
# INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela Técnica Superior de Ingeniería Informática

Curso académico 2009-2010

**Proyecto Fin de Carrera**

# Algoritmo evolutivo para la autolocalización de un robot móvil con láser y visión

**Tutor**: José María Cañas Plaza
**Autor**: Darío Rodríguez de Diego

*A mi familia y a Laura.*

*Gracias por vuestro apoyo.*

# Agradecimientos

En primer lugar quiero dar las gracias a todos los miembros del Grupo de Robótica de la Universidad Rey Juan Carlos por su apoyo y por todo lo que he aprendido en este tiempo que formado parte de él. En especial a Julio, Edu, Sara, Pablo, Luismi y Gonzalo por la ayuda y por los buenos ratos que hemos pasado.

Una mención especial a mí tutor José María Cañas por todo el trabajo y por su ayuda prestada durante el desarrollo de este proyecto.

También quisiera dar las gracias a mis compañeros de carrera Jose, David, Isaac, Julian, Carlos, Antonio, Toni, Sara, David y Gema por todo lo que hemos compartido durante estos años tanto fuera como dentro de clase.

Además quiero dar las gracias a mis amigos de toda la vida que siempre están ahí y la los que tanto debo Víctor, Rodri, Javi, Carlos, Diego, Eva, Dani, Fan, Cris y a otros muchos que también se lo merecen.

Seguro que se me olvida alguien, sólo espero que se de por aludido y que entienda estas páginas como algo suyo.

Muchas gracias a todos!

# Resumen

El problema de la autolocalización es clásico en el ámbito de la robótica autónoma ya que es vital conocer la posición del robot en todo momento para que los comportamientos del robot puedan escoger las mejores acciones a realizar en cada momento. Existen muchas formas de conseguir que el robot se localice dentro de un entorno, típicamente se han usado sensores específicos (GPS, inerciales, etc) o sensores no específicos (láser, sónares, cámaras, etc) y un mapa del entorno sobre el que se trabaja para poder estimar la posición del robot.

En este proyecto se usan dos tipos de sensores externos, uno de distancia (láser) y otro de visión (cámara), para obtener las mejores características de cada uno. Con el láser conseguiremos información morfológica del entorno y con la cámara información visual. Además usaremos un mapa con la información visual y morfológica del entorno. También contaremos con sensores odométricos llamados *encoders* para capturar los movimientos del robot. Con nuestro algoritmo devolvemos la posición y orientación del robot (x,y,$\theta$) dentro del entorno de acción. El algoritmo que hemos usado es un algoritmo con muestreo llamado *algoritmo genético multimodal*. Este algoritmo se divide en dos partes, en la primera se explora el entorno y en las segunda se realiza una exploración local en las zonas donde es más probable que se encuentre situado el robot. En todo el proceso de ejecución del algoritmo están muy presentes un *Modelo de observación*, que extrae información en forma de probabilidad de los datos sensoriales y del mapa, y un *Modelo de movimiento* que nos permite replicar los movimientos que realiza el robot en el algoritmo.

Se han realizado experimentos para validar cada una de las decisiones que se han tomado hasta llegar a la solución final obteniendo un algoritmo robusto y ágil. Todos los experimentos han sido realizados con el simulador *Gazebo* simulando un mundo parecido al Departamental II de la UJRC.

# Índice general

# Índice de figuras

# Capítulo 1
# Introducción

En el campo de la robótica autónoma es de vital importancia para la generación de comportamientos del robot el conocimiento de su ubicación dentro del entorno. A este problema, que consiste en estimar la posición y orientación del robot dentro de su entorno, se le denomina autolocalización y es uno de los problemas clásicos de la robótica móvil. En este proyecto fin de carrera hemos abordado la autolocalización de un robot que se mueve dentro de un entorno de oficinas con una cámara y un sensor láser. En este primer capítulo de introducción se ofrece una contextualización sobre la robótica y más concretamente sobre la robótica móvil y el problema a tratar en este proyecto. También se presentan algunas de las técnicas típicas de autolocalización.

## 1.1. Robótica

Se define Robótica como la rama de la tecnología que se dedica al estudio y diseño de máquinas que deben realizar tareas propias de los humanos o que requieren un uso de la inteligencia. A estás máquinas se las conoce como *robots*. La palabra Robot se deriva de la palabra de origen checoslovaco *robota*, que quiere decir *siervo*. Dicha palabra apareció por primera vez en la literatura en la obra R.U.R (1921) (Robot Universalis de Rossum), de Karel Capek.

Los robots se componen esencialmente de tres tipos de dispositivos: sensores, procesadores y actuadores. En un robot los sensores son los encargados de recoger la información del entorno. En este grupo se situarían: el láser, el sonar o las cámaras. Estos dispositivos equivaldrían a nuestros sentidos humanos. Por otro lado se encuentran los procesadores, encargados de analizar los datos que le son suministrados por los sensores, también son los encargados de elaborar una respuesta a estos datos y enviar la acción que deba llevarse a cabo a los actuadores, son como nuestro cerebro. Por último los actuadores, principalmente motores eléctricos, se encargan de interactuar con el entorno del mismo modo que lo hacen nuestros músculos.

El desarrollo tecnológico ha hecho posible la construcción de diferentes modelos robóticos capaces de ayudar o sustituir a las personas en la realización de tareas peligrosas, repetitivas, difíciles o que requieren precisión. En sectores como la industria del automóvil la utilización de brazos robóticos que sustituyen al hombre en la cadenas de montaje ha sido una constante desde que en 1967 se instalaran los primeros brazos robóticos en una factoría de General Motors. Hoy en día todas las empresas de la

industria del automóvil utilizan esta tecnología en sus procesos de fabricación. En este ámbito los robots se encargan de labores de soldadura, pintado, ensamblaje de piezas, etc. En la figura 1.1 se ve una cadena de montaje de automóviles en los que trabajan múltiples brazos robóticos.

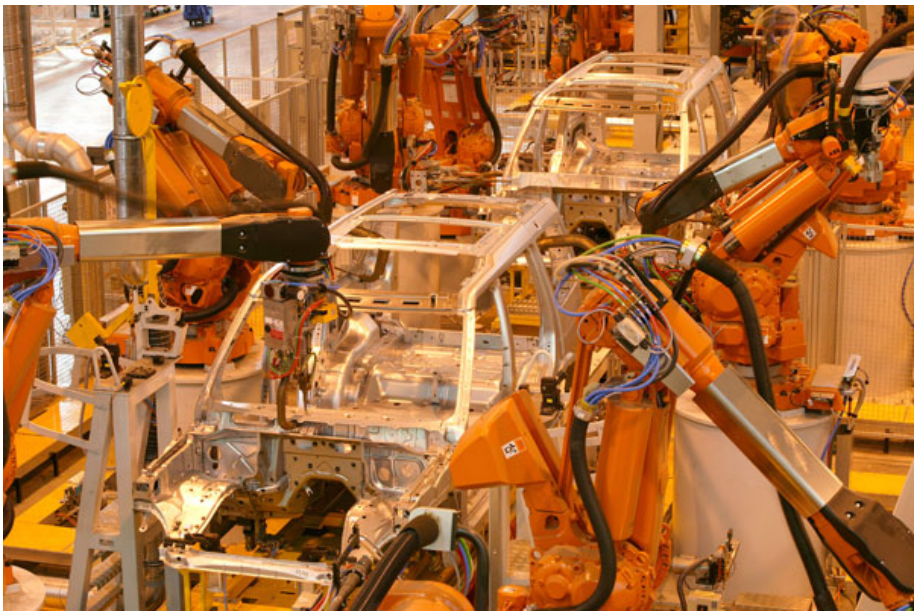


Figura 1.1: Cadena de montaje de automóviles

La combinación de la ingeniería robótica con las distintas técnicas de la visión artificial genera componentes de gran interés para el ser humano. En la industria alimenticia el sistema desarrollado por ABB para procesos industriales, aumenta la productividad de las empresas. El brazo robótico 1.1 analiza los productos mediante su cámara y los ordenarlos por lotes, para su posterior empaquetado.

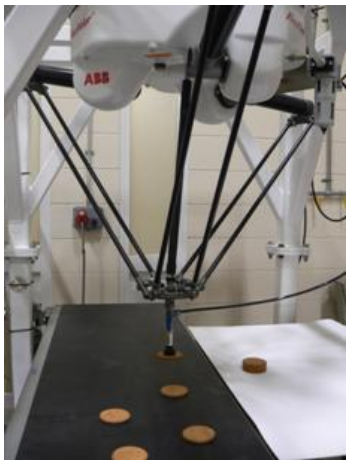



Figura 1.2: Robot de ABB para la industria de la alimentación

En nuestra vida cotidiana los robots están tomando un papel de mayor relevancia, desde hace unos años la empresa norteamericana *iRobot* comercializa el robot *Roomba*

(figura 1.1) para la ayuda doméstica. Este robot es capaz de aspirar una habitación de forma autónoma. *Roomba* es el primer robot autónomo para la ayuda doméstica comercializado para el gran público, y sin duda ha cosechado un reconocido éxito de ventas con más de 2 millones de unidades vendidas.

Figura 1.3: Robot doméstico *Roomba*

Otro éxito comercial en lo que a robótica doméstica se refiere se ha conseguido con robots enfocados al entretenimiento. Los kit robóticos *Lego Mindstorm* comercializados por la empresa de juguetes danesa *Lego* cuentan con legiones de seguidores. Estos kits permiten la construcción y programación de robots como el que aparece en la figura 1.1.

Figura 1.4: Robot *Lego Mindstorm*

En la actualidad existe una fuerte tendencia en ir aún más lejos. En este sentido, las grandes corporaciones japonesas como *Toyota* u *Honda* trabajan en el desarrollo de robots humanoides autónomos, con la intención de solventar un problema acuciante en la sociedad japonesa, como es el rápido envejecimiento de la población y su falta de cuidado en el ámbito doméstico. El propio Bill Gates, responsable de la llegada de los ordenadores personales a los hogares de todo el mundo, afirmaba en el artículo *A robot in Every Home*, para la revista *Scientific American* [Gates, 2007], que la situación de

la robótica actual es muy similar a la del ordenador personal hace ya varios años, y que él confía en que a medio-largo plazo empezaremos a tener robots en nuestras casas, y a verlos progresivamente en las calles de nuestras ciudades.


Figura 1.5: (a) Robot *Asimo* desarrollado por Honda y (b) prototipo de Toyota

Con el objetivo de favorecer la investigación en el campo de la robótica autónoma, diferentes empresas han desarrollado sus propios modelos de robot humanoides, el robot *Asimo* (figura 1.5), diseñado por Honda, y su homólogo el prototipo diseñado por Toyota (figura 1.5), cuentan con una gran agilidad motriz y utilizan sus cámaras como sensores principales, siendo el modelo de Honda capaz de reconocer y aprender nuevos objetos.

Otra de las aplicaciones para las que se han desarrollado robots es para la exploración espacial. La NASA ha enviado varias sondas espaciales para realizar tareas de exploración de mundos lejanos como precedente a una posterior colonización de los planetas. Las sondas *Spirit* (figura 1.6) y *Opportunity*, enviadas por la NASA a Marte hace más de cinco años, han ofrecido gran cantidad de imágenes del planeta rojo y a día de hoy siguen plenamente operativas como estaciones geológicas y de exploración.

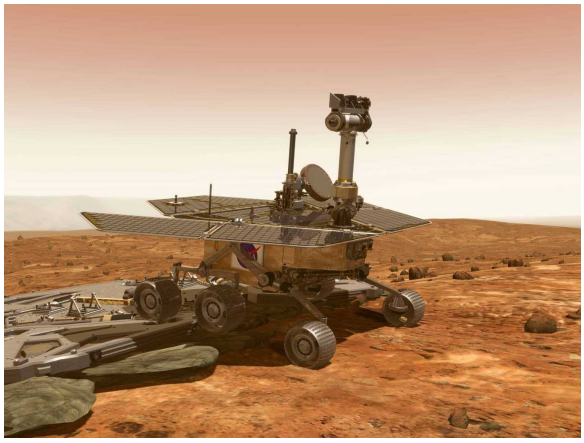


Figura 1.6: La sonda espacial *Spirit*.

Dentro del contexto de la exploración existen los robots autónomos que a diferencia de los robots teleoperados son capaces de tomar sus propias decisiones. Un buen ejemplo de aplicaciones donde son protagonistas los robots móviles autónomos son las competiciones Grand Challenge [1] y Urban Challenge [2], que se pueden ver en la figura 1.7.


(a) (b)

Figura 1.7: (a) Competición Grand Challenge (b) Competición Urban Challenge

Las dos pruebas nacieron como un reto por parte de DARPA [3] (Agencia de Investigación de Proyectos Avanzados de Defensa de EE.UU) para utilizar coches no tripulados en misiones de transporte. En este punto cobra mucha importancia la localización ya que el robot debe saber en todo momento su situación dentro del recinto para poder ejecutar las acciones oportunas.

Otra aplicación es los *robot guía* ya que deben conocer su posición para poder llevar personas de un lugar a otro. En este campo hay varios trabajos dignos de mención. El robot Minerva [4] creado por la Carnegie Mellon University [5], figura 1.8 (a), diseñado para educar y guiar a la gente a través de un museo explicando qué es lo que están viendo durante el trayecto. Fue instalado en 1998 en el Museo Nacional de historia americana Smithsonian y durante dos semanas de operación el robot recorrió más de 44 km interactuando con miles de personas. Otro ejemplo es el robot Xavier [6], figura 1.8 (b), diseñado por alumnos de la Carnegie Mellon University como un robot para operar durante largos periodos de tiempo de manera autónoma y que es capaz de aprender y adaptarse al entorno e interactuar con las personas, donde la localización es de vital importancia.

[1] http://www.darpagrandchallenge.com
[2] http://www.urbanchallenge.com
[3] www.darpa.mil
[4] http://www.cs.cmu.edu/~minerva/
[5] http://www.ri.cmu.edu/
[6] http://www.cs.cmu.edu/~Xavier/

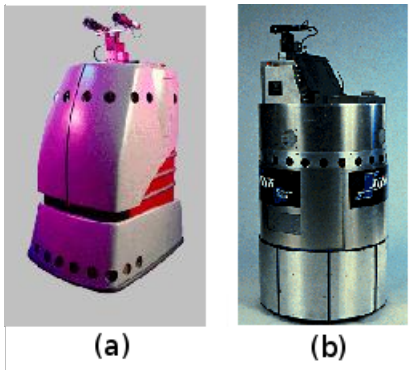



Figura 1.8: (a) Robot Minerva (b) Robot Xavier

En la misma linea de vistosos robots prototipo y mucho más actuales hay que destacar la Robocup[7](figura 1.9) una competición en la que diferentes tipos de robots compiten jugando partidos de fútbol y en la que el grupo de robótica de esta universidad participa con el equipo S$Pi$Team [8]. En esta competición la localización es muy importante ya que aporta al robot su posición en el campo y el conocimiento necesario para escoger la estrategia a seguir ya sea chutando a portería o defendiendo su campo.

Figura 1.9: Robots Nao compitiendo en la RoboCup

## 1.2. Localización de robots móviles

La localización de robot móviles, como ya hemos dicho, es un problema clásico y de vital importancia al que debe enfrentarse la robótica móvil autónoma. La localización es muy importante porque los comportamientos de los robot móviles dependen de la posición en la que se encuentra el robot para tomar buenas decisiones de movimiento.

Al tratar el problema de la localización de robots móviles nos podemos enfrentar a tres tipo de problemas distintos. El primero trata de localizar al robot dada una posición

[7] `http://www.robocup.org/`
[8] `http://www.teamchaos.es/index.php/TeamChaos`

inicial conocida, la solución consiste en ir estimando a partir de la odometría del robot la posición en cada momento compensando los errores odométricos (autolocalización incremental). El segundo problema es localizar al robot desconociendo la posición inicial (autolocalización absoluta). A este problema nos enfrentaremos en este proyecto. Por último un problema muy habitual es localizar la posición de varios robots que forman un grupo, este problema se pueden dar dependencias estadísticas entre las estimaciones de cada uno de los robots si estos son capaces de detectar a los miembros del grupo lo que hace que este problema también sea muy interesante, esto se conoce como autolocalización cooperativa.

### 1.2.1. Técnicas clásicas

Para poder localizarse el robot suele necesitar una representación del entorno en el que se está moviendo, dicha representación puede venir dada en forma de un mapa estático del entorno o bien se puede reconstruir dicho mapa por medio de sensores con la técnica SLAM(*Simultaneous Localization and Mapping*).

- Localización con sensores específicos: existen muchos tipos de sensores y muchos de ellos son de gran utilidad en tareas de localización, el ejemplo más simple son los encoders (figura 1.10) que proporcionan información sobre los movimientos del robot. Los encoders adolecen de un grave problema y es que son muy imprecisos debido a que los errores en la odometría son muy frecuentes por deslizamientos, holguras, falta de precisión, etc. y es necesario corregir estos errores odométricos ya que hace más precisa la información de la localización. También hay que tener en cuenta que los errores odométricos son acumulativos, esta es su principal desventaja, y por lo tanto si no se hace una corrección periódica de ellos podríamos obtener una localización pésima. Existen dos tipos de errores odométricos: *sistemáticos* que dependen de las características del robot y sus sensores y *no sistemáticos* que son impredecibles y son debidos a agentes a externos al robot.

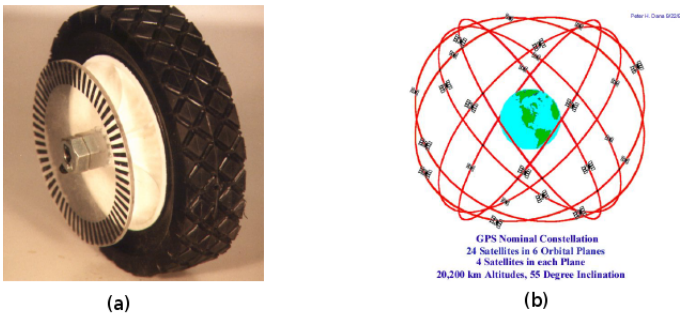



Figura 1.10: Encoders (a), GPS (b)

Como alternativa a los encoders tenemos los giroscopios o sensores inerciales, figura 1.11, que son unos sensores que son capaces de detectar los incrementos en la aceleración y con ello se pueden extraer datos de incrementos de movimiento.

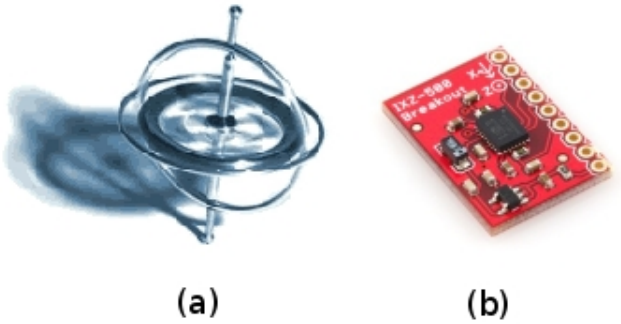



Figura 1.11: (a) peonza giroscópica, (b) giroscopio electrónico

El último ejemplo lo tenemos más cerca de lo que pueda parecer y es el GPS 1.10 que en entornos exteriores puede ofrecer una localización lo suficientemente precisa como para usarse en un robot. En interiores, sin embargo, no es un sistema muy útil ya que no se puede establecer una comunicación limpia con los satélites necesarios para usar este servicio.

- Localización con *balizas*: esta técnica permite localizar al robot en un entorno restringido mediante el emplazamiento específico de un determinado número de balizas con posiciones conocidas. Para estimar la posición se puede utilizar la *triangulación* basándose en el ángulo con que se ven las balizas y la *trilateración* basándose en la distancia a las balizas.

- El *scan matching*: es otra técnica que utiliza mapas locales para compararlos con lecturas sensoriales alineando estas lecturas con los diferentes mapas en posiciones cercanas a la que creemos que está el robot, necesitando para ello una estimación de la posición inicial del robot representada como una distribución gausiana que se va actualizando con las lecturas sensoriales. Esta técnica entra dentro de las técnicas de autolocalización incremental

- *Localización probabilística* [Thrun, 2000]: es adecuada para interiores ya que incorpora incertidumbre de acciones y observaciones que se acoplan a la incertidumbre que muestran los sensores. Este tipo de localización consiste en determinar la probabilidad de que el robot se encuentre en una determinada posición a través de sus lecturas sensoriales y movimientos a lo largo del tiempo. A cada posible posición se le asocia una probabilidad reflejando la verosimilitud de ser la posición actual del robot. Esta probabilidad se va actualizando con la incorporación de nuevas lecturas y movimientos del robot. Estas técnicas nos permiten localizar al robot aún desconociendo su posición inicial permitiendo representar situaciones ambiguas que se irán desambiguando posteriormente.

  Dos de las técnicas probabilísticas más importantes y que se utilizan para resolver la localización en este proyecto son: la probabilística *discretizada*, que implica mucho tiempo de cómputo debido a que almacena y actualiza la distribución de probabilidades para todas las posibles posiciones y *con muestreo*, que se utiliza para agilizar el tiempo de cómputo y hacer de la localización un proceso escalable

a grandes entornos. La intuición de la localización probabilística se puede apreciar en la figura 1.12.

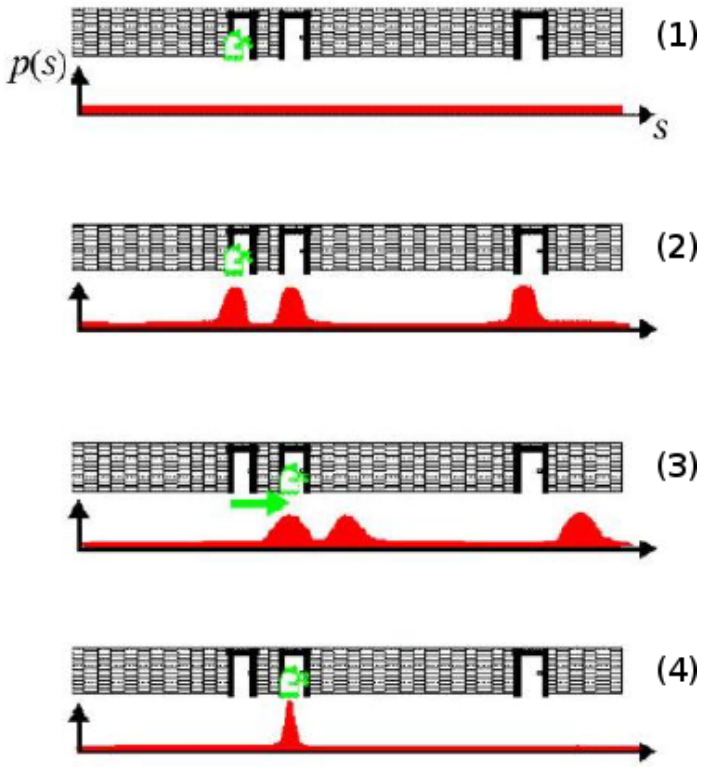



Figura 1.12: Localización probabilística

En un primer paso, se asume un espacio unidimensional en el que sólo se puede desplazar horizontalmente y se desconoce la posición inicial. El estado de incertidumbre se representa como una distribución uniforme sobre todas las posibles posiciones. En el siguiente paso, suponemos que el robot detecta mediante sus sensores que se encuentra enfrente de una puerta. Esto se refleja mediante el aumento de la verosimilitud en las zonas donde están las puertas y la disminución en zonas donde no hay puertas. La información disponible hasta el momento actual es insuficiente para poder determinar la posición del robot. Si el robot se desplaza, la distribución se desplaza de forma análoga como se refleja en el tercer paso. En el cuarto y último paso el robot ha detectado de nuevo, a través de sus sensores, que se encuentra frente a una puerta por ello aumenta la verosimilitud en esa posición que sumada a la acumulación del tercer paso, nos lleva a la conclusión de que es muy probable que el robot se encuentre en la segunda puerta.

Las técnicas más usadas para la localización probabilística son las muestreadas ya que son rápidas y ofrecen buenos resultados, entre ellas se encuentran los *métodos de Monte Carlo*. Las técnicas de Monte Carlo son un conjunto de técnicas desarrolladas inicialmente en el campo de la física entre 1945 y 1955 y son usados en muchas ramas de la ciencia. En el ámbito de la robótica se usa el algoritmo de condensación conocido como el *filtro de partículas*. Este algoritmo permite

muestrear la verosimilitud de las localizaciones posibles del robot mediante un número limitado de partículas que convergerán a una determinada posición a partir de sucesivos movimientos y observaciones del robot a lo largo del tiempo. La principal desventaja de este tipo de algoritmos es que sólo manejan un grupo de partículas y por ello son muy sensibles ante entornos de alta simetría, errando con mucha facilidad en el cálculo de la posición.

Para este tipo de técnicas se suelen usar sensores no específicos como el láser o la cámara 1.13 que por si mismos no aportan información de localización, pero que podemos conseguir a través de una inferencia normalmente ayudados de un mapa del entorno.


Figura 1.13: Ejemplo de cámara(a) y láser (b)

Existen tres técnicas probabilísticas con muestreo muy representativas:

- *Filtros de Kalman*: es una técnica que trata de estimar recursiva y periódicamente la posición de mínima varianza fusionando información parcial e indirecta sobre localización. Su principal limitación es que es una técnica unimodal y exclusivamente gausiana. Esta técnica no soporta bien el ruido de lecturas sensoriales ni entornos dinámicos y no es capaz de manejar múltiples hipótesis.
- *Discretizadas*: esta técnica trata de estimar todas las probabilidades del entorno de manera discreta y a medida que pasa el tiempo y se acumulan las probabilidades, estimar un única posición correcta. Estas técnicas son muy precisas, pero se vuelven impracticables para entornos demasiado grandes porque requieren mucha carga de proceso.
- *Muestreada*: es muy parecida a la técnica de discretización, pero sólo mantiene la probabilidad de ciertos lugares del entorno y a medida que pasa el tiempo converge a una sola posición. Dentro de estas técnicas están los algoritmos evolutivos y las técnicas de Monte Carlo.

## 1.3. Autolocalización probabilística y evolutiva en la URJC

Otra técnica de localización que no es probabilística pero es muy parecida son los algoritmos multimodales entre los que se encuentra el *algoritmo genético* que es un desarrollo propio del departamento de robótica de la URJC. Un algoritmo genético es un método de búsqueda dirigida basada en probabilidad manteniendo la condición de elitismo, es decir, el algoritmo siempre usa en la iteración t al mejor elemento de la población en t-1 sin hacerle ningún cambio. Gracias al elitismo y al paso de las iteraciones el algoritmo es capaz de converger hacia una posición con alta probabilidad. La ventaja de este tipo de algoritmos frente a los *algoritmos de Monte Carlo* es que estos son capaces de mantener varias estimaciones a la vez por ello son menos sensibles ante entornos altamente simétricos, por ello usaremos un *algoritmo genético* en este proyecto.

La resolución de la localización mediante estas técnicas nace de las motivaciones del grupo de robótica de la Universidad Rey Juan Carlos por generar comportamientos artificiales en robots, tales como un guía de personas o un robot capaz de jugar al futbol. En todos estos comportamientos la localización juega un papel muy importante en el funcionamiento de los mismos ya que sin localización no serían posibles.

Debido a que la localización no es un problema trivial ha sido abordado desde distintas perspectivas anteriormente en el Grupo de Robótica. Dando una perspectiva histórica estas han sido las diferentes aproximaciones al problema de la localización:

- María Ángeles Crespo en su PFC [Crespo, 2003] realizó un algoritmo de localización con ayuda de una cámara y el filtro de partículas de Monte Carlo sobre el entorno de la RoboCup
- Alberto López en su PFC [López, 2005] realizó un algoritmo de localización basado en información visual y con el método probabilístico de Monte Carlo.
- Redouane Kachach en su PFC [Kachach, 2005] abordo el este problema usando el algoritmo de filtro de partículas de Monte Carlo, pero esta vez obteniendo la información de un sensor láser.
- Ángel Cortés en su PFC [Cortés, 2007] aplicó el *algoritmo genético* desarrollado en el grupo de robótica al problema de la localización en entornos de alta simetría apoyándose en sensores láser.
- Julio Vega en su PFC [Vega, 2008] abordó el problema de la localización usando el *algoritmo genético*, una cámara y balizas situadas en el techo para estimar la posición del robot en una habitación.
- José Manuel Domínguez en su PFC [Domínguez, 2009] usó el filtro de partículas de Monte Carlo y la información de un cámara para estimar la posición del robot en entornos de alta simetría.

- Eduardo Perdices en su PFC [Perdices, 2009] y en un TFM [Perdices, 2010] abordó el problema de la localización del robot en el entorno de la RoboCup con varios algoritmos y la información de una cámara.

Como se puede ver en los proyectos anteriores se han usado los sensores no específicos de distancia (láser) y de visión (cámara) por separado. Además se han usado diversos algoritmos para realizar la tarea. En este proyecto se da un paso más y se combinan los dos sensores no específicos, láser y cámara, que se unen al *algoritmo genético* ya que ha sido el que ha dado mejores resultados históricamente.

En los siguientes capítulos de esta memoria primero se van a tratar los objetivos principales, el entorno y la plataforma de desarrollo donde se realizará el proyecto, el método de *Mallas de probabilidad*, el uso del *Algoritmo genético* y por último las conclusiones y las lineas futuras que deja abiertas este proyecto de fin de carrera.

# Capítulo 2
# Objetivos

Una vez expuesto un contexto general en el capítulo 1 vamos a fijar los objetivos concretos que llevan a la realización de este proyecto fin de carrera y los requisitos que han condicionado su desarrollo.

## 2.1. Descripción del problema

El objetivo principal de este proyecto es el desarrollo de un algoritmo que permita al robot localizarse en su entorno de navegación. El robot apoyándose en sus sensores deberá localizarse en un entorno altamente simétrico como es el Departamental II. Para realizar todo esto contamos con un robot Pioneer dotado con los sensores oportunos y un mapa del entorno a escala representado en escala de grises que se puede ver en la figura 2.1

El objetivo principal lo hemos dividido en subobjetivos que desarrollaremos a lo largo del proyecto.

- *Procesar la información de posición de los sensores no específicos*, se estudiarán diferentes métodos de extraer la información tanto del láser como de la cámara para obtener datos instantáneos que posteriormente se compararán con el mapa del entorno que posee el robot.

- *Implementación de mallas de probabilidad*, nos permitirá validar los métodos escogidos para procesar la información de los sensores y escoger el que mejores resultados nos aporte, probando que la probabilidad converge con el paso de las iteraciones a la posición correcta sin muestreo. Con esto obtenemos el modelo de observación que se usaremos en el algoritmo genético con visión y láser.

- *Implementación del algoritmo genético*, nos aportará flexibilidad y mayor control sobre el algoritmo de localización para añadir más precisión, mejorar la robustez y ganar en velocidad para acercarnos al tiempo real.

- *Experimentos*, con ellos validaremos todas los algoritmos y métodos que usaremos en este proyecto sobre un entorno simulado(Gazebo).

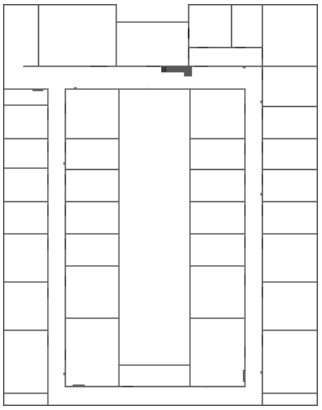


Figura 2.1: Imagen a escala del departamental en escala de grises

Por su extensión queda fuera las pruebas experimentales del algoritmo sobre el robot real, quedando como una línea de acción para el futuro.

## 2.2. Requisitos

Una vez descrito el problema al que se quiere dar solución hay que definir los requisitos que debe tener el algoritmo implementado. Los requisitos que pediremos a nuestro algoritmo de localización son los siguientes:

1. *Uso de la plataforma jderobot 4.3* y por ello será necesario unos conocimientos mínimos del lenguaje C y de sistemas operativos Linux. Además serán necesarios unos conocimientos mínimos de visión computacional y estadística.

