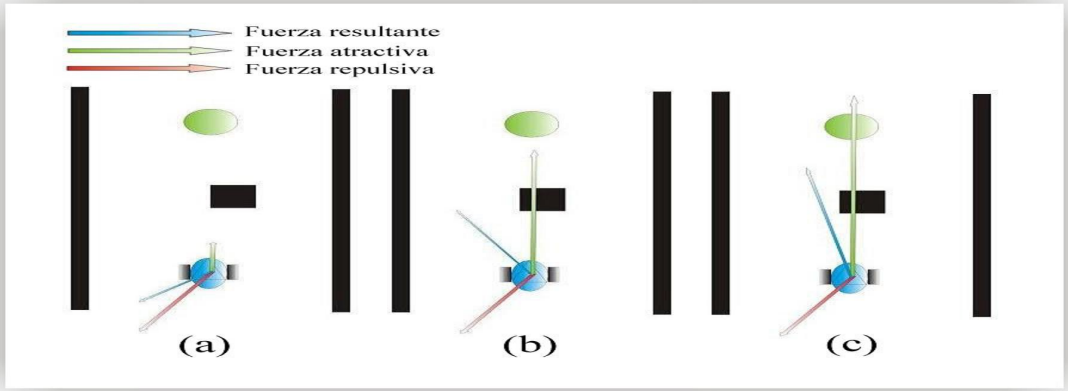

Figura 5.3: Diferentes módulos de la fuerza atractiva: Pequeño-Bueno-Grande.

5.2. Análisis de la construcción de mapas

Para comprobar que el algoritmo funciona bien se han realizado construcciones sobre el simulador con odometría ideal, es decir, sin ruido. Posteriormente hemos realizado pruebas con odometría ruidosa para comprobar cómo influye en la construcción de los mapas.

5.2.1. Comportamiento sin ruido

Probando el esquema constructor de mapas en condiciones perfectas se ha comprobado su buen funcionamiento construyendo de manera eficiente tanto mapas rectilíneos (figura 5.4) como curvilíneos (figura 5.6). En la figura 5.5 se observa la correcta construcción de un mapa de difícil construcción, con rectas, curvas y objetos pequeños.

Los resultados obtenidos indican que la representación de curvas con segmentos es más difícil ya que los segmentos son rectilíneos. Comprobamos que son necesarios más segmentos para representar el mundo puesto que la forma de realizar las curvas es mediante la utilización de segmentos pequeños (figura 5.6). No obstante, aunque se

necesiten más segmentos, representan el mundo perfectamente.

Posteriormente se probó la calidad de los mapas generados en mundos grandes o pequeños. En estas pruebas se ha determinado que el número de segmentos utilizados no crece proporcionalmente al tamaño del mapa. Esto se debe al algoritmo de fusión, que hace que rectas muy grandes se representen con un único segmento.

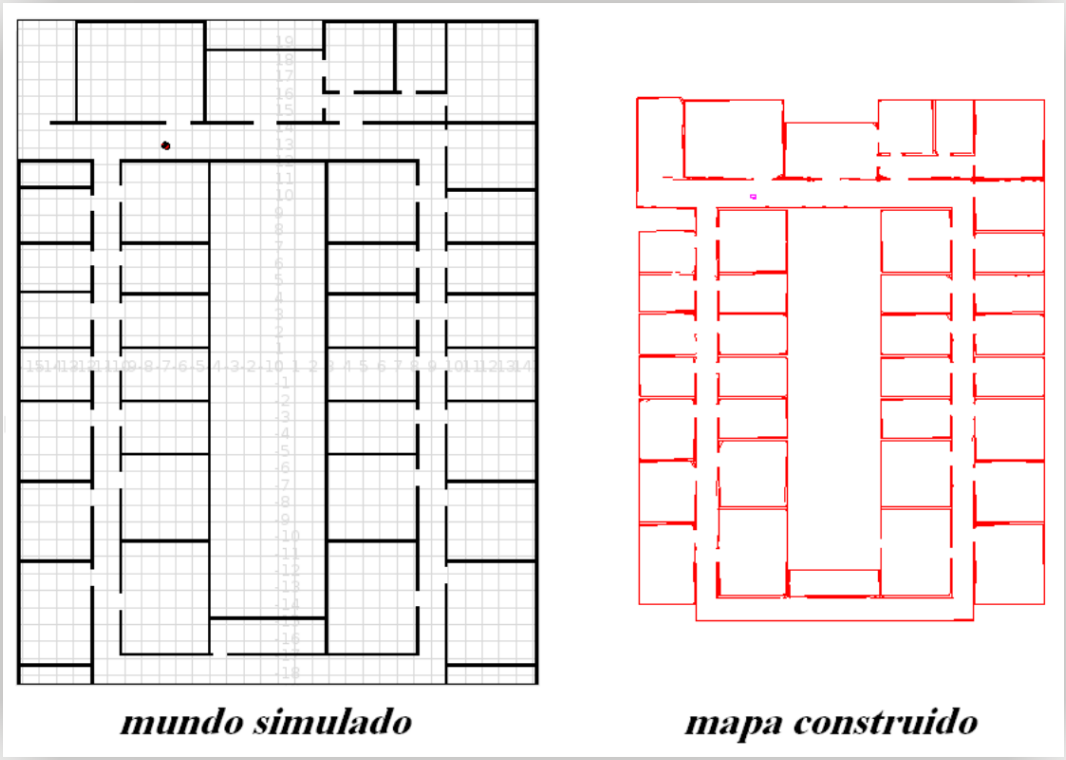

Figura 5.4: Construcción del departamental II.

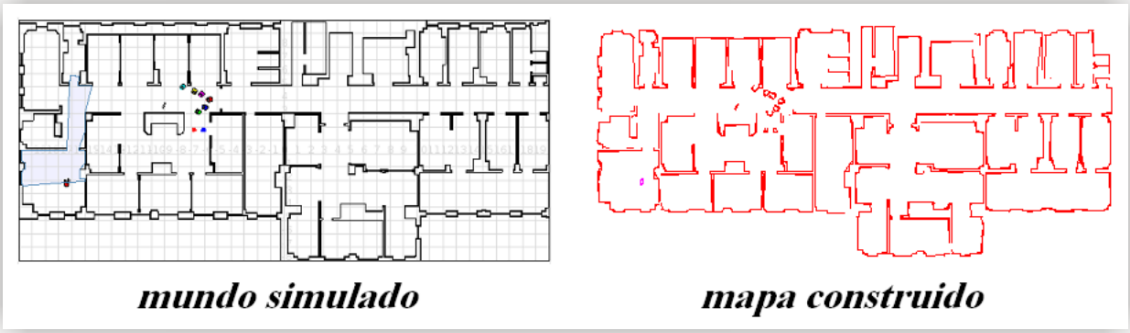

Figura 5.5: Construcción de mundos simulados.

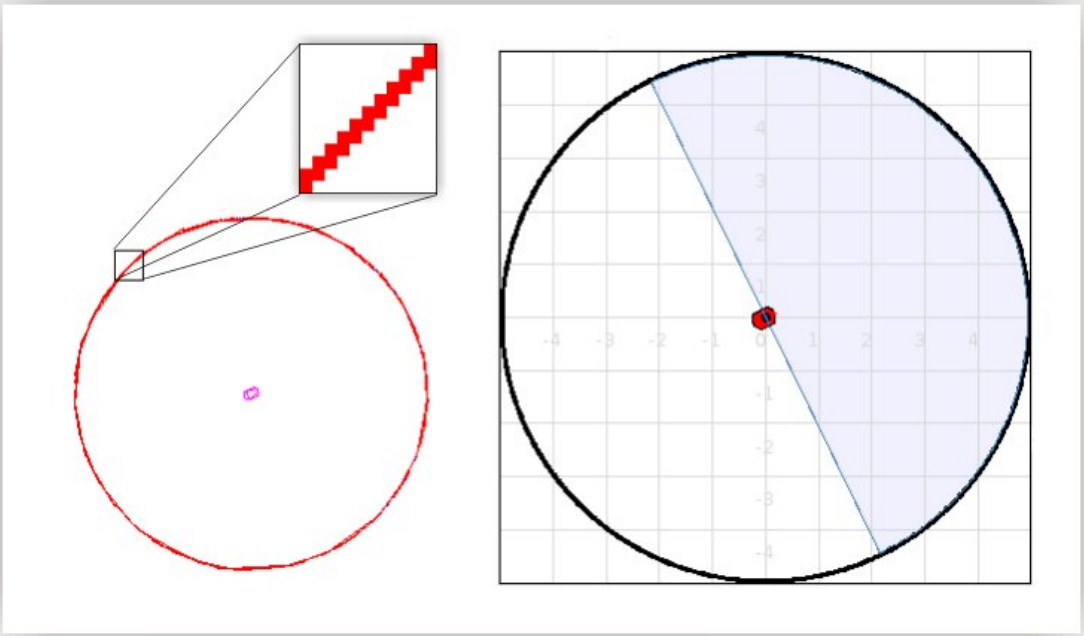

Figura 5.6: Construcción de mapa en mundo curvilíneo.

Por ejemplo, el mapa del departamental II leído por el simulador tiene 70500 celdillas, de las cuales 4083 están ocupadas. Para construir el mapa de este entorno el esquema ha necesitado 721 segmentos y representa 6199 celdillas ocupadas. Hay más celdillas ocupadas porque el mapa construido no es perfecto y tiene pequeñas imperfecciones como paredes más gruesas. Las 4083 celdillas ocupadas del mapa real también están ocupadas en el construido.

5.2.2. Comportamiento con ruido

Como se ha explicado en secciones anteriores (ver sección 4.4.5), el ruido en la posición es un gran problema a la hora de la construcción de los mapas. Al acumularse ruido, los segmentos no se sitúan en las posiciones absolutas reales. Por esto precisamente se necesita un localizador para resolver este problema. Para comprobar cómo afecta el ruido a la hora de construir el mapa, se ha comparado el mapa construido sin ruido (figura 5.7a) y el construido a partir de la odometría ruidosa (figura 5.7b). El robot realiza un recorrido en U invertida para construir los mapas. Podemos ver como el mapa de la figura 5.7b aparece desfigurado, producto de la acumulación de ruido en la odometría. Aunque el robot vuelve al punto de partida, éste no se da cuenta al estar mal colocados los segmentos por el ruido.

