Universidad Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

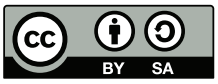
Ingeniería de Telecomunicación -
Ingeniería Técnica en Informática de Sistemas

PROYECTO FIN DE CARRERA

Algoritmo de Autolocalización Evolutiva Multimodal para Robots Autónomos

Autor:Luis Roberto Morales Iglesias
Tutor: Prof. Dr. José María Cañas Plaza

Curso académico 2013/2014



Que no perdáis el rumbo
y os encontréis en
vuestro camino
de la vida

Agradecimientos

Tras teñir las líneas escritas en este libro no solo de lo aprendido y desarrollado durante esta última etapa de la carrera, sino también de la ilusión, alegrías, quebraderos de cabeza, sensación de satisfacción y otras muchas emociones y recuerdos de esta época universitaria, ha llegado el momento de escribir lo que serán las primeras páginas de este texto: aquellas que curiosamente suelen escribirse al final pero con más empeño para reservarles un lugar especial al principio del todo.

Son muchas las personas que por diversos motivos deberían aparecer aquí: algunas que cuando tenga este libro en mis manos, pensaré en por qué se me olvidaría mencionarlas directamente o que no podré hacerlo por no escribir una sección que ocupe más que el propio proyecto. A todas ellas, decirles que en algún punto de este texto, aunque no aparezca su nombre, están incluidas con la misma importancia en estas líneas como las que sí.

En primer lugar agradecerles a mis padres por ayudarme a llegar a donde estoy; por el apoyo incondicional en toda la carrera; por esas horas de escuchar - entendiendo unas veces más y otras menos - las presentaciones y trabajos; por esas noches en las que me recordaban que dormir bien es igual de importante que querer dedicarle tiempo a algo. . . en definitiva, por estar a mi lado siempre apoyándome. También a mi familia, por su preocupación, apoyo y ánimos.

En segundo lugar agradecer a Jose María Cañas - mi tutor en este proyecto - todo el apoyo, dedicación, cercanía y paciencia - mucha paciencia - durante todo este tiempo. Nunca llegué a tenerle como profesor en las asignaturas de la carrera, pero no me cabe duda de lo bueno que es como profesor y como persona.

También quisiera dar gracias a todos los profesores que me han enseñado durante la carrera: a los que se han implicado más, a los que se preocupaban por amenizar las clases, a los que lo ponían difícil para que nos esforzáramos, a los que ofrecían una imagen más cercana y a los que piensen que no pertenecen a ninguno de estos grupos.

No quisiera olvidarme de mis compañeros de universidad de todos estos años, con los que he compartido no solo clases, sino otros momentos y emociones. En especial de aquellos con los que he pasado más momentos y de los que últimamente cuando me veían, su primera frase después de saludar era *"y el proyecto... ¿para cuando?"*; entre ellos, a los que más la han repetido últimamente: Ángel *(no "estudies" demasiado, ya sabes a lo que me refiero...)*, Dani *(suerte con tus* $_{sub}{}_{sub}$*becarios y no los maltrates mucho)*, Bea *(suerte en el norte y ven a vernos más veces)*, Guadalupe *(ánimo con las tazas y las certificaciones)*, Juan *(ánimo en tu proyecto, que queda poco)*.

Gracias a todos vosotros, los que me habéis marcado en algún momento o seguís haciéndolo; aunque no os nombre ya sabéis quien sois, gracias.

Y gracias a tí, por leer estas líneas y las que vienen a continuación; todas escritas con ilusión. Espero que te aporten algo nuevo, te ayuden a alcanzar una nueva meta y/o te ayuden a *encontrarte en el escenario en el que estás*.

Índice general

Índice de figuras

1 Introducción

Desde el diseño de los primeros autómatas, el desarrollo y aplicación de la robótica se ha ido extendiendo a diversos ámbitos de nuestra vida, desde los ambientes más industrializados hasta los domésticos. Un conjunto interesante de estas aplicaciones son las que conllevan la capacidad de un robot de desplazarse de forma más o menos autónoma por su entorno, siendo de vital importancia para este tipo de aplicaciones que el dispositivo conozca su posición y orientación dentro del espacio de trabajo.

En este proyecto se trata este problema desde el punto de vista de un robot que dispone de un sensor de desplazamiento y uno de medida de distancias y puede desplazarse libremente en un entorno tridimensional conocido. Como caso base de este robot genérico, se tomará un robot que se desplaza a ras de suelo y que dispone de encoders y un sensor de distancia multipunto.

Este capítulo introduce en el contexto de conceptos como la *robótica* y la *autolocalización* y que servirán de base para el resto de capítulos.

1.1. Robótica

La *robótica* - término acuñado por el escritor Isaac Asimov - es la rama de la tecnología dedicada al diseño, estudio, construcción, operación y aplicación de máquinas que realizan tareas con cierto nivel de automatismo y se conocen con el nombre de *robots*. Este término deriva de *robota*, palabra de origen eslavo que puede adoptar la acepción de trabajo o *siervo*, y se utilizó por primera vez en la literatura en la obra de Karel Čapek *Robot Universales Rossum* de 1920.

La robótica combina diversas disciplinas como la mecánica, la electrónica, la informática, la inteligencia artificial, la ingeniería de control y la física para hacer posible el funcionamiento autónomo de los robots.

La historia de los robots se puede considerar que comienza con los primeros autómatas mecánicos, pero no es hasta 1954 cuando George Devol inventa el primer robot autónomo programable, el Unimate. Este robot, vendido en 1961 a General Motors, tenía la función de mover piezas industriales a altas temperaturas.

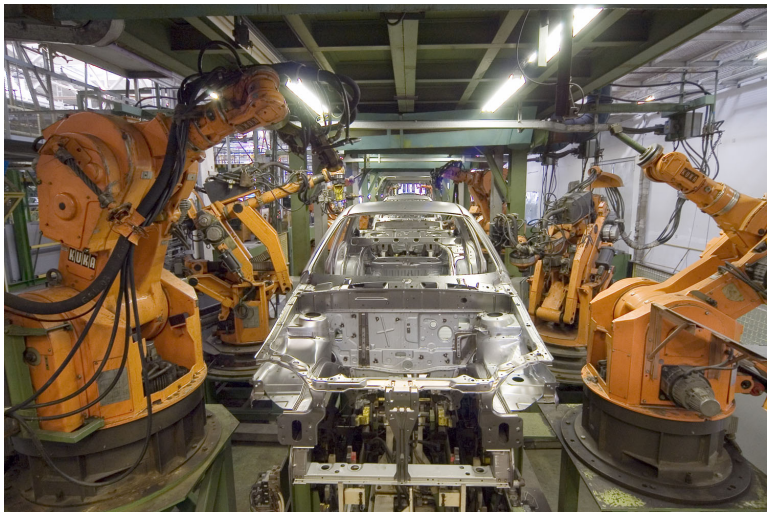


Figura 1.1: Ejemplo de robot industrial de ensamblaje automotriz. [1]

A partir de 1970 comenzó a desarrollarse el campo de la robótica, creándose en 1973 Famulus, el primer robot con 6 ejes electromecánicos, y en 1975 PUMA, el primer brazo mecánico programable. Desde 1980 comienza a extenderse el uso de los robots en el ámbito industrial, por su rapidez, precisión y por la minimización de riesgos en tareas potencialmente peligrosas para los obreros.

Conforme se han realizado avances en el ámbito de la robótica, se han ido diseñando robots para todo tipo de propósitos, incluso fuera del campo industrial. Así, se pueden ver robots diseñados para asistir en operaciones de rescate, en medicina, vigilancia, tareas domésticas, transporte, investigación y en el campo militar.

Un ejemplo notable de la robótica aplicada a la medicina es el *Sistema Quirúrgico Da Vinci* (Figura 1.2), desarrollado por Intuitive Surgical. Este robot está diseñado para que un cirujano maneje desde una consola fuera del campo estéril del quirófano una serie de brazos que disponen de instrumentos quirúrgicos y una cámara, aumentando el nivel de precisión del cirujano y facilitando los pasos de la operación.

La aplicación más conocida de la robótica en ambientes domésticos es la de la aspiradora robótica *Roomba* (Figura 1.3), de la empresa iRobot. Lanzado por primera vez en 2002, este robot utiliza sensores de contacto e infrarrojos para desplazarse por el área designada para aspirar, evitando posibles caídas.

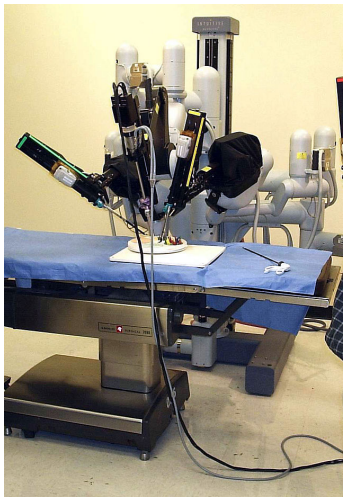


Figura 1.2: Aplicación de la robótica a la medicina: El Sistema Quirúrgico Da Vinci. [2]

Figura 1.3: Robot doméstico Roomba para aspiración [3].

Dentro de las aplicaciones en robótica móvil se puede destacar también el uso en misiones espaciales de exploración, en las que los robots enviados se diseñan para operar de la forma más autónoma posible, permitiendo enfocar más recursos al análisis de los datos recogidos. Un ejemplo de este tipo de robots es el *Opportunity* (Figura 1.5), un vehículo de exploración enviado a Marte en 2003 junto a su hermano *Spirit*, de 185 Kg de masa y con múltiples tipos de sensores para análisis de la superficie del planeta. Este vehículo aterrizó en Marte el 25 de enero de 2004 y a fecha de 8 de mayo lleva recorridos 39.45 km.

Estos robots que requieren desplazarse por una zona, no necesariamente conocida, sobre un terreno, en el agua o incluso volando, deben enfrentarse al reto de saber dónde deben ir y dónde están con la información captada por sus sensores.

Figura 1.4: Robot multipropósito militar PackBot

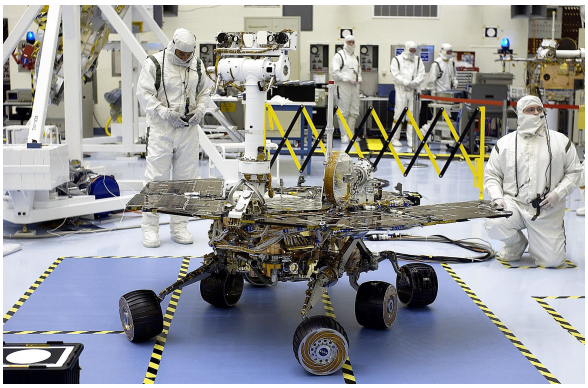


Figura 1.5: Vehículo de exploración espacial Opportunity de la NASA.

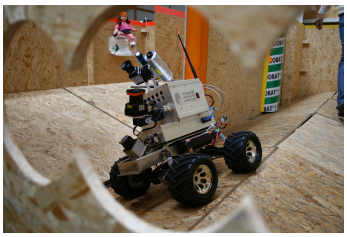
Con el fin de fomentar la investigación y el desarrollo dentro del mundo de la robótica móvil, diversas organizaciones organizan competiciones en las que se intenta resolver en cierta medida algunos retos. Entre estas competiciones destacan las organizadas por DARPA[1] (*Agencia de Defensa de Proyectos de Investigación Avanzados*), una agencia perteneciente al Departamento de Defensa estadounidense. Algunos de estos retos son el *DARPA Grand Challenge*, en el que vehículos autónomos tenían que cruzar parte del desierto de Mojave siguiendo la Interestatal 15; el *DARPA Urban Challenge*, en el que se tenía que superar un circuito urbano con obstáculos propios del entorno y respetando las normas de tráfico; y el *DARPA Robotics Challenge*, en la que robots humanoides semiautónomos tienen que completar una serie de tareas en condiciones peligrosas o de catástrofe.