2. Tener el *Departamental II* como entorno de pruebas usando balizas naturales del entorno, es decir, balizas que sean estáticas en el entorno y que estén en él sin tener que modificarlo.

3. *Robusto frente a simetrías*, es decir, el algoritmo implementado debe poder lidiar con los problemas que surgen en entornos simétricos a la hora de estimar la posición.

4. *Debe converger* en el caso medio y en el mejor de forma rápida y manteniendo la localización correcta. En el caso peor debe tardar más pero de igual manera debe estimar una localización correcta del robot.

5. *Precisión* estimando la posición del robot con un error entorno a 400mm en las mejores estimaciones.

6. *Tiempo real*, debemos obtener un algoritmo lo suficientemente rápido como para funcionar en tiempo real.

## 2.3. Metodología y plan de trabajo

Para llevar a cabo el proceso software de este proyecto nos hemos basado en un modelo de desarrollo software *incremental*. Dado que se trata de un proyecto amplio, decidimos dividir las distintas partes que conforman el mismo en sucesivos incrementos representativos de cada parte. De este modo, tanto el desarrollo (visto bajo un enfoque global) como las entregas semanales al tutor corresponden a incrementos cuya funcionalidad cumple con los objetivos marcados la semana previa.

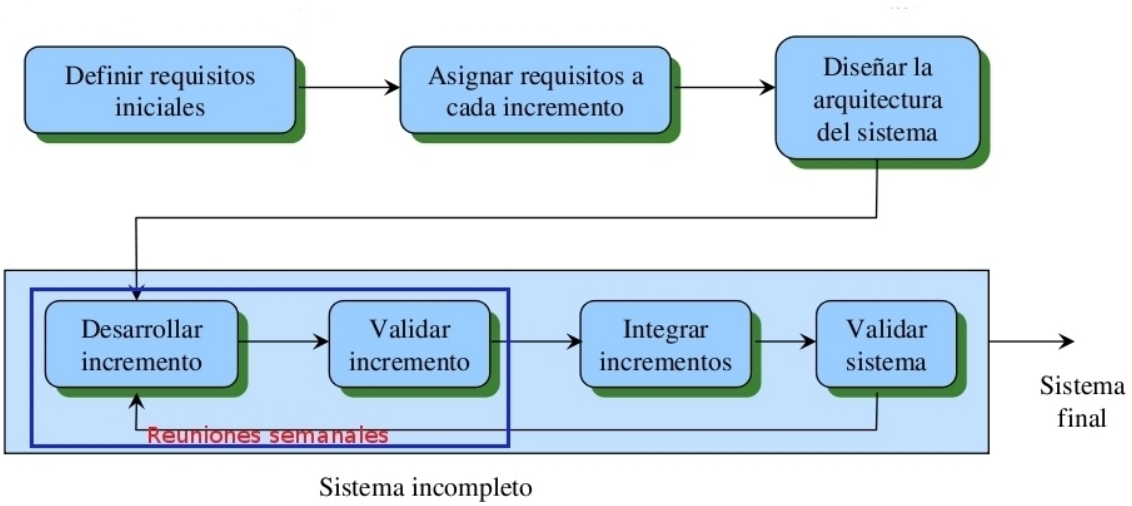



Figura 2.2: Modelo incremental de desarrollo software.

Obviamente, los requisitos de prioridad más alta se incluyen en los incrementos más tempranos. De forma que, cuando el desarrollo de un incremento comienza, sus requisitos son fijos; mientras que los requisitos de incrementos posteriores pueden ir siendo discutidos.

Basándonos en esta metodología definimos el plan de trabajo con los siguientes *incrementos*:

- *Formación básica y familiarización con la infraestructura software jderobot.* Este incremento corresponde a la familiarización con la plataforma de desarrollo *jderobot* así como la ampliación de conocimientos del lenguaje C y además el aprendizaje de otras tecnologías como GTK+, OpenGL, GSL y otras librerías.
- *Modelo de observación.* En este incremento se estudiaron varias formas de recoger los datos de los sensores no específicos y cómo afecta la resolución de los mismos a la calidad de los datos. Se explica en el capítulo4.

- *Técnica de localización por mallas de probabilidad.* En este incremento se estudió e implementó en algoritmo de localización probabilístico basado en mallas de probabilidad explicado en el capítulo4.
- *Técnica de localización por algoritmo genético* Para este incremento se fijo el objetivo de el estudio e implementación de un algoritmo de localización probabilístico basado en algoritmos genéticos que se describe en el capítulo5.

En cada uno de los incrementos anteriores se realizó una fase de experimentación para validar las técnicas implementadas y obtener los resultados que se analizan en los apartados dedicados a experimentos en cada capítulo.

En todo momento se documentó el proceso de realización del proyecto en el siguiente *MediaWiki*[1]. En él están los vídeos y los resultados experimentales de todas las pruebas realizadas además es posible observar el progreso y cómo se han ido completando los hitos marcados.

[1] http://jde.gsyc.es/index.php/User:D.rodriguez

# Capítulo 3
# Entorno y plataforma de desarrollo

En este capítulo se van a describir tanto la plataforma hardware, basada en el robot Pioneer, como software , basada en el middleware *jderobot* que se han utilizado para desarrollar la aplicación de autolocalización del proyecto. Además comentaremos las librerías auxiliares que necesitamos para realizar ciertas tareas como *gridslib* para las rejillas de probabilidad, *OpenGL* y *GTK* para aspectos de visualización.

Para el desarrollo del proyecto se ha escogido como plataforma hardware de referencia el robot Pioneer, figura 3.1 (a) desarrollado por la empresa Active Media. En nuestro caso usaremos el robot simulado de la figura 3.1 (b) que consta, al igual que su homólogo real, de dos ruedas motrices y otra omnidireccional de estabilización, además posee encoders en cada una de las ruedas para poder medir los movimientos que realiza el robot. Adicionalmente simularemos un sensor láser y una cámara sobre la plataforma del Pioneer para poder extraer la información del mundo simulado como lo haríamos en el real.

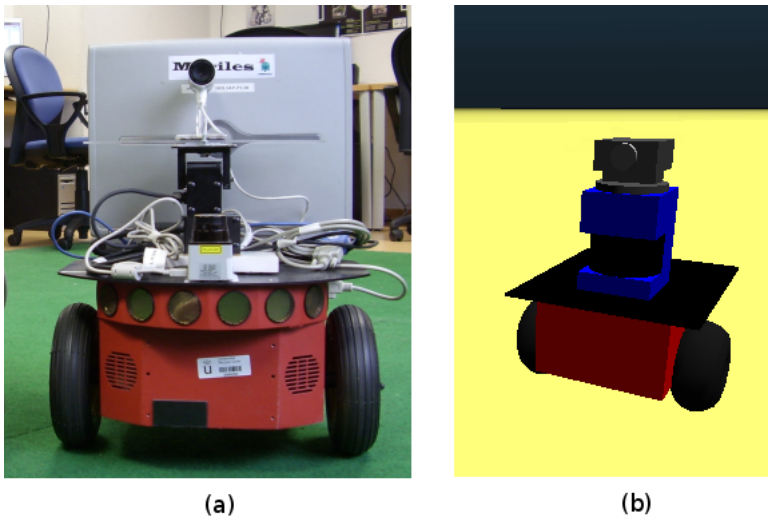



Figura 3.1: (a) plataforma hardware de referencia, (b) robot simulado

## 3.1. Jderobot

La plataforma *jderobot*[1] es un middleware creado por el Grupo de Robótica de la Universidad Rey Juan Carlos que permite acceder de forma sencilla a los distintos sensores de un robot o a diferentes dispositivos como láser, cámaras, cuellos mecánicos, etc. Esta plataforma hace de intermediario entre el hardware y el software que controla el robot como se puede ver en la figura 3.2, aportando cierta transparencia al desarrollador ya que ofrece una interfaz genérica independientemente del hardware. *Jderobot* consta de dos tipos de componentes básicos, los esquemas y los drivers. Un esquema es el código que genera el comportamiento del robot y el driver es el encargado de ofrecer el interfaz de control de un dispositivo.

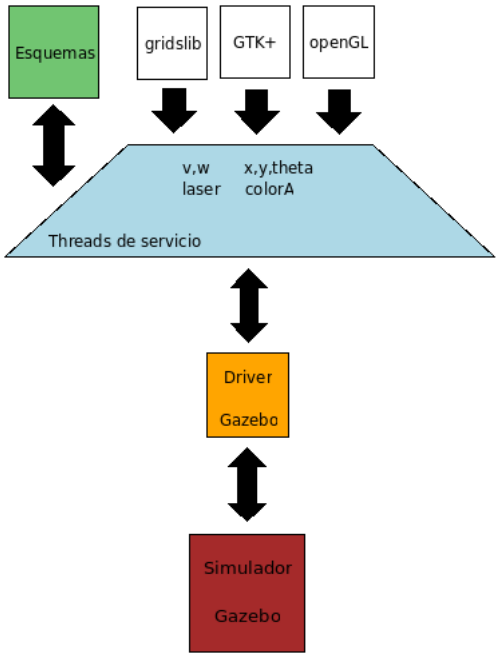



Figura 3.2: Esquema de la plataforma jderobot

Para nuestro proyecto usaremos varias interfaces que nos permiten acceder a los diferentes dispositivos.

- *Interfaz cámara*. Nos permite acceder a las imágenes provenientes de cámaras reales ya sean usb o firewire, de flujos de reproducción de vídeo y del simulador *Gazebo*.
- *Interfaz láser*. Nos brinda las medidas recogidas desde un sensor láser en un vector de números reales con 180 posiciones, independientemente del tipo de sensor láser y con total transparencia para el desarrollador.
- *Interfaz motors*. El interfaz motors nos permite mover el robot asignando valores a una velocidad lineal (V) y a una velocidad angular (W),

[1] `http://jderobot.org`

- *Interfaz encoders*. Con el interfaz encoders obtenemos un vector que nos muestra los incrementos de movimiento, tanto lineal (X,Y) como angular ($\theta$).

El *driver gazebo* nos suministra tanto el *interfaz encoder* como el *interfaz motors* por ello nos apoyaremos en el para comunicar la plataforma *jderobot* con el simulador *Gazebo* y así obtener los datos de las interfaces mencionadas anteriormente de forma transparente.

Nuestra aplicación de autolocalización es un esquema que se apoya en el driver gazebo y en las demás interfaces para obtener los datos del entorno.

## 3.2. Simulador Gazebo

El simulador *Gazebo* [2] es parte del proyecto *Player/Stage* [3] y es capaz de simular varios tipos de robot entre ellos el Pioneer que usamos como referencia y muchos tipos de sensores. Es capaz de simular en un entorno tridimensional de forma realista los sensores y los movimientos de los robots. Usaremos el simulador *Gazebo* para sustituir al mundo real en las pruebas de experimentación ya que simula el comportamiento del robot en un entorno virtual controlado, un ejemplo de la simulación del Departamental II con el robot Pioneer quedaría como se ve en la figura 3.3

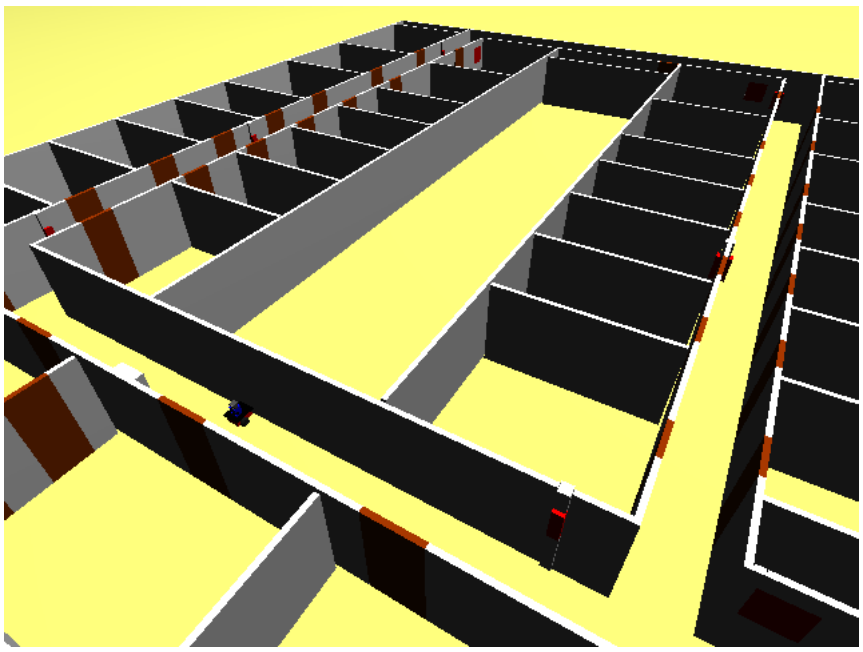


Figura 3.3: Ejemplo simulador Gazebo

[2] `http://playerstage.sourceforge.net/index.php?src=gazebo`
[3] `http://playerstage.sourceforge.net/`

## 3.3. Gridslib

Usaremos la librería *Gridslib* que también forma parte del proyecto *jderobot* y que sirve para crear y manejar rejillas de ocupación. Con esta biblioteca podemos generar una rejilla cuadrada con celdas regulares de un tamaño fijo especificando los parámetros en un archivo de configuración. En el proyecto la usaremos para guardar diversas estructuras de datos como son el mapa del entorno o las rejillas de probabilidad usadas en el algoritmo de mallas de probabilidad.

## 3.4. Librerías gráficas

Para el desarrollo del proyecto hemos necesitado usar unas librerías gráficas que nos han servido de ayuda en tareas de visualización y depuración. Las librerías usadas son las siguientes:

### 3.4.1. OpenGL

La librería gráfica OpenGL es una especificación estándar que define un API multilenguaje y multiplataforma. Es capaz de realizar escenas tanto en 2D como en 3D. Usaremos la librería OpenGL para crear escenas 3D que nos permitan visualizar de una forma clara y eficiente las estructuras internas de los algoritmos, por lo que nos servirá como método de depuración. Concretamente hemos usado la librería para representar el mapa y las estructuras de los algoritmos dentro del interfaz.

### 3.4.2. GTK+ y GTK

GTK+ es un conjunto de bibliotecas y rutinas para desarrollar interfaces gráficas. Hemos escogido esta librería porque se usa como librería estándar en Gnome que es el entorno de escritorio del que disponemos. Por otra parte la biblioteca GTK es el componente de GTK+ que nos permitirá crear la interfaz gráfica gracias a las funciones que nos provee y al editor WYSIWYG *glade* que nos facilita la creación del interfaz con diferentes componentes como menús,botones,diales,etc. Nosotros lo hemos usado para crear el interfaz gráfico que nos ayuda a visualizar en estado del algoritmo en todo momento y nos permite seleccionar distintos parámetros del mismo.

# Capítulo 4
# Mallas de probabilidad

En este capítulo describimos el método probabilístico de localización con mallas de probabilidad que nos permite estimar la posición del robot aun careciendo de una referencia inicial. Este método se apoya en una sucesión de observaciones y de movimientos del robot para, por medio de acumulaciones de probabilidad, obtener la zona del entorno donde se encuentra situado el robot. Con este método validaremos el modelo de observación y el modelo de movimiento comprobando que las estimaciones convergen a la posición correcta.

## 4.1. Fundamentos teóricos

Para conseguir nuestro objetivo, que el robot sea capaz de autolocalizarse, se ha optado por métodos probabilísticos porque es una técnica que ha dado buenos resultados y está muy aceptada dentro de la comunidad robótica. La idea de la localización probabilística es estimar por medio de las funciones de densidad de probabilidad una única posición, que se corresponda con la posición real del robot, con la mayor precisión posible. Para poder estimar correctamente la posición del robot debemos trabajar con observaciones independientes para luego poder realizar la acumulación de probabilidad. En todo momento usamos funciones de densidad de probabilidad para representar la posición estimada del robot, la información de los sensores contenida en observación (probabilidad instantánea) y el modelo de movimiento (desplazando las probabilidades). Una forma de representar estas funciones son las mallas de probabilidad usadas en este proyecto.

El algoritmo de mallas de probabilidad consta de tres pasos que se repiten reiteradamente:

- *Modelo de movimiento.* Se capturan los incrementos de movimiento medidos por la odometría y realiza estos mismos movimientos en la probabilidad contenida en la malla.

- *Modelo de observación.* Se realiza a partir de las observaciones sensoriales tanto del láser como de la cámara y siempre apoyándose en el mapa del entorno para realizar las comparaciones en todas las posiciones. Con cada nueva observación realizamos unos cálculos para obtener una probabilidad instantánea, de tal manera que las posiciones más compatibles con la observación actual del robot tienen una probabilidad más alta que las demás.

- Acumulación de evidencias.Se sigue del desarrollo matemático completo de la Regla de Bayes [Thrun, 2000]. A través de este desarrollo se llega a una fórmula incremental que nos permite actulizar la probabilidad acumulada añadiendo la última observación sensorial de una forma muy sencilla.

## 4.2. Diseño general del algoritmo

La implementación del algoritmo se realiza por medio de un esquema de *jderobot* con una hebra que realiza las tareas de recolección y los cálculos sobre los datos y otra hebra que nos ayuda a la visualización y depuración mediante una interfaz gráfico. El esquema usa las variables proporcionadas por la plataforma *jderobot*, en concreto las variables que contienen la información de los encoders *robot[0,1,2]*, las lecturas del sensor láser *laser[180]* y las imágenes provenientes de la cámara *varcolorA*. La relación del esquema con la plataforma se puede ver en la figura 4.1. Además de tener acceso a las variables proporcionadas por *jderobot* el esquema tiene el mapa y la rejilla 3D que hace las funciones de cubo de probabilidad 4.2.

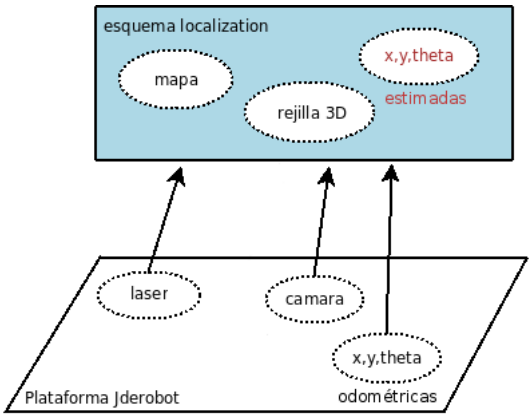



Figura 4.1: Esquema de comunicación con la plataforma jderobot

Nada más arrancar el algoritmo la probabilidad de todas las posiciones será muy parecida y a medida que pasen las iteraciones y gracias a las acumulaciones por medio de la Regla de Bayes conseguiremos destacar la posición correcta. Para guardar los datos con la probabilidad para todas posiciones y orientaciones posibles usamos la rejilla tridimensional que conceptualmente se puede ver en la figura 4.2 y en la que realizaremos los desplazamientos y los giros según señale el modelo de movimiento.

En el código fuente el cubo está implementado con ayuda de la librería *gridslib*, tenemos una rejilla con las posiciones (x,y) para cada orientación $\theta$. En concreto usaremos 36 rejillas para representar todas las orientaciones en incrementos de 10º, cada una de las rejillas tiene casillas de 200x200 mm que representan todas las posiciones (x,y) del mundo de manera discretizada, con un tamaño del mundo de 23500x30000 mm tenemos 22801 casillas en total.

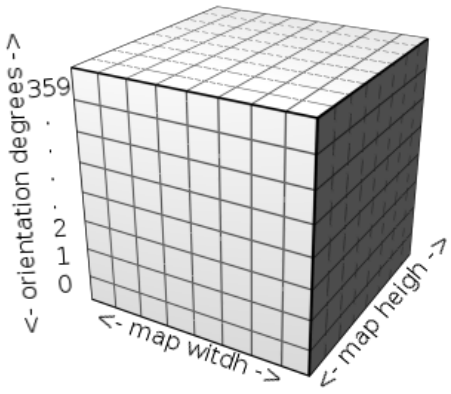



Figura 4.2: Cubo 3D

Por razones de visualización usamos dos cubos con las características anteriores, uno para visualizar los datos de la iteración actual y otro para procesar los datos que se visualizarán en la siguiente iteración, esta técnica que se conoce como *Doble Buffering* está representada en la figura 4.3.

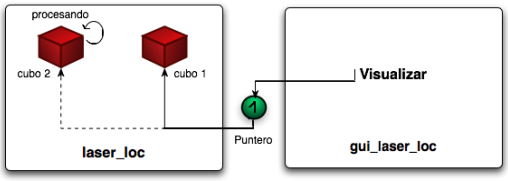



Figura 4.3: Cubo 3D

El *Doble Buffering* es una técnica usada en el mundo de tratamiento de gráficos e imágenes y consiste en mantener dos buffers en paralelo, uno para visualizar el resultado de la operación anterior y en el otro se calcula el resultado de la siguiente operación que verá el usuario. De esta forma el usuario siempre ve el contenido de una forma estable quedando ocultos los estados intermedios del buffer mientras se realizan los cálculos. Una vez terminados los cálculos se intercambian los roles de los buffers y se visualiza sobre el que antes se realizaban los cálculos.

A bajo nivel hay dos cubos de probabilidad y se mantiene un puntero que apunta al cubo que se visualiza y otro puntero apuntando al cubo usado para los cálculos, estos punteros son los que se intercambian ganando en eficiencia ya que el intercambio no es más que una asignacion de variables, como se ve en la figura 4.3.

El funcionamiento del algoritmo viene explicado por el diagrama de flujo que se ve en la figura 4.4 y que se explica a continuación.

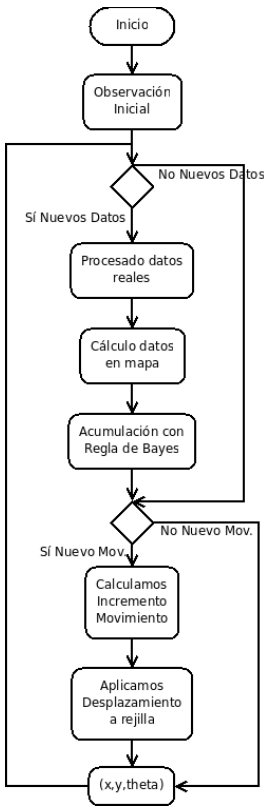



Figura 4.4: Flujo ejecución del algoritmo

1. Inicialización del algoritmo con una primera observación
2. Pasos reiterativos:
   *a*) Incorporación del movimiento (sección 4.3).
   *b*) Actualización de datos, cálculo de las nuevas probabilidades basadas en nuevas observaciones sensoriales (sección 4.4) y acumulación de las probabilidades (sección 4.5).

Para poder calcular las probabilidades necesitamos algo con lo que comparar los datos sensoriales obtenidos a partir del láser y de la cámara y por ello tenemos un mapa de colores a escala del escenario en una imagen PGM con la siguiente codificación:

- 56: para la papelera.
- 48: simboliza las puertas.
- 101: corresponde a los extintores
- 89: se refiere a la pared
- 0: es una posición vacía, transparente

A la hora de usar esta imagen necesitamos tenerla anclada con las dimensiones reales del mundo y para ello nos apoyamos en la librería *gridslib*. Por dentro esta estructura se puede ver como un gran tablero de ajedrez con la información de color correspondiente a esa posición del mapa en cada casilla. Esta es la estructura que consultaremos cada vez que necesitemos obtener los datos del mapa tanto para las imágenes como para el láser.

Para poder establecer una probabilidad realizaremos una comparación entre los datos sensoriales y los extraídos del mapa tanto para el láser como para la cámara como se describe en el apartado del modelo de observación.

La finalidad de este algoritmo es obtener una posición (x,y,$\theta$) estimada donde se encuentra el robot, para ello siempre daremos como posición estimada la que tenga mayor probabilidad en la rejilla.

## 4.3. Modelo de movimiento

El modelo de movimiento es el encargado de actualizar en la rejilla los movimientos que realiza el robot a partir de la información que se obtiene de los *encoders*. Los *encoders* cuentan las vueltas que dan las dos ruedas motrices del robot y devuelven unos valores de (x,y,$\theta$) aproximados. Con este sensor podremos determinar de modo aproximado cuánto y cómo se desplaza el robot siendo los desplazamientos tanto lineales (en línea recta) o angulares (giros). Para calcular los incrementos de movimiento se han usado las siguientes ecuaciones que calculan el incremento de movimiento de un instante a otro y que aplican un ruido a los incrementos para asemejar las medidas medidas a las obtenidas con sensores reales.

$$\Delta_{lineal} = Mov_{lineal} + Ruido_{lineal} \tag{4.1}$$

$$\Delta_{angular} = Mov_{angular} + Ruido_{angular} \tag{4.2}$$

$$Mov_{lineal} = \sqrt{(x_t - x_{t+1})^2 + (y_t - y_{t+1})^2} \tag{4.3}$$

$$Mov_{angular} = \theta_t - \theta_{t+1} \tag{4.4}$$

$$Ruido_{lineal} = \sigma(Mov_{lineal} * \%_{Ruidolineal}) \tag{4.5}$$

$$Ruido_{angular} = \sigma(Mov_{angular} * \%_{Ruidoangular}) \tag{4.6}$$

En algunos algoritmos de localización además de recibir los datos de los *encoders* se tienen en cuenta las ordenes de movimiento enviadas. En nuestro caso no tenemos en cuenta ninguna orden enviada a los actuadores del robot sino que sólo recogemos las observaciones de los *encoders* de esta forma nuestro algoritmo es más flexible ya que no bloquea los movimientos del robot que pueden ser comandados por otro esquema y eso permite navegación y localización simultáneas.

Este modelo nos permite incorporar los movimientos realizados por el robot a nuestras rejillas de probabilidad. Aunque los movimientos son continuos la aplicación de los mismos se realiza de manera discreta e incremental cuando estos incrementos superan un tamaño mínimo. Se han tomado como valores de compromiso 1800 mm

para el movimiento lineal y 10º para el movimiento angular. Un desplazamiento lineal del robot equivale a desplazar las probabilidades de las rejillas de unas casillas a otras en la misma loncha del cubo y para un movimiento angular deberemos cambiar las probabilidades de una loncha del cubo a otra ya que éstas simbolizan diferentes orientaciones, estos incrementos se aplican de acuerdo a estas ecuaciones:

$$\Delta r_x = \frac{\Delta m_x + \Delta m_x(\frac{Grid.resolucion}{2})}{Grid.resolucion} \tag{4.7}$$

$$\Delta r_y = \frac{\Delta m_y + \Delta m_y(\frac{Grid.resolucion}{2})}{Grid.resolucion} \tag{4.8}$$

$$\Delta r_\Theta = \frac{\Delta m_\Theta}{(\frac{umbral}{2})} \tag{4.9}$$

Con las ecuaciones anteriores obtenemos la equivalencia entre el incremento de movimiento del robot y el incremento de movimiento que debemos aplicar en la rejilla. Y con ello podemos mover las probabilidades de los cubos replicando los incrementos de movimiento obtenidos de los *encoders*. En los casos en los que el movimiento haga que una casilla se mueva a una nueva del exterior de la rejilla, simplemente no moveremos la probabilidad a ninguna ubicación nueva.

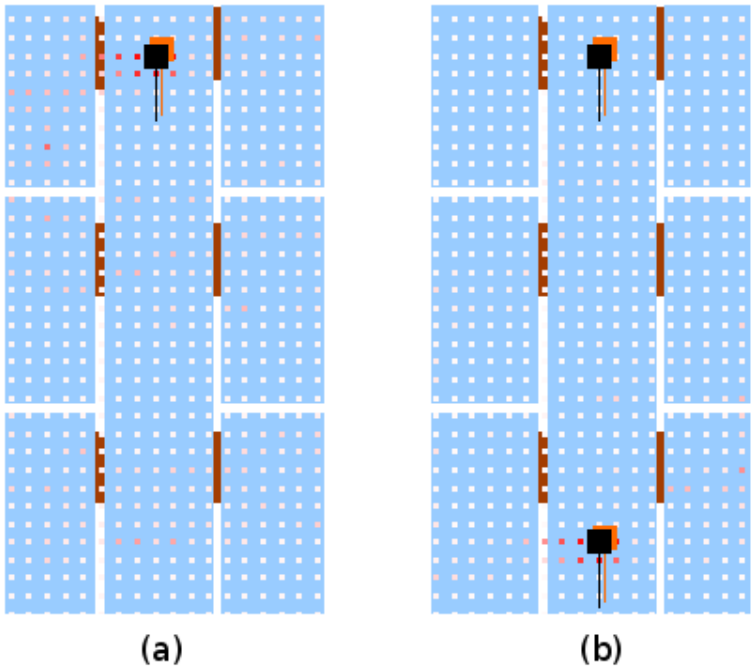



Figura 4.5: (a) Antes del movimiento; (b) Después del movimiento

El resultado del movimiento de la rejilla se puede ver en la figura 4.5 en la que se realiza un movimiento lineal sin aplicar ruido odométrico y se puede ver cómo las probabilidades más altas(rojo) que estaban en la posición inicial (a) siguen estando en la zona del robot después de aplicarles el incremento de movimiento hasta la posición (b).

## 4.4. Modelo de observación

Nuestro modelo de observación consta de dos sensores que son el láser y la cámara que hemos combinado para obtener las mejores características de cada uno. Una forma de obtener información a partir de estos sensores no específicos es comparar la observación teórica extraída del mapa con la observación real que aportan los sensores. Como tenemos dos tipos de sensores es necesario definir un modelo de observación distinto para cada uno de ellos.

### 4.4.1. El láser

Comenzaremos con el láser explicando cómo obtenemos los datos tanto del sensor real como del mapa y qué método hemos escogido para sacar la información de este sensor. De la obtención de datos extraeremos información en forma de probabilidad del parecido entre el láser observado por el robot y el láser en diferentes posiciones del mapa, de esta forma cuanto más parecidas sean una posición del mapa y la del robot mayor será la probabilidad y viceversa.

#### Obtención del láser real

Para el láser observado la plataforma *jderobot* nos brinda un vector de números reales con 180 valores que corresponden a las medidas del láser una por cada grado de la semicircunferencia.

#### Obtención del láser teórico

Por otro lado, tenemos que sacar los datos pertinentes del mapa y para ello calcularemos qué distancia existe desde un punto dado hasta el primer obstáculo de nuestro mapa, en otras palabras, comprobaremos la rejilla del mapa desde el punto dado en cada grado hasta encontrarnos con un valor que se corresponda con alguno de los designados como colores. Para calcular los valores del láser a partir del mapa seguiremos los siguientes pasos:

1. Calculamos el punto máximo teórico del láser, es decir la distancia máxima que podríamos alcanzar con el láser, con las siguientes ecuaciones:

$$B_x = A_x + RADIO\_LASER \cdot cos(\Theta) \tag{4.10}$$
$$B_y = A_y + RADIO\_LASER \cdot sin(\Theta) \tag{4.11}$$

   Donde el valor de la constante RADIO_LASER se corresponde con el radio máximo de nuestro láser que son 4 metros.

2. Una vez que tenemos el punto de destino (B) calculamos la recta desde el origen (A) e iremos recorriendo dicha recta buscando obstáculos con un factor de $\lambda$ cada vez como se ve en las siguientes ecuaciones:

$$P_x = A_x + \lambda(B_x - A_x) \tag{4.12}$$
$$P_y = A_y + \lambda(B_y - A_y) \tag{4.13}$$

En el siguiente dibujo se aprecia de manera gráfica lo que expresan las ecuaciones anteriores y cómo se va recorriendo la recta en pequeños intervalos

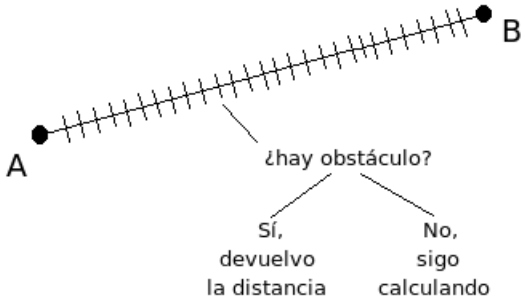



Figura 4.6: Representación del cálculo de la distancia a un objeto

Realizaremos este cálculo para cada uno de los ciento ochenta grados rellenando un vector que llamaremos *láser_teorico* y que usaremos para comparar con el que nos proporciona la plataforma *jderobot*. La representación del cálculo se puede ver en la figura 4.7. Por motivos de eficiencia limitamos el cálculo en lo referente al láser a 45º (realizamos el cálculo cada 4º) ya que comprobamos que no afectaba especialmente en el cálculo de la probabilidad y conseguíamos realizar los cálculos 4 veces más rápido.