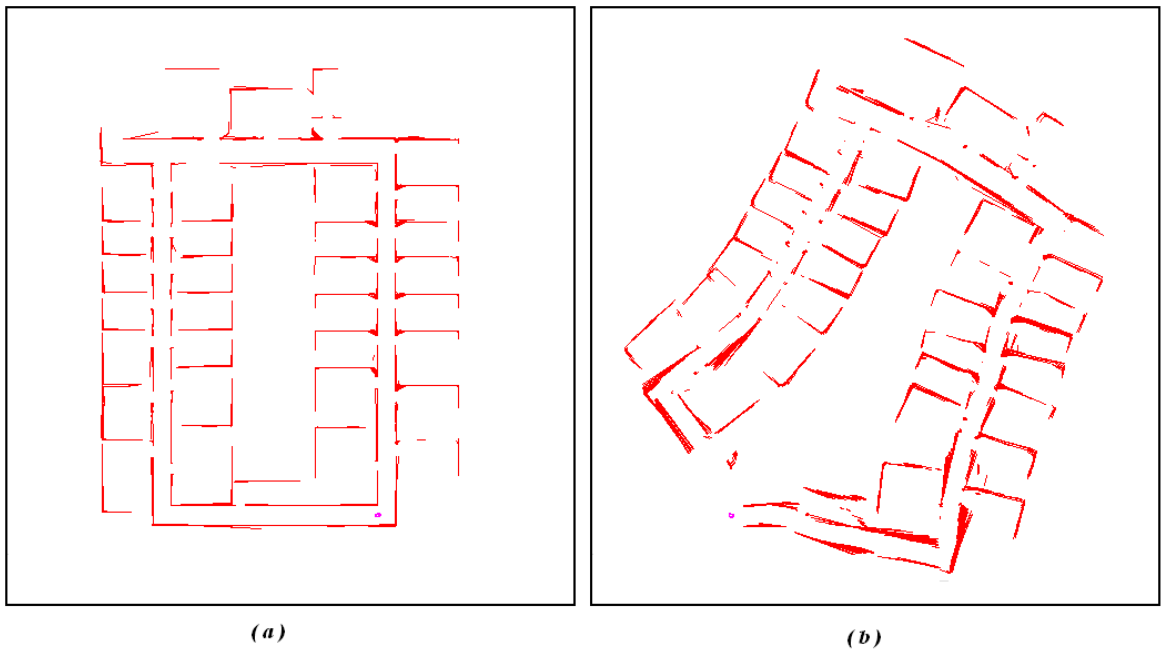

Figura 5.7: Construcción de departamental II sin ruido (a) y con ruido (b).

5.3. Análisis de la localización

Este esquema es probablemente el más importante. En esta sección se ha realizado una comparativa entre los dos métodos implementados: el filtro de partículas y el algoritmo evolutivo multimodal. Para la comparativa y las pruebas realizadas se han utilizado 1000 partículas con 90 rayos láser cada una. Además, se ha utilizado un mundo amplio (39 metros de largo y 30 de ancho).

En esta sección, además de experimentos que validan la implementación desarrollada del filtro de partículas se evalúan las mejoras sobre la técnica del filtro de partículas implementada en el proyecto de Redouane [Kachach, 2005],tanto a la hora de calcular el láser teórico, como a la hora de calcular la probabilidad de las partículas.

También se explican los experimentos que validan y justifican el desarrollo del algoritmo evolutivo multimodal.

5.3.1. Obtención del láser teórico

La obtención del láser teórico es fundamental ya que se deben calcular los 90 rayos láser teóricos para cada una de las 1000 partículas en cada iteración. Se debe optimizar su funcionamiento porque ineficiencias en este punto ralentizan

significativamente a todo el esquema.

Se han realizado y probado varias implementaciones comparando los tiempos obtenidos al final de esta sección (figura 5.11).

1. ***Haz***. En el proyecto de Redouane [Kachach, 2005] la generación del láser teórico se hacía lanzando rayos (haz) en cada ángulo, recorriéndolos con pequeños tics hasta encontrar un obstáculo (la celdilla por la que pasa está ocupada)(figura 5.8). Estos tics tenían que ser muy pequeños para que no se produjeran filtraciones (obstáculos que se saltaban). Al ser tics muy pequeños, la ejecución se ralentizaba, llegando a tiempos medios por iteración de 16 segundos.

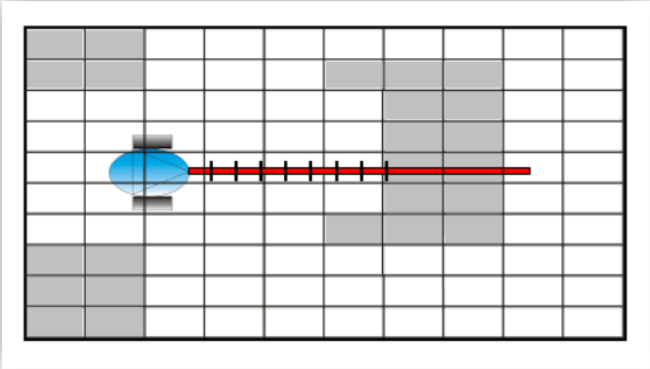

Figura 5.8: Haz láser.

2. ***Intersección***. La primera mejora que se introdujo fue utilizar la intersección de segmentos (figura 5.9a). De la ventana de celdillas que contienen el haz se comprueba si intersecta el rayo virtual con los bordes da la celdilla. El conjunto de celdillas se recorre ordenadamente según el ángulo del haz para parar en cuanto se encuentre una celdilla que intersecte. En el primer y cuarto cuadrante se recorren las celdillas de izquierda a derecha, ascendiendo hacia las celdillas superiores en el caso del primero y descendiendo en el caso del cuarto. Con el segundo y tercer cuadrante el recorrido es de derecha a izquierda, subiendo en el segundo y bajando en el tercero. Con esta mejora se consigue reducir los tiempos a una iteración, en media, cada 4,5 segundos.

3. ***Proyección***. La siguiente mejora que se introdujo fue utilizar geometría proyectiva para calcular la distancia de la celdilla al rayo virtual (figura 5.9b). Se recorren las celdillas de la ventana de cómputo del rayo virtual. Si la celdilla

está vacía se pasa a la siguiente. Si está ocupada y está a una distancia del rayo menor que la mitad de la diagonal de las celdillas, se calcula la proyección del centro de estas celdillas sobre el haz (se obtiene su λ). De todas las λ la menor es la del obstáculo más cercano. De nuevo se realiza un recorrido ordenado para parar en cuanto se encuentre la primera celdilla que intersecte con el rayo.

En la figura 5.10 se observa que en naranja y azul están coloreadas las celdillas que intersectan con el rayo láser virtual. Para comprobar que el recorrido en orden se efectúa de forma precisa pintamos en naranja la primera celdilla ocupada que se encuentra y en azul las posteriores. En una ejecución normal del algoritmo a las celdillas pintadas en azul no se llega porque en cuanto llega a la naranja se para, ya que ahí se sitúa el obstáculo más próximo. Con la implementación de esta mejora se consigue reducir los tiempos significativamente, a una iteración cada 766 milisegundos de media.

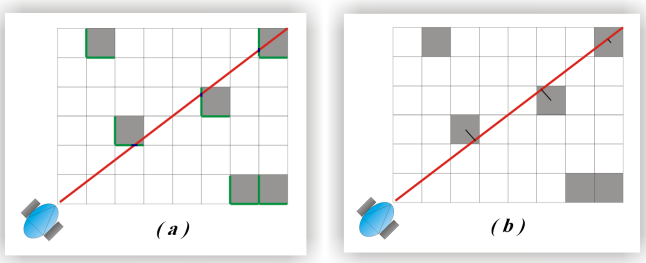

Figura 5.9: Utilización de las proyecciones para encontrar obstáculo.

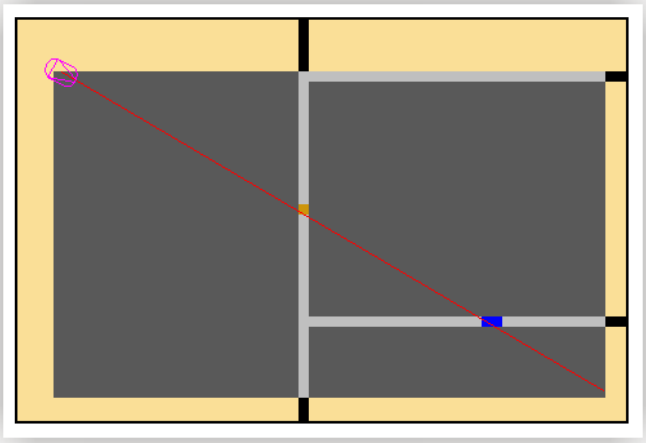

Figura 5.10: Rejilla del rayo láser.

4. ***Ocupadas***. El siguiente paso que dimos para intentar optimizar más fue generar al iniciar el esquema una lista de las celdillas ocupadas de todo el mapa. En vez de barrer todas las celdillas para calcular el láser teórico, sólo se recorren las ocupadas que se tienen en una lista. De estas celdillas, se calcula la λ de las que se encuentren a una distancia del haz menor que la mitad de la resolución de la celdilla. Escogemos la λ menor obtenida. Así se obtiene, en media, una iteración cada 30 segundos.

5. ***Ordenadas***. El siguiente paso dado en la optimización del láser teórico es ordenar la lista de celdillas ocupadas. Creamos dos listas diferentes de las mismas celdillas, *listaXY y listaYX* (sección 4.4.3). Esto nos sirve para usar una lista u otra según la orientación del haz, ya que según usemos una lista u otra habrá más o menos celdillas (figura 4.15). Así comprobamos sólo las celdillas cuya componente (x o y, según orientación) se encuentre en el rango de la ventana del haz. Con esta optimización se obtiene una iteración, en media, cada 11 segundos.

6. ***Búsqueda Binaria***. Por último al generar el láser teórico ordenamos también por la segunda componente, es decir, por ejemplo, en la lista ordenada por las x, cada entrada de x a su vez está ordenada por las y. Así sólo comprobamos las celdillas cuyas x e y están en los rangos de la ventana del haz. Además, realizamos una búsqueda binaria. Con esta mejora llegamos a una iteración, en media, cada 450 milisegundos.

La figura 5.11 muestra la tabla de tiempos de una iteración en segundos para 500, 1000 y 1500 partículas, así como una gráficas de los tiempos. Podemos ver que el tiempo crece con el número de partículas. No obstante, cuando el tiempo es bajo por iteración, aunque el número de partículas sea grande, este tiempo no crece en demasía. Analizando los tiempos, vemos que usando las listas de ocupadas, en un primer momento empeoran los tiempos pero al incorporar la ordenación por las dos componentes y la búsqueda binaria [6] se obtienen mejores resultados. De todas formas la opción de proyección [3] obtiene buenos tiempos también, aunque ligeramente superiores.