1.2. Autolocalización en robots móviles

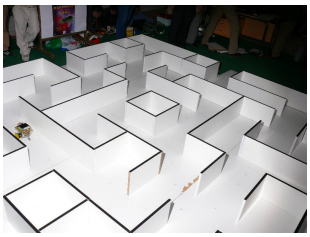
La *autolocalización en robots móviles* es un problema fundamental al que se enfrenta la robótica móvil autónoma, ya que los comportamientos de estos robots dependen en gran medida de su posición en cada momento.

Existe más de una variante del problema de autolocalización en función de los datos disponibles y suposiciones que se pueden asumir dentro del marco de funcionamiento del robot concreto. Dos variantes principales del problema son la de autolocalización en entorno conocido y la de autolocalización en un entorno inicialmente desconocido. Esta última es una variante interesante conocida dentro de los problemas de tipo SLAM[2].

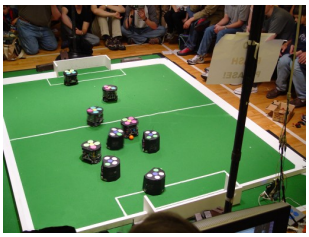
Otras variantes importantes del problema son la *autolocalización incremental*, en la que un sensor de odometría devuelve la posición o desplazamientos del robot respecto de una anterior en todo momento con cierto error; la *autolocalización absoluta*, en la que se carece de un sensor de odometría en el cual basarse; y la *localización de vecinos*, en la que es necesario conocer la posición relativa a otros robots. Estas y otras variantes no mencionadas pueden mezclarse dando lugar a nuevos tipos de soluciones de localización.



(a) SLAM



(b) Autolocalización



(c) Localización colaborativa

Figura 1.6: Ejemplos de variantes del problema de localización en robots móviles.

[1] `http://www.darpa.mil`

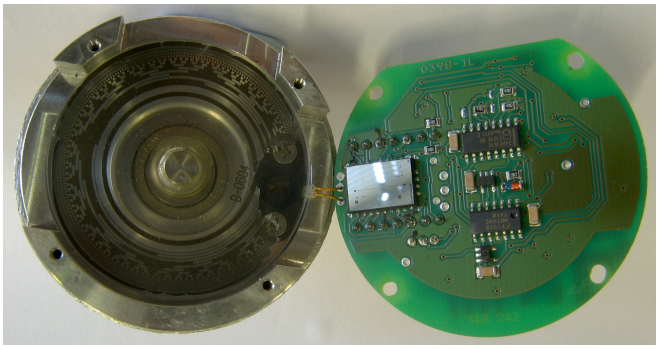
[2] *Simultaneous Localization And Mapping*, localización y mapeado simultáneos

En este proyecto se tratará la variante de *autolocalización absoluta* con un enfoque mixto entre la variante tradicional y la de *autolocalización incremental*, tomando lo mejor de las dos soluciones.

Independientemente de la variante, para localizase, el robot necesita de una u otra forma obtener una representación parcial o completa del área en el que se va a mover e información relativa al entorno que le rodea en cada instante. Generalmente, la representación del área viene dada por algún tipo de mapa que o bien se le suministra como dato al robot o bien se lo genera conforme va desplazándose (técnica utilizada en los ya mencionados problemas de *SLAM*).

Para relacionar la información relativa al entorno con la representación general, el robot móvil puede utilizar distintas soluciones:

Sensores específicos de localización Esta solución utiliza sensores que proporcionan información sobre los movimientos o posición concreta respecto a una referencia.



(a)

(b)

Figura 1.7: Los sensores de localización específicos más habituales: un encoder y un acelerómetro/giroscopio electrónico.

El principal ejemplo de este tipo de sensores es el *encoder*, que informa de los movimientos efectuados. Su principal problema es la imprecisión de las medidas debido a imperfecciones mecánicas o deslizamientos entre otros factores, dando lugar a un error sistemático predecible y a uno no sistemático que se acumula.

Otros ejemplos de este tipo de sensores son el giroscopio y el acelerómetro, que da información de aceleración y giro.

Caso especial son los sistemas de navegación por satélite (GPS, GALILEO, GLONASS...). La información final devuelta por estos dispositivos es una posición directa, pero para obtenerla, se hace uso satélites en órbitas y posiciones conocidas, utilizando estos datos y marcas temporales a modo de balizas de posicionamiento.

Balizas de posición conocida Esta solución requiere de una serie de elementos en posiciones conocidas con exactitud y que sean detectables por el robot de forma unívoca, lo cual requiere que el entorno se encuentre delimitado de antemano.

En este caso el robot mediante *triangulación* o *trilateración* localiza el ángulo de la visual con las balizas o la distancia a cada una de ellas e infiere su posición a partir de estos datos.

Scan matching Esta técnica requiere un mapa o conjunto de mapas locales y la lectura sensorial del entorno. En este caso se intenta alinear las lecturas obtenidas con los mapas disponibles, partiendo de una estimación de posición inicial.

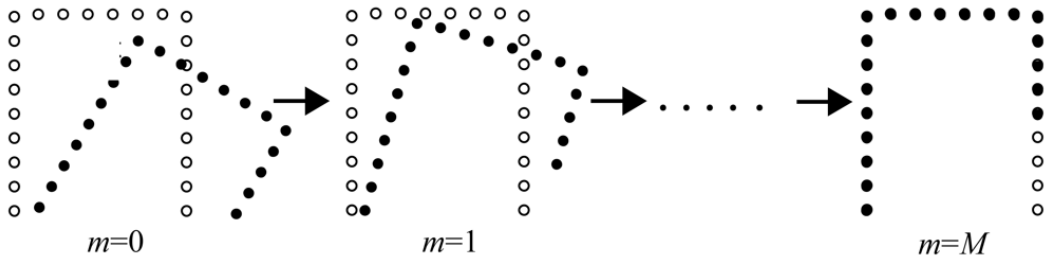


Figura 1.8: Representación de la técnica de scan matching.

Autolocalización basada en probabilidad Esta técnica calcula las posiciones más probables en base a las acciones efectuadas por el robot y las lecturas sensoriales obtenidas a lo largo del tiempo.

Al asignar una probabilidad a cada posición considerada se pueden manejar múltiples hipótesis de posición de forma simultánea, lo que permite manejar adecuadamente muchas situaciones de ambigüedad.

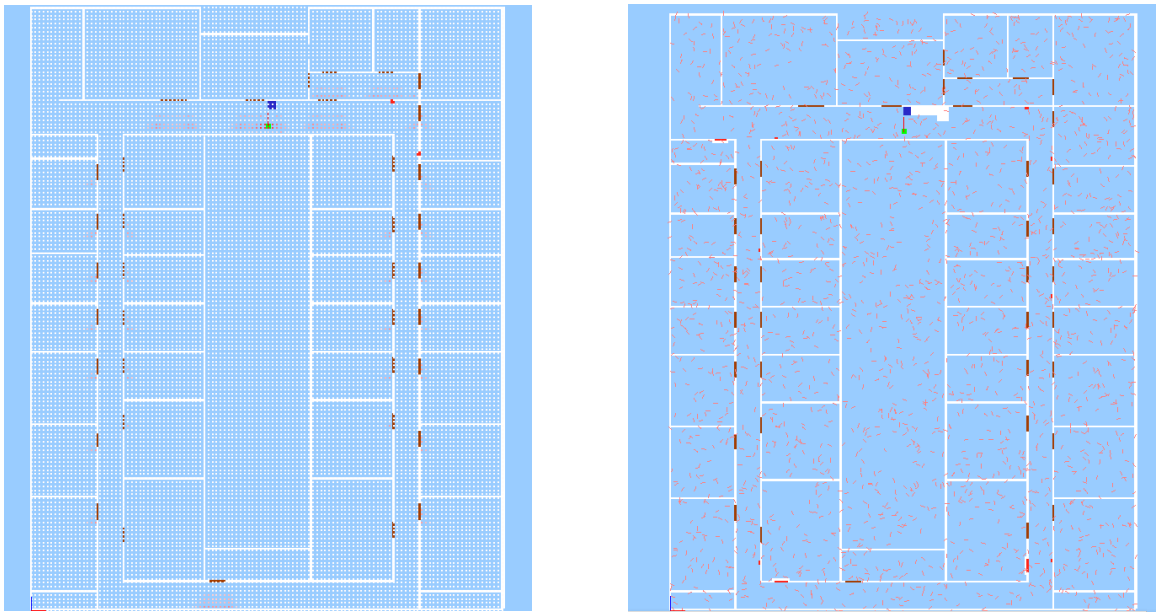
1.3. Autolocalización probabilísticas

Las *técnicas de autolocalización probabilísticas* consideran diversas hipótesis de localización generadas de distintas formas según la técnica concreta y les asignan cierta probabilidad de veracidad, lo que permite decidir como localización válida una de ellas, típicamente la de mayor probabilidad.

Estas técnicas se basan en modelos de incertidumbre que mediante una función de probabilidad representan la verosimilitud de que cierta posición sea la actual del robot y la actualizan de forma iterativa en base al Teorema de Bayes.

Partiendo de una función de probabilidad de posición base, en cada iteración se utilizan las lecturas sensoriales y la información de movimiento para relacionarlas con un modelo de observación, es decir, un modelo que refleja qué debería ver en los sensores el robot de encontrarse en una posición u otra. De esta relación se desprende una nueva función de probabilidad de verosimilitud de posición que, al combinarla con la función de probabilidad anterior, se reduce la incertidumbre en la localización estimada. Repitiendo esto en sucesivas iteraciones, la estimación de posición acabaría convergiendo a una posición.

En función de cómo plantean las hipótesis de localización se distinguen, entre otras, las técnicas de discretización y las de muestreo.



(a) Discretizada (b) Muestreo

Figura 1.9: Representación de técnicas probabilísticas [14].

Las *técnicas probabilísticas discretizadas* dividen el espacio de posibles posiciones en zonas con un valor de probabilidad propio que se va actualizando conforme pasa el tiempo. Generalmente, esta división suele consistir en algún tipo de rejilla en la que cada posición almacena su valor de probabilidad y el algoritmo decide que aquella con valor de probabilidad más alto es la posición verdadera. Por su modo de funcionamiento, este tipo de técnicas suelen ser costosas en cómputo y memoria y presentan problemas de escalabilidad, ya que el número de celdas de la rejilla crece de forma no lineal conforme se aumenta área considerada dentro del problema.

Las *técnicas probabilísticas muestreadas* consideran a la vez parte del espacio de soluciones, muestreando posiciones de interés y actualizando la localización de estas muestras en función de la probabilidad. Entre estas técnicas destacan las *técnicas de Monte Carlo* y, concretamente, el algoritmo de *filtro de partículas*. Este algoritmo genera un grupo limitado de muestras representativas de función densidad de probabilidad a las que denomina partículas, que se van actualizando, creando y destruyendo en función de los movimientos y observaciones del robot. Su principal inconveniente es su fragilidad ante entornos de alta simetría, en el que puede darse la situación de que el algoritmo tenga que decidir entre diversas hipótesis equiprobables y falle en la localización.