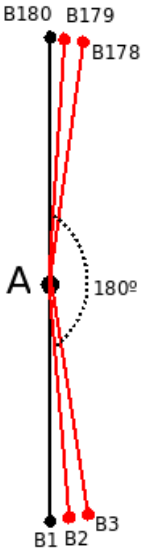



Figura 4.7: Cálculo del láser teórico

Para calcular la diferencia entre los datos sensoriales y los provenientes del mapa usamos la *Distancia de Manhattan* definida por la siguiente ecuación:

$$probLaser(x, y, \theta) = \frac{\sum_{i=1}^{45}(abs(laser\_real(i) - laser\_teorico(i)) < 20cm)}{45} \tag{4.14}$$

La función *Distancia de Manhattan* otorga un valor de 1 a cada par de medidas correspondientes a un determinado grado si estas se diferencian en menos de veinte centímetros, luego obtenemos la probabilidad dividiendo el número de pares de rayos válidos entre ciento ochenta que es el máximo. De esta manera cuantos más pares de rayos parecidos haya mayor será la probabilidad acercándose a 1 y si el número de pares de rayos válidos es muy pequeño la probabilidad se acerca a 0.

### 4.4.2. La cámara

En cuanto a las imágenes, no nos interesan todos los colores que puedan existir en una imagen que nos aporte la cámara real ya que en nuestro mapa sólo tenemos los colores que más información nos pueden aportar, por ello realizamos un filtrado y posteriormente resumimos la información aportada por la imagen observada para poder tratarla de un modo más eficiente.

#### Obtención de la imagen observada

La cámara nos proporciona una imagen cruda de 320*240 en formato RGB en un *array*. Lo primero que haremos será escoger los colores que nos interesan por medio de un filtrado. Para realizar el filtrado por colores el espacio RGB no es muy flexible frente a cambios en la iluminación, por ello para filtrar los colores que nos interesan hemos usado el espacio de colores HSV en el que podemos realizar un filtrado más robusto y nos permite separar la iluminación de la escena de los colores que queremos filtrar. Para filtrar estableceremos los valores de $H_{min}, H_{max}, S_{max}, S_{min}, V_{max}$ y $V_{min}$ para cada uno de los colores. El cambio de un espacio de colores a otro se realiza por medio de estas ecuaciones:

$$H = cos^{-1}\left(\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right) \tag{4.15}$$

$$S = 1 - \frac{3}{(R+G+B)} min\left(R,G,B\right) \tag{4.16}$$

$$V = \frac{1}{3}\left(R+G+B\right) \tag{4.17}$$

Una vez realizado el filtrado obtenemos una imagen de las mismas dimensiones que la original en un nuevo *array*, como se puede ver en la figura 4.8. La imagen (a) corresponde a la original capturada por la cámara y la imagen (b) es la que sale después de realizar el filtrado por los colores de interés.

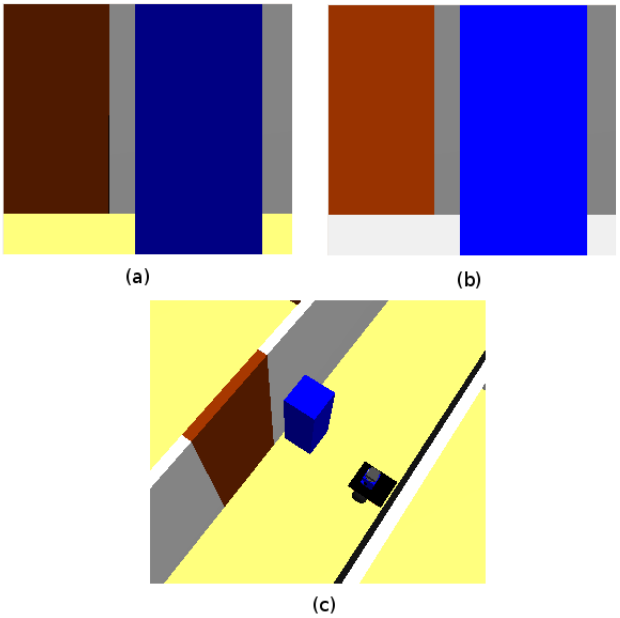



Figura 4.8: (a) imagen de la cámara, (b) imagen filtrada por colores, (c) posición del robot en el simulador

El siguiente paso es resumir la imagen filtrada, la convertiremos de un *array* de 320*240 posiciones a un vector con sólo 320 valores. La razón de resumir la imagen de un array bidimensional a un vector de una dimensión es para realizar los cálculos de manera más eficiente. El proceso de resumir la imagen consiste en recorrer el array de la imagen filtrada por columnas contando el número de píxeles del color de la columna para comprobar si hay suficientes píxeles para considerar que representan un objeto que nos interesa. Si obtenemos un número de píxeles relevante almacenamos la información de color en el vector con 320 posiciones, las diferentes etapas del proceso quedan ilustradas en la figura 4.9, donde vemos como pasamos de la imagen de la cámara (a) a la filtrada (b) y de esta última a un vector (c) resumido.

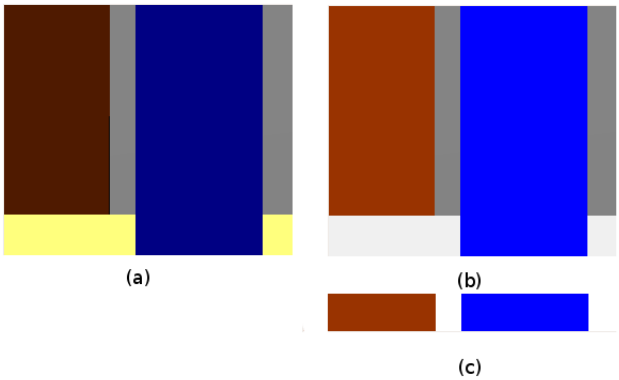



Figura 4.9: (a) imagen de la cámara, (b) imagen filtrada por colores y (c) la imagen resumida

### Obtención de la imagen teórica

Una vez tenemos los datos de la imagen obtenida de la cámara listos, necesitamos algo con lo que comparar estos datos y para ello tenemos nuestro mapa del escenario a escala. Para capturar la información visual de este mapa seguiremos el mismo método que hemos usado con el láser, pero esta vez nos ajustaremos al ángulo de visión de nuestra cámara. En nuestro robot tenemos una cámara que tiene un ángulo de visión de 70 grados y por ello calcularemos los puntos destino de los rayos de visión desde el centro del robot hasta treinta y cinco grados a la derecha y a la izquierda siempre en la dirección donde mira el robot, como se ve en la figura 4.10. En un principio se calcularon 320 muestras aunque luego se redujeron a 80 quedando este último como valor de compromiso por motivos de eficiencia.

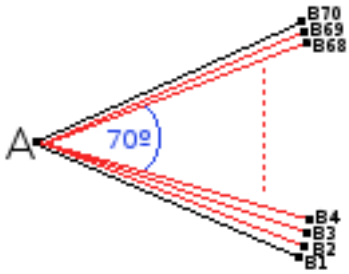



Figura 4.10: Cálculo de la imagen teórica

Una vez que tenemos calculados los puntos de destino usamos las mismas ecuaciones que se explicaron para el láser, pero con un rango de visión igual a treinta metros (hemos elegido esta distancia por que en interiores el robot nunca va a ver más de treinta metros y así nos facilita los cálculos) y con la salvedad de que cuando encontremos un obstáculo en vez de guardar la distancia de este al robot guardamos el color que hay en esa celdilla del mapa. El resultado de las mediciones lo guardaremos en un vector de 80 posiciones para poder compararlo con la imagen obtenida de la cámara resumida en otro vector, el resultado de esta medición se puede ver en la figura 4.11 ya transformado en forma de vector (b) y comparado con el vector que sacamos anteriormente de la imagen de la cámara (a).

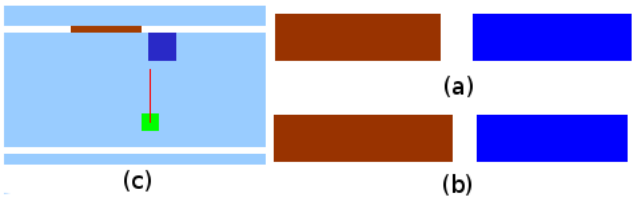



Figura 4.11: (a) vector imagen cámara, (b) vector imagen teórica y (c) posición del robot en el mapa

Ahora ya tenemos dos vectores que poder comparar y para ello usamos una función ya usada por [Domínguez, 2009] que analiza los colores de los vectores y los asocia a objetos de tal forma que si ve cualquier color distinto del blanco lo trata como un

objeto y posteriormente compara cómo son de parecidos los objetos de los dos vectores asignando una probabilidad alta si se parecen mucho y baja si no se parecen nada.

De cada objeto que detectamos nos quedaremos con los píxeles donde comienza y acaba el color además del propio color, como se puede ver en la imagen 4.12


Figura 4.12: Ejemplo de obtención de objetos del vector

En la figura 4.13 se ve cómo se realiza la comparación en las dos direcciones, mientras se comparan los objetos obtenemos el número de píxeles parecidos y los diferentes, y si concuerda o no el color de los objetos. Estos resultados se almacenan para poder operar con ellos y calcular la probabilidad.

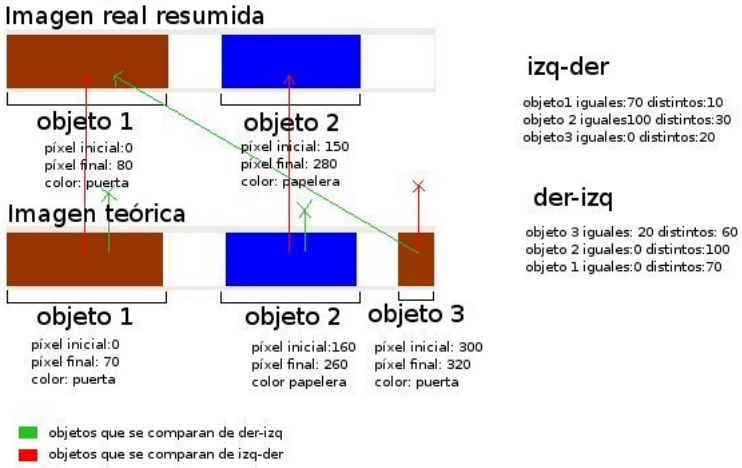



Figura 4.13: Ejemplo comparación de objetos

Una vez que tenemos el número de píxeles iguales y diferentes aplicamos la siguiente fórmula 4.18 para obtener el peso de cada objeto:

$$peso\_objeto = \frac{iguales - diferentes}{iguales + diferentes} \tag{4.18}$$

Para obtener el resultado en el rango [0,1] que nosotros necesitamos normalizamos aplicando a cada peso calculado esta fórmula 4.19:

$$peso\_objeto[0,1] = \frac{peso\_objeto + 1}{2} \tag{4.19}$$

Por último, para conseguir la probabilidad que nos exprese el parecido de las dos imágenes sumaremos todas las probabilidades obtenidas con el peso de los objetos ya normalizadas y las dividiremos entre el número de objetos total como se explica en la siguiente fórmula 4.20:

$$probabilidad\_funcion\_objetos\_sin\_cola = \frac{\sum_{i=0}^{num\_objetos} peso\_objeto[0,1]}{num\_objetos} \tag{4.20}$$

### 4.4.3. Combinando los dos modelos

Hasta aquí tenemos dos probabilidades, una para el modelo de observación láser y otra para el de la cámara, pero nosotros sólo necesitamos una probabilidad para cada posición de la rejilla tridimensional. Para pasar de múltiples probabilidades a una decidimos diferenciar dos casos posibles:

1. Si la imagen observada de la cámara no tiene objetos, simplemente usaremos la probabilidad que aporta el láser en las posiciones del mapa en las que la imagen teórica no contenga objetos y en las que sí contenga objetos daremos probabilidad cero.

2. Si la imagen observada de la cámara tiene objetos: si la imagen teórica obtenida del mapa no tiene objetos asignaremos a esa posición probabilidad cero y si la imagen del mapa tiene objetos entonces combinaremos ambas probabilidades con la siguiente fórmula:

$$prob = \sqrt{dist\_sin\_cola(camara, vision\_teorica) * dis\_manhattan(laser, laser\_teorico)} \tag{4.21}$$

## 4.5. Acumulación de probabilidades: Regla de Bayes

En nuestro modelo de localización necesitamos varias iteraciones para poder estimar la posición de manera correcta ya que no podemos decir con una única observación que el robot esté localizado porque podemos obtener múltiples posiciones con probabilidades altas. Para eliminar esta ambigüedad será necesario acumular de alguna forma las probabilidades para que las posiciones buenas mantengan una alta probabilidad y las malas reduzcan su probabilidad.

Al realizar la acumulación de probabilidades nos apoyamos en la Regla de Bayes del marco probabilístico, que nos ofrece robustez y que nos permitirá ir acumulando las probabilidades de cada iteración. En un principio realizamos la primera observación sin realizar acumulación ninguna ya que en el instante inicial consideramos que no había probabilidad anterior, pero para las siguientes iteraciones usaremos esta regla para fusionar la probabilidad existente con la nueva probabilidad calculada.

Para calcular la nueva probabilidad acumulada en el instante (t) fusionaremos los ratios de la probabilidad acumulada en el instante (t-1) y de la nueva probabilidad que nos dan nuestros modelos de observación en el instante (t) mediante la relación 4.24 que se deduce del desarrollo de la Regla de Bayes:

$$P(x, y, \Theta/obs_1, obs_2...obs_N) \sim P(x, y, \Theta/obs_1, obs_2...obs_{N-1}) \cdot P(x, y, \Theta/obs_N) \quad (4.22)$$

$$\rho ac(x_t) = \rho ac(x_t) \cdot \rho(obs_t) \quad (4.23)$$

$$\ln \rho ac(x_t) = \ln \rho ac(x_t) + \ln \rho(obs_t) \quad (4.24)$$

Donde:

- P es la probabilidad de esa posición.
- $\rho$ es el ratio de la probabilidad determinado por $\frac{P}{1-P}$.
- $x_t$ representa la posición en el instante actual.

En la figura 4.14 podemos ver cómo el robot realiza varios movimientos tanto lineales como angulares y cómo se van realizando las acumulaciones por medio de la Regla de Bayes y cómo se aplican los incrementos de movimiento en la rejilla. También se puede observar cómo las zonas de probabilidades altas (rojas) se estrechan hacia el robot haciendo que la probabilidad converja.

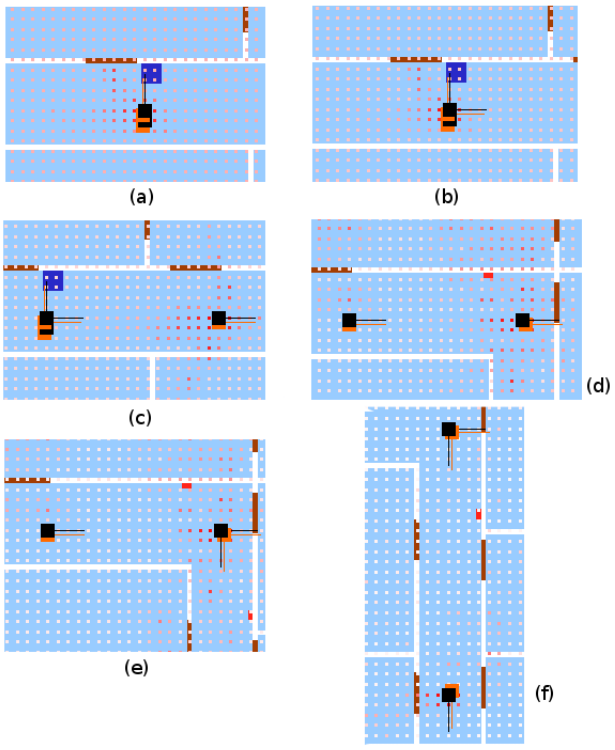



Figura 4.14: Ejemplo aplicación de la Regla de Bayes

La ejecución que se muestra en la figura 4.14 comienza en el apartado (a) después de varias mediciones del robot en un mismo lugar del mapa, en estas mediciones sólo se acumula la primera evidencia con la Regla de Bayes. En las siguientes observaciones se incorporan evidencias si el robot se desplaza más de 1800 mm o gira más de 5º. Posteriormente en (b) el robot realiza un giro de 90º a la derecha y por ello se gira la rejilla tridimensional en el mismo sentido y realizamos una nueva observación para acumular las probabilidades nuevas con las antiguas siguiendo la Regla de Bayes. Después en el apartado (c) el robot realiza un movimiento lineal en el eje x lo que hace que se muevan las probabilidades dentro de la rejilla y además realizamos una nueva observación y la pertinente acumulación. El apartado (d) corresponde a otro movimiento lineal en el eje x y el (e) a otro giro de 90º hacia la derecha en cada uno, también realizamos la observación y la acumulación de las evidencias. Por último, en el apartado (f) el robot avanza haciendo un movimiento lineal en el eje y acumulando las probabilidades con la nueva observación.

A medida que pasan las diferentes iteraciones se puede ver cómo en los sucesivos apartados gracias al movimiento del robot y a la acumulación de evidencias las zonas de probabilidad más alta (siempre en rojo) se estrechan en torno al robot lo que nos da una posición más fiable para su localización.

## 4.6. Experimentos

En los apartados anteriores se ha explicado la teória y la implementación software que hay detrás del modelo de observación y del modelo de movimiento, además se ha explicado la teoría que nos permite acumular la probabilidad gracias a la Regla de Bayes. En este apartado se explican los experimentos que nos condujeron a escoger los diferentes modelos y que nos permitieron validarlos con el algoritmo de mallas de probabilidad.

### 4.6.1. Experimentos con el modelo de observación

En la sección anterior se han descrito los fundamentos teóricos de los modelos de observación escogidos para ambos sensores. Para escoger estos modelos se realizaron una serie de experimentos con diferentes modelos de observación, tanto para el láser como para la cámara que se describen en este apartado.

**• Comparación entre láser teórico y observado**

Antes de escoger la función de la *Distancia de Manhattan* para comparar los datos sensoriales del láser con los datos del mapa probamos estas otras alternativas:

1. *Distancia de Mahalanobis*: consiste en aplicar la distancia de Mahalanobis con la siguiente ecuación:

$$distanciaMahalanobis\_laser = \sqrt{\left(\frac{laser_{sensor} - laser_{teorico}}{\sigma}\right)^2} \tag{4.25}$$

Donde $\sigma$ es la varianza de los datos.

Ahora bien, debemos normalizar para que el valor sea entre 0 y 1, y para ello dividimos la distancia de Mahalanobis obtenida entre el valor máximo que se puede obtener.

$$probabilidadMahalanobis = \frac{\sum_{i=0}^{180} dMa_i}{(\sqrt{3} * 4) * 180} \tag{4.26}$$

2. *Coeficientes de Correlación*: consiste en ver cómo son de parecidos los vectores comparando su producto escalar con sus módulos con la siguiente ecuación:

$$probabilidadCorrelacion = \frac{vector\_sensor\_laser \cdot vector\_laser\_teorico}{|vector\_sensor\_laser| * |vector\_laser\_teorico|} \tag{4.27}$$

3. *Distancia de Manhattan*: consiste en asignar un o cero a cada comparación entre láser teórico y real y luego normalizar el sumatorio entre el número de medidas comparadas como se describe en la siguiente ecuación:

$$probabilidadManhattan = \frac{\sum_{i=1}^{180}(abs(laser\_real(i) - laser\_teorico(i)) < umbral)}{180} \tag{4.28}$$

En los experimentos se comprobaron las tres funciones en diferentes posiciones para comprobar dos propiedades que nos interesan especialmente: ¿cómo es de discriminante la función? y ¿sitúa bien el máximo de la probabilidad?.

Para caracterizar la discriminación realizamos una prueba para observar la distribución espacial de la probabilidad, que se puede ver en la figura 4.15, siendo las posiciones con un color rojo más intenso las que tienen una probabilidad más alta y el cuadrado verde la posición del robot real en el en Departamental II.

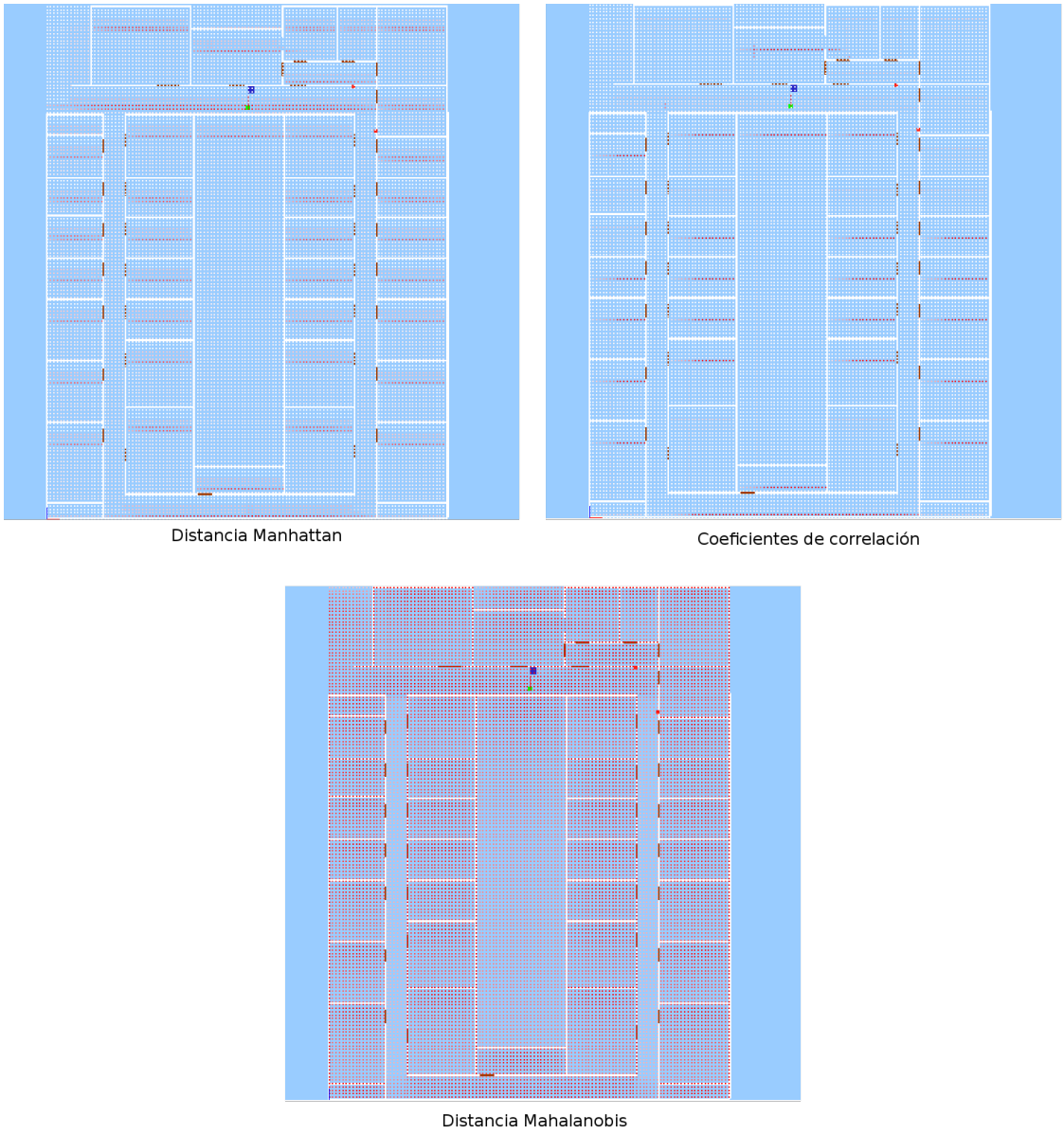



Figura 4.15: Mapas distribución espacial probabilidad láser

En esta prueba pudimos observar para una posición dada la distribución espacial de la probabilidad para cada función. Como se puede ver, la función de *Coeficientes de Correlación* no es capaz de discriminar lo suficiente una posición de otra muy distinta en el mapa. También vemos que la probabilidad que arroja la función de *Distancia de Mahalanobis* es poco discriminante dando demasiadas posiciones por válidas. Por último se puede ver la función que mejor se comporta es la función de *Distancia de Manhattan* ya que muestra lugares rojos en zonas parecidas a la correcta y además emplaza una alta probabilidad en la zona donde se encuentra situado el robot.

Para responder si situa bien el máximo de probabilidad realizamos una prueba analizando cómo cambia la probabilidad ante los giros, siempre manteniendo estática la posición en el mapa consiguiendo los resultados que se ven en la figura 4.16.

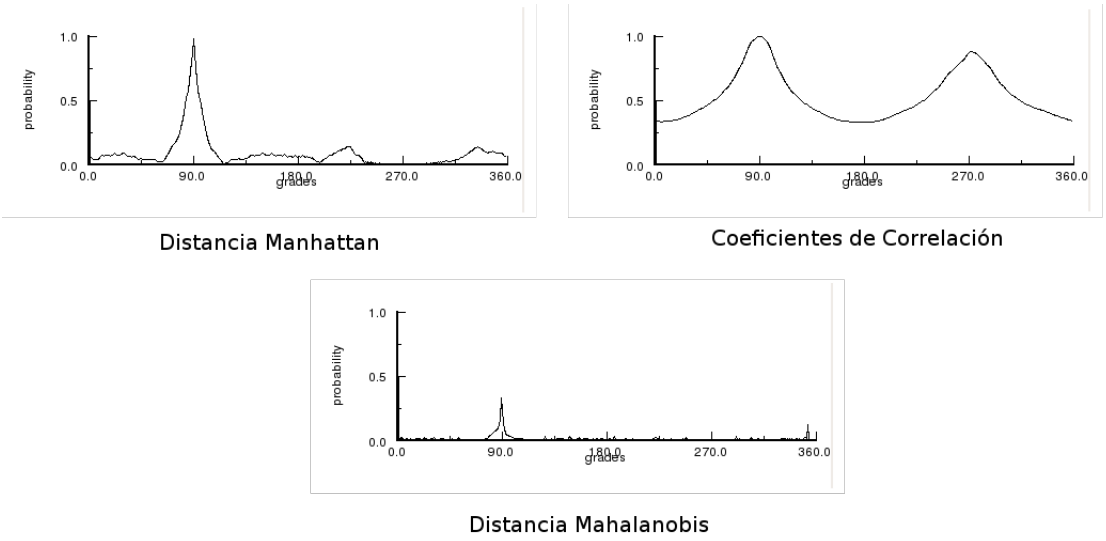



Figura 4.16: Gráficas distribución angular probabilidad láser

Con esta prueba se pretende ver si las funciones de probabilidad establecen el máximo en la orientación adecuada. Como podemos ver, la función de *Correlación* no es buena ya que tiene alta probabilidad tanto en 90º, la orientación correcta, como en 270º. La función *Distancia de Mahalanobis* emplaza correctamente el máximo de la probabilidad en 90º, pero la función de *Distancia de Manhattan* además de emplazar el máximo en 90º nos da una probabilidad mucho mayor lo que quiere decir que es capaz de distinguir esa orientación como la buena.

Como se puede observar escogimos la función *Distancia de Manhattan* porque es la que mejor resultados mostró en las pruebas realizadas siendo lo suficientemente discriminante y situando el máximo de la probabilidad en la posición correcta.

**• Efectos de la resolución sobre el láser**

En un principio se realizaron las pruebas con los 180 rayos nos da el sensor láser, pero por motivos de eficiencia se decidió probar a usar un número menor de rayos láser y ver cómo afecta a los resultados de la función *Distancia de Manhattan*. Las pruebas arrojaron los resultados que se ven en la figura 4.17.

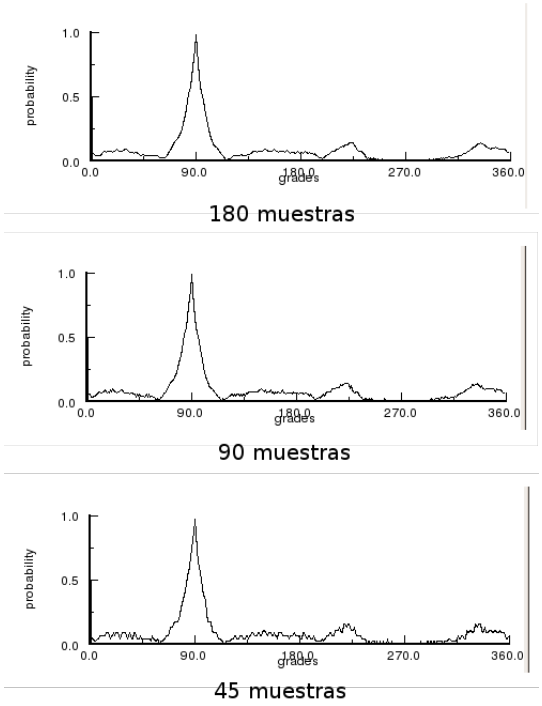



Figura 4.17: Gráficas comparativas resolución láser

Como se puede ver en la prueba vimos que al disminuir el número de rayos usados en los cálculos el máximo se mantenía en el lugar correcto y lo único que se añadía era algo de ruido en los valores de probabilidad, pero sin enturbiarlos demasiado.

Por ello al final se decidió usar la función *Distancia de Manhattan* usando 45 rayos del láser para realizar los cálculos.

**• Comparación unidimensional entre imagen teórica y observada**

Para escoger una función de observación probamos varias funciones para calcular la probabilidad. Las podemos dividir en dos subtipos, las basadas en píxeles y las basadas en objetos.