Partículas	Haz	Intersección	Proyección	Ocupadas	Ordenadas	B.Binaria
500	8	2,5	0,5	11	1	0,3
1000	16	4,5	0,766	20	11	0,450
1500	34	8,5	1,1	38	20	0,750

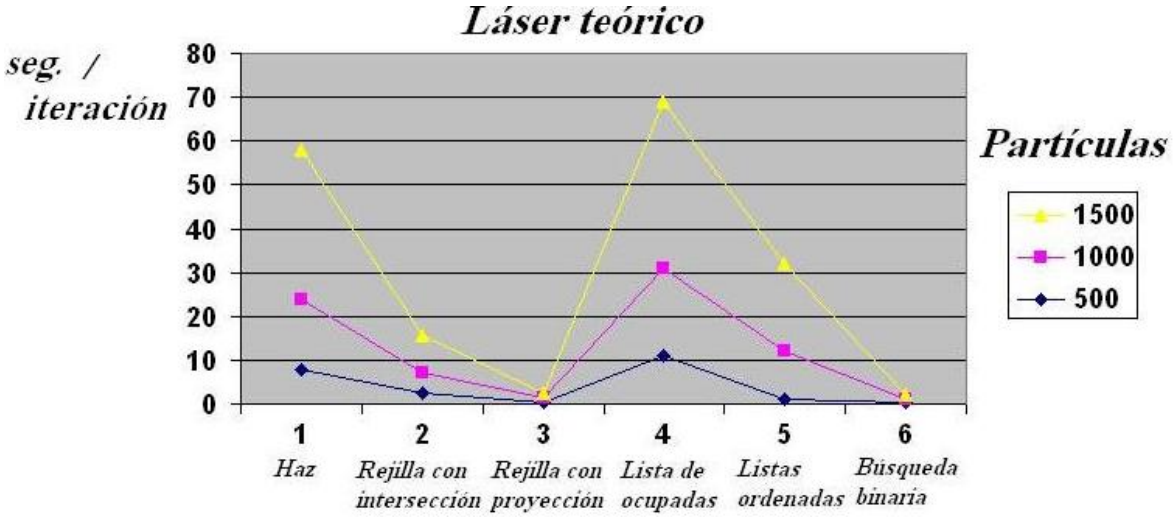

Figura 5.11: Comparativa de la evolución de optimizaciones.

5.3.2. Obtención de la probabilidad/salud

La obtención de la probabilidad nos determina el grado de semejanza entre la posición en la que se encuentra el robot y la representada por la partícula. Con lo cual hay que calcularla de una forma eficiente para que la localización funcione. Esto es así tanto para el filtro de partículas como para el algoritmo evolutivo. De hecho, cuando nos refiramos a probabilidad tal como se usa en el filtro de partículas, también nos estamos refiriendo a la salud tal y como se usa en el algoritmo evolutivo multimodal.

La forma de determinar la probabilidad de una partícula consta de dos funciones separadas e independientes. En primer lugar, una función distancia que determina la diferencia existente entre el láser real y el láser teórico de la partícula. En segundo lugar, una función traduce esta distancia a probabilidad de ser la posición real del robot, de tal forma que cuanto más grande es la distancia más pequeña es la probabilidad.

Hemos probado 6 opciones distintas para el cálculo de la probabilidad (ecuaciones 5.1, 5.2, 5.4, 5.5, 5.7, 5.8). A la hora de pasar la distancia a probabilidad hay dos funciones: *Potencias*, que se basa en elevar estas distancias a k; y *Exponentes*, que utiliza la distancia negada como exponente. k es una variable parametrizable y RES es el número de rayos teóricos del haz láser, siendo su valor 90.

1. ***Simple con Potencias***: *Simple* es la función de distancia realizada en el proyecto de Redouane [Kachach, 2005]. Se basa en la comparación del láser teórico con el real punto por punto. Así, si la diferencia de dos puntos es mayor al umbral

de semejanza, se aumenta en uno la distancia. Con lo cual, valores de distancia próximos a 0 significan que la medida es prácticamente igual que la real y valores cercanos a RES significan que son totalmente diferentes.

La función de probabilidad es:

$$prob = (\frac{RES - distancia}{RES})^K \tag{5.1}$$

2. ***Simple con Exponentes***: La función distancia es la misma que la anterior, pero la función de probabilidad es:

$$prob = e^{\frac{-distancia}{RES*K}} \tag{5.2}$$

3. ***Diferencia con Potencias***: La función distancia es *diferencia*. Esta función se basa en el sumatorio de las diferencias de los pares de medidas láser. De esta forma se diferencian medidas láser que con la primera función no se diferenciaban, ya que la primera función sólo cuenta el número de pares de medidas láser no similares en vez de determinar el grado de diferencia. El láser real se representa con (l_r) y el láser teórico con (l_t).

$$distancia = \sum_{i}^{n} \mid l_t(i) - l_r(i) \mid \tag{5.3}$$

La función de probabilidad es:

$$prob = (\frac{RES * 8000 - distancia}{RES * 8000})^K \tag{5.4}$$

4. ***Diferencia con Exponentes***: La función distancia es la misma que la anterior, pero la función de probabilidad es:

$$prob = e^{\frac{-distancia}{RES*8000*K}} \tag{5.5}$$

5. ***Correlación con Potencias***: La función de distancia es *correlacion*. La correlación mide el grado de semejanza de dos funciones. En nuestro caso vamos a medir la semejanza del láser real (l_r) y el láser teórico (l_t) mediante el coeficiente de correlación de Pearson.

$$distancia = \frac{RES * \sum l_r * l_t - \sum l_r * \sum l_t}{\sqrt{(RES * \sum {l_r}^2 - (\sum l_r)^2) * (RES * \sum {l_t}^2 - (\sum l_t)^2)}} \tag{5.6}$$

La función de probabilidad es:

$$prob = distancia^k \tag{5.7}$$

6. ***Correlación con Potencias***: La función distancia es la misma que la anterior, pero la función de probabilidad es:

$$prob = e^{(1-\frac{1}{distancia})*k} \tag{5.8}$$

Para decidir con qué función de distancia y probabilidad quedarnos se ha evaluado la capacidad de discriminación de cada una de ellas. Para una observación láser real se han obtenido los valores que daban estas funciones para todas las posibles posiciones técnicas del mundo. En la figura 5.12 se compara la probabilidad calculada para las posibles posiciones del mundo sin modificar el ángulo (a mayor probabilidad, color rojo más intenso). Esta comparación se realiza entre la función *simple con potencia*, realizada en el proyecto de Redouane (figura 5.12b), y la función de *diferencia con potencias*, que es la que mejor resultados ha obtenido (figura 5.12a). Se ve como la función *simple* es discriminante en exceso y sólo da mucha probabilidad a las posiciones muy cercanas a la real (dentro de la habitación). Por el contrario la función de *diferencia* es más suave y otorga más probabilidad que la función *simple* a posiciones cercanas a la real, ayudando a guiar a las partículas a converger a un sitio bueno.

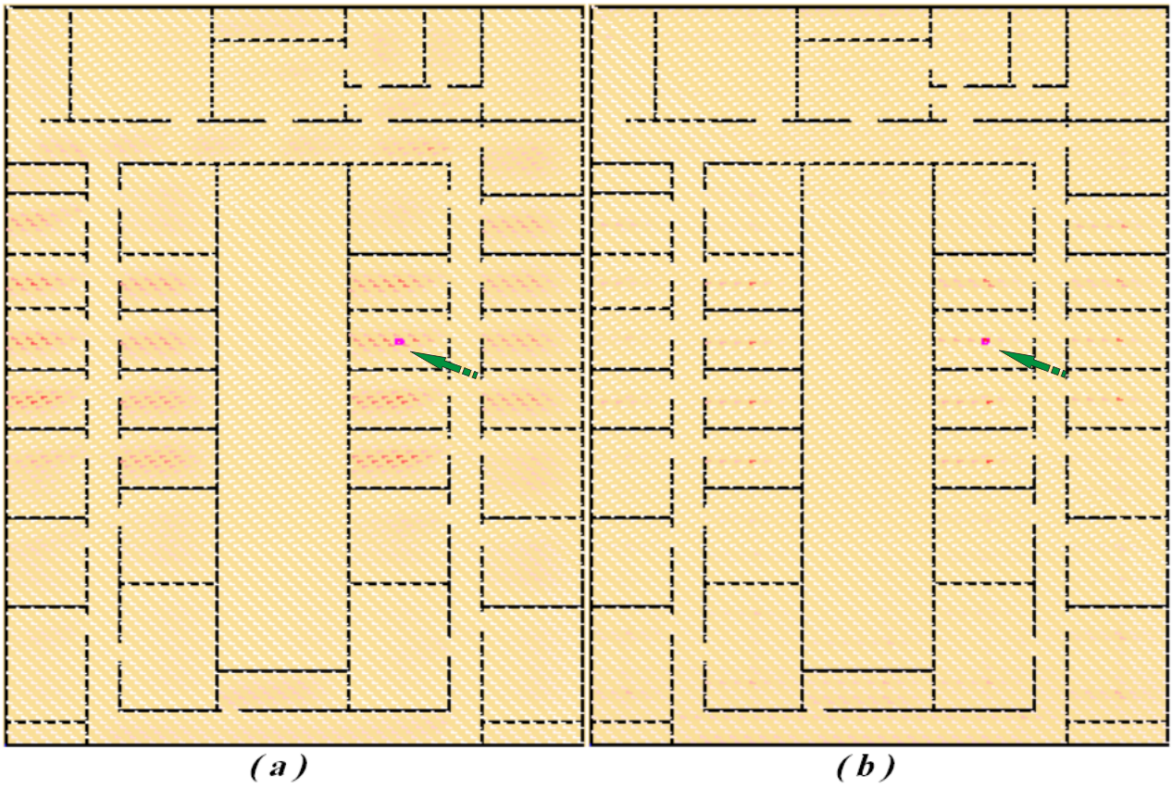

Figura 5.12: Probabilidad para *diferencia con potencia*(a) y *simple con potencia* (b) modificando x e y.

Para comprobar el grado de suavidad de las funciones comprobamos también como

influye el ángulo en el cálculo del láser teórico. Para ello generamos la probabilidad de estar en una posición del mundo, fijando la x y la y, y variando el ángulo. En la gráfica de la figura 5.13a podemos ver que *simple* da menos probabilidad a posiciones cercanas a la real que *diferencia*, la cual otorga buena probabilidad a la posición real y a las próximas (la probabilidad más alta pertenece a la posición real). En la figura 5.13b vemos como *simple* sólo genera buena probabilidad a la posición real. En cambio, la función *diferencia* da más probabilidad. Incluso da probabilidad a posiciones con el ángulo opuesto, siendo está menor por estar más lejos de la pared. Con lo cual la función *simple con potencia* es más discriminante que la función *diferencia con potencia*, que es más suave.