1.4. Localización en el Grupo de Robótica de la URJC

Dado el interés y lo poco trivial que resulta el problema de la localización, dentro del Grupo de Robótica de la Universidad Rey Juan Carlos se ha abordado este problema desde distintas perspectivas y mediante distintos métodos de solución, con el fin de conseguir comportamientos robóticos como pueden ser servir de guía o jugar al fútbol.

Siguiendo un orden cronológico, algunas de estas aproximaciones son:

- En 2003, María Ángeles Crespo presenta en su PFC [6] un algoritmo de localización basado en un filtro de partículas de Monte Carlo y una cámara como sensor principal; montado todo sobre el entorno de la RoboCup.
- En 2005, José Alberto López y Redouane Kachach abordan en sus PFC [7, 8] el problema de la localización mediante los métodos de mallas de probabilidad y fíltro de partículas, utilizando una cámara y un láser respectivamente.
- En 2007, Ángel Cortés aplica en su PFC [9] el método genético al problema de localización y mapeado simultáneos (SLAM), utilizando un sensor láser en un entorno controlado.
- En 2008, Julio Manuel Vega utiliza en su PFC [10] un algoritmo genético para localizar un robot en una habitación, utilizando una cámara y una serie de balizas de posición en el techo.
- En 2009, José Manuel Dominguez presenta en su PFC [11] un sistema de estimación de posición de un robot en entornos de alta simetría simulados basado en un filtro de partículas de Monte Carlo y una cámara como sensor.
- En 2009 con su PFC [12] y en 2010 con su TFM [13], Eduardo Perdices trata el problema de localización de un robot en el entorno de la RoboCup con distintos algoritmos y utilizando como sensor una cámara.
- En 2010, Darío Rodríguez presenta en su PFC [14] una implementación del algoritmo evolutivo para autolocalizar un robot utilizando un sensor láser y una cámara.

Se puede observar que en estas aproximaciones se ha optado por sensores de distancia y/o visión, probando la efectividad de distintos algoritmos a la hora de resolver el problema. En este proyecto se utilizará la base del algoritmo evolutivo, por haber sido el que ha dado mejores resultados históricamente, mejorándolo y diseñando el sistema para que soporte soluciones en el espacio tridimensional completo, usando como base sensores de distancia.

Los próximos capítulos explican de forma más detallada los objetivos principales del proyecto, la infraestructura utilizada y el desarrollo de la solución. Después, se expondrán los experimentos diseñados y realizados para la validación del algoritmo, conclusiones obtenidas y algunas líneas de desarrollo futuro que deja abiertas este proyecto.

2 Objetivos

En este capítulo se analizará el objetivo general de este proyecto fin de carrera, los requisitos subyacentes e impuestos a la solución y la metodología seguida para alcanzar el resultado que se describirá en secciones posteriores.

2.1. Descripción del problema

Este proyecto trata de solucionar en mayor o menor medida el *problema de autolocalización de un robot móvil autónomo en un espacio tridimensional*, utilizando para ello un *algoritmo multimodal que sigue un esquema evolutivo* y apoyándose en los sensores de posición y distancia de los que dispone el robot.

En base al objetivo general de este proyecto, se ha articulado el problema de forma más elaborada en los tres subobjetivos que se comentan a continuación:

- *Desarrollo de un entorno para evaluar la respuesta de un algoritmo de localización.*
  Para poder evaluar el funcionamiento de cada una de las versiones del algoritmo de localización hasta alcanzar la versión definitiva y compararla con otras versiones, es necesario algún entorno que facilite la puesta en marcha y evaluación de la autolocalización. El entorno se encargará de mostrar en tiempo real el estado del robot y del algoritmo, de manejar las entradas sensoriales y mapeado que servirán de base, y controlar el funcionamiento general del mismo. Además, deberá ofrecer herramientas que permitan comprobar la respuesta de diversos subcomponentes, con el fin de facilitar su desarrollo y depuración.

- *Diseño e implementación del algoritmo de autolocalización multimodal.*
  Este subobjetivo es el núcleo del problema, ya que una vez desarrollado y depurado, puede resolver el problema descrito sin necesidad del resto de elementos. Se tomará un esquema algorítmico de tipo evolutivo como base de implementación, adaptándolo y mejorándolo en función de los requisitos y mecánica del escenario.

- *Diseño y ejecución de pruebas experimentales de evaluación.*
  Para comprobar la respuesta del algoritmo desarrollado, es necesario diseñar una serie de pruebas experimentales que sitúen las diferentes soluciones que se van alcanzando durante el ciclo de desarrollo de la implementación, en situaciones tanto normales como límites. Estas pruebas serán en primera instancia simulaciones, para evitar posibles daños tanto en un robot físico como en elementos de su entorno durante las primeras fases de implementación. Una vez se alcanza cierto nivel de estabilidad y bondad en la respuesta del algoritmo, se realizan también pruebas en entornos reales controlados.

2.2. Requisitos

Se desprenden los siguientes requisitos para la solución final:

- *Implementación de todo el sistema basada en la plataforma JdeRobot.*
  Este requisito facilita y reduce el tiempo necesario para el desarrollo de todos los elementos del proyecto. Fijar este requisito permite, entre otros, abstraer el problema de adquisición de información de los sensores en función del robot, facilitar un desarrollo modular interoperable y aprovechar componentes ya desarrollados en la plataforma. Se tratarán las ventajas de esta plataforma en el apartado 3.1.

- *Funcionamiento de todo el sistema en tiempo real.*
  Por la naturaleza del problema, un tiempo de respuesta elevado o un funcionamiento *offline* no permitiría resolver la cuestión de autolocalización. Por tanto, la solución debe ser capaz de responder en tiempo real utilizando como plataforma hardware máxima un ordenador de prestaciones medias-elevadas. Aunque se tendrá en consideración, no se tratará la optimización de la implementación del algoritmo para que utilice menos recursos del hardware.

- *Convergencia del algoritmo dentro de un margen de error determinado para casos no críticos.*
  Para considerar completamente válida la implementación, esta deberá devolver con cierto grado de precisión la posición tras ejecutar varias iteraciones del algoritmo con el robot en movimiento. Se permite no obstante que en casos límite como, falta de información sensorial o demasiados puntos de simetría sin información adicional en el recorrido del robot, el algoritmo pierda precisión o devuelva una posición errónea mientras mantiene en consideración otras posibles soluciones.

- *Implementación como algoritmo genérico en el espacio tridimensional.*
  Independientemente de los grados de libertad de movimiento del robot final contra el que se conecte la solución, la implementación debe ser capaz de considerar un robot móvil con los 6 grados de libertad $(x, y, z, \theta, \psi, \phi)$ y devolver posición y orientación del mismo dentro del espacio tridimensional.

- *Modularización de todos los componentes del sistema.*
  Para facilitar el uso de la implementación resultado de este proyecto fuera del entorno de pruebas y permitir la reutilización de este entorno tanto en las distintas fases de implementación como en futuros proyectos que puedan hacer uso del mismo, se ha decidido que sea requisito que la solución elaborada sea lo más modular y desconexa posible.

2.3. Metodología

A la hora de realizar o trabajar en un proyecto de cierta envergadura es necesario establecer un sistema de trabajo o *metodología* para mantener cierto orden durante todos los pasos y fases del desarrollo del mismo.

En el caso de este proyecto se ha seguido una variante del modelo de desarrollo en espiral. Este modelo plantea el proyecto en una serie de ciclos, donde cada uno termina con un prototipo más completo de lo que se disponía en el ciclo anterior. Cada ciclo se divide en las fases de planificación de objetivos, análisis de requisitos, diseño e implementación y pruebas.

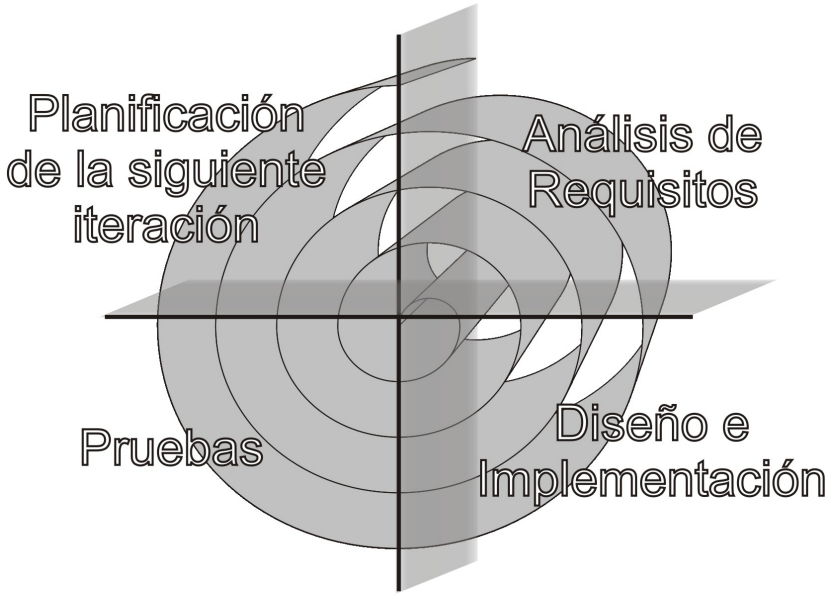


Figura 2.1: Representación del modelo de desarrollo en espiral

Planificación de objetivos En esta fase se deciden qué objetivos debe alcanzar la siguiente iteración para considerarla concluida y de qué manera se piensan abordar en base a cómo terminó el ciclo anterior. En el caso del primer ciclo, se definen los objetivos iniciales.

Análisis de requisitos En esta fase se estudia qué requisitos son necesarios para alcanzar los objetivos propuestos a través de la aproximación decidida en la fase anterior, los riesgos que podrían surgir y estrategias para evitarlos o paliarlos.

Diseño e implementación En esta fase se diseña e implementa el producto o prototipo esperado en el ciclo en base a los objetivos, requisitos observados y la planificación de las fases anteriores.

Pruebas En esta fase se diseñan y ejecutan una serie de pruebas para validar y depurar el producto para dar el ciclo por concluido.

Para las fases de planificación y análisis de requisitos se han mantenido reuniones semanales con el tutor, en las que se revisó el estado de los puntos de cada etapa y se establecían nuevos objetivos. De cara a documentar los hitos en el desarrollo del proyecto, se han mantenido un mediawiki [5] y un repositorio de código fuente *subversion* [4] públicamente accesible.

2.4. Plan de trabajo

El desarrollo de este proyecto ha seguido el siguiente plan de trabajo:

1. Aprendizaje de la infraestructura de JdeRobot.
   Con el objetivo de familiarizarse con el funcionamiento de JdeRobot se plantea la elaboración de varios componentes sencillos que se conecten a sensores tipo cámara o que devuelven valores simulados y operen con los datos sensoriales.

2. Aprendizaje de la infraestructura del simulador Gazebo.
   Para saber controlar e incluir elementos controlables en el simulador que se utilizará en el proyecto, se plantea la elaboración de un plugin que simule tanto los sensores con ruido como el manejo de uno de los robots utilizados posteriormente en experimentos.

3. Elaboración de un entorno de pruebas modular para el algoritmo.
   Con la idea de visualizar y simplificar los siguientes ciclos, se establece el objetivo de diseñar un entorno de pruebas que sea capaz de ejecutar una implementación del algoritmo y muestre algunos detalles del mismo. Se establecen como objetivos el dotar de capacidad de carga de módulos al entorno de pruebas, permitiendo mayor rapidez en el desarrollo y prueba del algoritmo en siguientes ciclos, y mejorar las representaciones 3D y de gráficas, para poder visualizar y analizar mejor sus resultados.