Las funciones basadas en píxeles tienen en común que usan el color del píxel para determinar el parecido entre la imagen de la cámara y la extraída del mapa. Las técnicas de este tipo que probamos fueron:

1. *Distancia Manhattan*: Es la más sencilla de todas, y consiste en ir comparando píxel a píxel si son iguales o no lo son. Al final si las dos imágenes son exactamente iguales su parecido será del cien por cien. Aplicamos la siguiente formula porque al final tenemos que obtener un valor entre 0 y 1

$$prob_{dist\_Manhattan} = \frac{numero\_pixeles\_iguales}{320} \tag{4.29}$$

2. *Distancia Manhattan con suavizado*: Muy parecida a la anterior pero aplicamos un suavizado sólo a la imagen teórica, la imagen que cogimos de la cámara no se

suaviza. Este suavizado consiste en coger un píxel, los dos de su izquierda y los dos de su derecha y aplicar la fórmula:

$$suavi_{p\_i} = \frac{(p_{i-2} * 1) + (p_{i-1} * 2) + (p_i * 3) + (p_{i+1} * 2) + (p_{i+2} * 1)}{9} \quad (4.30)$$

Conseguimos que los cambios de color no sean tan bruscos, ya que contamos también con los de su alrededor. La fórmula para calcular la probabilidad es la siguiente:

$$prob_{dist\_Manhattan\_suavizada} = \frac{numero\_pixeles\_iguales_{imag\_teo\_suavi}}{320} \quad (4.31)$$

3. *Distancia Manhattan sin blancos*: Consiste en ir comparando píxel a píxel la imagen teórica con la imagen de la cámara, sólo si el píxel de la imagen teórica no es blanco. La fórmula para calcular la probabilidad es la siguiente:

$$prob_{dist\_sin\_blancos} = \frac{numero\_pixeles\_color\_iguales}{numero\_pixeles\_color_{imagen\_teorica}} \quad (4.32)$$

4. *Distancia Manhattan sin blancos con suavizado*: Se parece mucho a la *Distancia Manhattan sin blancos* pero aplicamos a la imagen teórica el mismo suavizado que aplicamos en la distancia Manhattan y usaremos esta fórmula para calcular la probabilidad:

$$prob_{dist\_sin\_blancos\_suavi} = \frac{numero\_pixeles\_color\_iguales_{imag\_teo\_suavi}}{numero\_pixeles\_color_{imag\_teo\_suavi}} \quad (4.33)$$

5. *Distancia de Mahalanobis*: Consiste en aplicar la función de distancia de Mahalanobis, en este caso sobre la imagen con la siguiente ecuación:

$$dMa_{pixel} = \sqrt{\left(\frac{R_{cam} - R_{teo}}{\sigma_1}\right)^2 + \left(\frac{G_{cam} - G_{teo}}{\sigma_2}\right)^2 + \left(\frac{B_{cam} - B_{teo}}{\sigma_3}\right)^2} \quad (4.34)$$

Donde R,G,B son los valores de cada color del píxel y $\sigma_i$ es la varianza de la componente i de los datos.
Debemos normalizar para que el valor esté entre 0 y 1, y para ello como hicimos con el láser dividimos la distancia que nos ha dado entre la máxima posible.

$$prob\_dMa = \frac{\sum_{i=0}^{320} dMa_{pixel}}{(\sqrt{3} * 4) * 320} \quad (4.35)$$

6. *Distancia euclídea*: Consiste en aplicar la función de distancia euclídea. Esta función determina la similitud aplicando la distancia ordinaria en un espacio euclidiano con la siguiente fórmula:

$$dEu_{pixel} = \sqrt{(R_{cam} - R_{teo})^2 + (G_{cam} - G_{teo})^2 + (B_{cam} - B_{teo})^2} \tag{4.36}$$

De nuevo debemos normalizar para tener el valor entre 0 y 1 dividiendo el resultado obtenido ente el máximo posible.

$$prob\_euclidea = \frac{\sum_{i=0}^{320} dEu_{pixel}}{(\sqrt{3} * 255) * 320} \tag{4.37}$$

7. *Coeficientes de correlación*: Consiste en aplicar la misma función que usamos para el láser, pero sobre el espacio RGB de la imagen con las siguientes fórmulas

$$prob\_R = \frac{vector\_R\_camara \Delta vector\_R\_teorica}{|vector\_R\_camara| * |vector\_R\_teorica|} \tag{4.38}$$

$$prob\_G = \frac{vector\_G\_camara \Delta vector\_G\_teorica}{|vector\_G\_camara| * |vector\_G\_teorica|} \tag{4.39}$$

$$prob\_B = \frac{vector\_B\_camara \Delta vector\_B\_teorica}{|vector\_B\_camara| * |vector\_B\_teorica|} \tag{4.40}$$

$$\tag{4.41}$$

Y luego juntamos las probabilidades con la siguiente fórmula:

$$prob\_correlacion = prob\_R * prob\_G * prob\_B \tag{4.42}$$

Las funciones basadas en objetos en vez de comparar las imágenes por píxeles lo que hacen es comparar la abstracción objeto. Un objeto es un conjunto de píxeles contiguos y con el mismo color, este tipo de funciones es más laborioso ya que requiere la inferencia de los objetos a partir de las imágenes. Con este método hemos usado dos funciones:

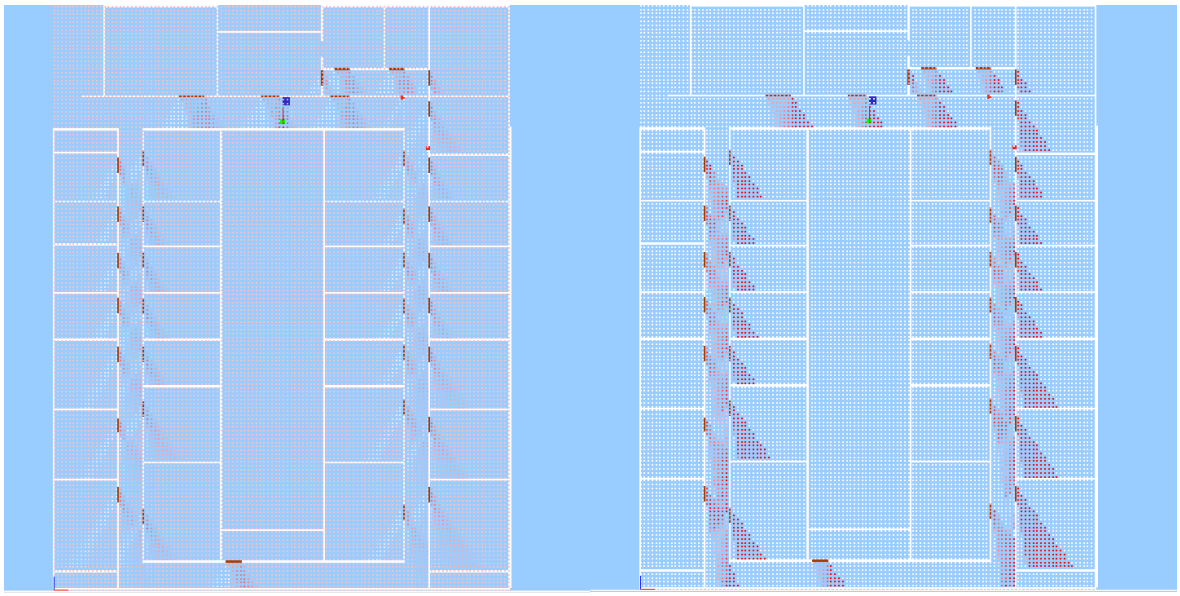
1. *Distancia con cola*: Esta técnica ya se usó en el proyecto [López, 2005] y en éste la hemos retomado para comprobar qué tal funciona en combinación con el láser. Lo primero que se calcula son los objetos que contiene la imagen de la cámara resumida. El segundo paso es comparar los objetos de la imagen de la cámara resumida con la imagen teórica, mirando si los píxeles del objeto de la imagen de la cámara resumida tienen el mismo color que los píxeles de la imagen teórica. El último paso es saber cuál es la distancia, para ello tenemos los dos casos, si no tenemos objetos diremos que la probabilidad es 0 y si tenemos objetos aplicamos la siguiente fórmula:

$$p_{x,y,\Theta/imagen} = \frac{1}{N} \sum_{i=0}^{N-1} p_i \cdot Peso_i \tag{4.43}$$

2. *Distancia sin cola*: Esta función es muy parecida a la *Distancia con cola*, pero sólo usa los objetos ignorando los espacios en blanco ya que no aportan información y realiza una comparación en dos direcciones de derecha a izquierda y al revés. Esta función es la que finalmente hemos escogido para el algoritmo y ya explicamos en la sección de teoria.

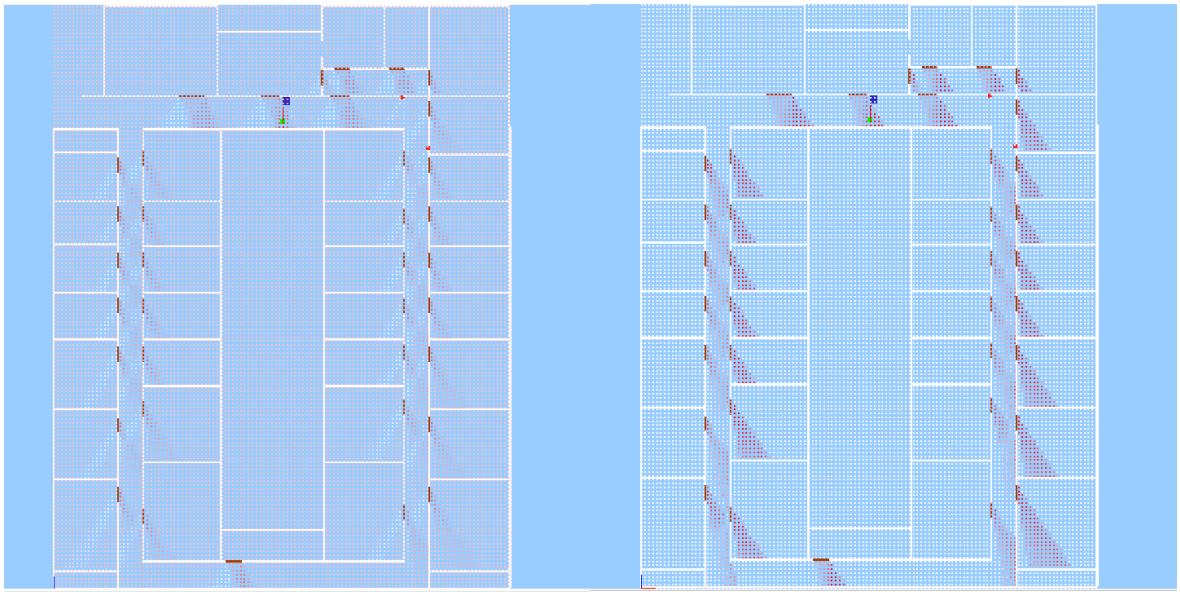
Al igual que con los experimentos realizados para el modelo de observación del láser debemos responder a dos preguntas:¿cómo son de discriminantes las funciones? y ¿dónde sitúan la máxima probabilidad?

Para responder a estas preguntas se realizaron diversas pruebas en varias posiciones del mundo al igual que con el láser. Aquí se muestra una prueba con los resultados más representativos. Primero comenzaremos analizando los resultados de las pruebas de distribución espacial de la probabilidad para poder determinar qué función es más discriminante.



Distancia Manhattan

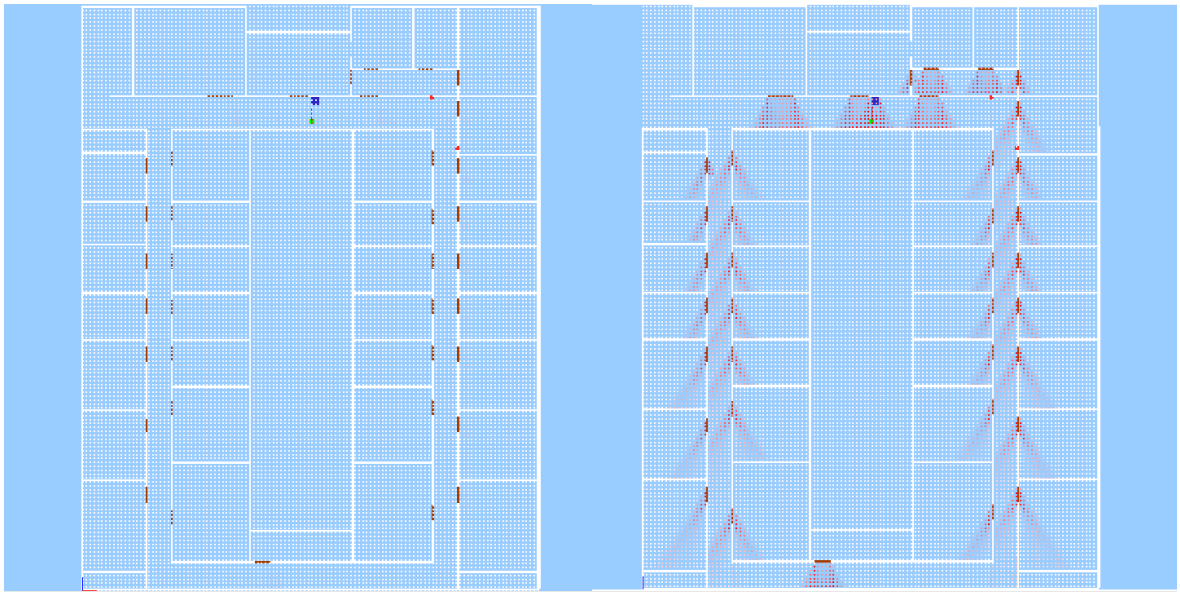
Distancia Manhattan sin blancos



Distancia Manhattan suavizada

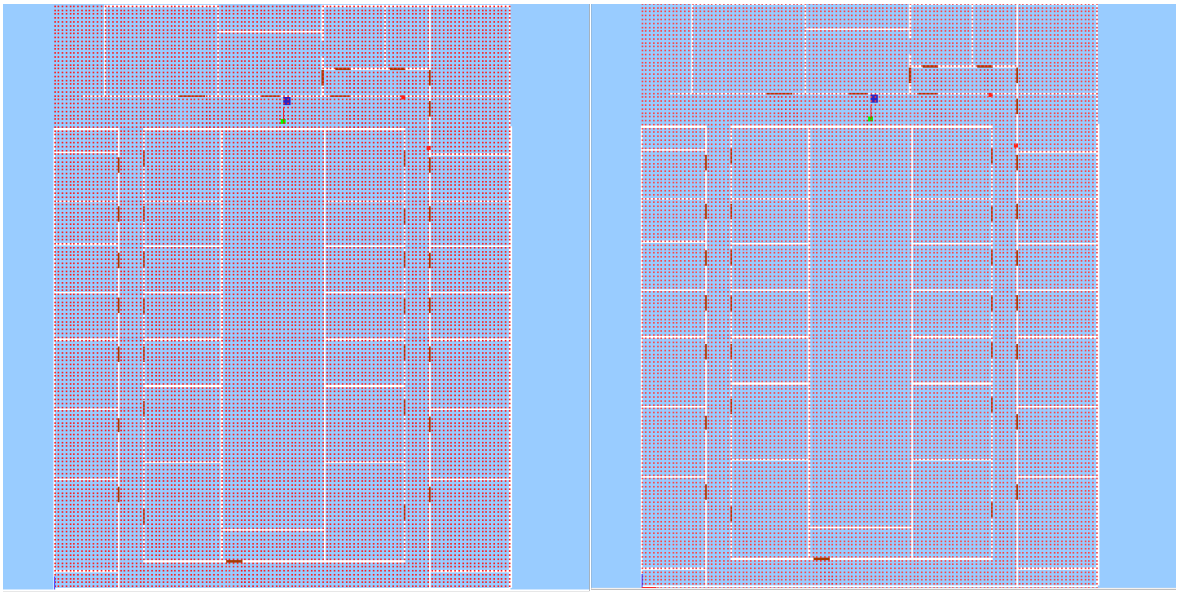


Distancia Manhattan sin blancos suavizada



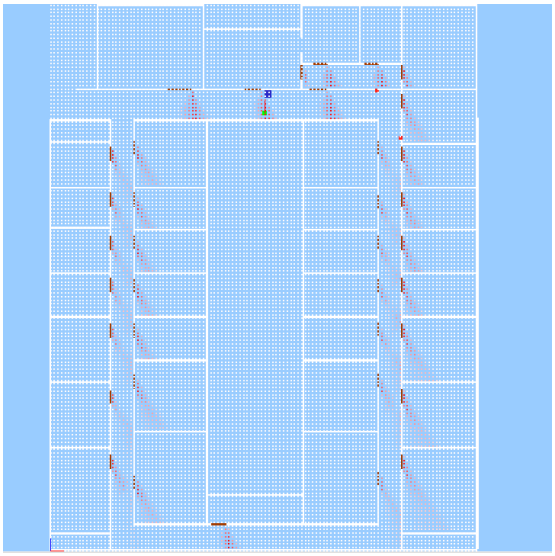
Distancia con cola

Distancia sin cola



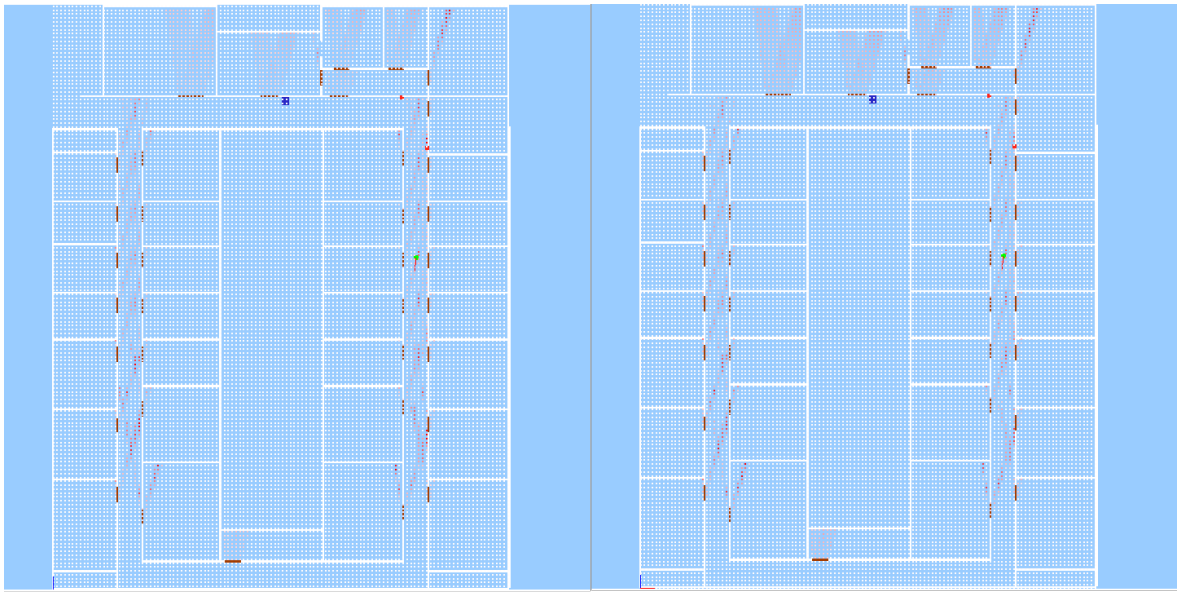
Distancia euclídea

Distancia Mahalanobis



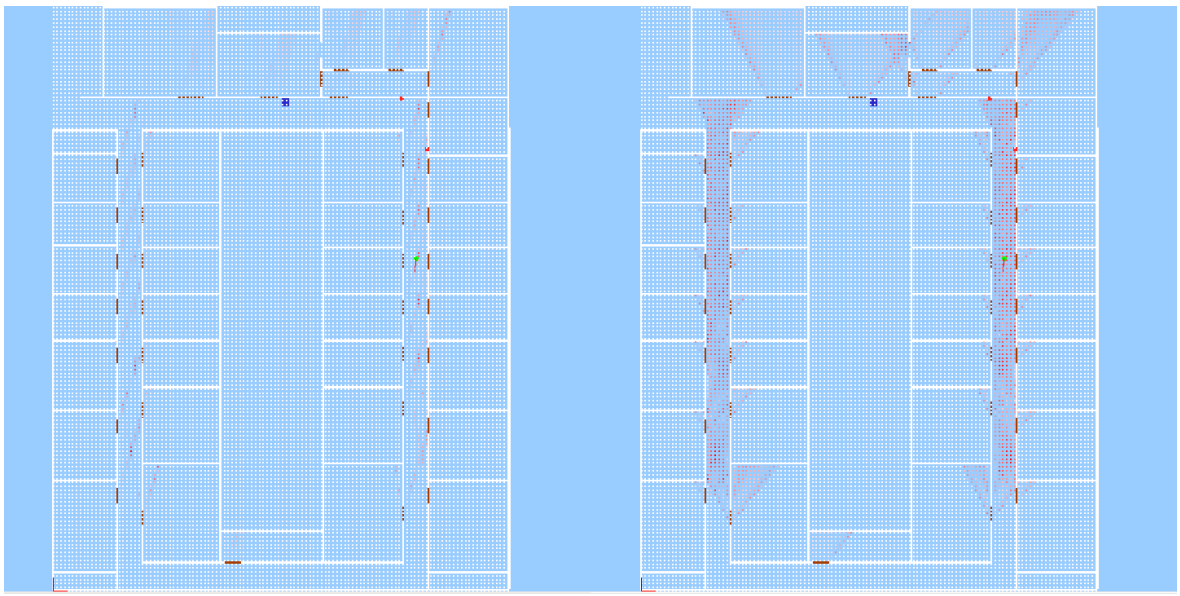
Coeficientes de Correlación

Podemos ver que la función más discriminante con diferencia es *Distancia con cola* que únicamente da un punto de alta probabilidad justo en la posición donde se encuentra el robot. Por otro lado tenemos las funciones *Distancia de Mahalanobis* y *Distancia Euclídea* que no son nada discriminantes, en este grupo también podemos meter las funciones *Distancia Manhattan* y *Distancia Manhattan suavizada* porque son muy poco discriminantes. Ninguna de las funciones mencionadas anteriormente nos vale porque buscamos una función que se encuentre en el término medio. Las demás funciones tienen características parecidas y para poder escoger una ellas probaremos en otro emplazamiento.



Distancia Manhattan sin blancos

Distancia Manhattan sin blancos suavizada

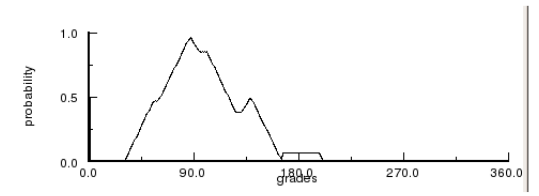


Coeficientes de correlación

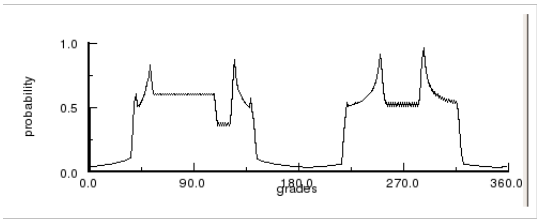
Distancia sin colas

Con la prueba de la figuras anteriores vemos que las funciones *Distancia Manhattan sin blancos* , *Coeficiente de Correlación* y *Distancia Manhattan sin blancos suavizada* son demasiado discriminantes y que la función *Distancia objetos sin cola* cumple los requisitos que queremos y por eso fue la que escogimos el último término. Además, en esta última prueba se puede apreciar muy bien cómo afecta un problema muy típico en la localización que es la simetría apareciendo con casi idéntica probabilidad las dos zonas de los pasillos.

Ya tenemos escogida una función que cumple el requisito de dispersión espacial, pero debemos comprobar cómo se comporta en los giros para responder a la segunda pregunta y ver si emplaza bien la máxima probabilidad.




Distancia sin cola papelera



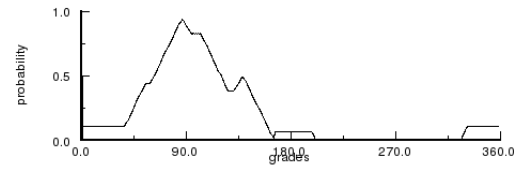

Distancia sin cola pasillo

En las pruebas representadas en la figura 4.6.1 podemos ver cómo en la prueba en frente de la papelera emplaza correctamente la probabilidad en 90º y en la prueba en la que el robot está situado en el pasillo aunque no emplaza correctamente la probabilidad más alta en 270º, sí tiene un valor bastante alto y hay que tener en cuenta que el pasillo es un lugar con muchísima simetría y por lo tanto es muy difícil estimar la probabilidad de manera perfecta.

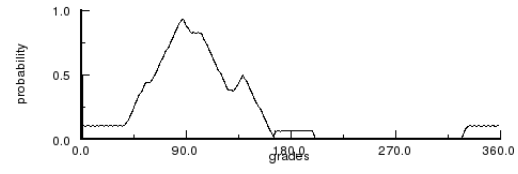
**• Efectos de la resolución sobre la cámara**

Para completar las pruebas con el modelo de observación visual experimentamos usando menos muestras a la hora de calcular la probabilidad, como ya hicimos con el láser.

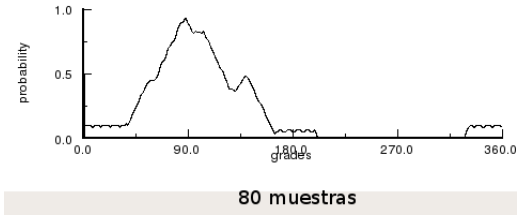
En las pruebas se cogieron 320 muestras, 160 muestras y 80 muestras respectivamente y se observó que la función con 80 muestras emplaza correctamente la probabilidad más alta, el ruido que se añade no entorpece a la función y sigue dando una probabilidad correcta en todo momento. Como se puede observar en las figuras a continuación:




320 muestras

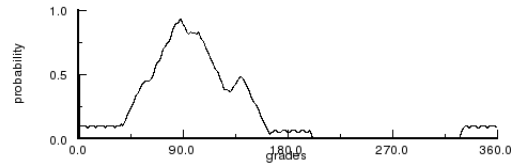



160 muestras

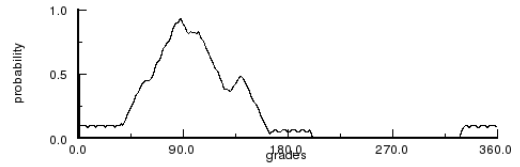



80 muestras

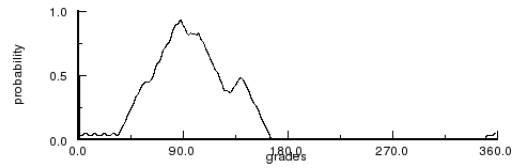
Además, en las pruebas de resolución probamos varias distancias 30, 15 y 3 metros como se ve en la figura 4.6.1 y de nuevo vemos que los resultados de la función y la estimación de la probabilidad no se ven demasiado afectados. Como valor de compromiso se escogió 30 metros ya que la diferencia temporal calculando 80 muestras no era muy grande y al tener más distancia siempre tendremos una observación más rica incluso con el robot en espacios más abiertos.




30 metros




15 metros




3 metros

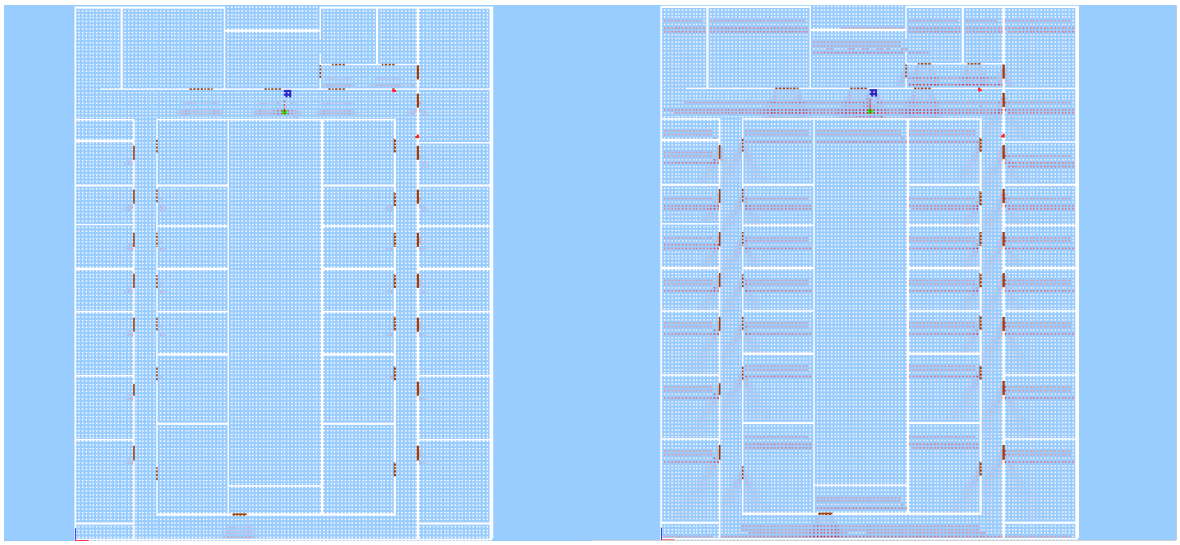
**• Combinación de los modelos de observación**

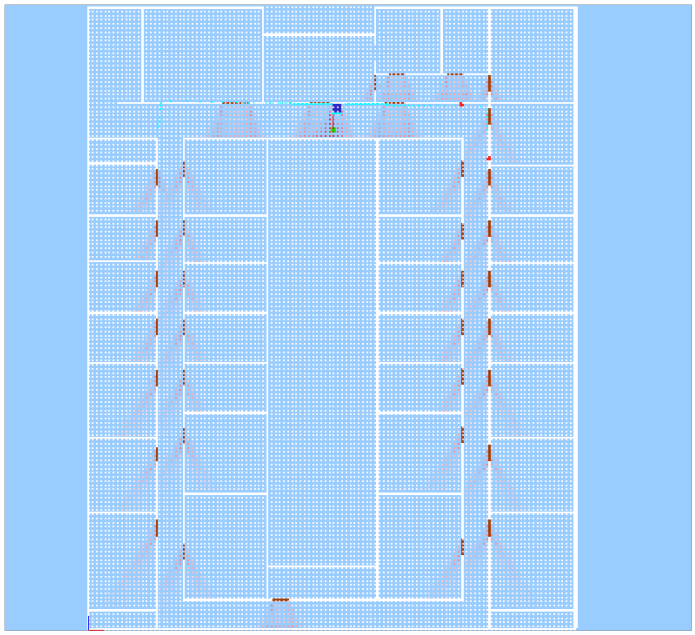
Antes de llegar a usar a la solución para combinar las dos probabilidades anterior se probaron tres maneras de hacerlo:

1. Condicionada (si vemos objetos usamos visión, si no vemos objetos usamos láser)
2. Media
3. Multiplicación

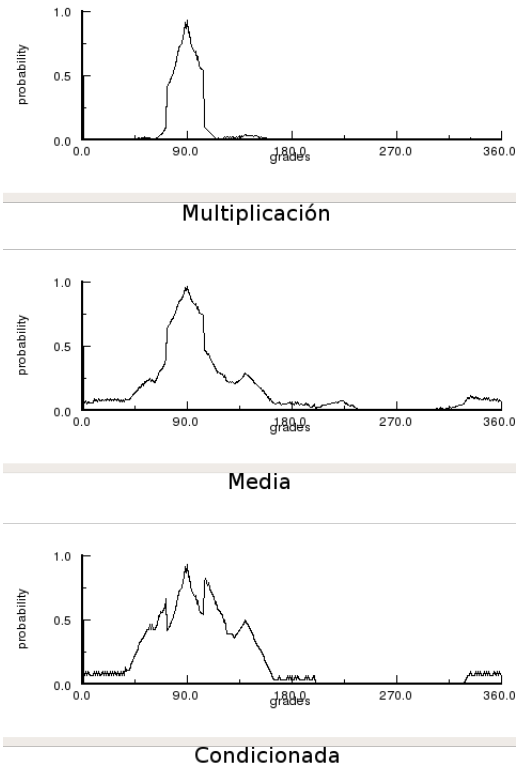
Con las siguientes pruebas contestaremos a las dos preguntas clásicas: ¿cómo son de discriminantes las funciones? y ¿dónde sitúan la máxima probabilidad?.

En la figura 4.6.1 se exponen la dispersión de probabilidad con la combinación de funciones de probabilidad. Lo que buscamos es una dispersión lo más discriminante posible, pero sin perder mucha información. Por ello en este caso nos interesa el resultado que nos da al unir las probabilidades mediante la multiplicación frente a las otras alternativas, que son menos discriminantes y las descartamos porque con esas obtenemos una menor precisión.




Para ratificar esta decisión sólo tenemos que ver cómo se comporta ante los movimientos angulares y para eso realizamos la siguiente prueba que se puede ver en la figura 4.6.1 donde se puede ver cómo la multiplicación se comporta de una manera correcta.




### 4.6.2. Precisión espacial

Para ver que el algoritmo funciona realmente bien se ha puesto a prueba desde diferentes posiciones en el mapa y se han observado los resultados obtenidos, errores máximos y mínimos tanto lineales como angulares. Para realizar las pruebas se ha usado el simulador *Gazebo* y se han realizado dos tipos de pruebas, sin ruido odométrico y con ruido odométrico para probar la robustez de la función de probabilidad.

Para poder comprender bien las pruebas hay que tener en cuenta dos cosas, la figura naranja representa la posición en (x,y,$\theta$) del robot real, en ella la orientación viene dada por una recta que sale del cuadrado que simula el cuerpo del robot, la figura negra representa la posición estimada por el algoritmo también en (x,y,$\theta$) y la orientación se representa de la misma manera con una línea negra. Por otro lado los puntos rojos representan la mayor probabilidad en cada posición (x,y) del mapa para cualquier orientación posible, cuanto más fuerte sea el color rojo más alta es la probabilidad en esa posición.

**• Ejecución típica sin ruido odométrico**

Una ejecución típica del algoritmo puede comenzar en una posición como es delante de la papelera y consiste en comprobar cómo una vez se ha conseguido estimar la posición del robot de manera correcta ésta se mantiene durante toda una vuelta al Departamental II, todo esto sin aplicar ningún tipo de ruido odométrico en el modelo de movimiento, simplemente usando la odometría perfecta que nos da el simulador.