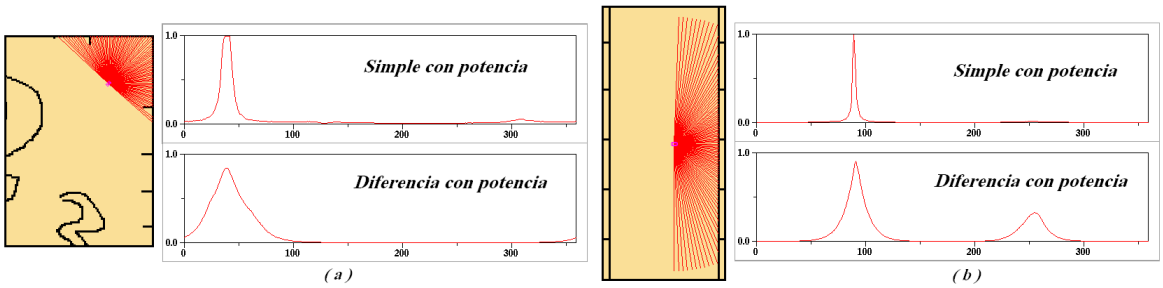

Figura 5.13: Probabilidad de las partículas para todos los ángulos, fijando x e y.

Una vez analizado los resultados obtenidos se determina que las funciones potenciales [1, 3, 5] actúan de forma mejor que las exponenciales [2, 4, 6], ya que las funciones exponenciales son más estrictas y dan menos probabilidad que las potenciales en zonas próximas a la real. Las funciones exponenciales necesitan que las partículas caigan de entrada en posiciones muy cercanas a la real para que se de una probabilidad alta.

Dentro de las potenciales el mejor resultado lo ofrece la diferencia [3] ya que la función de correlación [5] da mucha probabilidad a muchas posiciones y la función simple [1] da poca probabilidad a muchas posiciones. En la figura 5.14b vemos como la función *correlación* da mucha probabilidad (muy poco discriminante) y la función *diferencia* se comporta mejor 5.14a. Se ha determinado usar la función de distancia *diferencia* y la función de probabilidad *potencial*.

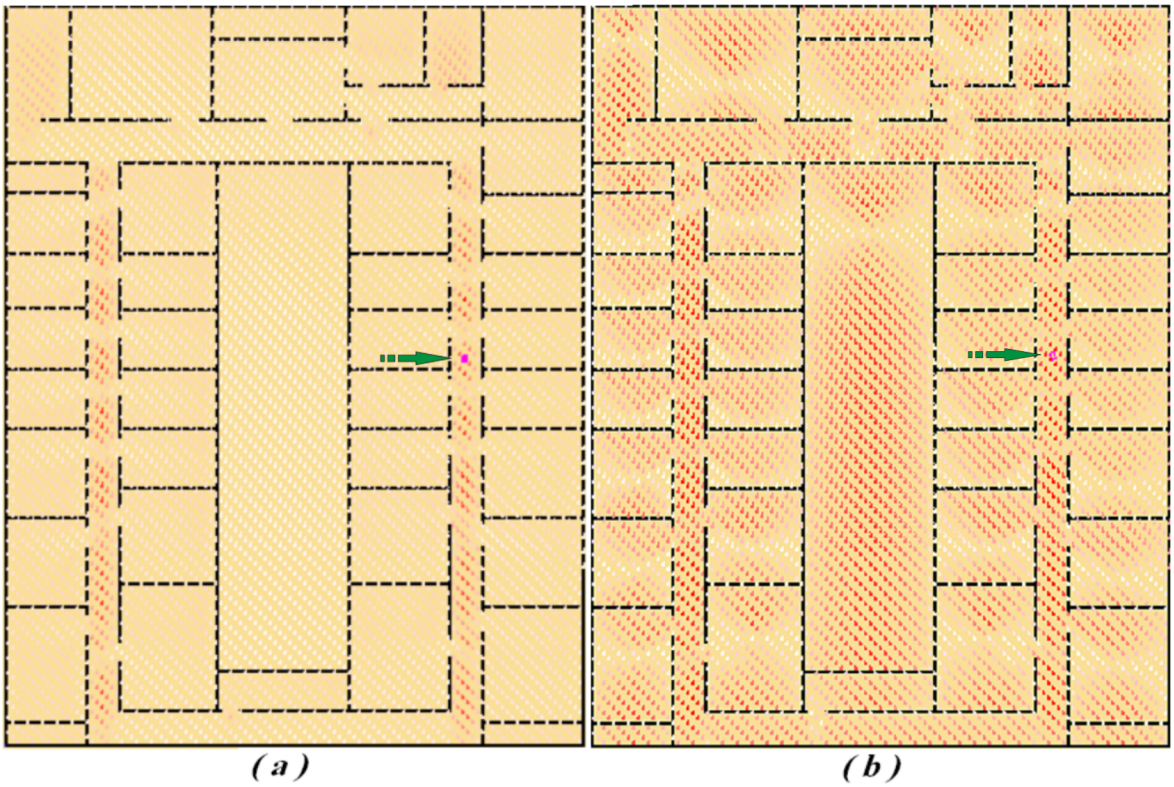

Figura 5.14: Probabilidad para *diferencia con potencia* (a) y *correlación con potencia* (b) modificando x e y.

5.3.3. Resultados del filtro de partículas

Para probar el buen funcionamiento del filtro de partículas se han realizado pruebas sobre el simulador ante mundos asimétricos y simétricos. Se ha usado la función *diferencia con potencia*. Se ha utilizado una población de 1000 partículas.

En primer lugar el filtro de partículas se ha probado en mundos asimétricos. En la figura 5.15 podemos ver cómo las partículas convergen hacia la posición correcta del robot según este gira sobre sí mismo. El robot está situado encima de la palabra Angel.

Por el contrario en la figura 5.16 podemos ver cómo las partículas han convergido hacia una posición no correcta, sino hacia una posición parecida a la real. El robot se localiza en el interior de la cruz, y sin embargo, las partículas convergen al interior del número cuatro.

En las pruebas realizadas en estos mundos asimétricos se consigue localizar el 90 % de las veces. Las zonas que no se conseguía localizar eran las ubicadas dentro de los

números o signos. Las zonas pequeñas de este mundo para el robot son muy parecidas y por lo tanto simétricas, en cuanto encuentran una convergen hacia ella, aunque no sea la buena.

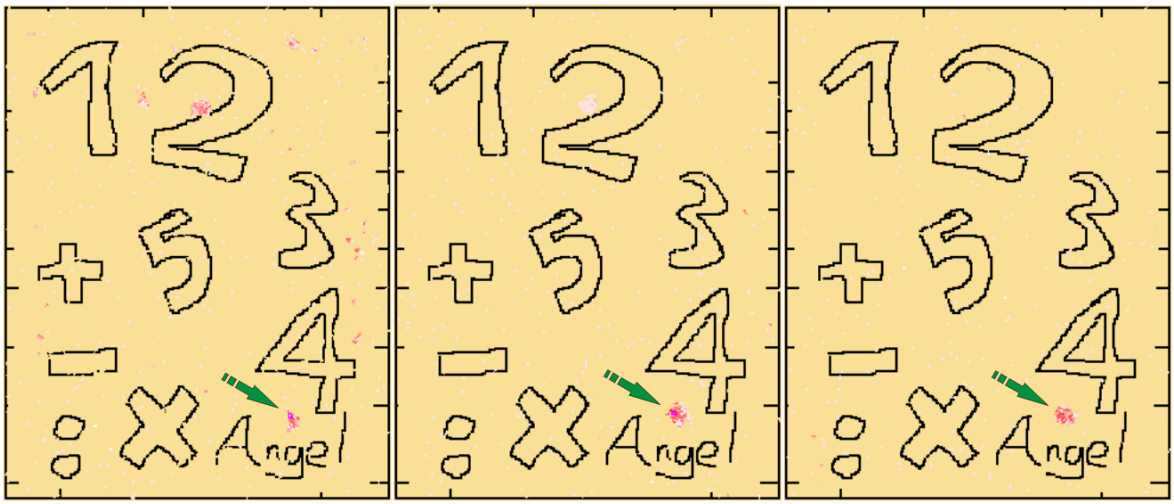

Figura 5.15: Localización eficiente con el filtro de partículas en un mundo asimétrico.

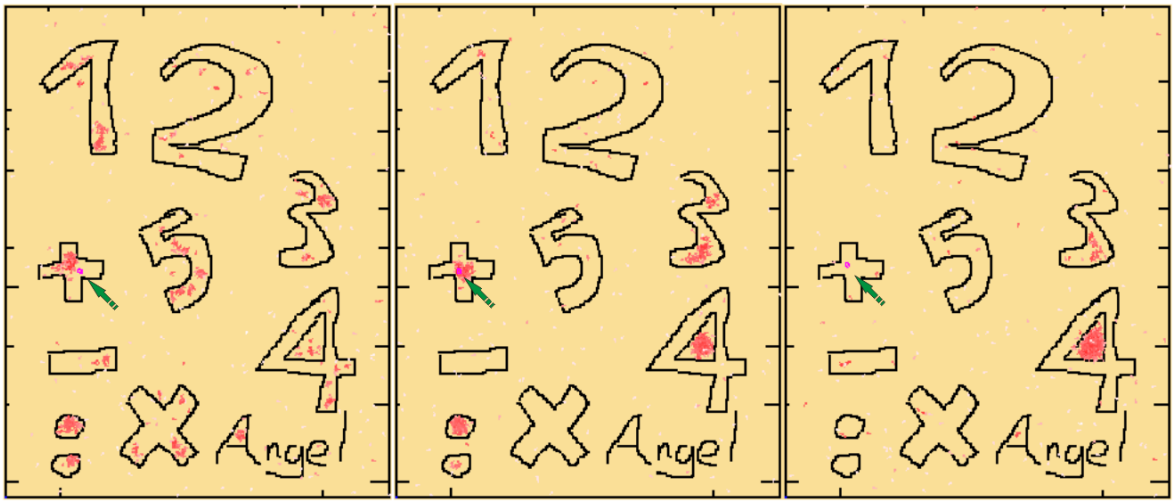

Figura 5.16: Localización fallida con el filtro de partículas en un mundo asimétrico.

También se ha probado el algoritmo en el departamental II de la ESCET-URJC como ejemplo de entorno con mucha simetría. En la figura 5.17 se muestra la ejecución del algoritmo localizando de forma eficiente al robot. El robot avanza por el pasillo superior del departamental II localizándose al final de este.