4. Diseño e implementación del algoritmo de autolocalización.
   En este ciclo se planifica una primera implementación de lo que será el algoritmo de autolocalización, permitiendo a su vez realimentación sobre el funcionamiento del entorno de pruebas. Como segundo paso se establece la transformación de todos los elementos del algoritmo que asumían un plano bidimensional como área de trabajo en elementos que soportasen una visión tridimensional completa.

5. Ciclos de mejora y experimentación de la implementación del algoritmo.
   Esta serie de ciclos se orienta a mejorar iteración a iteración la respuesta del algoritmo, mejorando sólo lo imprescindible el entorno de pruebas respecto al ciclo anterior. También se elaboran, ejecutan y verifican los experimentos tanto simulados como reales que permitirán la validación del algoritmo como solución del problema.

6. Recopilación de documentación generada y escritura de la memoria.
   En este ciclo se comprueba, ordena y clasifica la documentación generada y consultada a lo largo del proyecto para su inclusión en la memoria del proyecto. También es el ciclo donde se redacta y comprueba esta memoria.

3 Infraestructura

Para la realización de este proyecto se ha utilizado una serie de bibliotecas, aplicaciones y plataformas *software* para centrar el desarrollo principal en el núcleo del proyecto.

A continuación se describirán la plataforma base, *JdeRobot*, sobre la que se ha desarrollado la aplicación del proyecto; *Gazebo*, programa de simulación de robots y entornos; *Eigen*, la biblioteca utilizada para cálculos vectoriales y matriciales; *OpenGL*, API sobre la que se ha desarrollado la visualización del mundo 3D en la aplicación; *GTK+*, bibliotecas para diseñar y mostrar la interfaz de usuario; y *Boost*, utilizado para ciertas funcionalidades adicionales.

3.1. JdeRobot

JdeRobot[1] es una plataforma software desarrollada por el grupo de robótica de la Universidad Rey Juan Carlos destinada a facilitar la creación de aplicaciones en el ámbito de la robótica, visión artificial, domótica, y en general, aplicaciones basadas en sistemas inteligentes que hagan uso de sensores y actuadores variados.

La plataforma se basa en distintos componentes modulares e independientes con una funcionalidad concreta que pueden trabajar de forma distribuida e interoperan entre sí a través de una serie de interfaces estandarizados. De esta forma, los módulos encargados de obtener información de sensores o enviar una orden a los actuadores abstraen su funcionamiento interno y pueden ofrecer interfaces similares a aquellos que hacen uso de estos elementos.

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

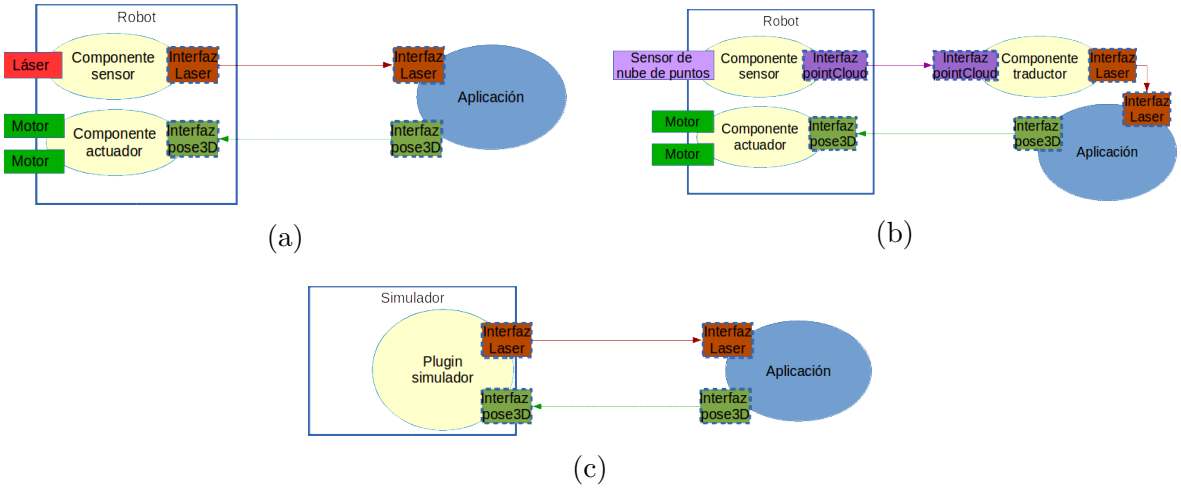


Figura 3.1: Esquema ejemplo de comunicación intermodular para una aplicación basada en JdeRobot. Se muestran las comunicaciones para un robot real (a), un sensor que será visto como otro tipo de sensor por la aplicación (b) y una simulación (c).

Además de interfaces y componentes básicos, la plataforma ofrece una serie de bibliotecas que simplifican distintas formas de intercomunicación entre componentes u operaciones relacionadas con el ámbito de aplicación de JdeRobot. Un ejemplo de estas bibliotecas es la que ofrece soporte de interpretación, manipulación y representación de ficheros en formato COLLADA[2]

Un *componente* es la unidad modular funcional e independiente más básica dentro de la plataforma JdeRobot. De acuerdo a su funcionalidad, los componentes pueden diferenciarse en los siguientes tipos:

Drivers

Están destinados a interactuar con dispositivos finales, sean reales o simulados. Entran en esta categoría componentes que actúan como fuentes de información sensorial, recogiendo y tratando los datos de los sensores para poder enviar su información a través de una o más interfaces. También pertenecen a este tipo sumideros que reciben información por una o más interfaces y la traducen en órdenes para los actuadores. Un ejemplo de los *drivers* genéricos que ofrece la plataforma es `CameraServer`, que sirve información de un sensor de tipo cámara.

En este proyecto, se han utilizado *drivers* específicos para poder acceder a los sensores laser y manejar los motores tanto de los robots reales como de los simulados.

[2] Formato basado en XML para intercambio de recursos 3D. Ver `https://collada.org`

Herramientas
: Están pensados para conectarse a otros y extender su funcionalidad o permitir la explotación de la misma. Al contrario que los anteriores, no suelen interactuar directamente con el dispositivo final. Entran en esta categoría componentes de tipo visor, controladores manuales, generadores y registradores de datos derivados de los sensoriales, y algoritmos para detección o actuación en general.

  Dos ejemplos notables, utilizados en este proyecto, son las herramientas `Recorder` y `Replayer`. La primera permite recoger la información de distintos sensores y volcarla a una serie de ficheros para tratarla posteriormente; la segunda, toma los ficheros generados por la anterior para replicar la fuente de datos original. De esta forma, se pueden comparar los resultados de distintas versiones del mismo algoritmo de autolocalización o distintos algoritmos asegurando que los datos sensoriales de entrada son exactamente los mismos para cada prueba.

Las *interfaces* son los elementos de intercambio de información entre componentes. Entre las distintas interfaces disponibles dentro de la plataforma, por su uso en este proyecto se destacan:

Pose3D
: Describe una posición y orientación en un espacio tridimensional como un vector (x, y, z, h) y un cuaternión (q_0, q_1, q_2, q_3).

Laser
: Describe la lectura de un sensor de tipo laser mediante un vector de distancias y un valor que indica cuántas medidas contiene dicho vector.

PointCloud
: Describe la lectura de un sensor de tipo nube de puntos mediante un vector de puntos tridimensionales con información de color RGB.

En JdeRobot las bibliotecas y componentes principales están programados en C y C++, aunque por su sistema de interconexión, permite el desarrollo de componentes en otros lenguajes de programación y su distribución sobre distintos sistemas operativos. Esta interconexión se basa en el *middleware* ICE[3], desarrollado por ZeroC y disponible bajo licencias GNU GPL o comercial. Está enfocado a facilitar el desarrollo de aplicaciones distribuidas cubriendo sus necesidades de comunicación entre módulos. Mediante el lenguaje *SLICE (Specication Language for ICE)* se definen las distintas interfaces de la plataforma, siendo después traducidas al lenguaje o lenguajes de programación de los componentes.

El proyecto hace uso de la versión 5.2 de esta plataforma para abstraer el tipo y funcionamiento de los sensores y motores de la implementación principal, permitiendo así su uso en distintos dispositivos y robots.

[3] `http://www.zeroc.com/ice.html`

3.2. Simulador Gazebo

Gazebo[4] es un simulador multi-robot tridimensional. Es capaz de simular una población de robots, sensores y objetos, así como las interacciones físicas producidas y las respuestas sobre los sensores en los mundos tridimensionales simulados. Gazebo está en desarrollo activo bajo los auspicios de la Open Source Robotics Foundation y disponible bajo licencia Apache 2.0.

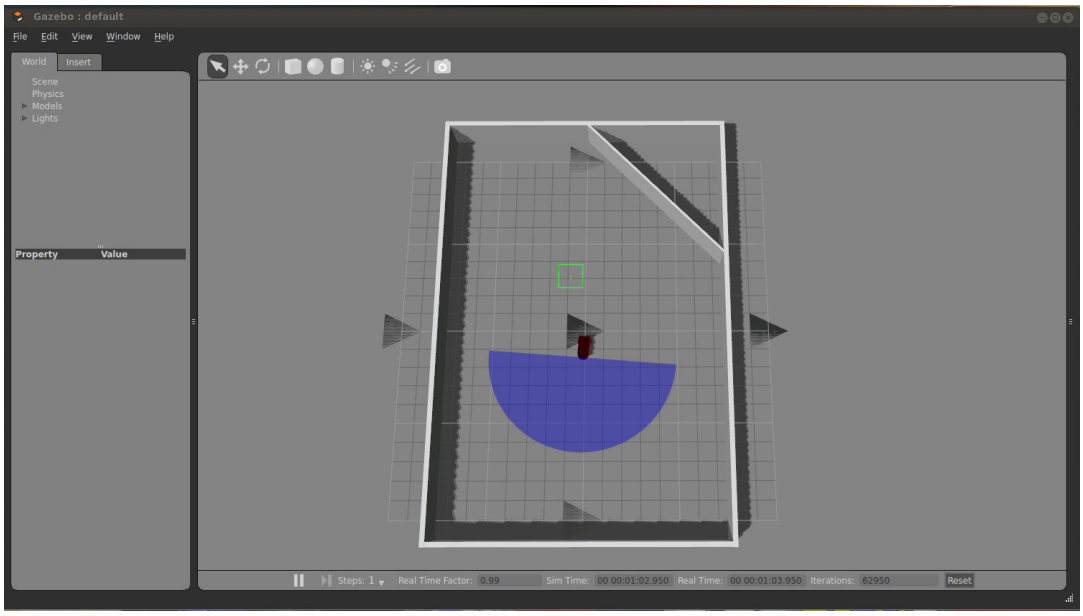


Figura 3.2: Ejemplo de simulación de un robot en un mundo tridimensional en Gazebo

El simulador Gazebo nació dentro del proyecto Player/Stage[5] como simulador 3D para el sistema Player, pero en el año 2011 se transformó en un proyecto independiente auspiciado por Willow Garage, un laboratorio de investigación robótica dedicado al desarrollo de software de código libre. En 2012 recibió financiación de parte de *DARPA (Defense Advanced Research Projects Agency)*, convirtiéndose en la base de la plataforma del DARPA Robotics Challenge[6]. Gracias a ello y a su uso extendido por parte de grandes proyectos de la comunidad robótica como ROS[7] se ha convertido en el simulador estándar de facto de esta comunidad.