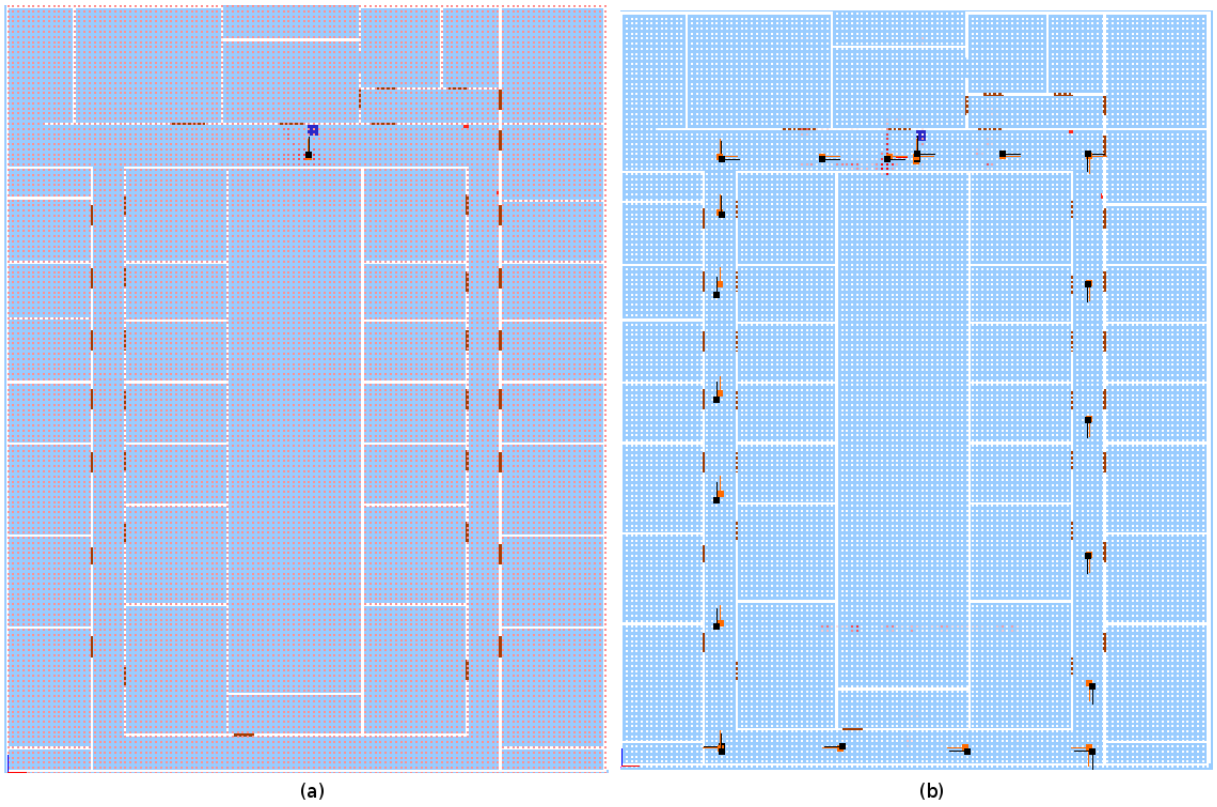



Figura 4.18: (a) Inicio de la vuelta, (b) Fin de la vuelta

En la figura 4.18 se puede ver en el instante inicial de la prueba (a) con la primera observación realizada, el apartado (b) de la imagen muestra el resultado final de la prueba después del recorrido. Como se puede observar, con esta prueba podemos ver que la probabilidad converge durante las sucesivas iteraciones y el algoritmo siempre es capaz de estimar una posición correcta. Con esta prueba podemos validar el modelo de observación que hemos escogido diciendo que nos permite estimar la probabilidad de una manera correcta y por lo tanto el modelo de observación es bueno.

**• Ejecución típica con ruido odométrico**

Una vez validado experimentalmente el modelo de observación sin ruido odométrico hace falta probar qué tal se comporta ante situaciones en las que la odometría presenta ruido ya que en el mundo real la odometría del robot está lejos de ser perfecta, debido a que se pueden dar fenómenos como deslizamientos o cambios de superficie lo que aporta un valor erróneo en los sensores odométricos. Para simular la imperfección de la odometría se ha hecho una función que a cada valor de la odometría dado por el simulador le añade un error cómo máximo del 10 % positivo o negativo como se ve en la siguientes ecuaciones:

$$\Delta r_x = \frac{\Delta m_x + \Delta m_x(\frac{Grid.resolucion}{2})}{Grid.resolucion} \pm 10\,\% \tag{4.44}$$

$$\Delta r_y = \frac{\Delta m_y + \Delta m_y(\frac{Grid.resolucion}{2})}{Grid.resolucion} \pm 10\,\% \tag{4.45}$$

$$\Delta r_\Theta = \frac{\Delta m_\Theta}{(\frac{umbral}{2})} \pm 10\,\% \tag{4.46}$$

De nuevo realizaremos una ejecución típica comenzando en frente de la papelera azul y dando una vuelta al Departamental II, esta vez con ruido en la odometría para poder compararla con la ejecución típica sin ruido odométrico.

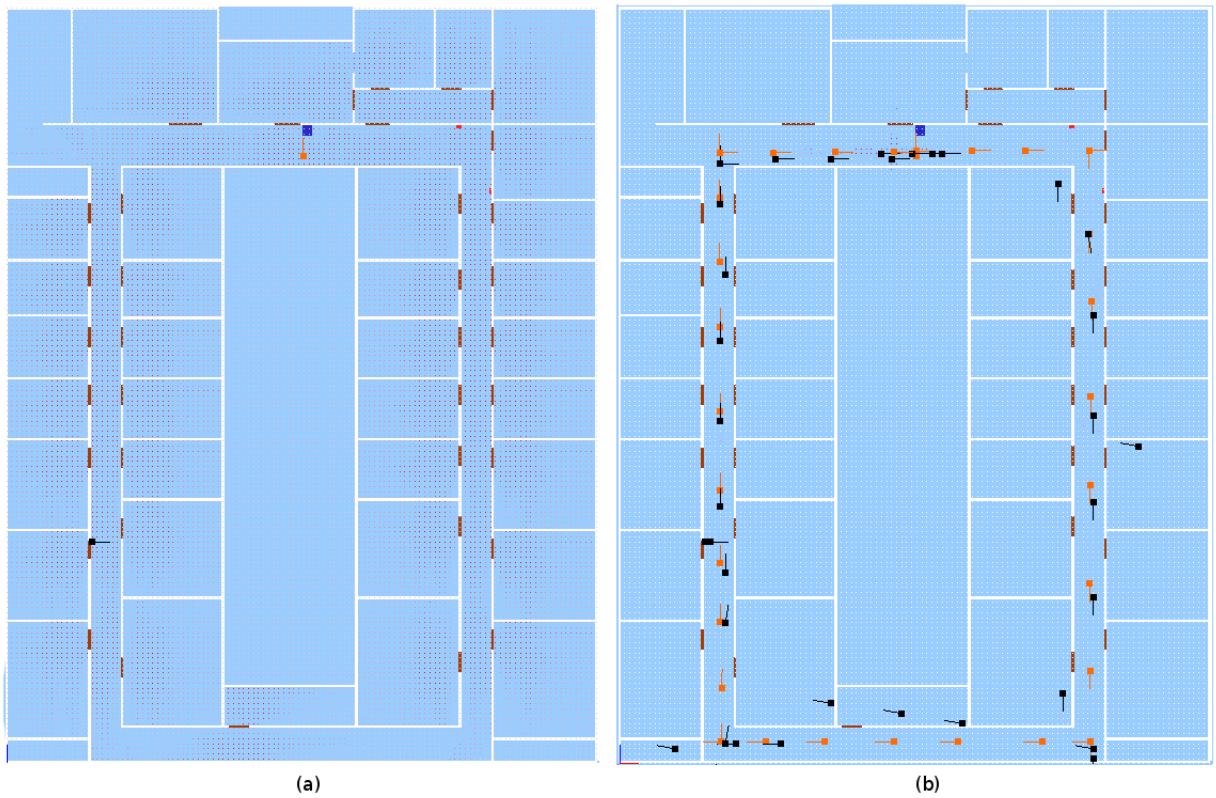



Figura 4.19: (a) Inicio de la prueba, (b) Fin de la prueba

Como se puede ver en la figura 4.19 en las primeras iteraciones el robot no se localiza. Pasadas unas cuantas iteraciones el algoritmo consigue una mejor estimación de la posición del robot, pero nunca con una buena precisión a causa del ruido odométrico incluso en algunas ocasiones el algoritmo estima posiciones alejadas de la localización correcta del robot. En la zona de abajo del mapa se produce un efecto deriva que hace que la posición estimada se aleje de la real debido al ruido odométrico angular. En el tramo final de la prueba la estimación de la posición del robot es más precisa aunque siempre tiene un error algo mayor que el recogido en la prueba con la odometría sin ruido.

A la vista de los resultados de esta prueba podemos decir que el algoritmo pierde precisión con el error odométrico, pero tampoco podemos decir que el algoritmo sea

malo ya que a pesar de que se producen fallos es capaz de recuperarse de ellos y acaba estimando una posición correcta.

Para ver cómo de preciso es el modelo de observación que hemos escogido y qué tal se comporta en movimiento analizaremos los errores en la estimación de las ejecuciones típicas con y sin ruido odométrico.

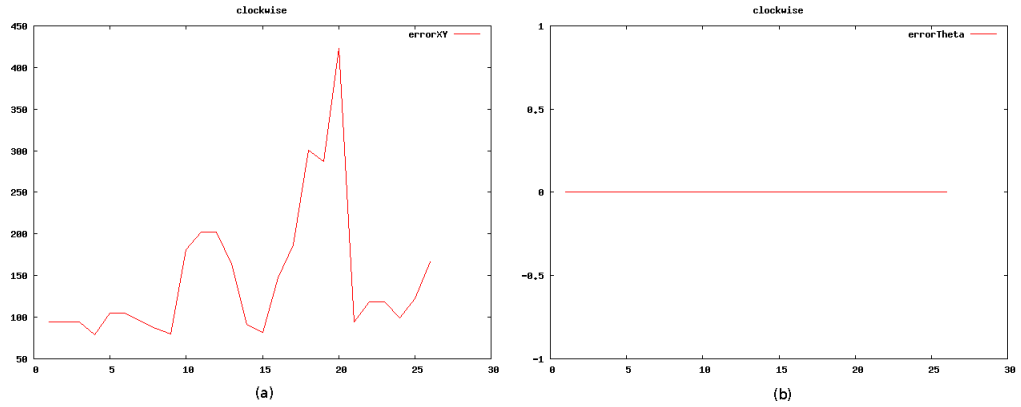



Figura 4.20: (a) Error lineal , (b) Error angular, ambos sin ruido odométrico

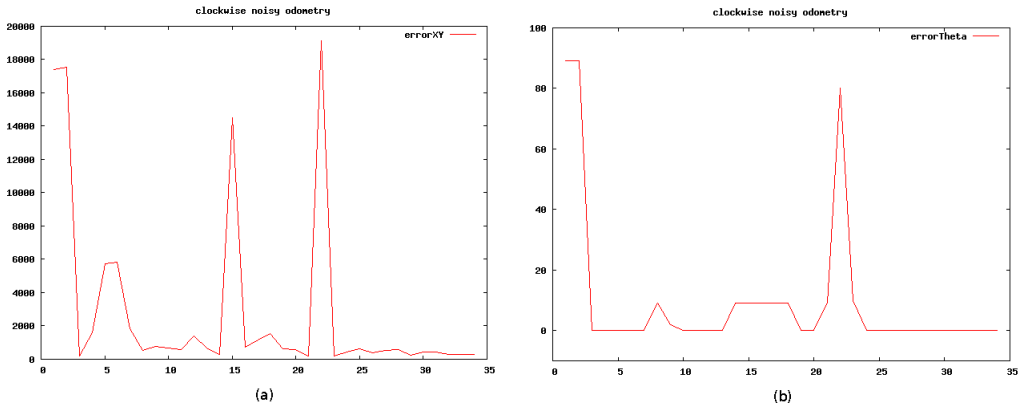



Figura 4.21: (a) Error lineal, (b) Error angular, ambos con ruido odométrico

Las figuras 4.20 y 4.21 muestran la evolución del error lineal en milímetros y del error angular en grados a lo largo de los experimentos sin ruido y con ruido en la odometría respectivamente.

La primera observación que se puede hacer a simple vista comparando las gráficas es que el algoritmo se ve afectado por el ruido odométrico, pero eso es normal ya que el ruido añade más dificultad a la hora de estimar la posición del robot. También podemos ver que el algoritmo es capaz de recuperarse de los errores debidos al ruido odométrico. Por lo tanto podemos decir que tenemos un algoritmo robusto y que podemos usar para resolver la localización del robot.

Si nos centramos en la precisión que conseguimos, tenemos que en el mejor de los casos (sin ruido odométrico) el error espacial medio en una ejecución típica es de 147 mm alcanzando un máximo de 423 mm y con un error angular medio de 0º. Con estos

datos podemos decir que el algoritmo localiza con una precisión muy buena al robot teniendo en cuenta la gran extensión del mundo.

Si observamos los datos para el caso con ruido odométrico vemos que al principio de la prueba el error es muy grande pero decrece con el paso de las iteraciones teniendo un valor medio al final de la prueba de 231 mm para el error lineal y 0º para el angular. Con esta prueba observamos que el ruido dificulta la localización, pero no la imposibilita por lo que podemos afirmar que el algoritmo es robusto frente al ruido.

### 4.6.3. Caracterización temporal del algoritmo

Una característica muy importante de un algoritmo de localización es el coste temporal que tiene ya que normalmente convive junto con otros algoritmos en ejecución en el robot con más prioridad, por ejemplo un algoritmo de navegación local que esquive obstáculos dinámicos. Por ello debe tardar el menor tiempo posible, muy cercano al tiempo real, para dejar el máximo tiempo en activo a los comportamientos más prioritarios.

El algoritmo propuesto para validar el modelo de observación y la función que calcula la probabilidad no se puede usar junto con otros algoritmos ya que cada iteración del mismo tarda aproximadamente 1 minuto y 50 segundos. Con estas cifras una ejecución de los casos típicos como los que se han mostrado en las pruebas anteriores con unas 35 iteraciones puede tardar aproximadamente 1 hora. Por eso se vio la necesidad de usar otro método más ligero computacionalmente que se desarrolla en el siguiente capítulo.

### 4.6.4. Comportamiento ante la simetría

Para comprobar si el modelo de observación es lo suficientemente robusto debe enfrentarse a unos problemas clásicos que se dan en la localización de robots móviles. Uno de estos problemas clásicos es la simetría y se da cuando el algoritmo de localización da prioridades más altas a lugares del entorno parecidos a la ubicación real del robot pero alejados de la misma, es decir, el algoritmo se confunde y asigna una alta probabilidad a lugares parecidos al correcto pero que son erróneos.

Para ilustrar este efecto se realizó entre otras una prueba que se puede ver en la figura 4.22 donde el robot comienza en el pasillo de la izquierda, uno de los sitios con mayor simetría dentro del Departamental II, y se mueve hacia el pasillo pequeño de abajo, todo ello con ruido en la odometría.

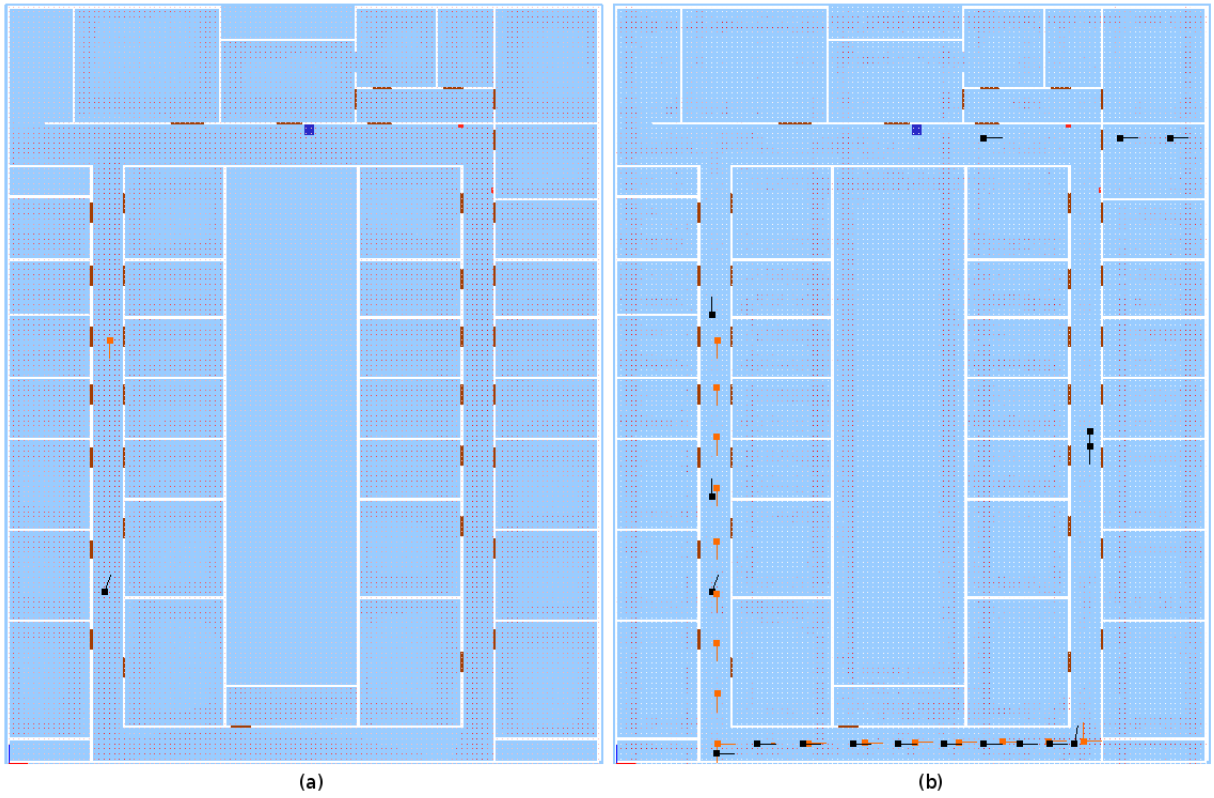



Figura 4.22: (a) Inicio de la prueba, (b) Fin de la prueba

En la figura 4.22 podemos ver cómo el robot comienza a avanzar por el pasillo y el algoritmo estima erróneamente su posición en el pasillo de la derecha, este error no se corrige hasta que no llega a la esquina de abajo y gira, es entonces cuando el algoritmo es capaz de corregir la estimación y romper la simetría existente decantándose por la posición correcta.

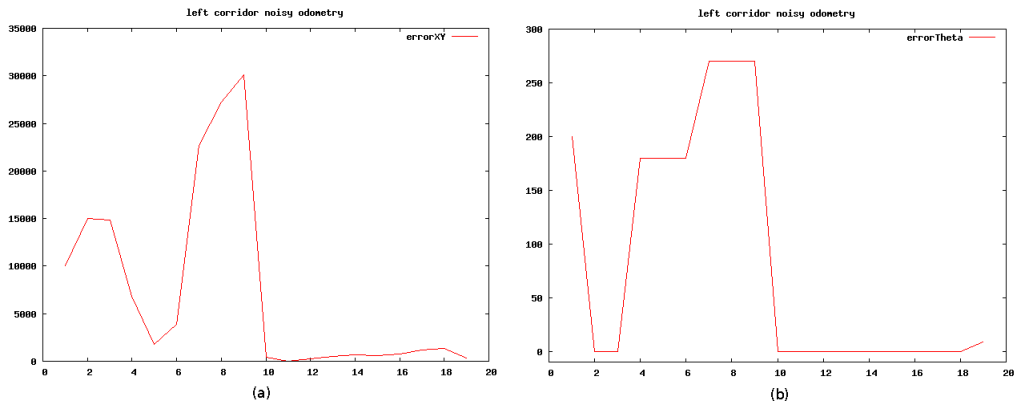



Figura 4.23: (a) Error lineal, (b) Error angular

En la figura 4.23 vemos los errores lineal y angular para esta prueba, donde se puede apreciar de una forma muy clara la forma que tienen las gráficas de error cuando se producen simetrías. En un principio tenemos un error alto que se corrige en una determinada iteración y a partir de ese momento el algoritmo vuelve a estimar

la posición correcta con unos valores del error más contenidos. Lo importante de esta prueba es observar que el algoritmo es robusto y capaz de recuperarse de las simetrías.

### 4.6.5. Comportamiento ante el secuestro

En el siguiente experimento probaremos el otro problema clásico al que deben enfrentarse los algoritmos de localización y es el secuestro. El problema consiste en que una vez localizado el robot se cambia la ubicación del mismo en el entorno sin que los sensores odométricos sean capaces de percibir ese incremento de movimiento y por lo tanto no se podrá aplicar en el algoritmo de localización. Por ello el algoritmo debe corregir la estimación errónea apoyándose sólo en el modelo de observación que debe corregir esta situación con el paso de las iteraciones. De nuevo la prueba que se comenta contiene el error en los sensores odométricos lo que añade aún más dificultad.

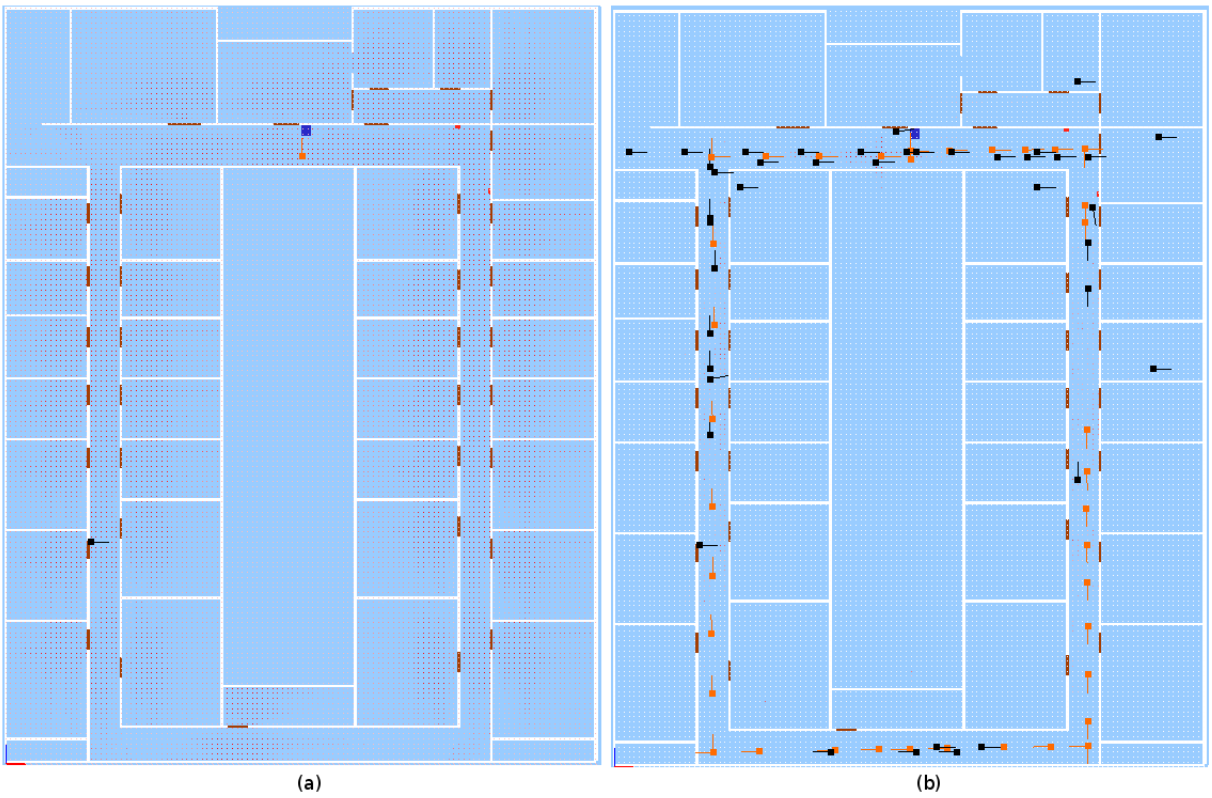



Figura 4.24: (a) Inicio de la prueba, (b) Fin de la prueba

Como se puede ver en la figura 4.24, el robot ha comenzado una vuelta en sentido horario al Departamental II y lo hemos secuestrado después de dos iteraciones en el pasillo derecho avanzándolo hasta la mitad del pasillo donde se ha vuelto a poner en marcha el algoritmo. Se puede ver cómo el algoritmo pasa de estimar la posición correcta a perderse por completo debido al secuestro, según van avanzando las iteraciones se van corrigiendo las probabilidades, pero el algoritmo no se recupera del secuestro hasta la parte final de la prueba. Que el algoritmo tarde tanto tiempo en recuperarse no se debe a otro motivo que no sea la baja densidad de iteraciones y por lo tanto esta recuperación se produciría mucho más rápido si se pudieran ejecutar las iteraciones más rápido.

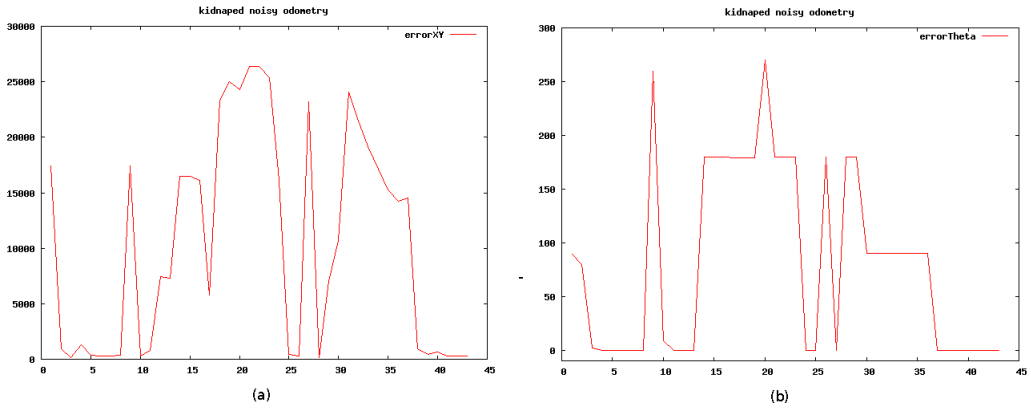



Figura 4.25: (a) Error lineal, (b) Error angular

En la figura 4.25 vemos las gráficas del error que muestran un error inicial y la posterior estimación correcta de la posición y orientación del robot durante unas cuantas iteraciones. También podemos ver de forma clara cómo aumenta el error al producirse el secuestro y cómo el algoritmo se recupera del secuestro al final cuando se reduce el error dentro de las cotas asumibles. Una vez vistos estos datos podemos decir que el algoritmo también pasa la segunda prueba clásica de la localización y por lo tanto podemos decir que es lo suficientemente robusto como para intentar llevarlo al tiempo real.

# Capítulo 5
# Algoritmo genético

En el capítulo anterior describimos el algoritmo de Mallas de Probabilidad y vimos que no era apto para ser usado en tiempo real. Por eso en este capítulo exponemos como alternativa el algoritmo genético, por qué nos decidimos por este tipo de algoritmos, cómo lo usamos y por supuesto también los experimentos que prueban las mejoras introducidas por este método.

## 5.1. Fundamentos teóricos

Los algoritmos genéticos son un subtipo de los algoritmos metaheurísticos. Un algoritmo metaheurístico se aplica cuando no existe un algoritmo específico para resolver el problema o bien como en nuestro caso cuando el algoritmo no es capaz de resolver el problema en un tiempo asumible. Una metaheurística es un proceso iterativo maestro que guía y modifica las operaciones de una heurística subordinada para producir eficientemente soluciones de alta calidad. Las metaheurísticas pueden manipular una única solución completa (o incompleta) o una colección de soluciones en cada iteración. La heurística subordinada puede ser un procedimiento de alto (o bajo) nivel, una búsqueda local, o un método constructivo.

Un algoritmo genético es un método de búsqueda dirigida basada en la salud de los individuos, nosotros nos referiremos indistintamente a salud y probabilidad ya que manejaremos la salud como una probabilidad acotada entre 0 y 1 para poder reutilizar el modelo de observación visto en el capítulo 4. Bajo una condición muy débil (que el algoritmo mantenga elitismo, es decir, que guarde siempre al mejor elemento de la población sin hacerle ningún cambio) se puede demostrar que el algoritmo converge en probabilidad al óptimo. En otras palabras, al aumentar el número de iteraciones, la probabilidad de la población óptima tiende 1. El funcionamiento de un algoritmo genético es el siguiente: hace evolucionar a una población de individuos (en nuestro caso posiciones tentativas del robot) sometiéndolos a unos ciertos patrones aleatorios (la posición y orientación en el mundo) y seleccionando los mejores individuos (para nosotros los mejores individuos serán los que estén en una posición más parecida a la del robot real).

## 5.2. Diseño general del algorimo genético

En este proyecto usaremos un *algoritmo genético* ya usado en el Grupo de Robótica en proyectos como [Perdices, 2010] para realizar una búsqueda de la posición del robot sobre el mundo adaptándolo a los requisitos específicos del problema de localización. En nuestra implementación cada individuo es evaluado por su función de salud que viene dada por el valor que devuelva el modelo de observación para ese individuo. Con la función de salud podemos determinar que individuos son los mejores para evolucionar a partir de ellos.

El algoritmo consta de los siguientes componentes:

- *Individuos*: Los individuos son análogos a las posiciones individuales dentro del cubo de probabilidad y simulan la situación de un hipotético robot dentro del mundo. Guardan la posición del robot $(X, Y, \theta)$, la probabilidad obtenida de la función salud, y un parámetro para indicar si es elitista.

- *Razas*: Cada raza es un conjunto de individuos que representan una posible solución a la localización del robot real en el mundo. En el algoritmo existen varias razas que compiten entre sí para ser la ganadora en una iteración determinada, la raza ganadora es considerada la posición actual del robot.

  Los parámetros almacenados en la raza son la posición del robot $(X, Y, \theta)$, la salud de la raza y el número de victorias.

- *Exploradores*: Individuos independientes encargados de buscar posiciones en las que generar nuevas razas.

- *Explotadores*: Cada uno de los individuos que forman parte de una raza, encargados de analizar en profundidad una posible posición y sus alrededores en busca de la mejor solución.

El algoritmo tiene dos tipos de iteraciones bien diferenciadas ,como se puede ver en la figura 5.1, cuando se lanzan exploradoras y cuando sólo se usan las explotadoras para verificar que la posición y orientación son correctas.

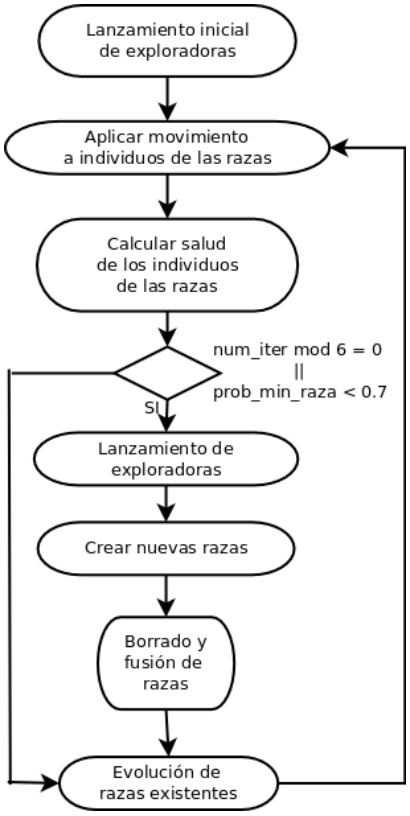



Figura 5.1: Flujo ejecución algoritmo genético

A continuación vamos a describir algunos de los pasos más importantes realizados en cada iteración:

- Cálculo de salud: Recogemos los datos de los sensores no específicos y obtenemos la información de posición, como vimos en el capítulo 4.
- Creación de exploradores: Creamos y dejamos evolucionar nuevos exploradores para buscar posiciones dentro del entorno general en las que crear nuevas razas. Este proceso lo veremos en detalle más adelante.
- Gestión de razas: Creamos, fusionamos o eliminamos razas en función de la posición y de la salud de éstas.
- Evolución de razas: Se crean los explotadores de la raza o, en caso de ya existir, se hacen evolucionar mediante operadores genéticos.