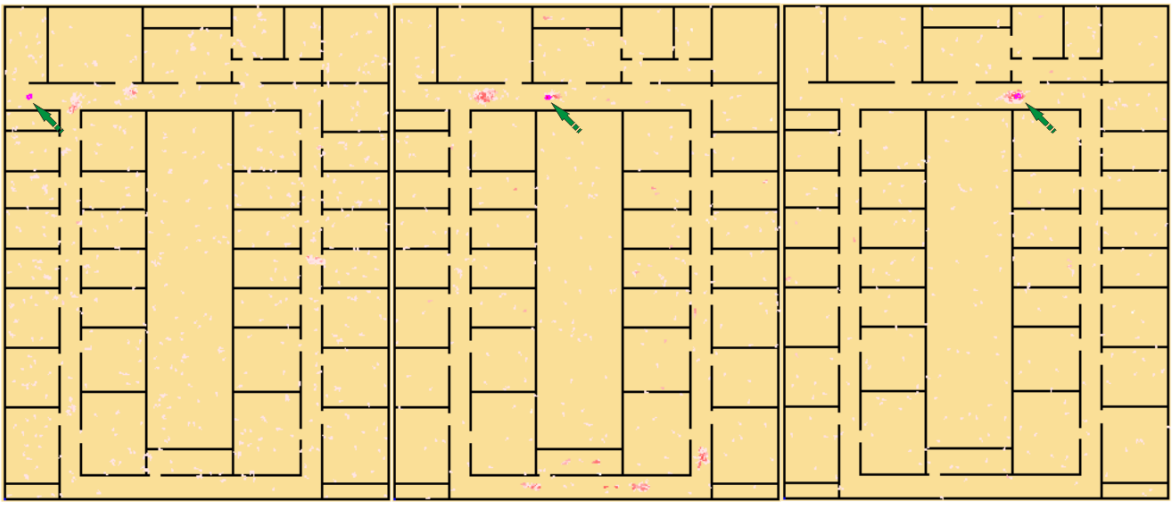

Figura 5.17: Localización eficiente con el filtro de partículas en el departamental II.

No obstante el filtro de partículas no consigue llegar a un resultado exitoso en todas las ocasiones. El problema es que el departamental II tiene numerosas simetrías y el algoritmo no es capaz de mantener de manera estable varias posiciones posibles, así que en cuanto encuentra una posición que puede ser la real, las partículas convergen hacia ella, aunque no sea la real. Este es un gran problema puesto que en nuestras pruebas el algoritmo sólo es capaz de localizarse en la posición correcta el 50 % de las veces. La explicación que hemos encontrado para este bajo porcentaje reside en que cuando el algoritmo converge hacia una posición simétrica que no es la real (convergencia prematura) esta posición se pierde al avanzar el robot, puesto que al no ser la real siempre se llega a un punto donde esta simetría se pierde. Debido al carácter monomodal del filtro de partículas ya no se recupera, no encuentra la posición real.

Relacionado con esto, otro problema observado en las pruebas es el comportamiento ante el secuestro. El secuestro consiste en soltarlo en otra posición del mundo sin que sepa donde está. De nuevo debe localizarse y el filtro de partículas sólo el 50 % de las veces se recupera de él.

En la figura 5.18 se representa una ejecución que converge hacia una posición simétrica, no la real. Mientras el robot está en el pasillo inferior, el algoritmo converge hacía una posición simétrica.

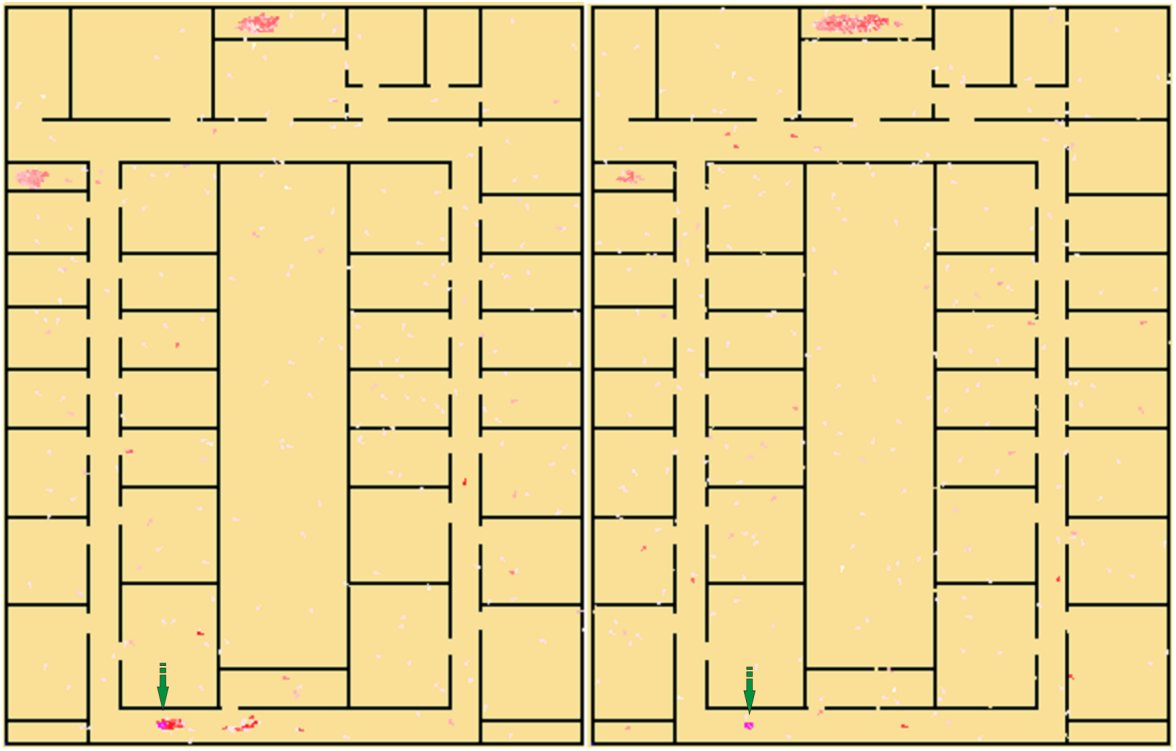

Figura 5.18: Localización fallida con el filtro de partículas en el departamental II.

5.3.4. Resultados del algoritmo evolutivo multimodal

En esta sección se analizan los experimentos realizados al algoritmo evolutivo multimodal y que justifican su desarrollo. Se ha utilizado una población de 1000 exploradores y se han permitido un máximo de 10 razas de 50 explotadores cada una.

Primero hemos probado el algoritmo ante mundos asimétricos. En la figura 5.19 se muestra cómo el algoritmo se localiza en un mundo asimétrico. El robot se encuentra en el interior del cuatro.

También hemos probado el algoritmo en mundos simétricos. En la figura 5.20 vemos la ejecución del algoritmo en el departamental II. Se muestran las razas existentes durante la ejecución del algoritmo y movimiento del robot a lo largo del pasillo superior y entrando en la primera habitación. La raza con más salud y por la que se decanta el algoritmo es la que tiene el círculo rojo y la flecha verde indica la posición verdadera del robot.

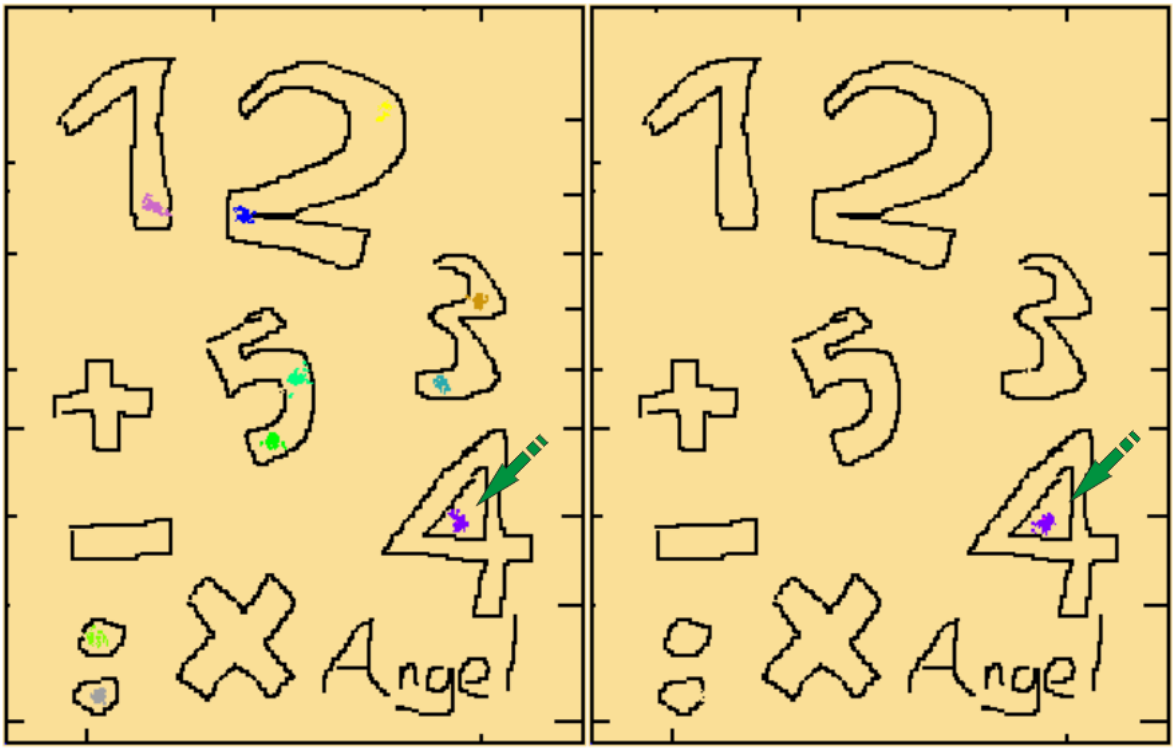

Figura 5.19: Localización en un mundo asimétrico con algoritmo evolutivo multimodal.

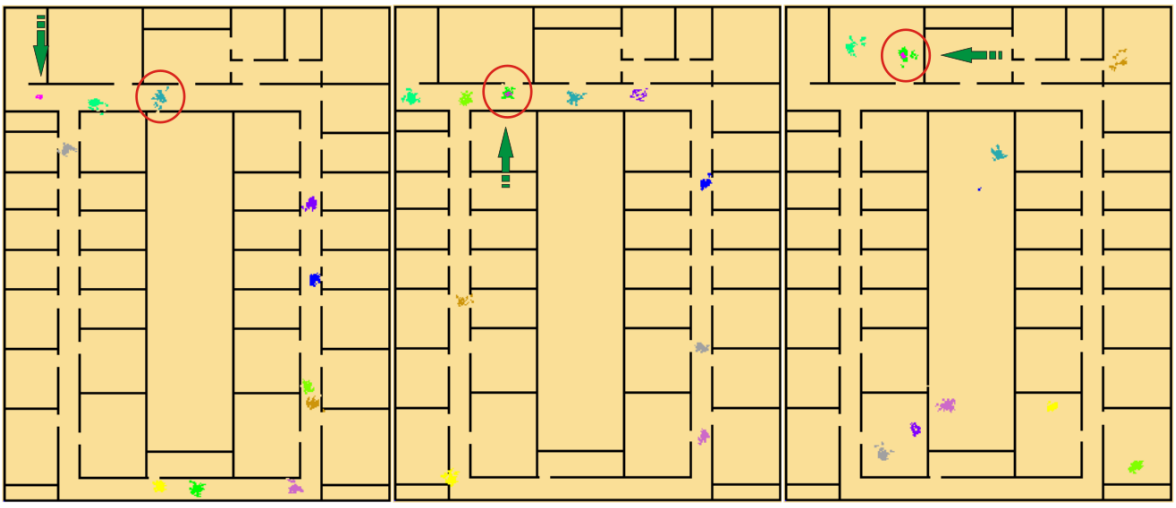

Figura 5.20: Localización eficiente con algoritmo multimodal en el departamental2.