El simulador dispone de diversos modelos predefinidos y permite definir nuevos modelos y mundos, además de integrar *plugins* para definir el comportamiento de los diversos elementos simulados, como pueden ser las respuestas de los sensores ante estímulos del mundo o de los actuadores a las órdenes recibidas.

[4] `http://gazebosim.org`
[5] `http://playerstage.sourceforge.net`
[6] `http://gazebosim.org/wiki/DRC`
[7] Robot Operating System: `http://www.ros.org`

En este proyecto se hace uso de Gazebo para simular el comportamiento de los robots sobre los que se probará el algoritmo desarrollado de forma realista antes de probarlos sobre las versiones reales de los mismos. Esto permite realizar pruebas sobre los algoritmos en distintos mundos y situaciones límites de forma controlada, ya que disponemos de los datos veraces sobre la simulación del robot (*verdad absoluta*).

Con el fin de mantener una compatibilidad con el resto de componentes, a pesar del rápido desarrollo de versiones del simulación, se ha optado por utilizar versión 1.8 de Gazebo.

3.3. Eigen

Eigen[8] es una biblioteca basada en plantillas de C++ para álgebra lineal. Define clases, algoritmos y funciones para matrices, vectores, análisis numéricos y otros elementos relacionados con este área. Eigen está disponible bajo licencia MPL 2.0.

Esta biblioteca está diseñada no sólo de cara a la versatilidad de uso, que ofrece a través del soporte para distintos de tamaños y tipos de vectores, matrices y algoritmos, sino que además se preocupa de la fiabilidad de estos y trata de ofrecer la mayor rapidez sin comprometer en la medida de lo posible la anterior. Además, ofrece una API bastante fácil de comprender y utilizar.

Dentro de este proyecto se hará uso de los vectores, matrices, cuaterniones y operaciones básicas relacionadas ofrecidas por esta biblioteca en su versión 3.2.

3.4. OpenGL

Open Graphics Library[9] es una API multiplataforma y multilenguaje para la generación de gráficos vectoriales bidimensionales y tridimensionales. Generalmente se utiliza para interactuar con una GPU, pero también hay disponibles implementaciones de la API basadas en software.

Esta API permite definir el trazado de escenas bidimensionales o tridimiensionales relativamente complejas a partir de primitivas sencillas como puntos, rectas o triángulos, además de su presentación con funciones que definen la iluminación, color y otras propiedades de la escena en cuestión.

[8] `http://eigen.tuxfamily.org`
[9] `http://www.opengl.org`

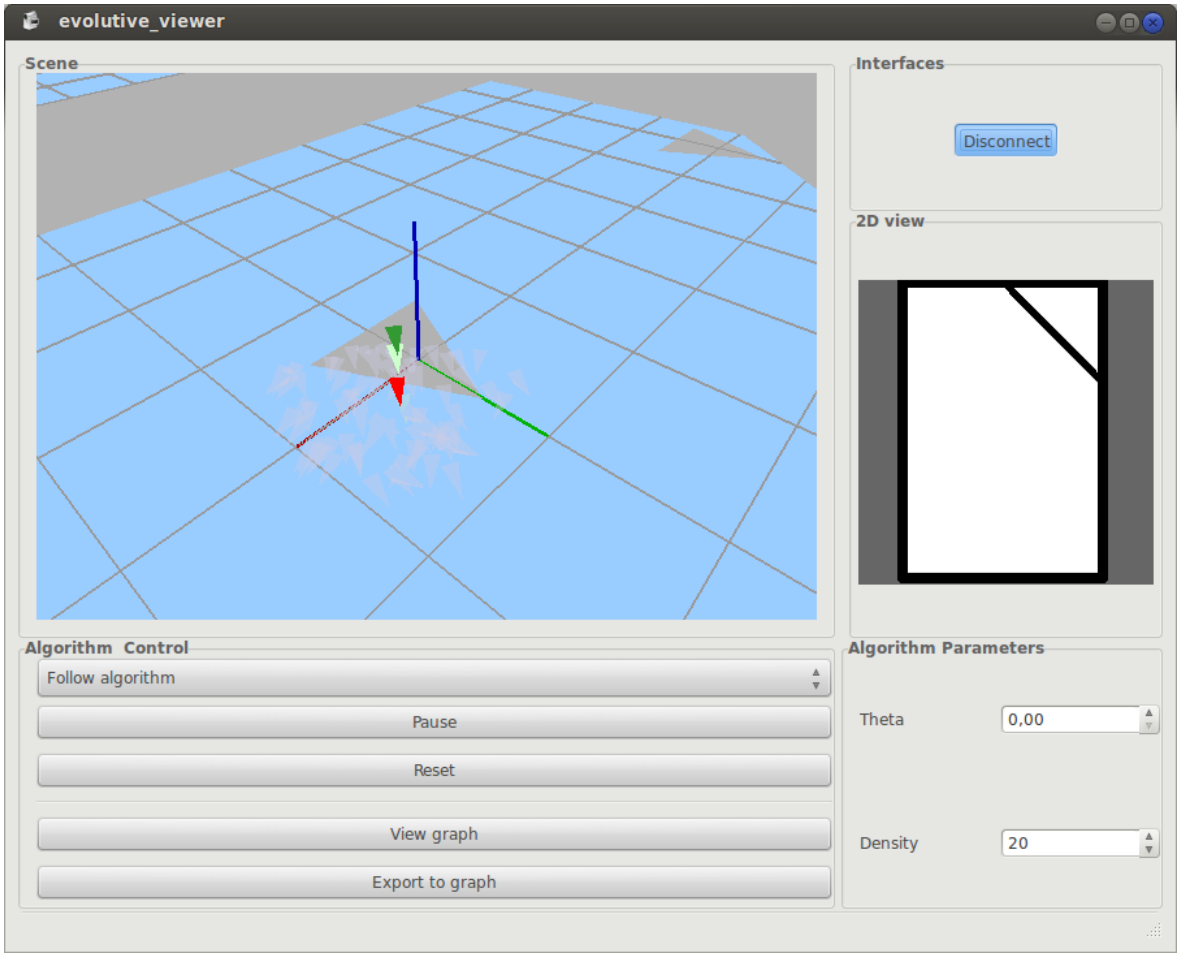


Figura 3.3: Interfaz de usuario de la aplicación principal del proyecto

Tal y como se observa en la figura 3.3, en este proyecto se han aprovechado las capacidades de OpenGL para generar una representación de las distintas áreas o mundos donde se utilizará el algoritmo de autolocalización, así como una rejilla base para facilitar la localización visual dentro de los mismos y otros elementos de interés como son la localizaciones real, estimada y ofrecida por encoders del robot, las partículas consideradas y los indicadores de razas.

3.5. GTK+

GTK+[10] o The GIMP Toolkit es un conjunto de bibliotecas multiplataforma diseñada con el objetivo de desarrollar interfaces gráficas de usuario. Licenciado bajo los términos de LGPL, GTK+ es software libre y es parte del proyecto GNU.

Aunque está escrita en C, su diseño permite el uso de otros lenguajes como C++, Fortran, Java, Perl, PHP, Python o Ruby. Concretamente, en este proyecto se utilizará gtkmm[11] que es la interfaz para C++.

De las bibliotecas incluidas bajo GTK+, cabe destacar GTK que es la que ofrece de forma directa todos los componentes gráficos para generar una interfaz, como pueden ser botones, menús y cuadros de texto e imagen.

Este proyecto hace uso de GTK+ (en su versión 2.24), como se puede ver en la figura 3.3, para ofrecer la interfaz de usuario principal a través de Glade[12], un editor y bibliotecas que permiten definir ventanas y componentes gráficos a través de ficheros XML externos a las aplicaciones para su carga dinámica desde estas.

3.6. Boost

Boost[13] es un conjunto de bibliotecas diseñadas para extender las capacidades de C++, añadiendo funcionalidades más o menos básicas no disponibles en la biblioteca estándar o extendiendo las ya existentes.

Originalmente desarrollada para suplir carencias en el lenguaje C++ y basado fundamentalmente en cabeceras y plantillas, Boost está pensado para soportar su uso en múltiples sistemas. Su carácter abierto y su licencia (Boost Software License) contribuye a su uso extendido tanto en aplicaciones abiertas como propietarias y a la revisión y contribuciones de personas inicialmente ajenas al proyecto.

Este proyecto hace uso en gran medida de algunas de las bibliotecas de punteros inteligentes, hilos y conversión de cadenas de texto en otros tipos básicos entre otras en las aplicaciones desarrolladas para el mismo. La versión de Boost utilizada ha sido la 1.53.

[10] `http://gtk.org`
[11] `http://gtkmm.org`
[12] `https://glade.gnome.org`
[13] `http://www.boost.org`

4 Autolocalización evolutiva multimodal

El núcleo de este proyecto consiste en el desarrollo de un algoritmo de autolocalización basado en el esquema evolutivo y con base multimodal. El algoritmo utilizará la base evolutiva para representar distintas posibilidades de localización del robot y decidir con la información sensorial recogida en todo momento cuál de todas es más probable que sea la posición real.

En este capítulo se expondrán la base fundamental y la implementación del algoritmo desarrollado en este proyecto.

4.1. Fundamentos de los algoritmos evolutivos

Los algoritmos evolutivos son un subconjunto de los denominados *algoritmos metaheurísticos*. Este tipo de algoritmos se utilizan cuando no existe un algoritmo de optimización exacta y específica para el problema o cuando no se puede implementar por restricciones técnicas o del propio problema. Se enfocan en obtener una solución al problema, sacrificando otros aspectos como la exactitud, la validez de la misma en todos los casos o el tiempo necesario para encontrarla.

Estos algoritmos siguen un proceso iterativo que busca, modifica o genera una operación basada en una heurística de menor nivel para encontrar la solución o soluciones más adecuadas posibles en base a las asunciones iniciales hasta alcanzar, si existe, una condición de parada.

Los *algoritmos evolutivos* son algoritmos de optimización que se inspiran en mecanismos de la evolución biológica como pueden ser reproducción, mutación, recombinación genética y la selección natural, con el fin de encontrar la mejor solución al problema.

El espacio de soluciones del problema se representa como un conjunto de *individuos* con una serie de propiedades que los distinguen entre ellos. Este conjunto de individuos o *población* parte de un estado inicial y con el paso del tiempo va *evolucionando*, es decir, modificando sus propiedades y mediante la creación de individuos más *saludables* y la extinción de los menos saludables. Conforme se produce esta evolución, los individuos se acercan paulatinamente a un estado donde alcanzarían la *salud máxima* que representa una solución válida dentro problema. Este comportamiento se puede representar de forma esquemática de la siguiente manera:

1. *Generación de la primera población.* Se generan los primeros individuos de forma aleatoria, posiblemente en base a ciertos parámetros iniciales.
2. *Evaluación de la salud de la población.* Mediante una función de salud, se evalua este indicador para todos los miembros de la población.
3. *Aplicación de los operadores genéticos.* Se aplican operadores que seleccionan, alteran y generan los individuos que pertenecerán a la siguiente generación.
4. *Eliminación de la población sobrante.* Los miembros que tengan menor salud son eliminados hasta que el conjunto total de la nueva generación tenga un número de miembros predefinido.
5. Repetir desde 2 hasta la condición de finalización.

La *función salud* varía de un problema a otro y se basa en una serie de características puntuables para cada individuo de la población. Estas puntuaciones se ponderan y suman en función de su importancia de cara a la salud final del individuo, de tal forma que a mayor salud, más cercano a la solución ideal se encontrará el mismo.