  En el caso de que el robot se mueva, también aplicamos un operador de movimiento a cada una de las razas al inicio de cada iteración. Este operador de movimiento se explica más adelante.

En las siguientes secciones se explican con más detalle los pasos del algoritmo, además también se explican los cambios en el modelo de movimiento.

### 5.2.1. Creación de exploradoras

En un principio la creación de exploradoras se realizaba de forma aleatoria por todo el mapa, pero pronto vimos que suponía demasiado coste computacional y que se exploraban zonas del mapa en las que el robot nunca iba a estar. En la figura 5.2 se puede ver las exploradoras lanzadas de forma aleatoria (a) y las exploradoras creadas con la alternativa escogida para el algoritmo (b).

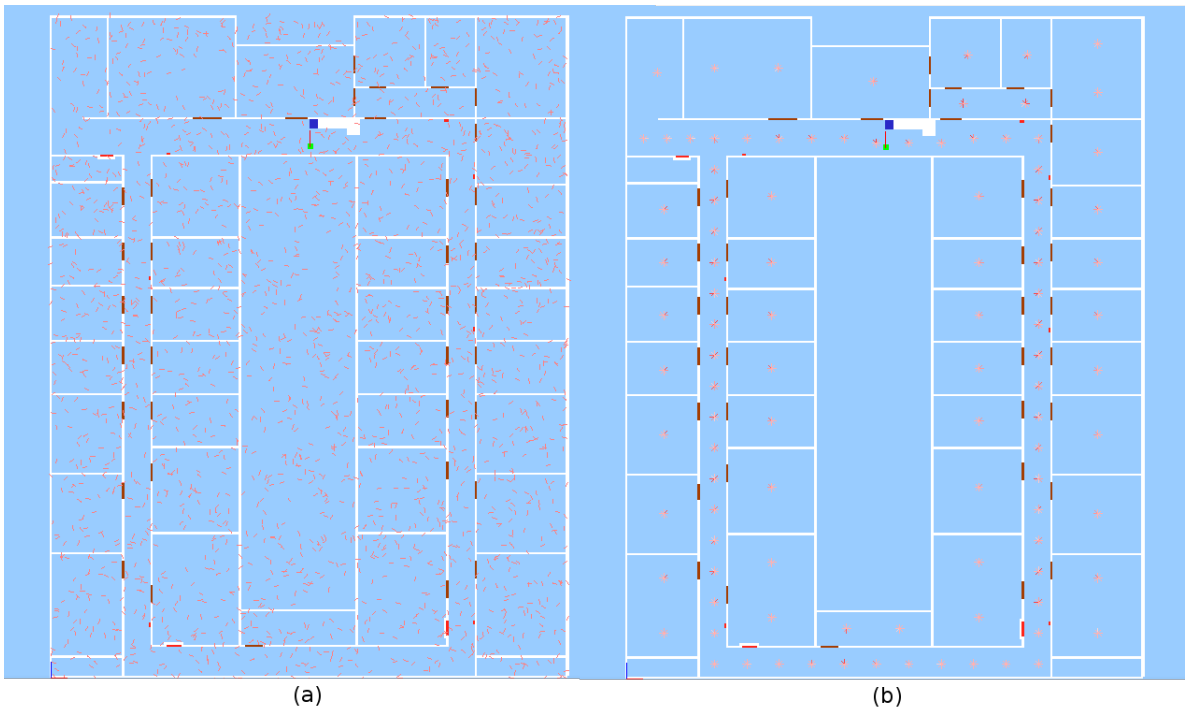



Figura 5.2: Comparación generación exploradoras

Para que el barrido de las exploradoras tenga más posibilidades de alcanzar zonas prometedoras barriendo la mayor extensión posible optamos por crear un archivo de configuración en el que se deberán poner las posibles localizaciones en (x,y) donde se quiera tener un grupo de partículas exploradoras. La persona que defina el archivo simplemente debe tener en cuenta que cuantas más posiciones menor será el rendimiento del algoritmo, pero debe cubrirse la mayor parte del mapa posible para mejorar la precisión. En cada una de las posiciones se crearán 8 partículas, una por cada 45º, con la misma posición (x,y) quedando como se ve en la figura 5.3.

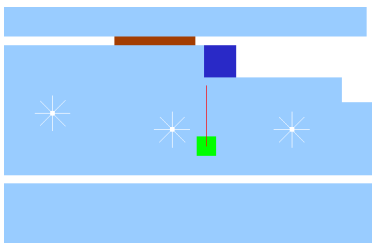


Figura 5.3: Ejemplo de generación de las nuevas exploradoras

Para mejorar aún más la parte de exploración, antes de crear las razas realizaremos una búsqueda local en 4 iteraciones de las mejores exploradoras. En cada una de las iteraciones generaremos unos nuevos grupos de partículas con forma de abanico y distribuidos en forma de cruz a partir de la partícula con alta probabilidad, como se puede ver en la figura 5.4. Con la forma de abanico conseguiremos afinar aún más la precisión de la orientación del robot y al distribuirlos en forma de cruz abarcamos el espacio colindante de la partícula de referencia para obtener más precisión en la posición (x,y).

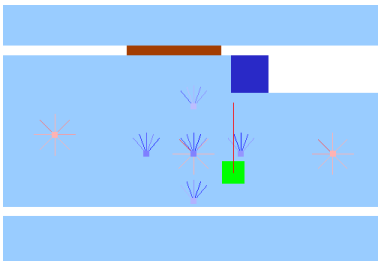


Figura 5.4: Cruz con los abanicos de partículas

La búsqueda local consta de 4 iteraciones en las que usamos las mejores 12 partículas de entre todas las disponibles en cada iteración, con ello lo que conseguimos es estimar la posición de las mejores partículas que luego crearán las razas más próximas a la situación del robot real. Se pueden ver las fases de estas iteraciones en la figura 5.5. Al final sólo queda un grupo de partículas alrededor de la posición más cercana a la del robot real.

Aún así el coste computacional de calcular las partículas exploradoras era bastante alto y por ello decidimos no lanzarlas en cada iteración del algoritmo sino cuando realmente fuese necesario. Para ello nos basamos en dos criterios: si han pasado 6 iteraciones del algoritmo sin lanzarlas o si la probabilidad máxima de las explotadoras es menor de 0.7, en esos dos casos lanzaríamos las exploradoras igual que hacemos en la iteración inicial.

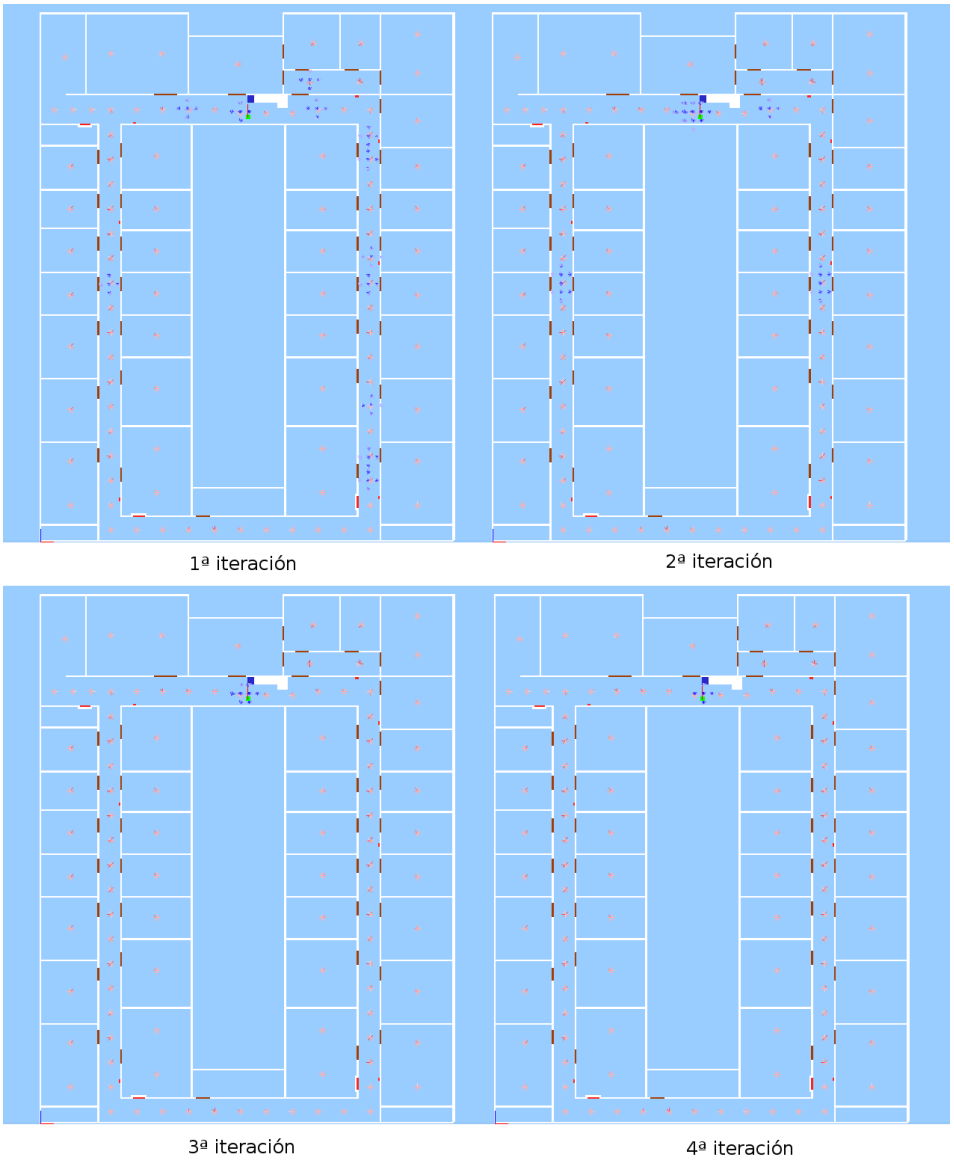



Figura 5.5: Iteraciones de la búsqueda local

### 5.2.2. Creación y selección de la raza ganadora

En esta sección se explican los criterios seguidos para crear nuevas razas a partir de las exploradoras y cómo se escoge la raza ganadora.

**• Creación de nuevas razas**

Una vez tenemos la salud de todas las exploradoras escogemos las que tienen un valor más alto de salud que pasan a ser las candidatas a generar una nueva raza de explotadoras. Para crear una raza de explotadoras se coge la exploradora de referencia y se generan a su alrededor de forma aleatoria las partículas que componen la nueva raza, de forma que se crea un grupo de partículas con similares características siguiendo los siguientes criterios:

1. Número de explotadoras por raza: 40
2. Radio de movimiento lineal con ruido gaussiano (x,y): 300mm
3. Radio de movimiento angular con ruido gaussiano ($\theta$): 40º
4. Número máximo de razas: 8

Todos estos valores han sido definidos de manera experimental para obtener unas razas que cubran una amplia superficie, pero sin ser demasiado grande y que cubran de manera eficiente el arco de visión. El número máximo de razas también fue definido como un valor de compromiso ya que con pocas razas se ignoran posibles localizaciones válidas y con muchas aumenta en gran medida la simetría. Un ejemplo de una raza generada a partir de una exploradora con un alta probabilidad sería como el que se ve en la figura 5.6.

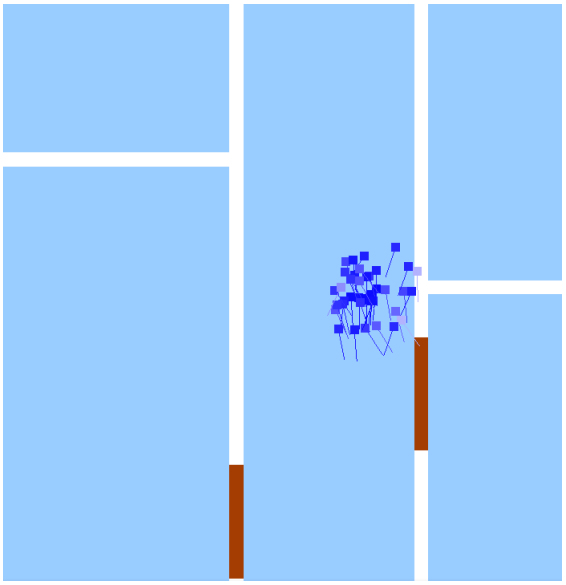


Figura 5.6: Ejemplo de grupo de explotadoras

**• Elección de la raza ganadora**

Para escoger la raza ganadora nos basaremos en un único criterio: tiene que ser la que más veces haya tenido la probabilidad más alta en el transcurso de las iteraciones. Para llevar esta cuenta usamos el parámetro *número de victorias* de las razas.

Cuando una raza es la que tiene mejor salud (probabilidad más alta) se suma una victoria al contador y cuando no es la mejor se resta una, de esta manera premiamos a la raza que sea ganadora durante el mayor número de iteraciones.

**• El efecto de la simetría**

En el algoritmo genético cobra especial protagonismo el efecto de un entorno altamente simétrico a la hora de estimar la localización del robot. A diferencia del

algoritmo con mallas de probabilidad en el que manteníamos la probabilidad de todas las posiciones del mapa y la simetría se resolvía por si sola con el paso de las iteraciones. En el algoritmo genético debemos tomar una decisión ya que es posible que la raza ganadora no sea adecuada porque durante unas iteraciones se encontraba en una posición parecida a la real, pero no la buena. Por ello, gracias al sistema de victorias que prima a la raza con probabilidad más alta y sanciona a las que tengan probabilidad más baja tenemos un indicador fiable de cuál es la posición válida conforme pasan las iteraciones. Esto nos sirve para evitar oscilaciones entre las razas ganadoras, como mecanismo de histéresis.

Tendremos simetría entre la raza ganadora y otra raza cuando el diferencial de las probabilidades de ambas sea menor de 0.2 como se ve en la ecuación :

$$haysimetria = |probganadora(t-1) - probganadora(t)| < 0{,}2 \tag{5.1}$$

Si nos encontramos en esta situación lo que haremos será aumentar el contador de victorias de la raza escogida como ganadora en esta iteración y decrementar el contador de la raza ganadora en la iteración anterior. La razón de seguir este criterio es que si es cierto que hay simetría, con el paso de las iteraciones la raza con la posición válida se impondrá sobre la que venía siendo ganadora que resultó errónea, y si no hay simetría con el paso de las iteración se impondrá la raza que venía siendo ganadora en las iteraciones anteriores ya que se romperá esa igualdad.

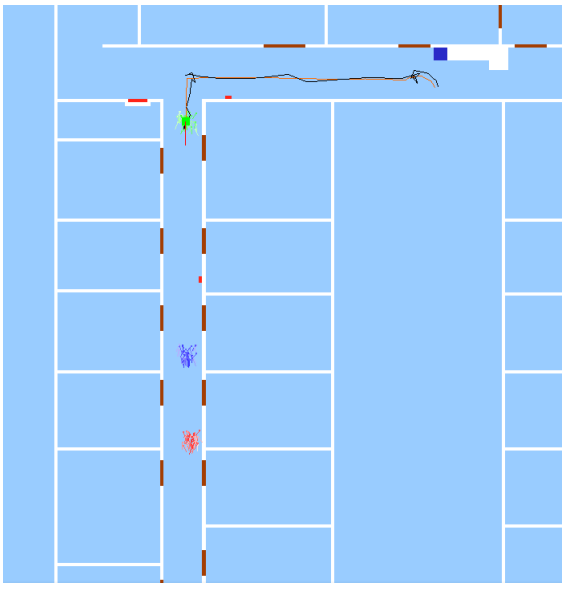


Figura 5.7: Ejemplo de razas simétricas

Podemos ver un ejemplo de simetría en la figura 5.7 en el que aparecen la raza ganadora en color verde y la posible simetría en color rojo. Como se puede ver, hasta ese momento la raza ganadora coincide perfectamente con la posición del robot, es decir, la localización de la raza ganadora es óptima, pero debido al entorno altamente

simétrico, como son los pasillos, surgen dudas con la posición que representa la raza roja, pero las dudas se despejarán a lo largo de las siguientes iteraciones.

La posibilidad de mantener varias posiciones simultaneas con alta salud y el método de cálculo de simetrías aportan al algoritmo genético una flexibilidad parecida a la que teníamos con el algoritmo de mallas de probabilidad y una recuperación ante secuestros bastante aceptable, pero además con el algoritmo genético obtenemos una velocidad de cómputo mucho más alta que con el cubo de probabilidad ya que el número de cálculos que debemos realizar se ha reducido de forma drástica.

### 5.2.3. Eliminación y fusión de razas

En esta sección se explica una parte muy importante de nuestro algoritmo, la parte que se ocupa de gestionar la eliminación y fusión de razas. Esta parte tiene mucha importancia porque que sólo podemos manejar un número limitado de razas en cada iteración.

**• Eliminación de razas**

La eliminación de razas no tiene otro propósito que descartar razas que tengan una baja salud para posibilitar la entrada en la ejecución del algoritmo a razas que tengan una salud mayor. En nuestro caso diremos que una raza tiene una salud muy baja cuando ésta sea menor de 0.2. Las razas que estén por debajo de este valor de sesgo serán eliminadas ya que no están situadas en una posición nada parecida a la que ocupa el robot real.

**• Fusión de razas**

La fusión de razas es necesaria para aglutinar razas que actúan en áreas muy cercanas del mundo de forma que si fueran reemplazadas por una sola raza no se notaría ninguna diferencia en los cálculos. Consideramos que dos razas son cercanas cuando una está a menos de 500mm de otra. En este caso se creara una nueva raza que reemplazará a las dos anteriores escogiendo como partícula de referencia la partícula de mayor probabilidad de las dos razas.

### 5.2.4. Modelos de movimiento y de observación para el algoritmo genético

El modelo de movimiento y el modelo de observación nos ayudan a la recogida de datos de los sensores no específicos y a materializar el movimiento del robot en los individuos de las razas.

**• Modelo de observación**

Usaremos exactamente el mismo modelo de observación que ya describimos en el capítulo4. El único cambio es que ahora la probabilidad que obtenemos con este modelo

de observación se corresponde con la salud de los individuos que usamos en el *algoritmo genético*.

• **Modelo de movimiento**

Para el modelo de movimiento partiremos de las ecuaciones 4.45 planteadas en el capítulo anterior y que corresponden a las ecuaciones del movimiento aplicando un ruido en la odometría para obtener unos resultados más cercanos a la realidad.

Tanto el modelo de movimiento como el modelo de observación lo aplicaremos a partículas, los individuos de la población, que simularán posiciones en el mapa en las que se puede encontrar el robot y cada partícula tendrá su propia probabilidad dependiendo de su situación en el mapa que vendrá dada aplicando estrictamente el modelo de observación de la sección ya explicado. Para aplicar el modelo de movimiento debemos adaptarlo a las nuevas partículas modificando su posición (x,y) y su ángulo ($\theta$) según sea necesario, como se ve en la figura 5.8.

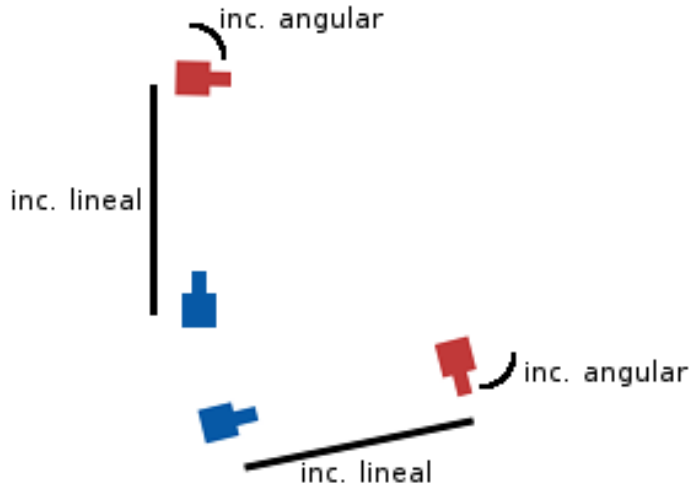



Figura 5.8: Ejemplo de movimiento de las partículas

### 5.2.5. El omnilaser

El concepto de omnilaser surgió para aumentar aún más la precisión a la hora de localizar al robot. Básicamente la idea es que si podemos obtener información sensorial de lugares por los que ya ha pasado el robot de forma reciente podremos afinar más la posición que tiene el robot en este instante. Quizás un ejemplo claro es que si el robot acaba de dejar una esquina y entra en el pasillo las zonas centrales del pasillo deberían dar poca probabilidad, pero la realidad es que debido a la simetría tenemos una probabilidad muy parecida en todo el pasillo. Ahora bien, si además de ver el pasillo desde un principio el robot pudiera ver que detrás aún tiene una esquina, claramente la probabilidad no sería igual en todo el pasillo y obtendríamos mayor precisión.

• **Cálculo del omnilaser**

El problema fundamental al que nos enfrentamos con el omnilaser fue cómo guardar la información de momentos anteriores proveniente del sensor láser del robot. Para solucionar este problema se implementó una memoria que almacena los datos provenientes del láser observado. En la figura 5.9 vemos los puntos amarillos que son un ejemplo de la memoria que contiene puntos almacenados en 360º y no sólo 180º como nos aporta el láser.

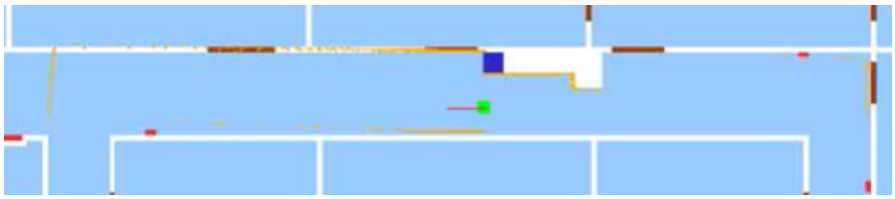


Figura 5.9: Ejemplo memoria laser

Para calcular el omnilaser usamos la misma técnica descrita en el capítulo4 para el cálculo del láser sobre el mapa, pero esta vez sobre la memoria de puntos y sobre 360º, así obtenemos un vector de 360 muestras con el valor del omnilaser observado. En la figura 5.10 podemos ver cómo se calcula el omnilaser (puntos rojos) sobre la memoria de puntos, en este ejemplo sólo tenemos datos para la parte delantera del robot por ello para las partes donde no hay memoria se establece por convenio la distancia máxima del láser.

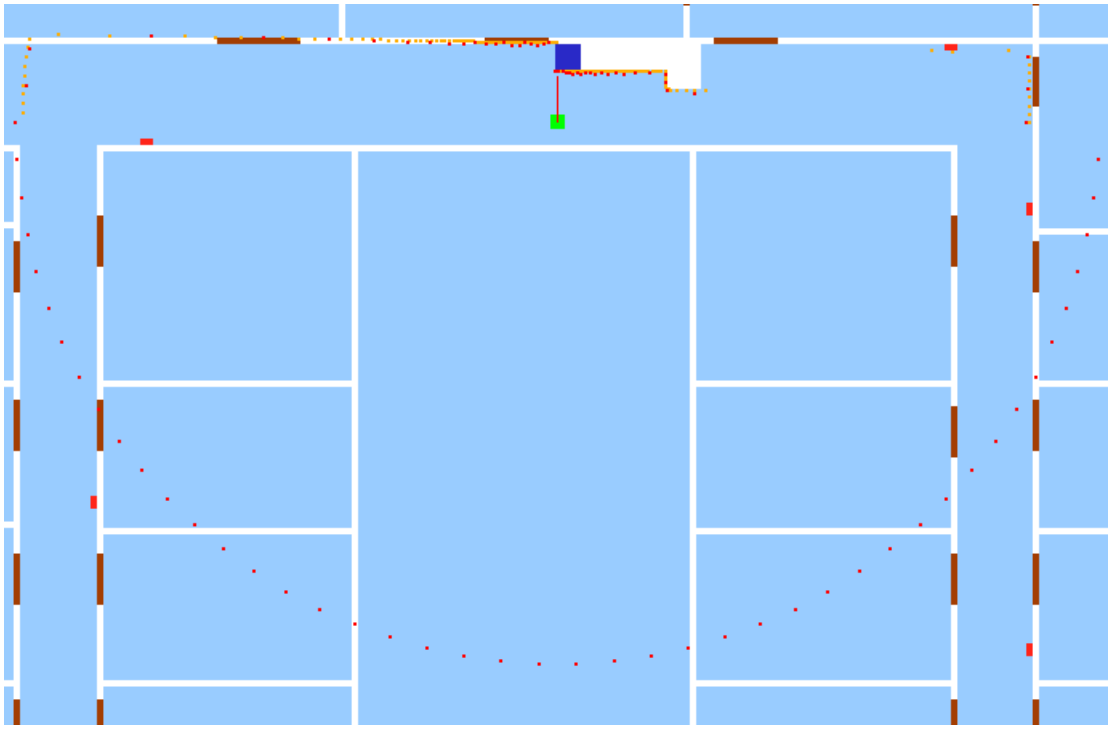


Figura 5.10: Ejemplo cálculo omnilaser sobre la memoria

Para recoger los datos del omnilaser teórico proveniente del mapa, simplemente hemos adaptado la técnica usada en el capítulo 4 para el cálculo del láser del mapa, pero en vez de hacerlo sólo para la parte delantera del robot lo hacemos también para la parte trasera del mismo. De esta forma obtenemos otro vector de 360 muestras con el omnilaser teórico. En la figura 5.11 se puede ver los valores arrojados por el cálculo del omnilaser teorico (puntos negros) sobre el mapa del Departamental II.

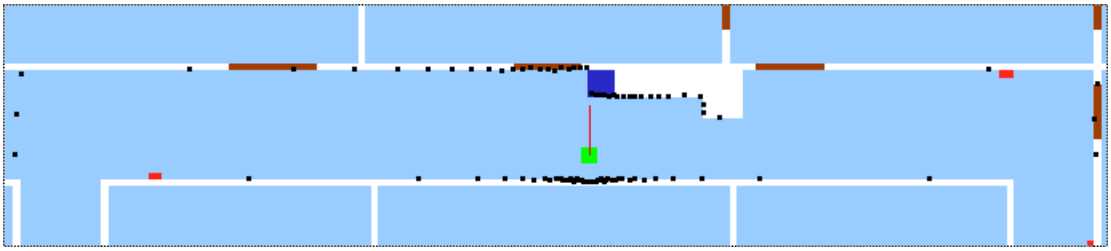


Figura 5.11: Ejemplo cálculo omnilaser teorico

Para obtener una probabilidad que poder asignar como salud a los individuos de la raza hemos usado la misma fórmula de la *Distancia de Manhattan* vista en el capítulo4, pero adaptada a las nuevas dimensiones de los vectores. Para mejorar la eficiencia en vez de usar las 360 medidas sólo se han usado 90 resultando la siguiente fórmula:

$$probLaser(x, y, \theta) = \frac{\sum_{i=1}^{90}(abs(laser\_real(i) - laser\_teorico(i)) < 20cm)}{90} \quad (5.2)$$

En el apartado de experimentos se detalla una de las pruebas realizadas con este nuevo modelo de observación para el láser junto con el modelo de observación para la cámara que ya usábamos anteriormente.

## 5.3. Experimentos

Como hicimos en el capítulo4 después de explicar la teoría y el algoritmo final que escogimos para desarrollar la tarea de localización del robot tenemos que validar las elecciones con pruebas que demuestren todo lo que se ha expuesto en la teoría. Por ello esta sección la dedicaremos a la explicación de las pruebas de precisión espacial, las caracterizaciones temporal y espacial y a ver cómo el algoritmo solventa los secuestros y las simetrías. Además, también someteremos al algoritmo a pruebas ante oclusiones esporádicas que introducen ruido sensorial.

Antes de comenzar a analizar los experimentos se debe explicar el código de colores y su significado para comprender bien las imágenes:

- El color verde corresponde a la raza ganadora, la que dicta la posición estimada del robot
- El color azul son razas con una probabilidad alta
- El color rojo corresponde a razas con una probabilidad muy parecida a la ganadora, razas simétricas

- La línea de color naranja es el camino que ha seguido el robot real durante la prueba
- La línea negra es el camino que ha seguido la raza ganadora durante la prueba, es decir la posición estimada del robot en cada instante

### 5.3.1. Precisión espacial del algoritmo final

La primera prueba que realizamos fue una ejecución típica del algoritmo para comprobar si converge a posiciones correctas con un error aceptable.

**• Precisión espacial del algoritmo final**

Como hicimos en el experimento anterior el robot comienza frente a la papelera azul en la parte superior del mapa y realiza una vuelta en sentido horario por todo el Departamental II.

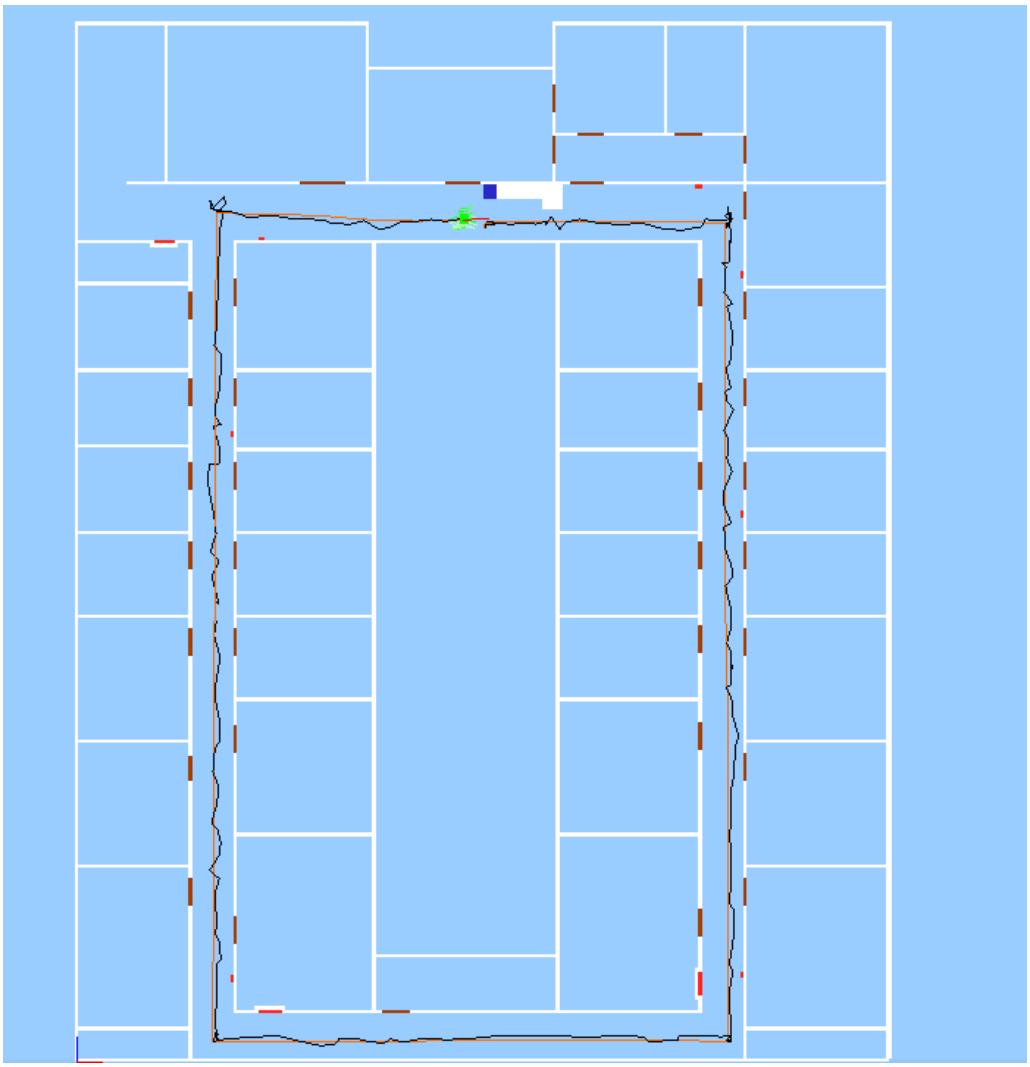


Figura 5.12: Resultado de la vuelta en sentido horario

Con un simple vistazo a la figura 5.12 vemos que el algoritmo mantiene en todo momento al robot localizado con bastante acierto y lo que es muy importante desde las primeras iteraciones. También se puede apreciar cómo la línea negra (posición estimada por el algoritmo) es más temblorosa en los pasillo debido a que la alta simetría dificulta la localización en estos lugares. A la vista de los resultados de la prueba podemos decir que el *algoritmo genético* es un método totalmente válido para ser aplicado en el problema de la localización con sensores no específicos ya que arroja unos resultados muy buenos sin verse afectado por el ruido odométrico.