En la figura 5.21 se muestra cómo el algoritmo es capaz de mantener varias posiciones simétricas y además sabe determinar cuál es la posición en la que se encuentra el robot. Estos modos se pierden si se encuentran observaciones incompatibles manteniéndose el bueno siempre entre ellos. La flecha verde indica la posición del robot.

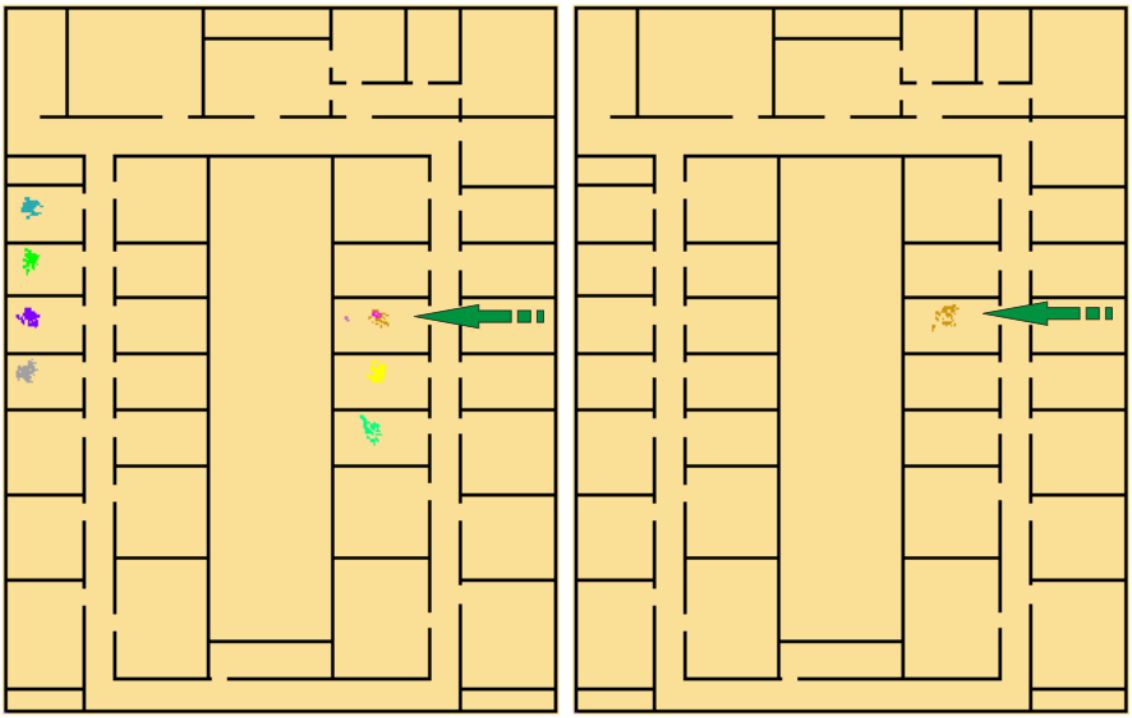

Figura 5.21: Elección de la mejor raza.

Gracias al algoritmo multimodal se ha conseguido resolver los problemas aparecidos con el filtro de partículas (ver sección 5.3.3): el problema del secuestro y la mala localización por la existencia de simetrías. En los experimentos realizados, los mismos a los que sometimos al filtro de partículas, el algoritmo multimodal ha conseguido localizarse en la posición real del robot el 100 % de las veces. Al ser capaz de mantener varios modos, aunque el algoritmo se decida en algún momento por una posición simétrica, el algoritmo se recupera rápidamente por haber mantenido el modo real. Como el filtro de partículas sólo puede mantener un modo, en cuanto se decide por una posición que no es la real, ya no encuentra la real, y el robot se desorienta.

El tiempo para localizar al robot depende del tiempo que se tarde en crear un explorador en la zona correcta del mundo para que los explotadores actúen en ella.

A pesar de los buenos resultados obtenidos se ha encontrado una limitación. Cuando el mundo es muy simétrico hay algún momento que puede alcanzar mayor probabilidad una raza que no es la real. Este problema se resuelve en cuanto se mueve el robot y recopila nuevas observaciones sensoriales. Aunque solamente sea durante una iteración, produce errores a la hora de construir el mapa puesto que se piensa que el robot se encuentra en una posición la cual no es la real. A la hora de construir el mapa este

problema se resuelve, ya que cuando se pasa por la zona del mapa mal construida estando bien localizado se corrige el mapa y se reconstruye bien. En la figura 5.22 se muestra en imágenes este problema, decantándose en un primer lugar por una raza incorrecta y eligiendo bien al desplazarse por el pasillo.

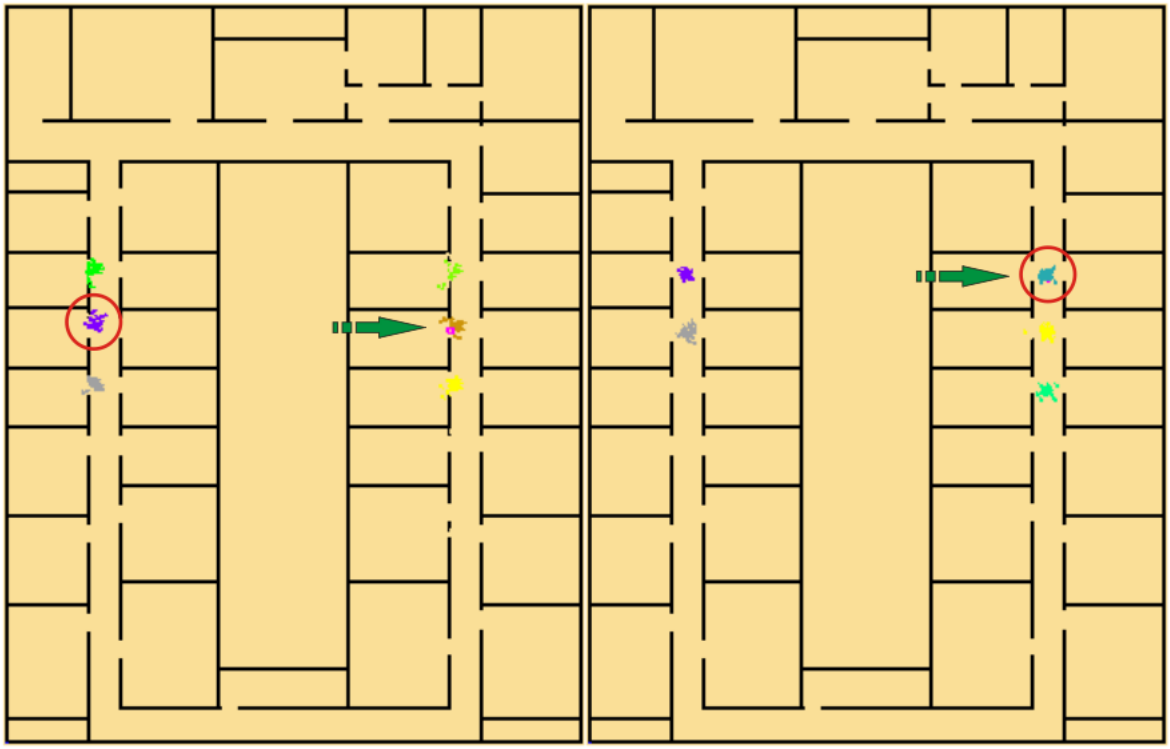

Figura 5.22: Elección de raza incorrecta y corrección posterior.

Como se puede observar en la figura 5.22 las razas si sitúan todas ellas en posiciones simétricas del mundo y coherentes con las observaciones actuales. La probabilidad de todas estas razas es mayor de 0,943, pero la que mayor probabilidad obtiene es una raza que no se sitúa en la posición del robot y que alcanza una probabilidad de 0,964. La diferencia es de sólo 0,02. Posteriormente el robot empieza a desplazarse por el pasillo y la raza que explota la posición real del robot mantiene e incluso aumenta su probabilidad (0,981), mientras que las demás disminuyen la suya. En este momento se tiene localizado al robot eficientemente pero en un primer momento el algoritmo determinaba como posición del robot una diferente a la real.

5.3.5. Comparativa filtro de partículas contra multimodal

En base a los experimentos realizados se ha comprobado que el filtro de partículas presenta dos problemas. El primero es que sólo es capaz de localizarse el 50 % de las veces en el departamental II por culpa de las simetrías. El segundo problema,

relacionado con el primero, que no resuelve es el del secuestro. Sólamente se recupera del secuestro el 50 % de las veces.

Por el contrario, se han hecho las mismas pruebas para el algoritmo multimodal y en este caso se ha localizado el 100 % de las veces y resuelve el problema del secuestro en todos los casos. En la figura 5.23 se muestran las estadísticas realizadas para determinar el comportamiento de cada algoritmo. En verde están sombreadas las celdillas cuando el resultado obtenido es el bueno y en rojo cuando el algoritmo falla. Las celdillas en amarillo se refieren a que el filtro de partículas no ha encontrado la posición correcta pero sí una simétrica.

	¿Se localiza?	¿Se localiza en posición simétrica?	¿Se recupera del secuestro?	¿Se pierde la posición del robot ?
FILTRO DE PARTÍCULAS				
1	SI	NO	SI	NO
2	NO	SI	NO	SI
3	SI	NO	SI	NO
4	NO	SI	SI	SI
5	SI	NO	NO	NO
6	NO	SI	NO	SI
7	SI	NO	NO	NO
8	SI	NO	SI	NO
9	NO	SI	NO	SI
10	NO	SI	SI	SI
ALGORITMO EVOLUTIVO MULTIMODAL				
1	SI	NO	SI	NO
2	SI	NO	SI	NO
3	SI	NO	SI	NO
4	SI	NO	SI	NO
5	SI	NO	SI	NO
6	SI	NO	SI	NO
7	SI	NO	SI	NO
8	SI	NO	SI	NO
9	SI	NO	SI	NO
10	SI	NO	SI	NO

Figura 5.23: Comparativa de los algoritmos localizadores.

A la vista de las pruebas, en ambos algoritmos una vez que se encuentra la posición correcta del robot no se pierde y la mantiene.

En conclusión el algoritmo multimodal es más robusto que el filtro de partículas. Además, la localización en mundos asimétricos se realiza más rápido que en mundos simétricos puesto que hay menos simétricas y por lo tanto menos posiciones del mundo confundibles con la real y en las que el algoritmo debe de decantarse.

5.4. Construcción de mapas y localización simultáneas

Hasta ahora se han construido los mapas con la localización resuelta y se ha localizado al robot con el mapa resuelto. En esta sección se explicarán los experimentos

y resultados preliminares sobre la unión de ambos comportamientos.