Los *operadores genéticos* permiten generar los individuos de la siguiente población. El criterio de generación es propio de cada problema y responden al propósito del algoritmo concreto, pudiendo definirse operadores específicos para cada caso. Algunos operadores clásicos utilizables son:

Elitismo selecciona los individuos más saludables de la población y los incluye en la nueva generación.

Cruce o reproducción genera un nuevo individuo a partir de características de dos individuos de la generación anterior.

Mutación genera un nuevo individuo alterando visiblemente y de forma aleatoria una o varias características de otro de la generación anterior.

Ruido térmico genera un nuevo individuo cercano a otro de la generación anterior.

Repulsión desplaza o destruye individuos demasiado cercanos entre sí.

4.2. Diseño general del algoritmo de autolocalización

El algoritmo desarrollado en este proyecto adapta el esquema evolutivo de forma que los individuos de la población representan estimaciones de distinta exactitud de la posición y orientación del robot.

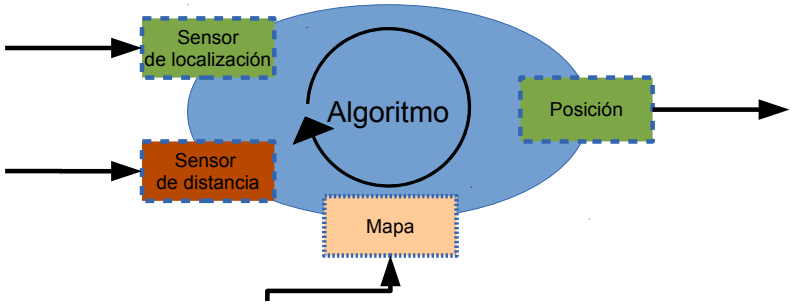


Figura 4.1: Diagrama de entradas y salidas del algoritmo

Para que este diseño funcione, se necesita: (1) un sensor de distancia informe de los obstáculos encontrados en las inmediaciones del robot, (2) un sensor que informe de forma aproximada de los desplazamientos y rotaciones que está realizando el robot y (3) algún tipo de mapa que permita establecer la relación entre todas las medidas y que establecerá la referencia para la posición que se devolverá como salida. Este algoritmo tienes un carácter cíclico por iteraciones, en el que cada iteración permitirá la evolución de los actores del algoritmo y refinará la solución ofrecida. Este esquema de entradas y salidas se puede observar en la figura 4.1.

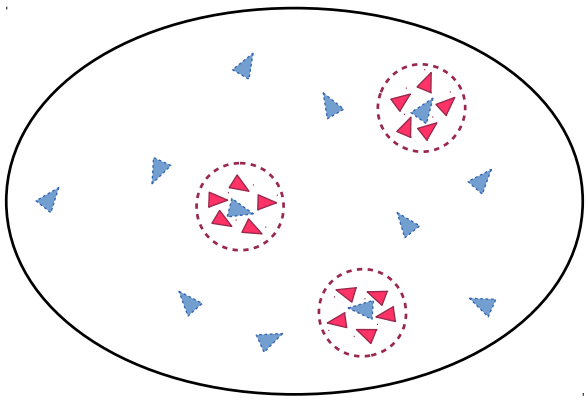


Figura 4.2: Representación esquemática de los actores del algoritmo dentro del espacio de soluciones. En azul con borde punteado, los *exploradores*; en rojo con borde continuo, los *explotadores*; círculos rojos punteados encierran miembros de la misma *raza*.

Para adaptar el esquema de algoritmo evolutivo a este problema utilizando las entradas y salidas descritas es necesario definir qué representarán cada uno de los conceptos y acciones en los que se basa el esquema genérico en el espacio del problema de la autolocalización. De esta forma el algoritmo considera los siguientes actores (figura 4.2):

Población es el conjunto completo de *individuos* relevantes para el problema.

Individuo describe una posible posición y orientación individual del robot tenidas en cuenta por el algoritmo. Está definido a partir de un par posición $\vec{p}$ y orientación $\vec{q}$ y se le asocia una salud l.

Raza es un conjunto de *individuos* con características de posición y orientación cercanas entre ellos que representa una de las posibles soluciones al problema de localización. Evolucionan a partir de un *explorador* suficientemente viable.

Explorador es un *individuo* independiente de otros encargado de realizar una búsqueda de grano grueso dentro del espacio de soluciones. Cuando localiza una posición y orientación suficientemente viables da lugar a la generación de una nueva *raza*.

Explotador es cada uno de los *individuos* pertenecientes a una *raza*. Su función es realizar una búsqueda fina de una posible solución, analizando la encontrada por un explorador y sus alrededores.

Básicamente, el algoritmo generará una serie de *exploradores* que permitirán la búsqueda de posiciones y orientaciones. Una vez que uno de estos exploradores encuentre un resultado viable, se generará una raza de *explotadores* utilizándolo como base y se buscará en esa zona la posición y orientación que de mejor resultado. De esta forma hay dos grupos de población diferenciados en cuanto a su papel dentro del algoritmo que se encargan de resolver el problema de localización a distinto nivel.

Definidos los actores y descrita su función, los pasos que sigue el algoritmo para cumplir con su cometido son (figura 4.3):

1. *Generación y actualización de exploradores.* Se actualizan los parámetros de los ya existentes de acuerdo a los cambios en el robot. A continuación se extinguen los que caigan fuera del espacio de soluciones.

2. *Evaluación de salud de exploradores.* Se evalúa la salud de todos los exploradores, extinguiendo aquellos que tengan un valor de salud muy bajo. A continuación se evolucionan los de mejores características hasta alcanzar la población máxima de exploradores.

3. *Generación de razas.* Se seleccionan los exploradores más prometedores que cumplan ciertas condiciones y se busca si ya existe una raza a la que pudieran pertenecer. De no existir, se genera una nueva raza a partir del explorador.

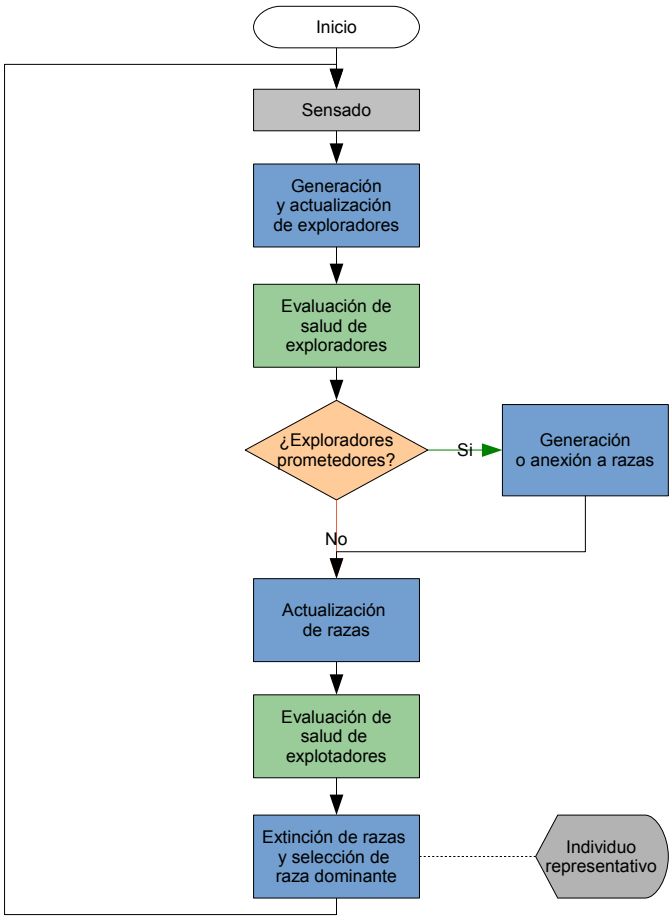


Figura 4.3: Esquema representativo del funcionamiento del algoritmo.

4. *Actualización de razas*. Se actualizan los parámetros de los explotadores existentes de acuerdo a los cambios en el robot. A continuación, en cada una de las razas se extinguen los individuos que caigan fuera del espacio de soluciones. Si alguna de las razas se queda sin explotadores que evaluar, se considera extinta y se destruye.

5. *Evaluación de salud de explotadores*. Se evalúa la salud de los explotadores dentro de cada raza. A continuación, en cada una de las razas se extinguen los individuos de baja salud y se evolucionan los de mejores características hasta alcanzar la población máxima. Nuevamente, si alguna de las razas se queda sin explotadores a partir de los que evolucionar a la siguiente generación, se considera extinta y se destruye.

6. *Selección de raza dominante*. Se establecen criterios de salud y ordenación entre razas. Aquella que ofrezca mejores resultados se selecciona como raza dominante y, por tanto, como respuesta definitiva del algoritmo en ese instante. Esta selección no extingue ni descarta las otras razas, que podrán pasar a ser dominantes en siguientes iteraciones. Por otro lado, si se ha llegado a superar el límite de razas, las menos saludables se consideran extintas y se destruyen.

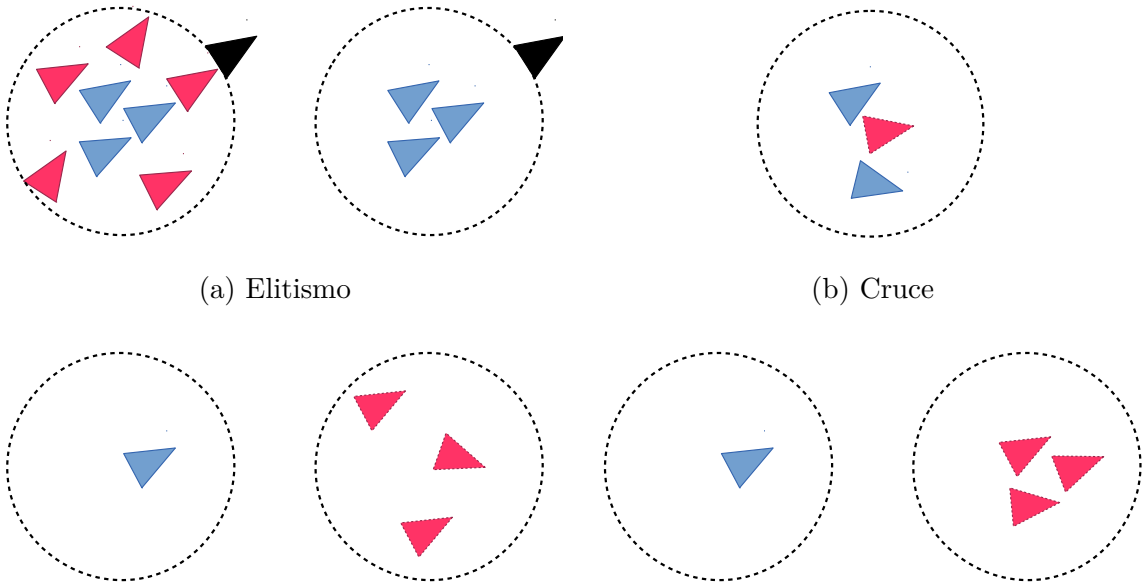
Para efectuar la *evolución* en exploradores y explotadores se aplican varios de los operadores genéticos descritos en la sección 4.1, concretados para el escenario de la autolocalización. De esta forma, los operadores aplicados en los pasos de evolución para los individuos son:

Elitismo selecciona los mejores individuos en base a su salud para incluirlos en la siguiente generación. Esta selección no varía ninguna de las características de los individuos seleccionados.