Desde una perspectiva más formal analizaremos los errores en la localización tanto lineal como angular que obtuvimos en esta prueba y que se representan en la siguientes gráficas.

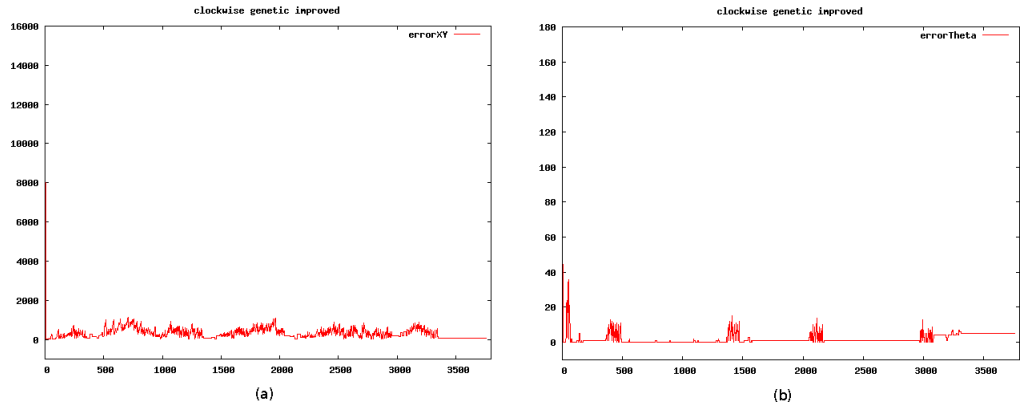



Figura 5.13: (a) Error lineal, (b) Error angular

En la figura 5.13 podemos comprobar de manera mucho más clara que el algoritmo localiza al robot desde las primeras iteraciones. Si miramos las gráficas podemos ver que ambos errores se encuentran en valores muy bajos con unos valores en media de 338.75 mm para el error lineal y 2.11º para el error angular. Estos valores nos dicen que el algoritmo localiza al robot con una precisión muy aceptable y dentro de los objetivos marcados, lo que refuerza aún más nuestra elección.

Una característica importante que se puede observar muy bien en la gráfica del error angular es el aumento del error angular cuando el robot realiza los giros. Cada unos de los cinco picos que se observan en la gráfica corresponde con uno de los giros que ha realizado el robot durante la prueba. Además podemos ver que el robot es capaz de recuperarse sin problemas ante los giros y los errores odométricos en los mismos.

**• Precisión espacial con exploradoras aleatorias**

En el experimento anterior usamos las exploradoras generadas en posiciones clave, que como comentamos en la sección de fundamentos teóricos, además de disminuir el tiempo de cómputo mejoraban la localización. A continuación veremos los resultados de la misma prueba anterior, pero usando las exploradoras aleatorias. El resultado de la prueba se puede ver en la figura5.14

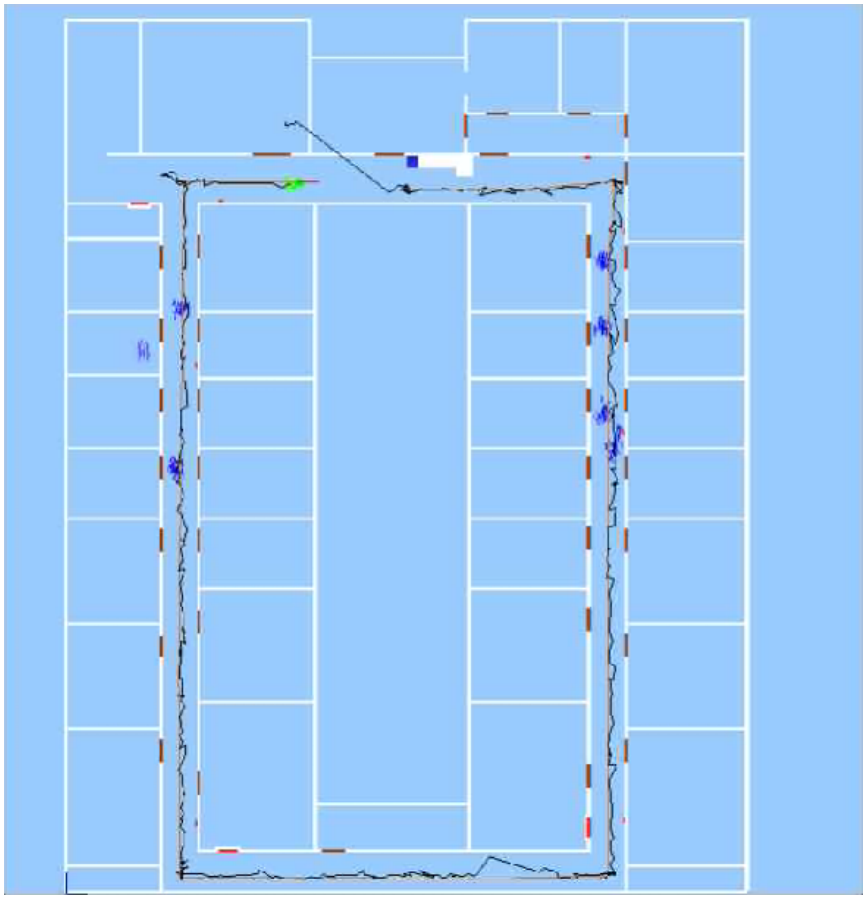


Figura 5.14: Resultado de la vuelta en sentido horario exploradoras aleatorias

Al comparar las dos pruebas podemos ver la mejora que introducen a la localización las exploradoras en posiciones claves. Al principio de la prueba debido a las exploradoras aleatorias tenemos una error en la estimación de la posición del robot ya que vemos cómo la línea negra no coincide con la naranja. Este error se corrige a medida que pasan las iteraciones y el algoritmo es capaz de recuperarse. Globalmente podemos ver que los resultados son parecidos en las dos pruebas aunque en la segunda con exploradoras aleatorias la línea negra es mucho más temblorosa lo que indica más imprecisión en la localización.

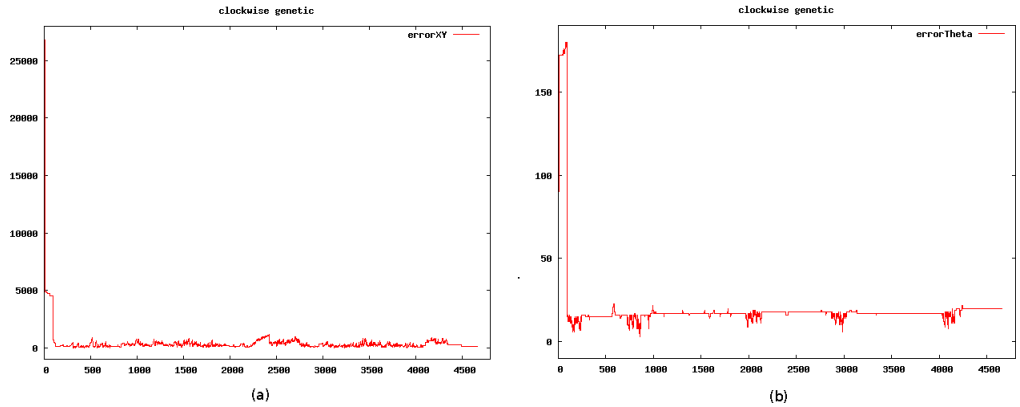



Figura 5.15: (a) Error lineal, (b) Error angular

En las gráficas de la figura5.15 se puede ver que los errores con las exploradoras aleatorias son más grandes. Concretamente el error lineal medio fue de 427.273 mm y el angular de 19.72º. El error lineal es sensiblemente más grande y podemos decir que asumible, pero el error angular no puede ser soportado ya que un error de cerca de 20º significa que el algoritmo no estima con la suficiente precisión la posición del robot. Este fue uno de los motivos por lo que escogimos como solución final el emplazamiento de las exploradoras en posiciones clave del mapa.

### 5.3.2. Caracterización temporal del algoritmo final

Como explicamos en la sección de teoría, el algoritmo tenía dos tipos de iteraciones, cuando se usaban las exploradoras y cuando no. En las iteraciones en las que se usan las exploradoras tenemos un coste temporal de unos 600 ms y en las iteraciones en las que sólo hacemos tareas de mantenimiento de razas unos 100 ms como máximo. En cambio si usáramos las exploradoras aleatorias el tiempo de las iteraciones al lanzarlas aumentaría a 1 segundo, este es el otro motivo por el que desechamos esta solución para el algoritmo final.

Es valores son aceptables y podemos decir que son aptos para el tiempo real ya que el algoritmo de localización no necesita ser ejecutado en cada iteración del sistema sino que es flexible en este sentido. Por ello el *algoritmo genético* es apto para ser usado en sistemas en los que conviven algoritmos de localización y navegación.

### 5.3.3. Comportamiento ante la simetría del algoritmo final

El siguiente experimento es para comprobar si el algoritmo es capaz de lidiar con un entorno altamente simétrico y cómo se recupera de los errores que introduce la simetría.

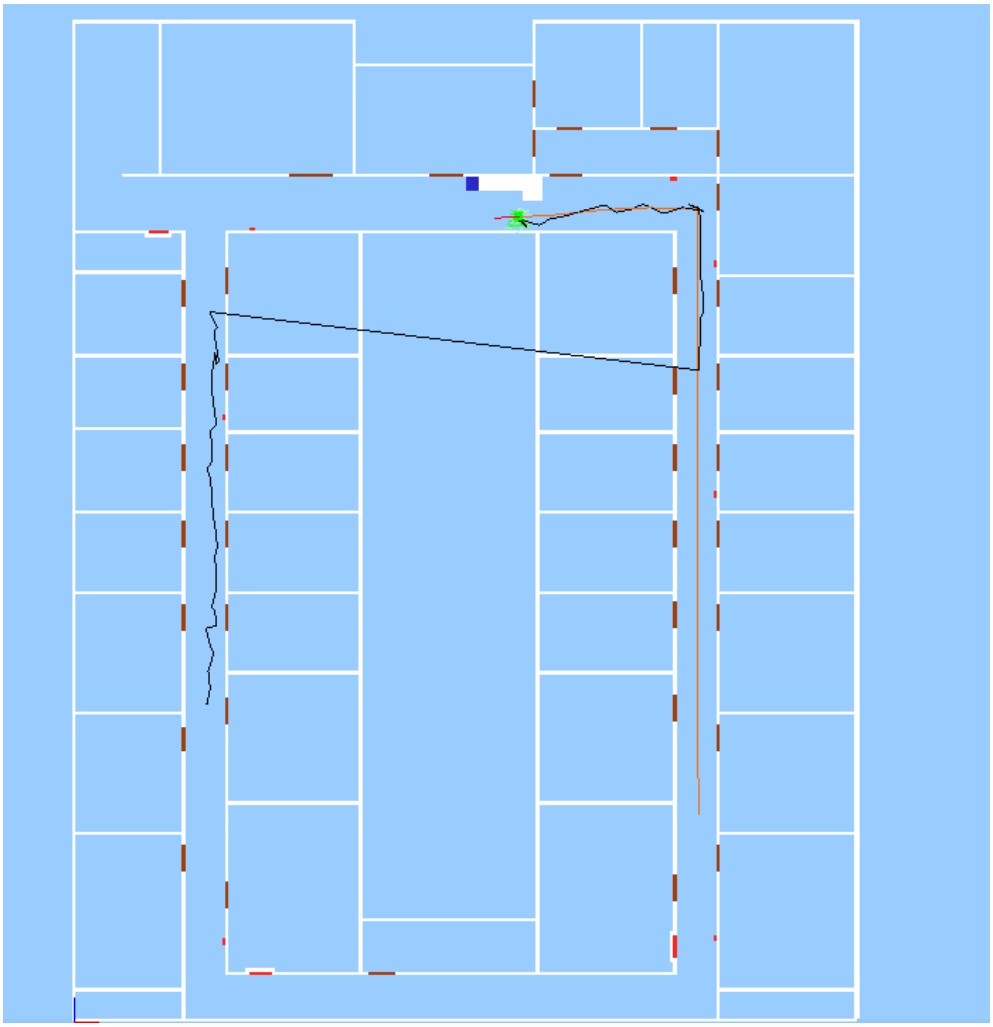


Figura 5.16: Resultado de la prueba comenzando en el pasillo de la izquierda

En la figura 5.16 vemos claramente el efecto que producen las simetrías en la localización. Durante casi la mitad de la prueba el algoritmo estima como posición correcta el pasillo izquierdo en vez de el derecho que es en el que se encuentra el robot. Al acercarse a la esquina superior derecha del pasillo el algoritmo es capaz de romper la simetría y en adelante estima correctamente la posición del robot. Veamos lo errores para apreciar mejor el efecto de la simetría.

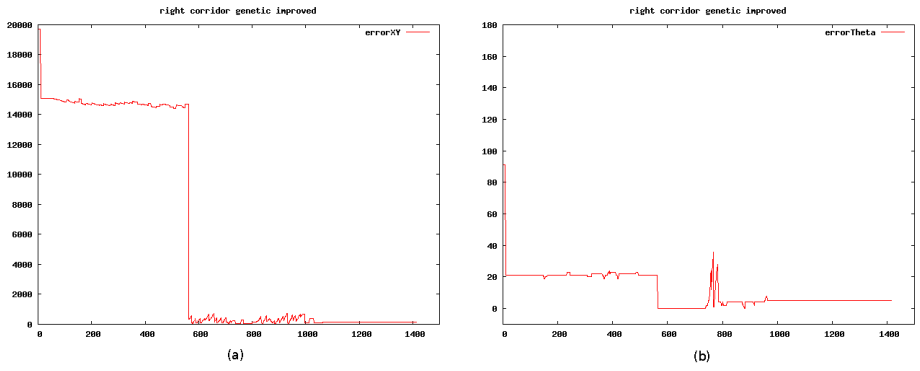



Figura 5.17: (a) Error lineal, (b) Error angular

En las gráficas de la figura 5.17 se puede ver que al principio de la prueba el error lineal es grande debido a la simetría y que sobre la mitad de la prueba se corrige volviendo a valores aceptables. Lo mismo ocurre con el error angular ya que debido a las simetrías el algoritmo falla, pero al final es capaz de corregirlo. Los errores medios se situaron en 6037.79 mm para el lineal y 11.47º para el error angular. Estos valores son muy aceptables teniendo en cuenta que el error angular no es muy alto y que a pesar del elevado error lineal medio en la parte final de la prueba el error lineal es bajo. Con estos datos podemos decir que el algoritmo es capaz de superar la prueba de las simetrías y localizar al robot en un entorno altamente simétrico sin necesidad de tener una buena localización previa.

### 5.3.4. Comportamiento ante el secuestro del algoritmo final

En el siguiente experimento enfrentaremos al algoritmo contra otro de los problemas clásicos en la localización, los secuestros. Para ello ejecutaremos el algoritmo normalmente y en un momento dado lo pararemos haciendo avanzar al robot, después activaremos el algoritmo y deberá recuperarse de los errores.

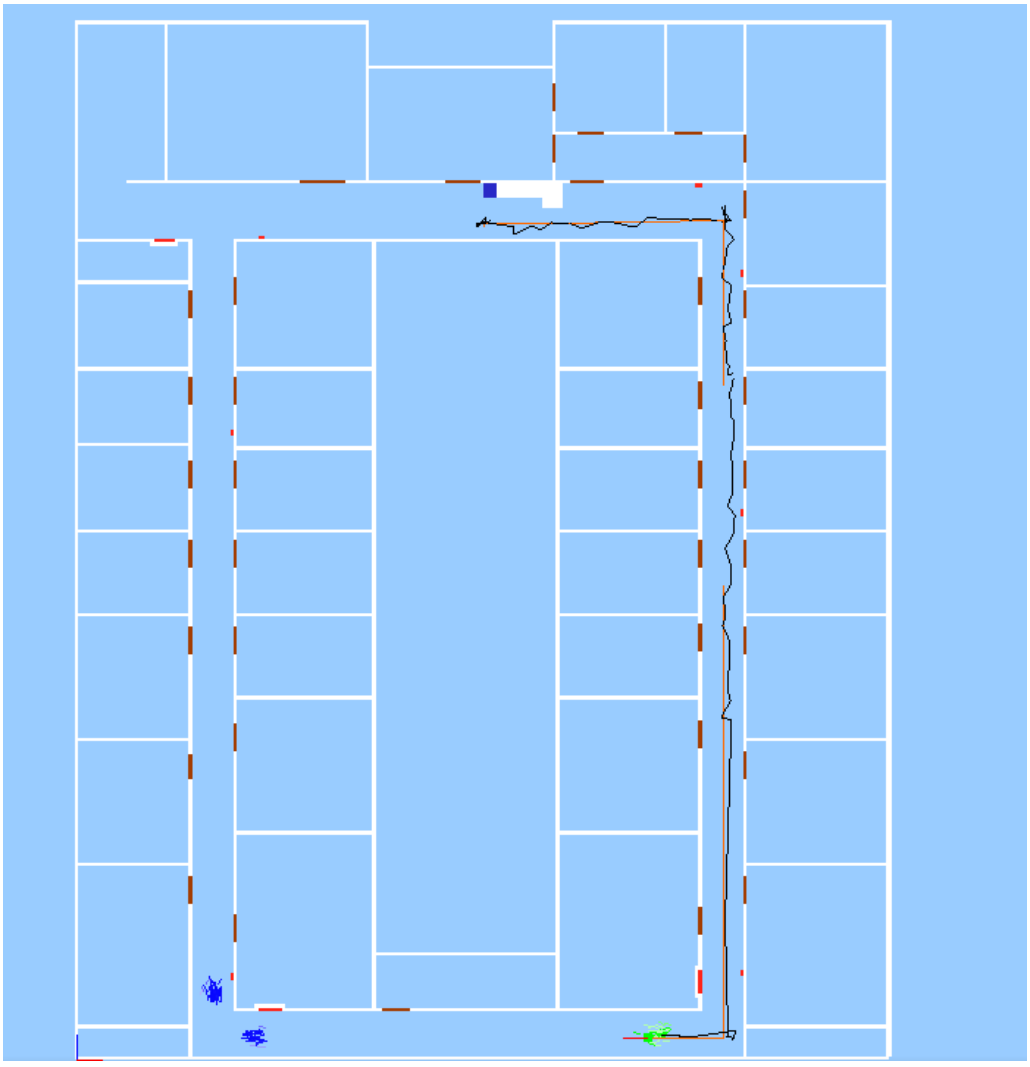


Figura 5.18: Resultado de la prueba de secuestro

En la figura 5.18 se puede observar cómo el algoritmo parte con el robot localizado con un error bajo y al producirse el secuestro pierde la posición del robot. Con el paso de unas iteraciones es capaz de recuperarse del error estimando de forma correcta la posición del robot.

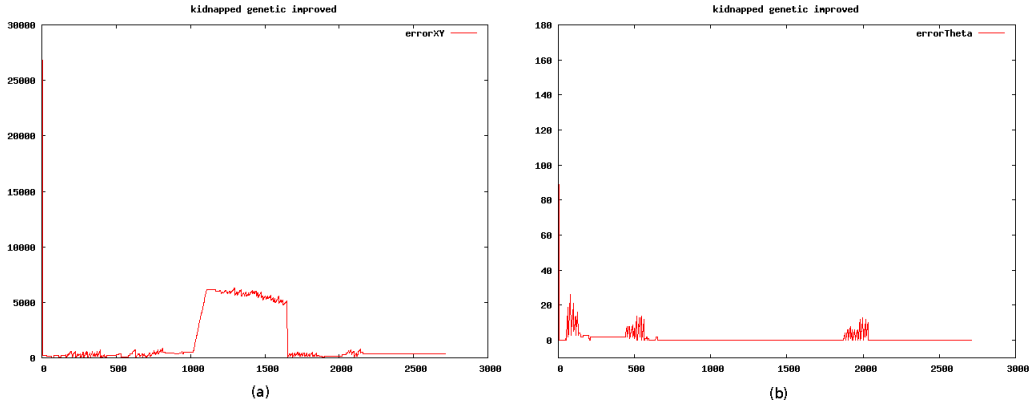



Figura 5.19: (a) Error lineal, (b) Error angular

En la figura 5.19 se puede ver mucho mejor el momento en el que el algoritmo está perdido debido al secuestro ya que aumenta el error lineal. También se puede ver cómo pasadas unas iteraciones el algoritmo se recupera y disminuye el error lineal. En cuanto al error angular vemos que gracias al emplazamiento de las exploradoras se mantiene bajo durante todo el experimento. El error lineal medio fue algo alto debido al secuestro con un valor de 1541.12 mm y el error angular fue bueno 1.27º. Estos valores son muy buenos ya que el error angular es bajo durante toda la prueba y el error lineal disminuye a valores normales recuperándose del secuestro. A la vista de los datos podemos decir que el *algoritmo genético* también supera la prueba de los secuestros.

### 5.3.5. Comportamiento ante oclusiones del algoritmo final

El último experimento al que sometimos el algoritmo final escogido fue las oclusiones esporádicas. Este experimento consiste en la situación de otro robot de color rojo en el mundo que confundirá al robot que está usando el algoritmo para localizarse.

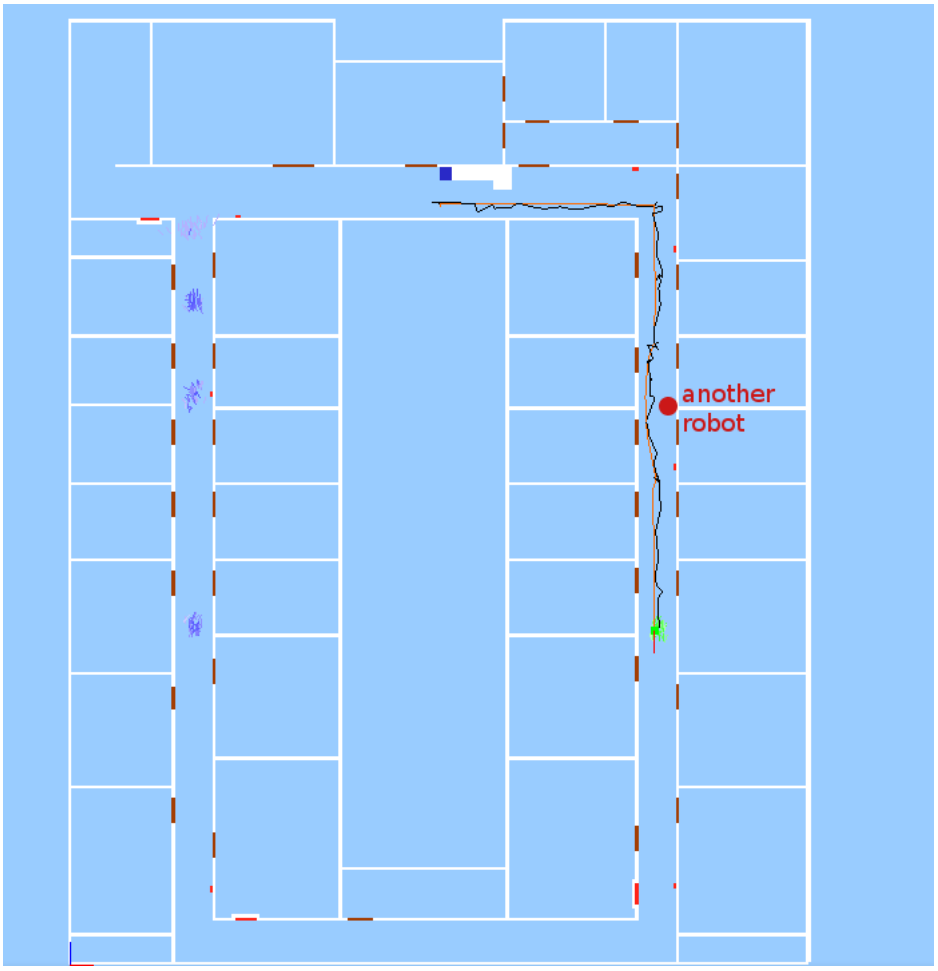



Figura 5.20: Resultado de la prueba de oclusión

Como se puede ver en la figura 5.20 el algoritmo estima correctamente la posición del robot desde las primeras iteraciones y la oclusión que genera el otro robot no afecta en gran medida al algoritmo. Este resultado es muy bueno ya que nos asegura que el algoritmo es robusto ante oclusiones momentáneas y sostenidas durante poco tiempo como pueden ser un objeto extraño en el mundo o una persona pasando por delante del robot.

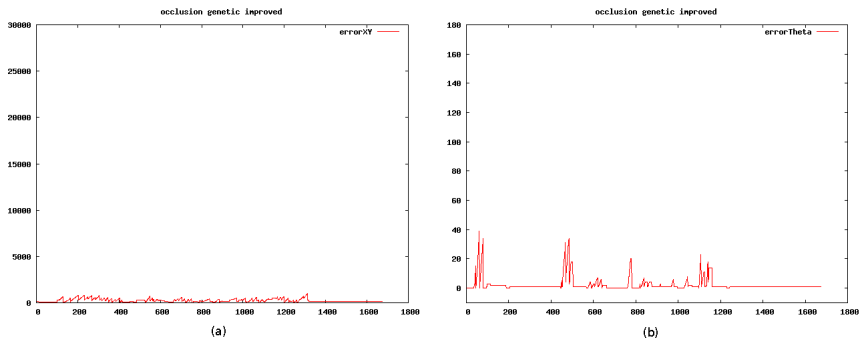



Figura 5.21: (a) Error lineal, (b) Error angular

En la figura 5.21 se puede ver cómo la oclusión no afecta al error lineal, aunque sí al error angular debido a los giros necesarios para esquivar el segundo robot. Los errores

medios dieron valores de 268.04 mm y 2.46º que nos dicen que el algoritmo es capaz de estimar la posición del robot de manera precisa a pesar de las posibles oclusiones que son muy frecuentes.

### 5.3.6. Modelo de observación con omnilaser

Igual que hicimos en el capítulo 4 con los otros modelos de observación, hemos realizado unos experimentos para ver qué tal se comporta el nuevo modelo de observación usando el omnilaser. En estos experimentos hemos probado cómo responde el modelo de observación con el omnilaser a las dos preguntas claves: ¿cómo es de discriminante? y ¿sitúa bien el máximo de la probabilidad?.

Para responder a la primera pregunta nos fijaremos en la figura 5.22 donde podemos ver cómo el nuevo modelo de observación es capaz de situar aún mejor la posición del robot siendo un poco más discriminante que el modelo de observación usando el láser normal.

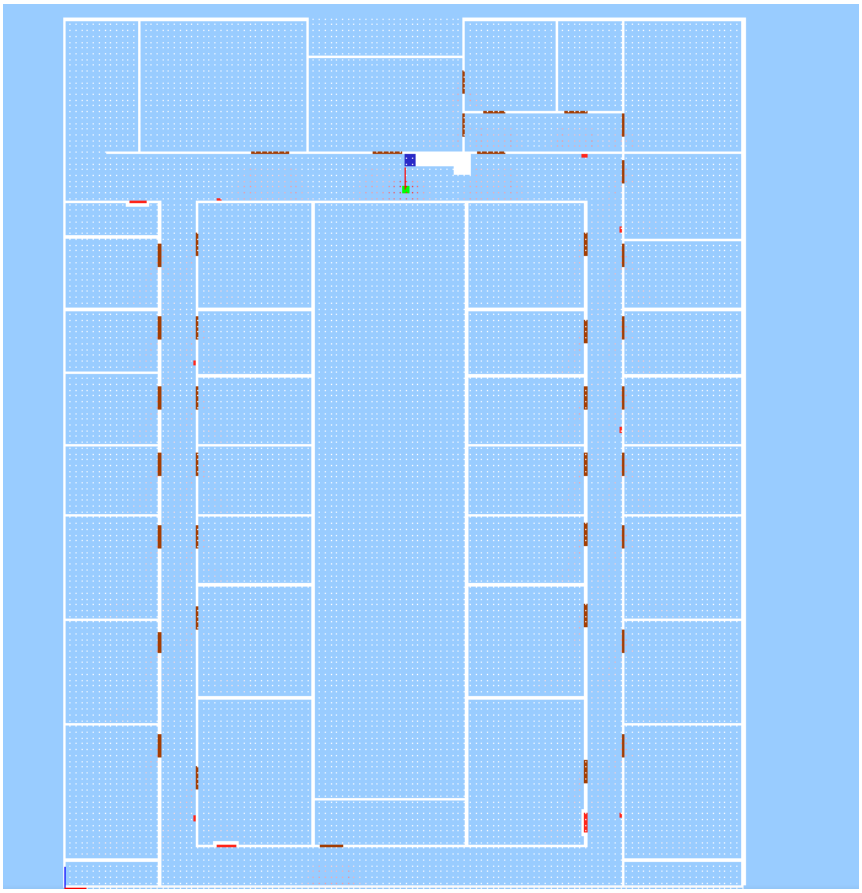


Figura 5.22: Discriminación espacial omnilaser

Para comprobar que el modelo de observación con omnilaser situa bien la probabilidad máxima nos fijaremos en la figura 5.23 donde podemos ver que el máximo de la probabilidad se situa correctamente a los 90º y que la probabilidad no es muy discriminante en las orientaciones próximas algo que es bueno.

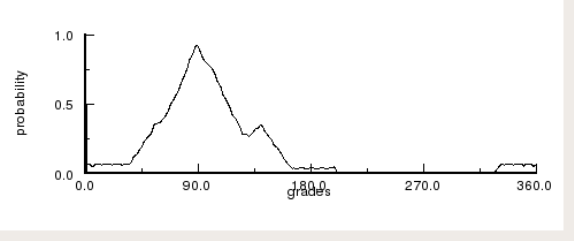



Figura 5.23: Discriminación espacial omnilaser

Ahora, al igual que hicimos con lo demás modelos de observación tenemos que probar qué tal se comporta en movimiento y para ello realizamos los siguientes experimentos.

### 5.3.7. Precisión espacial con omnilaser

En siguiente experimento realizado fue para probar la precisión espacial del algoritmo final, pero usando el modelo de observación con omnilaser explicado en el apartado de teoría anterior. Para el experimento de precisión espacial se realizó un recorrido típico del robot por el entorno comenzando desde el pasillo superior frente a la papelera azul y realizando una vuelta en sentido horario al Departamental II.