La primera prueba realizada como toma de contacto es construir el mapa a partir de la estimación de posición del robot del localizador, pero teniendo éste el mapa perfecto desde el principio. La construcción de mapas está totalmente integrada con el localizador. El localizador, sin embargo, no lo está porque no usa el mapa construido por *mapping*. Antes de empezar a construir el mapa, se realiza una etapa inicial para que el localizador localice al robot y no salte de un modo a otro. Una vez localizado se puede empezar a construir el mapa. En la figura 5.24(c) se puede ver como la desviación del mapa ruidoso de la figura 5.24(b) se ha corregido en gran medida.

Errores en la localización suponen errores en el mapa. Por éste motivo, a la hora de construir el mapa se va a generar ruido al incorporar segmentos desde una posición del robot errónea. Con la información del láser instantáneo se eliminan las incoherencias, pero la zona de detrás de las paredes no se puede detectar y no se eliminan. Esto se ve observando las diferencias entre el mapa construido desde la posición real en todo momento (figura 5.24(a)) y el construido en base a la estimación (figura 5.24(c)).

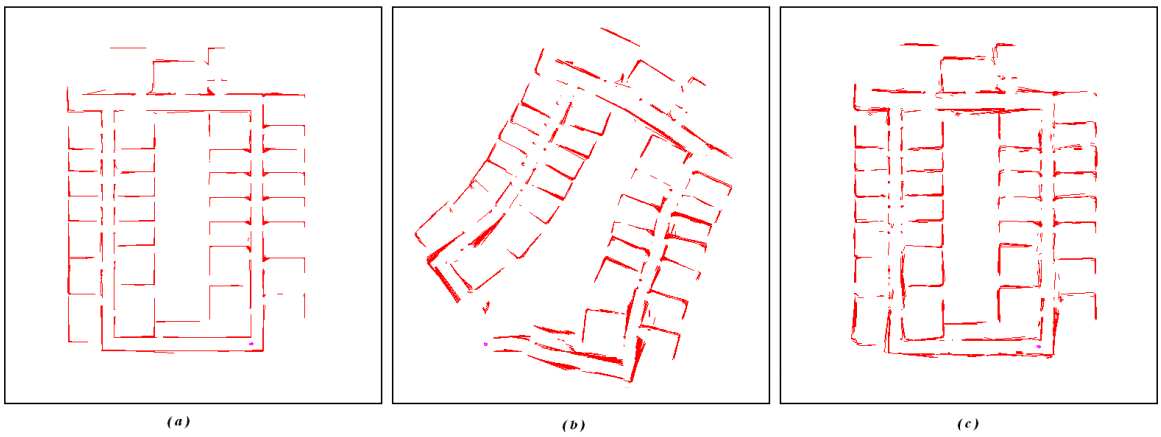

Figura 5.24: Mapa construido real (a) - ruidoso (b) - estimado (c).

Para poder realizar la construcción y localización simultáneas, el mapa que se usa para generar el láser teórico es el construido por el constructor de mapas, en vez de usar el mapa perfecto leído al principio de la ejecución del localizador (sección 4.4.1). De ésta forma tanto el constructor como el localizador están totalmente integrados. Al usar el mapa construido, en cada iteración del localizador se debe refrescar la rejilla que almacena el mapa, ya que el esquema *mapping* construye el mapa de forma continua.

Para calcular el láser teórico usamos dos listas que guarda de forma ordenada las celdillas ocupadas (opción *búsqueda* binaria de la sección 5.3.1). Se rellenan al principio de la ejecución del algoritmo y ya no se modifican. Ahora, como la rejilla se actualiza en cada iteración, también hay que actualizar estas listas en cada iteración. Esto supone el incremento de los tiempos del localizador. Como se necesita utilizar el algoritmo en tiempo real para hacer los experimentos de construcción y localización simultáneas, se ha utilizado la optimización *proyección* explicada en la sección 5.3.1. Calculando el láser usando la ventana de cómputo del rayo virtual se consiguen tiempos similares a los obtenidos sin actualizar ninguna lista. De todas formas, los tiempos aumentan algo (un par de segundos) debido a que se gasta tiempo en vaciar la rejilla y rellenarla en cada iteración.

A la vez que se va obteniendo el mapa obtenido se realiza la localización sobre este mapa de forma simultánea. El localizador usa el mapa construido por el constructor de mapas, y el constructor de mapas utiliza la estimación de posición suministrada por el localizador, la cual se actualiza de forma continua durante la iteración.

Al utilizar para la construcción del mapa la posición estimada por el algoritmo localizador, el mapa se construye mal puesto que al no estar el robot localizado desde el principio, la posición estimada por el algoritmo salta de una posición absoluta a otra. El mapa se va construyendo mal y como el láser teórico se calcula de este mapa mal construido, no puede entonces localizarse bien.

Para solucionar esto se ha dejado un periodo de guardería para que el algoritmo localice al robot. En este tiempo el mapa se va construyendo con la odometría que le llega del robot, que aunque tenga ruido, el mapa construido es más fiel al real.

Otro problema encontrado consiste en que el láser teórico calculado no cuadra con el real en las zonas cercanas al vacío. Esto se debe a que se calcula a partir de un mapa incompleto, ya que la rejilla solo se actualiza una vez por iteración. De esta forma, aunque haya una raza en la posición exacta del robot, cuando se calcula su salud, el láser teórico calculado de la raza que está en la posición del robot es distinto al láser real, puesto que el láser real se obtiene del mapa completo y el teórico se calcula de un mapa incompleto al que le falta la última parte del mapa construida. En la figura 5.25 se muestra como el mapa construido es incompleto en relación al mapa real cuando el robot avanza por un pasillo. Por este motivo el algoritmo localiza al robot en

posiciones simétricas parecidas a la real y que si se han construido ya en el mapa. En la figura 5.26 se ve como la raza elegida (la verde) se encuentra en posiciones simétricas ya construidas, debiéndose localizar en el final del pasillo que es donde está el robot avanzando y construyendo el mapa.

Por último, se ha comprobado que se debe ser menos exigente con el umbral para crear razas puesto que el mapa construido no es idéntico al real y por lo tanto el láser teórico calculado sobre la posición real del robot no va a ser tan parecido al láser real, como cuando se calcula sobre el mapa perfecto.

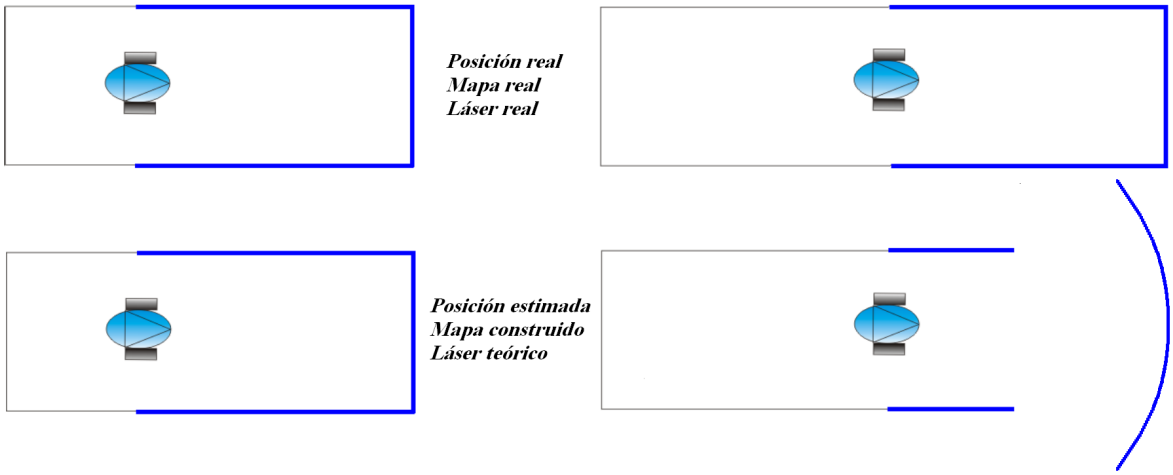

Figura 5.25: Mapa construido incompleto en comparación con el real.

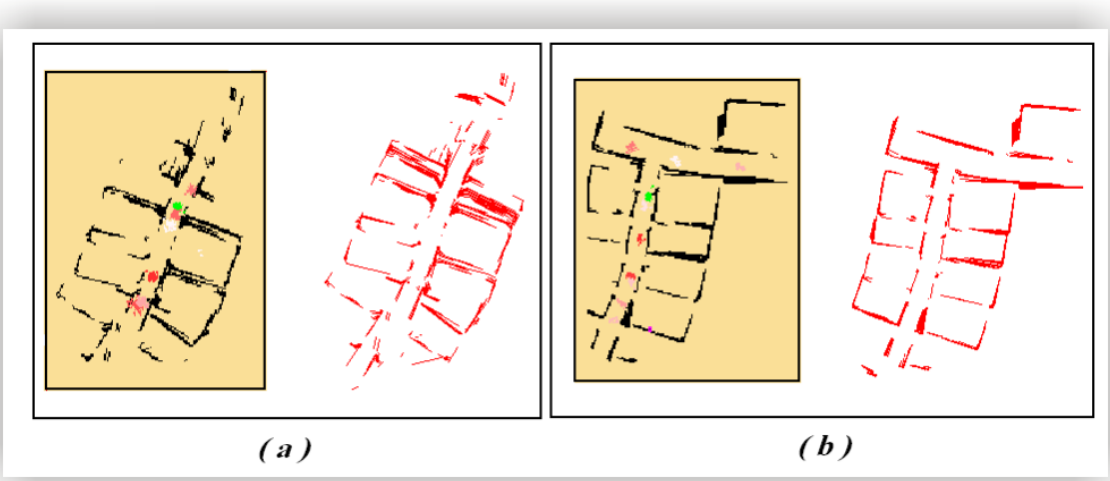

Figura 5.26: Localización en posiciones simétricas ya construidas.

Capítulo 6
Conclusiones y trabajos futuros

Después de explicar la implementación del sistema y pruebas realizadas con él, cerramos esta memoria comentando las conclusiones alcanzadas.

6.1. Conclusiones

En primer lugar hay que decir que se han cumplido los objetivos que se plantearon en el capítulo 2 (2.1). Para ello se han construido tres esquemas como se explica en la sección 4.1.

El primer objetivo consistía en lograr una navegación segura y reactiva del robot. La construcción del algoritmo de navegación se ha resuelto de forma satisfactoria mediante la implementación de un algoritmo VFF (campo de fuerzas virtuales) (sección 4.2). El esquema de navegación (4.4) se ha construido de tal forma que la velocidad del robot se obtiene de tres formas distintas: utilizando el teleoperador, usando VFF semiautónomo y VFF autónomo. Con VFF semiautónomo el objetivo se obtiene al pinchar en la ventana existente en la GUI. Con el VFF autónomo el objetivo lo obtiene directamente el algoritmo generándolo aleatoriamente delante del robot.