Cruce o reproducción toma como base los individuos elegidos mediante *elitismo* y realiza un número determinado de cruces entre sus características. Para ello, primero se toman dos individuos distintos I, con características $(\vec{p_I}, \vec{q_I})$, y J, con características $(\vec{p_J}, \vec{q_J})$. A continuación, se escogen unas nuevas características de manera aleatoria e independiente de forma que cada una se encuentre en el rango de valores formado por las características de los individuos escogidos, es decir, de forma que se cumpla que $(p_X \in [p_I, p_J], q_X \in [q_I, q_J])$. Este nuevo individuo X con características (p_X, q_X) formará parte de la siguiente generación.

Mutación toma como base individuos elegidos por *elitismo* y genera nuevos alterando completamente una de sus características. De forma más concreta, selecciona al azar una serie de individuos elitistas, toma su característica $\vec{p}$ de base y la modifica de forma aleatoria para generar nuevos individuos; a continuación se repite el procedimiento de selección, tomando esta vez su característica $\vec{q}$ para generar nuevos individuos.

Ruido térmico toma de base individuos *elitistas* y genera nuevos modificando *levemente* de forma aleatoria sus características. Dicho de otro modo, se selecciona una serie de elitistas y se genera un nuevo individuo por cada uno de la serie, utilizando para ello una modificación aleatoria incremental de su par $(\vec{p}, \vec{q})$ que no lo aleje mucho de los valores originales.



(a) Elitismo

(b) Cruce

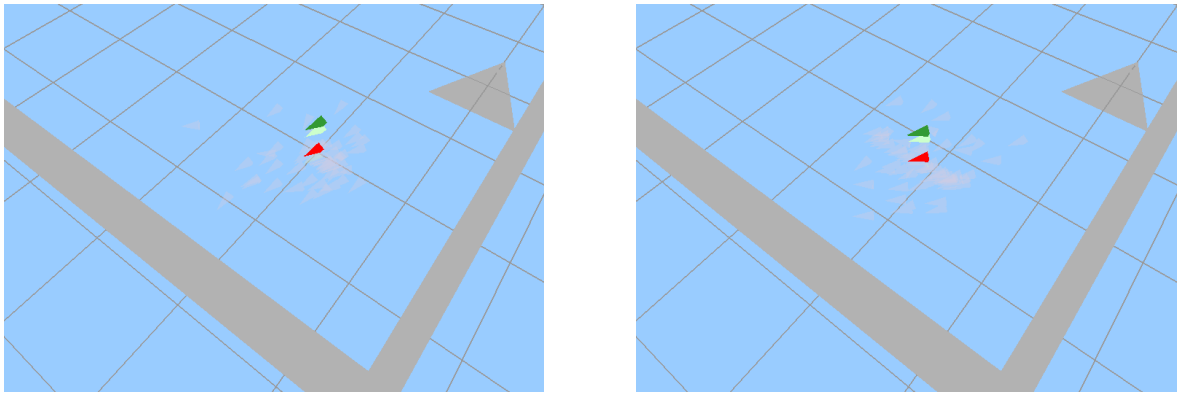
(c) Mutación

(d) Ruido térmico

Figura 4.4: Representación de los operadores genéticos utilizados en el algoritmo. Se representa como circunferencia discontinua la posición del robot con el margen de error aceptado y como flecha negra la orientación del robot (donde procede). En la figura (a), se representan en azul los mejores candidatos tomando de base la posición y orientación del robot; en la (b), se representa en rojo un posible cruce entre los individuos azules; en la (c), se representan en el lado derecho posibles mutaciones del individuo representado a la izquierda; y en la (d) se representan en el lado derecho posibles individuos resultantes de aplicar el operador de ruido térmico al individuo de la izquierda.

4.2.1. Generación y actualización de exploradores

Para crear la primera generación de exploradores se utiliza de base una posición inicial y se establece una dispersión aleatoria entre individuos tanto en posición como en orientación. De esta forma, se pueden aprovechar los datos de posición aportados por otras fuentes - por ejemplo, un *encoder* acoplado al robot - para iniciar la búsqueda.



(a) (b)

Figura 4.5: Ejemplo de actualización de exploradores, representados en color azul en degradado según su salud. Tras haber obtenido una generación de exploradores (a) en cierta iteración, en la siguiente se actualizan las posiciones (b) de acuerdo al movimiento del robot. Aquellos exploradores que quedarían fuera de las zonas válidas, se descartan.

En cada iteración del algoritmo se desplaza toda la población de exploradores la distancia recorrida por el robot según los sensores de odometría, respetando la orientación de cada individuo. Si tras el desplazamiento algún explorador cae fuera de la zona válida del mapa, se considera extinto a la hora de calcular la siguiente generación.

4.2.2. Evaluación de salud de exploradores

Una vez se han actualizado las posiciones y orientaciones de todos los exploradores se pasa a evaluar la salud de cada uno. En el algoritmo, la salud es una manera de medir la verosimilitud de la posición y orientación de un individuo en función de los datos del sensor de distancia del robot.

Para poder efectuar esta medida, el sistema se apoya en un *modelo de observación* que mediante el mapa que se dispone de la zona y la posición y orientación de un individuo, estima los valores que debería obtener el sensor de distancia para ese individuo. Según cómo sean de parecidas estas medidas a las obtenidas realmente por el sensor se establece el valor de la salud del individuo.

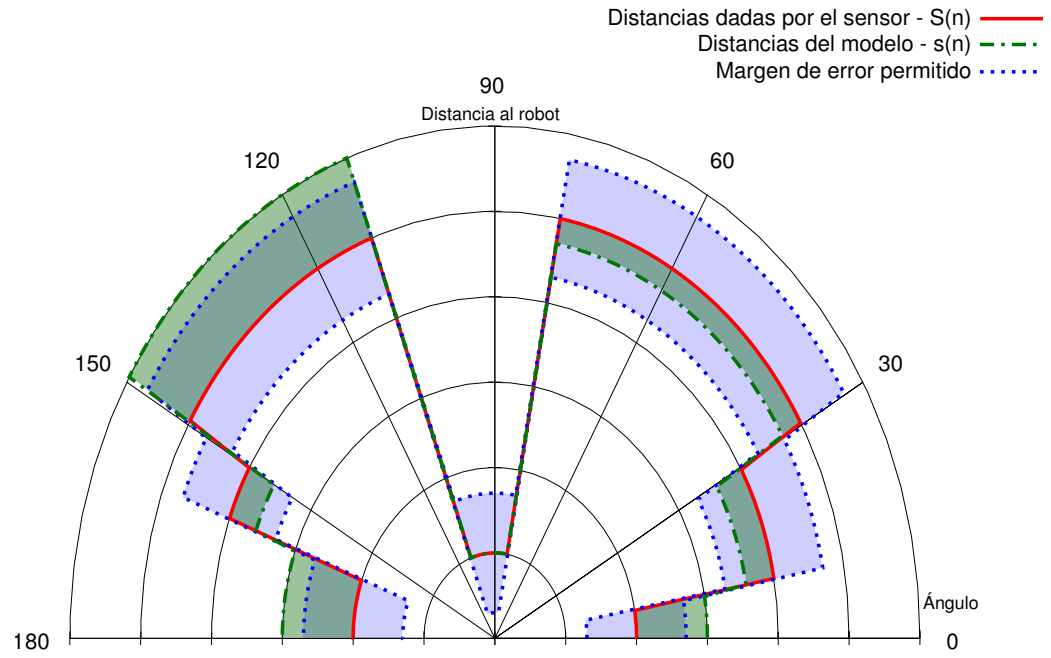


Figura 4.6: Modelo de medidas de distancia para la función salud. Se representan las medidas devueltas por el sensor del robot, las distancias contempladas por el modelo de observación y el error permitido (ϵ) dentro de la función salud. En degradados, el error cometido por el modelo (*verde*) y el margen de error permitido (*azul*).

Así, la función salud toma como entrada las distancias devueltas por los sensores y por el modelo de observación como vectores numéricos, los compara, y devuelve un valor (normalizado entre 0 y 1) con el porcentaje de distancias parecidas entre los vectores de entrada.

De manera más formal se puede definir esta función salud para un explorador como:

$$l = \frac{\sum_{n=1}^{N} \delta_L(n)}{N}$$

$$\delta_L(n) = \begin{cases} 1 & \text{si} |S_n - s_n| < \epsilon \\ 0 & \text{en otro caso} \end{cases}$$

En esta definición N corresponde al número de medidas que se obtiene por cada lectura del sensor de distancias; S_n es la medida de distancia n de los sensores reales; s_n corresponde a la medida n estimada a través del modelo de observación del robot para el explorador en cuestión y ϵ el error permitido entre ambas medidas.

Una vez obtenido el valor de salud para cada explorador, se buscan los individuos con el valor de salud más bajo y se declaran como extintos. Esta acción se realiza con el fin de mantener un número constante de exploradores y así evitar un coste computacional demasiado elevado en este paso. Además, permite descartar exploradores que potencialmente no proporcionarán una solución útil al problema de localización.

Tras estas acciones se obtienen los exploradores de la siguiente generación, aplicando los operadores genéticos - descritos en la sección 4.2 - de elitismo, cruce, mutación y ruido térmico. La proporción de generación de nuevos exploradores no es la misma para cada operador, siendo más importante el operador de mutación. Esto permite agrandar el espacio de búsqueda y no depender apenas de valores de exploración que puedan ser tratados por razas de explotadores.

4.2.3. Generación de razas

El sistema parte siempre de una raza inicial, siguiendo criterios similares a los de la primera generación de exploradores. Para el resto de iteraciones, una vez establecida la nueva generación de exploradores, se elige al mejor de ellos en función de su salud. Si existen demasiados exploradores que superen el criterio de salud, se decide que no hay un explorador *dominante* para esta generación. Si existe más de un explorador con alto nivel de salud, se elige como dominante a aquel que más se acerque a la posición estimada tras aplicar la información del sensor de odometría.

Si tras todas las consideraciones anteriores se decide que existe un explorador dominante, se comprueba si por sus características $(\vec{p}, \vec{q})$ puede ser incorporado a una raza ya existente. De no ser así, se genera una raza nueva tomando como base este individuo.

La generación de una raza a partir de un individuo se realiza mediante la creación de nuevos individuos con variaciones aleatorias de sus características, de forma similar a como funciona el operador de ruido térmico, con la salvedad de que la variación es más pequeña.

Este planteamiento implica que sólo puede aparecer como máximo una raza nueva por cada iteración, lo que a primera vista puede parecer una limitación importante; pero si se tiene en cuenta que se ejecutan varias iteraciones por segundo, esta limitación es poco apreciable frente la simplificación que supone en código y en coste computacional.

4.2.4. Actualización de razas

Una vez conocido el efecto del posible explorador dominante, se procede a actualizar los parámetros de todas las razas activas. Primeramente se desplazan todos los explotadores de la raza la distancia recorrida por el robot, respetando la orientación de cada individuo, tal y como se hizo en el apartado 4.2.1 para los exploradores. Tras el desplazamiento, cualquier explotador que tras la actualización quede fuera de la zona válida de localización se declara extinto, salvo el último explotador.

La decisión de mantener al último explotador de la raza permite disponer siempre de una respuesta del algoritmo, ya que la raza no se descarta en caso de ser la última aunque disponga de un único explotador, a la vez que si existen múltiples razas, se pueden extinguir las que estén dando las respuestas menos adecuadas al problema.