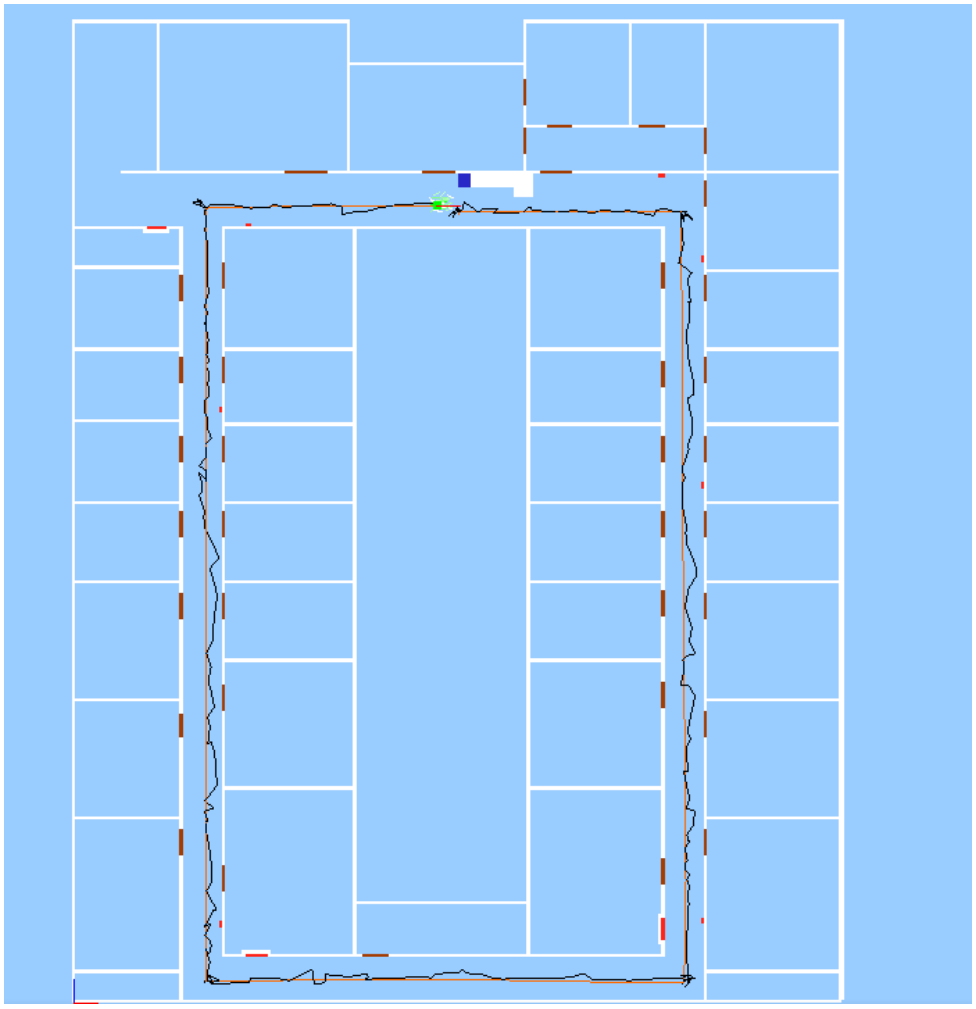


Figura 5.24: Resultado de la vuelta en sentido horario con omnilaser

Como se puede ver en la figura5.24 se parece mucho al resultado que obtuvimos en la prueba usando el algoritmo final con el láser. Aunque en los pasillos se nota un poco más de error debido al alto grado de simetría podemos decir que el algoritmo con el modelo de observación del omnilaser se comporta de manera aceptable. Para ver una diferencia más fiable veremos los valores de error aportados en este experimento.

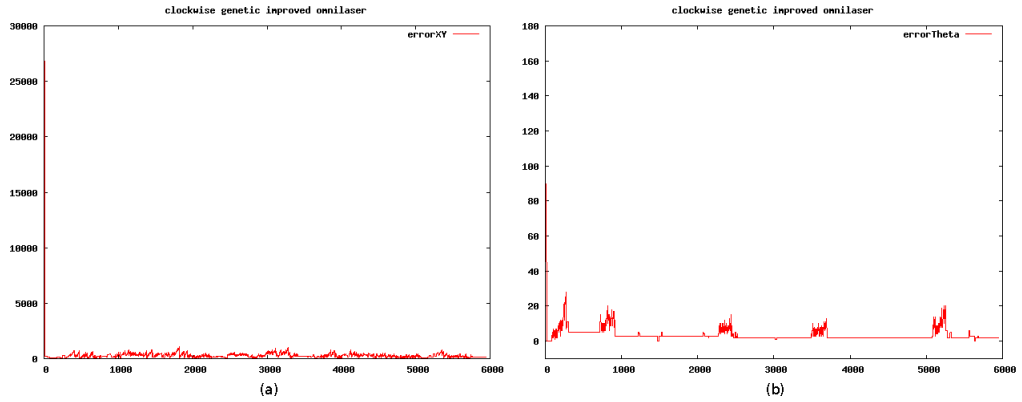



Figura 5.25: (a) Error lineal, (b) Error angular

En la figura5.25 se ven los errores tanto lineal como angular de este experimento. Si comparamos las gráficas del modelo de observación con el láser normal veremos que en esencia son parecidas y las dos han tenido valores de error muy bajos. Si miramos las cifras tenemos un error lineal medio de 344.58 mm para el omnilaser frente a 338.75 mm con el láser normal, para el error angular medio tenemos 3.83º con el omnilaser y 2.11º con el láser normal. Con estos resultados podemos decir que los dos modelos de observación son igual de válidos ya que localizan al robot con una precisión alta.

### 5.3.8. Caracterización temporal con omnilaser

El uso del nuevo modelo de observación se ha notado con una disminución del rendimiento. La disminución de rendimiento se debe en que ahora tenemos que realizar los cálculos para 90 rayos en lugar de 45 y por eso necesitamos el doble de tiempo para realizar los cálculos. Concretamente en la iteración en la que usamos los individuos exploradores necesitamos 1.2 segundos y para las iteraciones de mantenimiento de las razas necesitamos unos 180 ms.

### 5.3.9. Comportamiento ante la simetría con omnilaser

El siguiente experimento somete al algoritmo final usando el omnilaser ante el test de la simetría. Comenzará en uno de los pasillos del Departamental II, una de las zonas con más simetría.

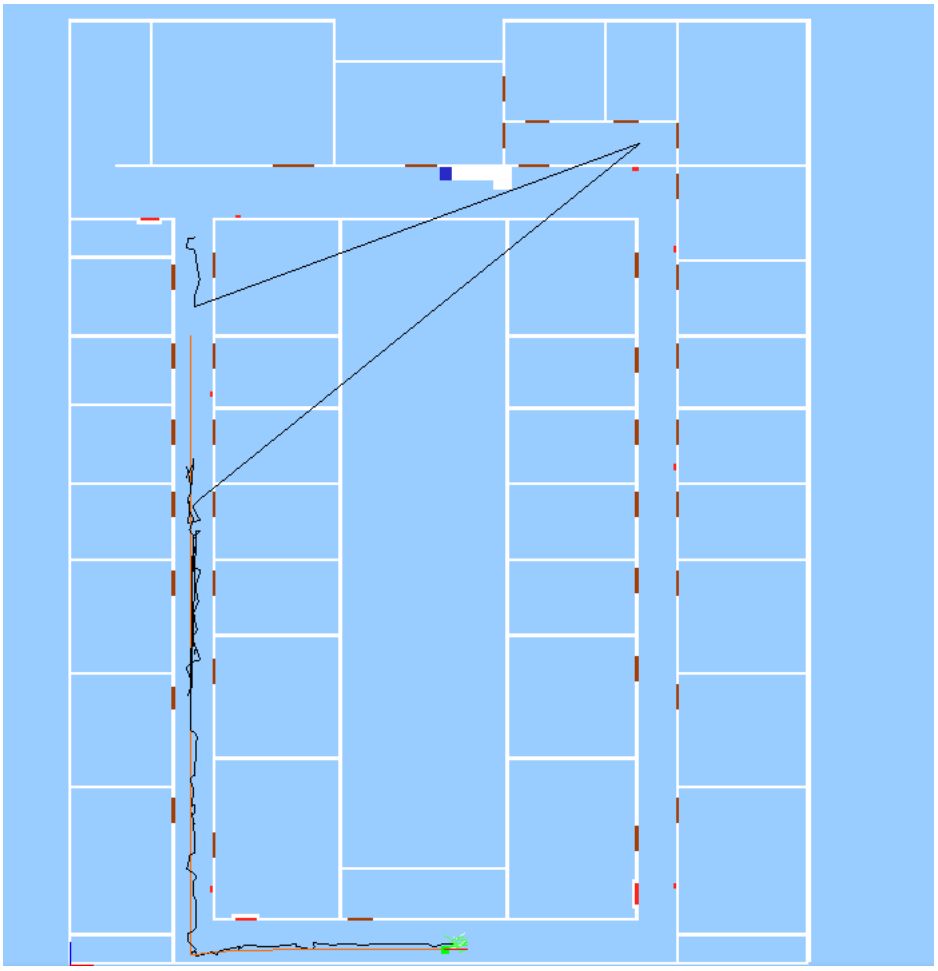


Figura 5.26: Resultado de la prueba comenzando en el pasillo de la izquierda con omnilaser

En la figura5.26 podemos ver el resultado final del experimento. Al principio, como es lógico, el algoritmo no es capaz de estimar la posición cometiendo errores bastante grandes, pero según pasan las iteraciones el algoritmo se recupera y estima normalmente la posición del robot. Para obtener más datos de este experimento vamos a ver las gráficas de error.

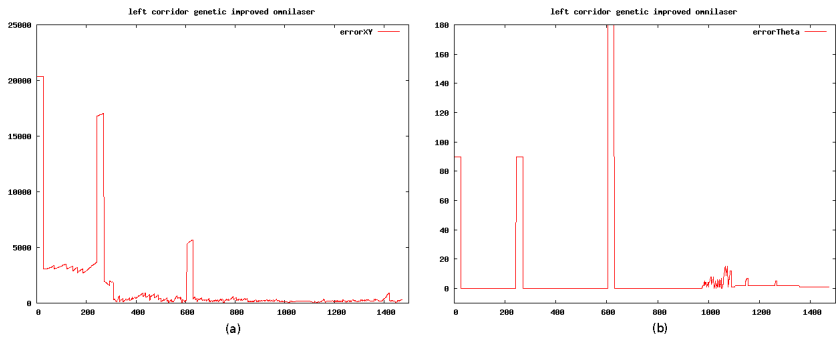



Figura 5.27: (a) Error lineal, (b) Error angular

En la figura5.27 vemos las gráficas de errores para este experimento. A simple vista si las comparamos con las gráficas aportadas por el mismo experimento con el láser

normal puede parecer que el omnilaser no ayuda a resolver la simetría mejor que el láser normal. En cifras tenemos un error lineal medio de 1514.84 mm para el omnilaser frente a los 1374 mm del láser normal y para el error angular medio tenemos 7.28º con el omnilaser y 1.50º con el láser normal.

Sin embargo si nos fijamos en el número de iteraciones en las que ambos modelos de observación consiguen llegar al valores de error bajos tenemos que el omnilaser tarda 200 iteraciones menos que el láser normal. Con esto vemos que el omnilaser se comporta mejor ante simetrías ya que resuelve la localización correcta del robot en menos tiempo.

### 5.3.10. Comportamiento ante el secuestro con omnilaser

El siguiente experimento se realizó para probar el modelo de observación con omnilaser frente al otro problema clásico de la localización, el secuestro. En este experimento el robot comenzará localizado y después del secuestro el algoritmo deberá estimar correctamente la posición del robot.

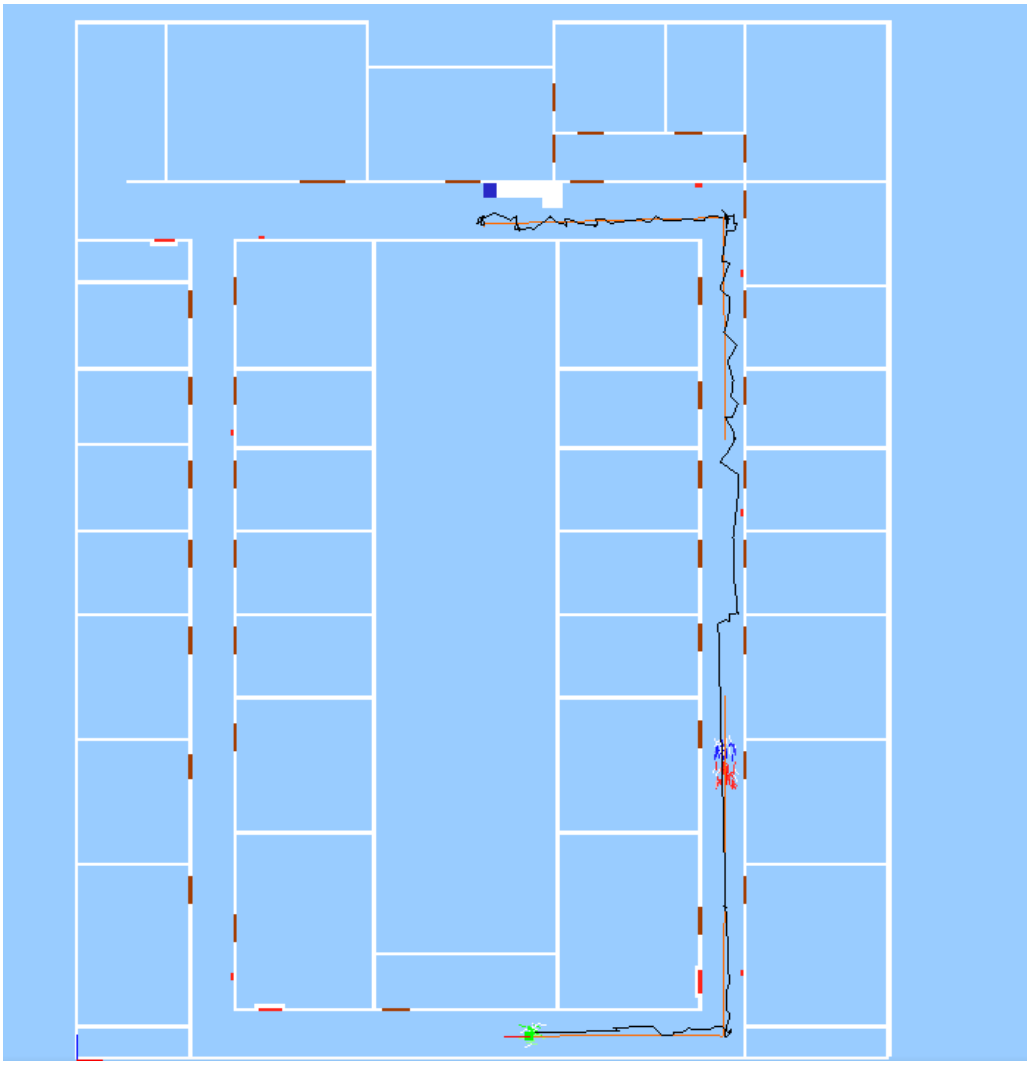


Figura 5.28: Resultado de la prueba de secuestro con omnilaser

Como se puede ver en la figura5.28 la prueba realizada con el omnilaser es muy similar a la realizada con el láser normal. Al principio el robot estima correctamente la

posición y al producirse el secuestro se produce un fallo en la estimación. Pasadas unas cuantas iteraciones el algoritmo con el omnilaser es capaz de recuperar una estimación correcta al igual que el algoritmo con el láser normal.

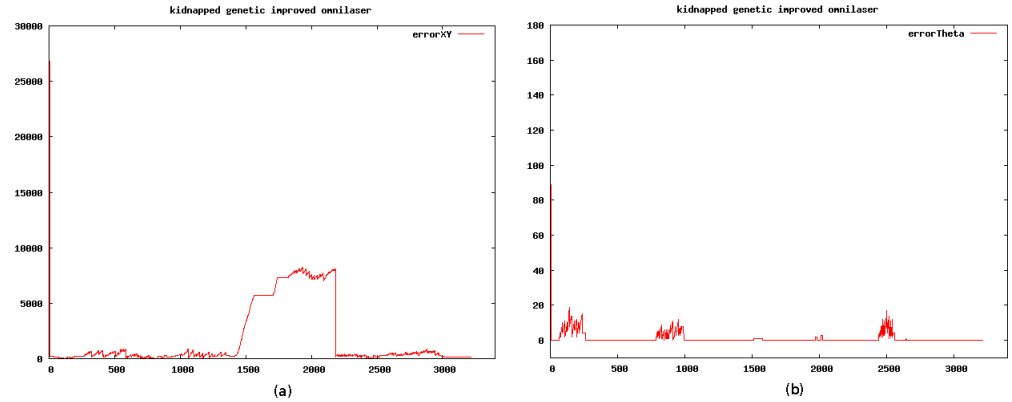



Figura 5.29: (a) Error lineal, (b) Error angular

En la figura5.29 podemos ver que los valores representativos de error arrojados por el experimento con omnilaser con muy parecidos a los que vimos en el experimento con el láser real. Las cifras para el error lineal medio con el omnilaser fueron 1791.78 mm frente a 1541.12 mm del láser normal, para el error angular medio tuvimos 1.24º con el omnilaser y 1.27º con el láser normal. Como vemos son cifras muy parecidas por lo que podemos decir que ambos modelos de observación son igual de efectivos al enfrentarse a secuestros, siendo un algo mayores las cifras del modelo de observación con omnilaser debido a que las iteraciones son más lentas.

A pesar de que el rendimiento es más bajo sería interesante explorar este modelo de observación ya que es mucho más rico que si sólo usamos el láser normal y la localización sería bastante más precisa si se consigue optimizar para reducir el tiempo de computo.

### 5.3.11. Comportamiento ante oclusiones con omnilaser

Para finalizar los experimentos de validación del algoritmo con el modelo de observación que usa el omnilaser realizamos un experimento que introduce oclusiones para someter al algoritmo a ruido en los sensores.

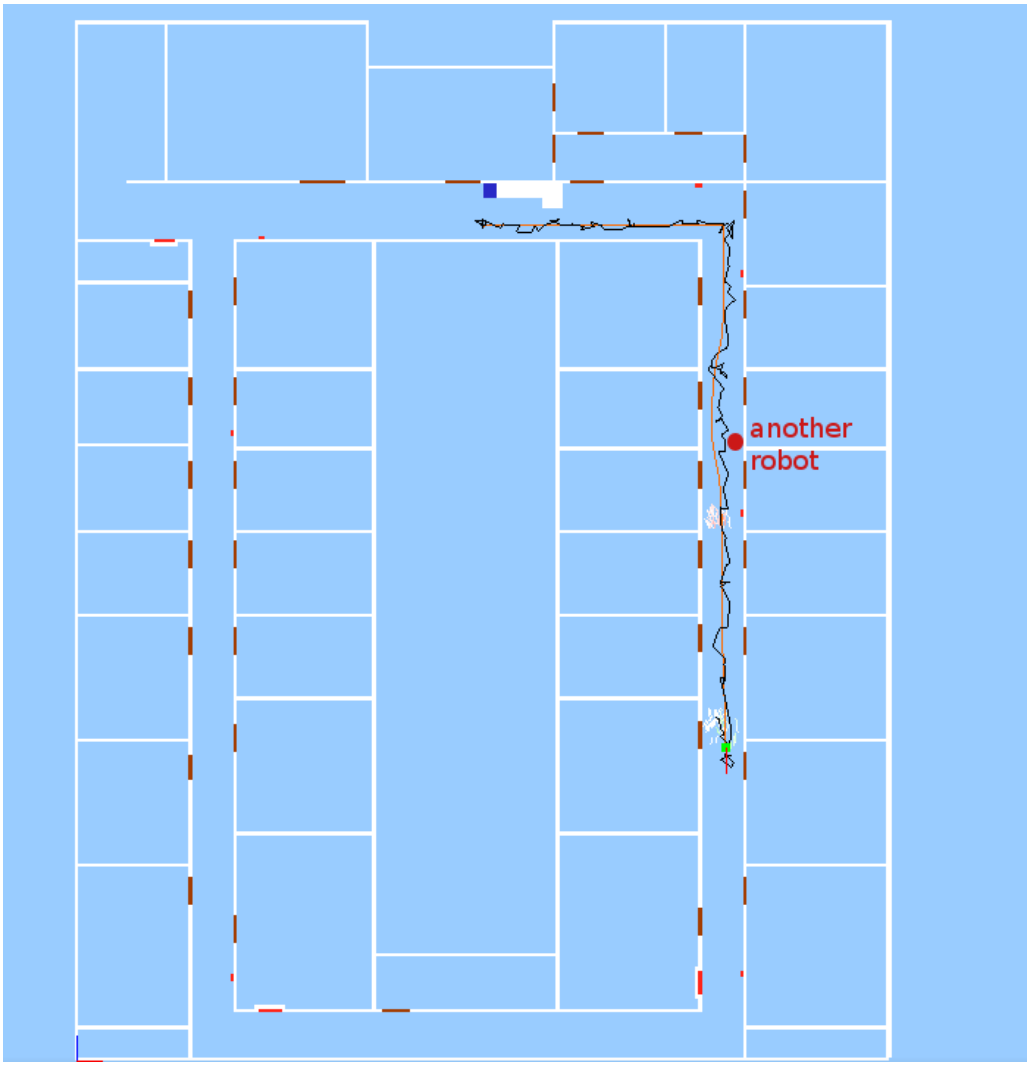



Figura 5.30: Resultado de la prueba de oclusión con omnilaser

Como vemos en la figura5.30 la línea negra, que corresponde a la estimación realizada por el algoritmo, es más temblorosa que la obtenida con el modelo de observación usando el láser normal. A pesar de que aparentemente el omnilaser es más sensible a las oclusiones podemos decir que estas no han afectado mucho al algoritmo durante el experimento ya que es capaz de estimar la posición del robot con bastante acierto.

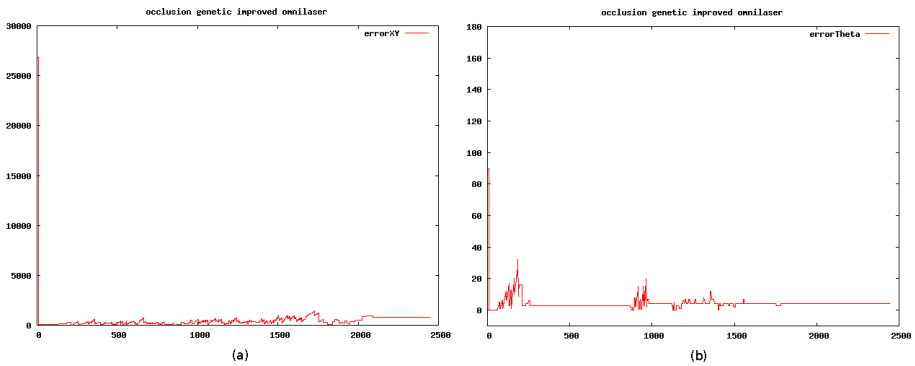



Figura 5.31: (a) Error lineal, (b) Error angular

En la figura5.31 tenemos los errores que hemos obtenido en el experimento. Las gráficas del error lineal son muy parecidas con errores bastante bajos, en la gráfica del error angular tenemos unos picos más pequeños lo que sugiere que tenemos más precisión que usando el láser normal. Las cifras para el error lineal medio con omnilaser fueron 524.05 mm frente a 268.04 mm usando el láser normal y para el error angular medio obtuvimos 4.48º con el omnilaser frente a 2.46º del láser. Los valores de error se ven enturbiados por un alto error en las iteraciones iniciales tanto para el error lineal como para el angular. Aún así podemos decir que el algoritmo de localización usando el omnilaser no se ve muy afectado ante oclusiones momentáneas.

# Capítulo 6
# Conclusiones y trabajos futuros

En los capítulos anteriores hemos descrito los problemas tratados en este proyecto y las soluciones a las que hemos llegado, junto con los experimentos que validan dichas soluciones. En este capítulo expondremos las conclusiones finales que hemos obtenido y las líneas de trabajo que se dejan abiertas como posible continuación del mismo.

## 6.1. Conclusiones

En esta sección haremos un balance general del proyecto y las conclusiones que hemos sacado del mismo, además visitaremos punto por punto los objetivos y requisitos que hemos cumplido.

**• Cumplimiento de objetivos**

Como conclusión general podemos decir que hemos conseguido cumplir los objetivos que fijamos en el capítulo2 que son los siguientes:

1. *Modelo de observación válido a partir de sensores no específicos(láser y cámara conjuntamente):* Hemos obtenido un modelo de observación para cada uno de los sensores no específicos, láser y cámara, que nos permite extraer información de posición en forma de probabilidad. Con estos modelos obtenemos una probabilidad alta cuando la posición es muy parecida a la real y baja cuando son distintas, tal y como se explica en el capítulo4. Por otro lado hemos obtenido una fórmula para combinar las probabilidades de que aportan los sensores por separado en una sola.

2. *Mallas de probabilidad como solución previa:* Hemos usado las mallas de probabilidad para validar el modelo de observación escogido y probar la robustez del modelo frente a los movimientos del robot comprobando que converge, como se describe en el capítulo4. Además, sometimos al modelo de observación a un escenario con ruido odométrico más parecido al escenario real y también pudimos dar por válido el modelo de observación ante los posibles errores odométricos ya que es capaz de recuperarse de los mismos. A pesar de que el algoritmo converge hacia la posición correcta es método no sirve ya que está lejos de ejecutarse en tiempo real, por lo que buscamos otra alternativa.

3. *Uso del algoritmo genético como solución final:* El uso del *algoritmo genético multimodal* descrito en el capítulo5 junto con nuestro modelo de observación combinado nos aportó un algoritmo ágil, robusto y con una precisión suficiente para poder localizar al robot. Además con este algoritmo conseguimos reducir el tiempo de proceso notablemente adaptándolo al tiempo real y haciendo posible el uso del algoritmo de localización junto con otros algoritmos de navegación en un mismo sistema.

4. *Validación experimental:* Los apartados de experimentación tanto del capítulo4 como del capítulo5 han servido para validar toda la teoría mostrada y junto con los demás experimentos realizados y expuestos en el *MediaWiki*[1] del proyecto demuestran que tanto el algoritmo final como los demás algoritmos desarrollados cumplen los objetivos que hemos marcado.

**• Requisitos cumplidos**

Además de cumplir los objetivos descritos en el capítulo2 se han cumplido los requisitos definidos en dicho capítulo con los siguientes resultados:

1. *Usar el Departamental II como escenario:* como se ha podido ver en los distintos experimentos del *algoritmo genético* con la correcta configuración del mismo se ha conseguido la autolocalización en un entorno de oficinas común sin dificultad y sin modificar el entorno.

2. *Robustez frente a simetrías:* con los diferentes experimentos del *algoritmo genético* se ha probado que es robusto ante un altísimo grado de simetría, recuperándose hasta en las peores situaciones como es al comenzar en los pasillos del Departamental II.

3. *Convergencia en el caso medio:* también hemos conseguido que el algoritmo sea capaz de converger hacia la posición correcta en poco tiempo, tanto en el caso mejor como en media. En el caso mejor desde el arranque del algoritmo se estima la posición correcta y en el caso peor pasan unos 5 minutos hasta que se obtiene la posición correcta del robot, para el caso medio pasan entre unos segundos y 1 minuto.

4. *Alta precisión:* en media se ha conseguido una precisión de unos 350 mm lo que es muy bueno teniendo en cuenta que tomamos como ejemplo un mundo amplio 35000mm x 25000 mm.

5. *Tiempo real:* podemos decir que hemos alcanzado el tiempo real con un tiempo máximo en cada iteración de 600 ms obtenemos unos valores aplicables en tiempo real.

[1] http://jde.gsyc.es/index.php/User:D.rodriguez

• **Casos de fallo**

Gracias a los experimentos hemos detectado que se deben mejorar ciertos aspectos, que a pesar de arrojar resultados positivos no son todo lo satisfactorios que deberían:

1. *Mayor velocidad de recuperación ante secuestros*: el algoritmo se recupera bien de los secuestros, pero dado que es un problema serio ya que se pierde toda referencia de la posición del robot sería de interés mejorar el tiempo de recuperación ante esta situación.

2. *Estimaciones erróneas a pesar de estar localizado*, en ciertas situaciones a pesar de provenir de un estado en el que el robot está localizado perfectamente se produce un cambio en la estimación del algoritmo hacia una posición errónea debido principalmente a simetrías del entorno. En este proyecto se ha planteado la solución del mecanismo de victorias para este problema, pero en algunas situaciones seguía surgiendo.

3. *Aumentar la precisión*, en este proyecto hemos conseguido una precisión muy aceptable para manejarnos un entorno tan grande, pero cuando se trata de localización en robótica unos centímetros pueden ser mucho por lo que cuanta más precisión tenga el algoritmo mejor será.

Haciendo balance podemos decir que hemos conseguido un algoritmo de localización que tiene como principales características: robustez, rapidez, precisión y que además es totalmente independiente del entorno ya que no precisa de balizas o señalización alguna específica en el entorno.

Sobre los conocimientos adquiridos realizando este proyecto podemos destacar las propias técnicas de autolocalización de robots autónomos. Además se han adquirido conocimientos de estadística y de algoritmos metaheurísticos aplicados a la robótica. También, también se han obtenido conocimientos básicos de tratamiento y filtrado de imágenes.

Por otro lado también se han adquirido conocimientos en el uso de software auxiliar y librerías para poder realizar gráficas y documentos como son: OpenGL, GNUPLOT y LaTeX Esto a sido posible gracias a que se trata de un proyecto heterogéneo.

Como ya hemos mencionado, en los primeros capítulos de esta memoria en este proyecto se pretendía dar un paso más desde los anteriores proyectos sobre autolocalización. Por ello se ha buscado obtener un algoritmo funcional combinando el sensor láser y la cámara.

## 6.2. Trabajos futuros

Para finalizar en esta sección se exponen algunas de las líneas de trabajo futuras más interesantes para ampliar este proyecto.

- **Llevar al robot real** este algoritmo, que ha mostrado ser válido sobre el simulador, lo que supondrá enfrentarse a problemas reales en la odometría y a problemas a la hora de obtener las imágenes con la calidad adecuada.

- **Usar sólo cámaras** para recoger la información del medio. Esta mejora sería interesante ya que la visión artificial es algo que actualmente tiene mucho peso en la robótica.

- Otra línea muy cercana dentro del Grupo de Robótica de la URJC sería **fusionar el algoritmo de autolocalización con el sistemas de robot guía** creado por [Vega, 2008] permitiendo además de la navegación del robot la autolocalización del mismo.

- Por último y para redondear el tema de la autolocalización del robot sería interesante crear un sistema para que el robot siempre estuviera **localizado tanto en exteriores como en interiores**. Por ejemplo fusionando un sensor específico como es el GPS para exteriores con el algoritmo aquí desarrollado para la localización en interiores.

# Bibliografía


[ActivMedia, 2002] Robotics ActivMedia. *Pioneer 2. Operations Manual.* ActivMedia Robotics, 2002.

[Cañas Plaza *et al.*, 2007] José M. Cañas Plaza, Antonio Pineda, Jesús Ruíz-Ayúcar, José A. Santos, y Javier Martín. *Programación de robots con la plataforma jdec.* URJC, 2007.

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

[Cortés, 2007] Ángel Cortés. Localización y construcción de mapas en un robot de interiores. *Proyecto fin de carrera Ing. Técnica Informática de Sistemas, Universidad Rey Juan Carlos*, 2007.

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

[Domínguez, 2009] Jose Manuel Domínguez. Autoocalización probabilística visual de un robot en el simulador gazebo. *Proyecto fin de carrera Ing. Informática, Universidad Rey Juan Carlos*, 2009.

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

[Gates, 2007] Bill Gates. A robot in every home. *Scientyfic American*, 2007.

[Kachach, 2005] Redouane Kachach. Localización del robot pioneer basada en láser. *Proyecto fin de carrera Ing. Tec. Informática de sistemas, Universidad Rey Juan Carlos*, 2005.

[López, 2005] Alberto López. Localización de un robot con visión local. *Proyecto fin de carrera Ing. Técnica Informática de Sistemas, Universidad Rey Juan Carlos*, 2005.

[Mackay, 1999] D.J.C. Mackay. Introduction to monte carlo methods. In *In M. Jordan, Learning in Graphical Models, MIT Press*, pages 175–204, 1999.

[Perdices, 2009] Eduardo Perdices. Autolocalización viusal en la robocup basada en la detección 3d de las porterías. *Proyecto fin de carrera Ing. Informática, Universidad Rey Juan Carlos*, 2009.

[Perdices, 2010] Eduardo Perdices. Autolocalización viusal en la robocup con algoritmos basado en comparaciones. *Proyecto fin de master, Universidad Rey Juan Carlos*, 2010.

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

[Vega, 2008] Julio Manuel Vega. Navegación y autolocalización de un robot guía de visitantes. *Proyecto fin de carrera, URJC*, 2008.