Para generar las fuerzas del algoritmo VFF se utiliza una memoria de puntos para almacenar la información sensorial en el tiempo, porque a vista de los experimentos, el comportamiento era mejor que usando el VFF sobre la información instantánea de *laser(t)* (sección 4.2.1).

El segundo objetivo era la construcción autónoma de mapas. La construcción del mapa se ha conseguido con la creación de una memoria de segmentos (sección 4.3.2) que almacena a lo largo del tiempo la información sensorial del mundo que rodea al robot. La memoria de segmentos, además de contener los segmentos almacenados durante la ejecución del constructor de mapas, etiqueta cada uno con su fecha de creación. Esto

se ha realizado así para poder eliminar, si se desea, los segmentos viejos.

Las funciones desarrolladas más destacables son la fusión de segmentos y la eliminación de segmentos incoherentes con la última información láser. Estas dos funciones son muy rápidas por utilizar geometría proyectiva plana y trabajar con sumas y restas a la vez de evitar divisiones por cero (sección 4.3.2 y anexo 6.2).

La construcción de mapas se puede realizar utilizando tres medidas de posiciones distintas: la odometría ideal, la odometría ruidosa y la estimada por el localizador. De esta forma se ha podido comprobar cuán diferentes son los mapas construidos con ruido y sin ruido en la odometría. A la vez, nos demuestra la ventaja del algoritmo localizador a la hora de construir el mapa comparando el mapa construido con la odometría ruidosa y el construido con la estimación de posición del localizador (figura 5.24).

El simulador no incorpora ruido de odometría. Por ello, la odometría ruidosa se ha creado artificialmente incorporando a la odometría ruido gaussiano. Se ha creado para simular las condiciones reales a la hora de construir el mapa (sección 4.4.5).

El tercer objetivo era conseguir de forma robusta la localización del robot en un mundo simulado extenso. A la vez, uno de los requisitos más importantes era la necesidad de tiempo real. Para ello se ha utilizado el filtro de partículas realizado en el proyecto de Redouane Kachach [Kachach, 2005] mejorando el cálculo del láser teórico y la obtención de la probabilidad de una partícula. Además, se ha diseñado y desarrollado un nuevo algoritmo localizador (algoritmo evolutivo multimodal) (sección 4.4).

La nueva forma de calcular el láser teórico utiliza dos listas de celdillas ocupadas. Estás listas se recorren con un orden predeterminado, y se accede a ellas usando búsqueda binaria. De está forma se consiguen dos iteraciones por segundo. Con la forma implementada en el proyecto de Redouane se llegaban a tiempos por encima de 15 segundos (secciones 4.4.3 y 5.3.1).

También se ha cambiado la forma de obtener la probabilidad de las partículas. Se han probado 6 formas distintas, probándolas en entornos simétricos (en el departamental II) y asimétricos. La forma elegida determina en un primer lugar la distancia entre el láser real y el teórico mediante el sumatorio de las diferencias de sus medidas. Posteriormente eleva ésta distancia a una potencia calculada

experimentalmente (sección 5.3.2). De ésta forma se ha mejorado la convergencia de las partículas, y por lo tanto, la localización.

También se ha realizado la localización con el algoritmo evolutivo multimodal. Éste se basa en dos poblaciones: exploradores y explotadores. La labor de los exploradores es rastrear el mundo para encontrar zonas de alta probabilidad de que el robot se encuentre en ella. La función primordial de los explotadores es rebuscar al máximo una posición encontrada por los exploradores y sus alrededores para determinar si en esa posición se sitúa el robot. La principal ventaja que supone éste algoritmo es su carácter multimodal, lo que le permite mantener varios modos.

Se han comparado los resultados de los dos algoritmos. El algoritmo del filtro de partículas no logra la localización del robot de forma robusta en ambientes simétricos (5.3.3). Por la existencia de las simetrías, en el 50 % de las veces el robot se localiza en posiciones simétricas (convergencia prematura). De igual forma, y por el mismo motivo, no es capaz de recuperarse del secuestro. Por el contrario, con el algoritmo multimodal se consigue la localización del robot de forma eficiente. Esto se debe al carácter multimodal. De ésta forma, aunque el algoritmo se decante por otro modo, se mantiene el modo bueno, al que se vuelve en cuanto se rompen las simetrías. En el filtro de partículas esto no pasa y cuando las partículas convergen a una zona simétrica no logra recuperarse.

Finalmente, se han realizado pruebas preliminares de SLAM para realizar una primera toma de contacto en la integración de la construcción de mapas y localización simultáneas. El comportamiento que se ha obtenido ha sido malo.

El primer problema encontrado consiste en una localización inicial inestable al saltar el localizador de un modo a otro. De ésta forma el mapa se construye mal y no se localiza al robot de forma robusta. Éste problema se ha resuelto dejando un periodo de guardería donde se construye el mapa a partir de la odometría ruidosa. De ésta forma el mapa construido es más parecido al real.

El segundo problema aparece al generar el láser teórico. El láser teórico de las zonas cercanas al vacío no se parece al real porque se calcula a partir de un mapa incompleto (figura 5.25). Por éste motivo el localizador cree que el robot se encuentra en zonas simétricas ya construidas (figura 5.26).

En conclusión, se han logrado todos los objetivos planteados de forma eficiente, consiguiendo una navegación reactiva segura del robot, una construcción de los mapas de forma eficiente y una localización robusta gracias al algoritmo multimodal.

6.2. Trabajos futuros

Sin lugar a dudas, los trabajos futuros más interesantes a los que conduce este proyecto son la implementación de estas técnicas de construcción y localización en el robot real y profundizar más en los experimentos de SLAM. De esta forma, el robot podrá explorar y construir el mapa de zonas no conocidas y estar localizado en todo momento a pesar del ruido existente en los sensores.

Otro posible trabajo es probar como se comportan estos algoritmos ante obstáculos dinámicos, como puedan ser personas.

También se podrá agilizar más los tiempos conseguidos (por ejemplo, calcular el láser teórico desde la memoria de segmentos y no desde la rejilla) o mejorar el problema de los obstáculos tras las paredes del constructor de mapas (5.4).

Anexo.

Cálculos matemáticos con geometría proyectiva planar.

En geometría proyectiva planar se representan los puntos P y las rectas R con tres coordenadas:

$$P = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$R = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Para saber si un punto pertenece a una recta el producto escalar de estos debe dar 0:

$$ax + by + c = 0 \tag{6.1}$$

$$\vec{r} \cdot \vec{p} = 0 \tag{6.2}$$

El punto de corte de dos rectas se calcula como el producto vectorial de dos rectas: $((r_a, r_b, r_c)$ y $(s_a, s_b, s_c))$.

$$\vec{r} \times \vec{s} = \vec{p} \tag{6.3}$$

Un punto P pertenece a un segmento (A,B) si el producto escalar de los vectores formados por los extremos y el punto de corte es menor o igual que 0 (ver figura 6.1):

$$\vec{AP} \cdot \vec{BP} \leq 0 \tag{6.4}$$

Para comprobar si dos segmentos $(A,B)(D,E)$ se cortan en el punto C, el punto de corte de las rectas que los contienen debe pertenecer a ambos. En la figura 6.1a se observan dos segmento que se cortan y en la figura 6.1b dos que no se cortan.

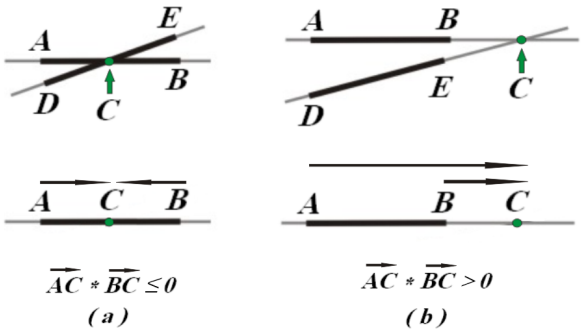

Figura 6.1: Punto de intersección de dos segmentos.

Para calcular la proyección P de *P3* sobre la recta definida por *P1* y *P2* nos basamos en la ecuación de la recta definida por dos puntos:

$$\vec{P} = \vec{P1} + \lambda * (\vec{P2} - \vec{P1}) \tag{6.5}$$

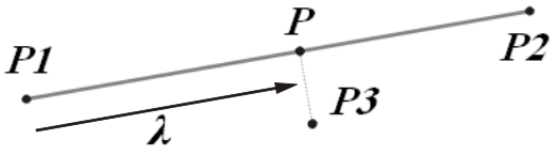

Figura 6.2: Mínima distancia punto-recta.

Como los vectores $P\vec{3}P$ y $P2\vec{P}1$ son perpendiculares, su producto escalar debe ser 0:

$$(P3 \vec{-} P) \cdot (P2 \vec{-} P1) = 0 \tag{6.6}$$

Con lo cual sustituyendo en la ecuación 6.6 la ecuación 6.5 obtenemos:

$$(P3 - \vec{P1} - \vec{\lambda} * (\vec{P2} - \vec{P1})) \cdot (P2 \vec{-} P1) = 0 \tag{6.7}$$

Ahora ya podemos despejar λ.

$$\lambda = \frac{(x_3 - x_1) * (x_2 - x_1) + (y_3 - y_1) * (y_2 - y_1) + (z_3 - z_1) * (z_2 - z_1)}{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{6.8}$$

Sustituyendo ésta λ en la ecuación 6.5 obtenemos la proyección de *P3* sobre ésta recta.

Bibliografía

[Borenstein, 1989] Y. Koren J. Borenstein. Real-time obstacle avoidance for fast mobile robot. *IEEE Journal of Robotics and Automation*, 1989.

[Cañas Plaza *et al.*, 2006] José María Cañas Plaza, Antonio Pineda, Pablo Barrera, y David Lobato. Programación de robots con la plataforma jde.c. *URJC*, 2006.

[Cañas Plaza y García, 2002] José María Cañas Plaza y Lía García. Construcción de mapas dinámicos: comparativa. *URJC*, 2002.

[Isado, 2005] Raúl Isado. Navegación global de un robot usando el método del gradiente. *Proyecto fin de carrera, URJC*, 2005.

[Kachach, 2005] Redouane Kachach. Localización del robot pioneer basada en láser. *Proyecto fin de carrera, URJC*, 2005.

[Lobato, 2003] David Lobato. Navegación local con ventana dinámica para un robot móvil. *Proyecto fin de carrera, URJC*, 2003.

[López, 2005a] Alberto López. Localización de un robot con visión local. *Proyecto fin de carrera, URJC*, 2005.

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

[Martínez, 2007] Iván García Martínez. Reconstrucción 3d visual con algoritmos evolutivos. *Proyecto fin de carrera, URJC*, 2007.

[Palacios, 2006] Ricardo Palacios. Representación rica de la escena 3d alrededor de un robot móvil. *Proyecto fin de carrera, URJC*, 2006.

[Plaza, 2003] José María Cañas Plaza. *Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo*. PhD thesis, Universidad Politécnica de Madrid, 2003.