4.2.5. Evaluación de salud de explotadores

Tras el paso de actualización, se recalcula la salud para todos los individuos de la raza y se decide quién será el *explotador dominante*. Los principales criterios para decidir la dominancia dentro de una raza consisten en la *salud instantánea* de los miembros y en un "valor de confianza"que recibe la denominación de *crédito*.

Para entender el uso de los conceptos de *salud instantánea* y *crédito*, hay que recordar que este algoritmo es de base iterativa y que, por tanto, en lugar de pretender obtener una respuesta absoluta y veraz al final del mismo, trata de obtener una solución potencialmente válida e irla refinando tras cada iteración. De esta forma, mediante la *salud instantánea* se puede evaluar cómo de buena es la posición de un explotador respecto a otro en una misma iteración, lo cual no implica que en la siguiente tenga que seguir siendo así; mientras que con el *crédito* se tiene en cuenta durante cuántas iteraciones un explotador ha dado la mejor respuesta dentro de su raza para mantener cierta confianza en él.

Con este sistema se premia a un explotador que continuamente proporcione la mejor solución dentro de su raza, frente a uno que fortuitamente haya alcanzado un valor de salud similar en una iteración. Esto permite que el algoritmo tenga una buena tolerancia a fallos espúreos y a parecidos circunstanciales entre explotadores de la misma raza.

Sin más medidas este método corre el riesgo de acabar *confiando a ciegas* en un mismo explotador por el hecho de haber proporcionado la mejor solución durante varias iteraciones; siendo necesario que proporcione durante el mismo tiempo peores medidas que el resto para decidir el cambio de dominancia.

Para evitar esta situación se toman dos medidas: la primera es que se restringe el valor máximo que puede alcanzar el *crédito*, lo que limita el tiempo necesario para que otro explotador sea el nuevo dominante; y la segunda es que este valor decrece en función de la diferencia entre la *salud instantánea* del explotador dominante y la del explotador de mayor salud dentro de su misma raza, permitiendo que si un dominante produce valores demasiado erróneos sea más fácil que otro ocupe su lugar.

Todo este planteamiento en su conjunto proporciona estabilidad a la estimación ofrecida por la raza, evitando así oscilaciones innecesarias en la misma (*jittering*).

De una manera más formal, se puede expresar el cálculo del crédito de la siguiente manera:

$$c(t) = L_d - max(\vec{l})$$

$$C_s(t) = C_s(t-1) + c(t)$$

$$C_T(t) = \begin{cases} 0 & \text{si } c(t) < c_{min} \\ C_s(t) & \text{si } C_s(t) \leq C_{max} \\ C_{max} & \text{en otro caso} \end{cases}$$

Donde $c(t)$ es el *crédito instantáneo* para la iteración t, L_d la *salud del explotador dominante* actual $\vec{l}$ el vector de valores de *salud de los explotadores de la raza*, C_{max} el *límite máximo* que puede alcanzar el crédito total y $C_T(t)$ el *crédito total* del explotador dominante para la iteración t.

Así, un explotador se considerará dominante si su crédito total $C_T(t)$ se mantiene por encima de 0 y si su crédito instantáneo $c(t)$ no es demasiado bajo. En otro caso, el explotador con mayor salud tomará la dominancia y se reiniciará el valor de crédito total a su máximo.

Si una vez realizado este procedimiento se detecta que una raza no goza de salud suficiente y no es la última disponible, el algoritmo decide que la raza está *extinta* y deja de ser considerada en siguientes iteraciones. Tras finalizar la actualización en todas las razas, si éstas superan el número máximo para el sistema, aquellas de menor salud serán igualmente consideradas extintas hasta alcanzar el máximo de razas definido.

Por último, se aplican los operadores genéticos descritos en la sección 4.2 sobre los individuos de cada raza para formar la siguiente generación de explotadores.

4.2.6. Selección de raza dominante

Conociendo los individuos representativos de cada raza tras su actualización, se pasa a decidir cuál de todas representa mejor la solución al problema de localización, es decir, se selecciona la *raza dominante*.

El sistema de selección está basado en el sistema de *salud instantánea* y *crédito* descrito para los individuos explotadores en el apartado 4.2.5. En este caso se establece como *salud instantánea* de cada raza la de su explotador dominante y en cada iteración se enfrenta a la raza que tiene mayor salud instantánea con la que en ese momento se considera raza dominante. Tomando como referencia el procedimiento ya descrito, se aplica a esta situación para obtener la *raza dominante* para cada iteración. De nuevo, este procedimiento permite evitar el *jittering*, aunque esta vez a más alto nivel.

La raza que sea identificada como dominante será la respuesta que ofrezca el algoritmo al problema de localización; sin embargo, el resto seguirán siendo consideradas para próximas iteraciones, pudiendo ser en un futuro la siguiente raza dominante.

4.3. Implementación

Para la implementación y estudio del funcionamiento del algoritmo de localización anteriormente descrito, se ha desarrollado un componente JdeRobot escrito en C++ cuyo esquema de entradas y salidas sigue al que se ha presentado en el apartado de diseño. Este componente está escrito en más de 4900 líneas de código (de las cuales unas 1400 corresponden al módulo del algoritmo).

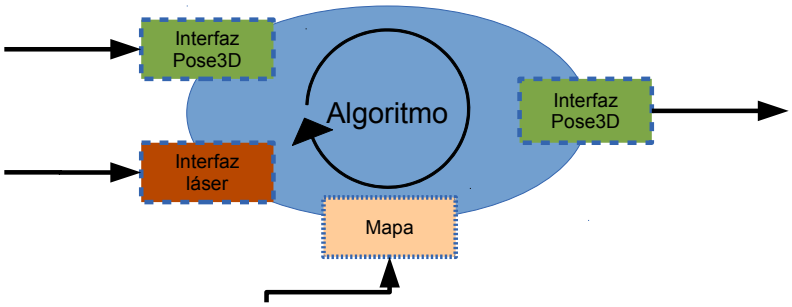


Figura 4.7: Diagrama de caja negra de implementación del algoritmo

Para cumplir todos los requisitos necesarios, el componente implementado debe tener una doble función: principalmente tiene que permitir conectar la implementación del algoritmo a una fuente sensorial cualquiera y producir una salida de posición, lo que supone el funcionamiento normal esperado del componente; y adicionalmente, debe permitir tanto la depuración del algoritmo como el estudio de los resultados que está produciendo.

El funcionamiento normal de la implementación sólo necesita ser capaz de manejar una entrada de datos de sensor de posición, una entrada de sensor de distancia y devolver una posición, que traduciéndolo a los elementos de la plataforma JdeRobot quiere decir que son necesarias las siguientes interfaces de entrada y de salida:

- Una interfaz ICE *Pose3D* para obtener una posición y orientación de entrada desde un sensor de posición, típicamente un sensor de odometría.
- Una interfaz ICE *Láser* para obtener un vector de distancias desde un sensor de distancias.
- Una interfaz ICE *Pose3D* para devolver la posición y orientación de salida.
- Una fuente de entrada de mapas *COLLADA* para obtener un mapa del entorno del robot.

Para cubrir la segunda funcionalidad, son necesarios un interfaz para monitorizar y controlar lo que está ocurriendo dentro de la implementación del algoritmo de localización y herramientas que permitan estudiar la respuesta de dicha implementación a distintos niveles.

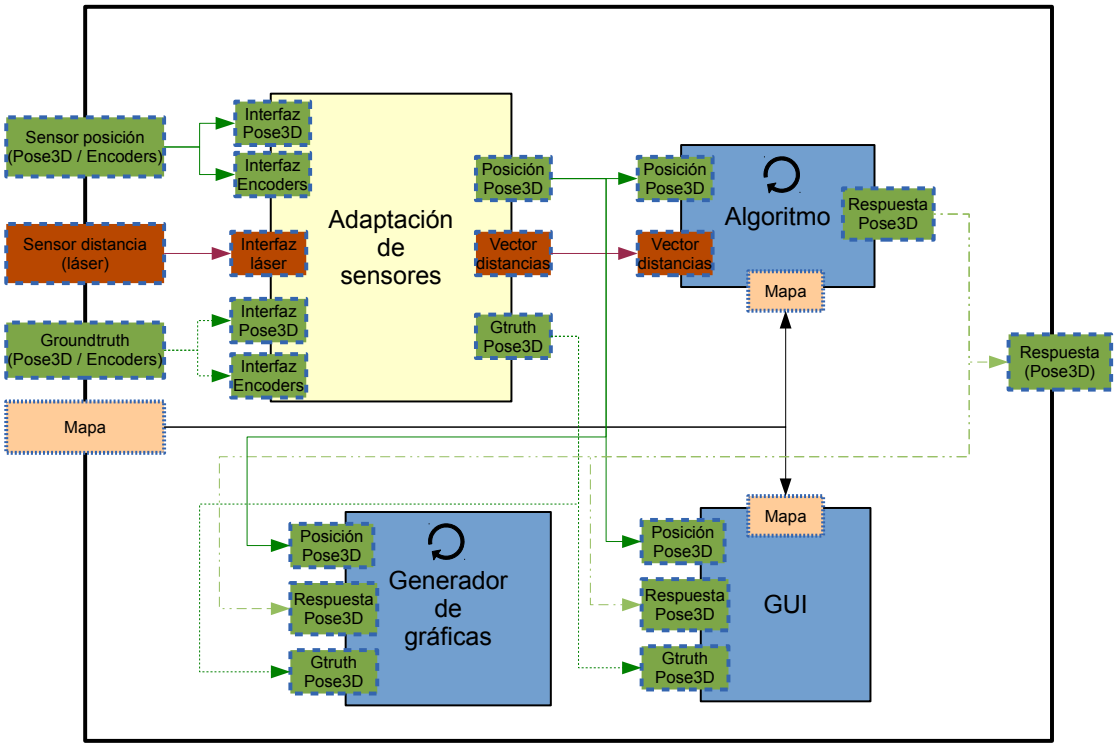


Figura 4.8: Diagrama interno de bloques de implementación. La figura muestra una vista esquemática de los bloques y sus conexiones que forman el componente.

De esta forma se ha optado por desarrollar un componente formado por distintos módulos (Figura 4.8), cada uno con una funcionalidad concreta:

- *Adaptación de sensores*. Encargado de conectarse a las distintas interfaces de entrada, su misión es hacer un tratamiento previo de la información de estos sensores si es necesario y asegurar que en su salida se ofrece siempre la información de la misma forma al resto de módulos. Al aislar la conexión de las interfaces del resto de bloques, también permite que ante un fallo en la comunicación, el componente pueda tratar de recuperarse sin necesidad de reiniciarlo.

  En esta implementación, se espera información de un sensor de posición a través de una interfaz *Pose3D* o *Encoders*, información de un sensor de distancia a través de una interfaz *láser* y, opcionalmente, información de referencia de verdad absoluta de posición (*groundtruth*) a través de una interfaz *Pose3D* o *Encoders*.

- *Algoritmo.* Es la implementación del algoritmo de localización propiamente dicha. Toma como entrada la información de un *Pose3D* en formato de vector y cuaternión $(\vec{p}, \vec{q})$ y el vector de medidas de distancia $\vec{S}$ desde el bloque de adaptación; por otro lado utiliza el mapa que se le ha suministrado al componente. En cada iteración que ejecute este bloque, utiliza esta información de entrada para devolver la posición estimada en el mismo formato de vector y cuaternión $(\vec{p_r}, \vec{q_r})$.
- *Interfaz Gráfica de Usuario.* Bloque de activación opcional, muestra una ventana que permite monitorizar las informaciones de posición del sensor del robot, la estimada y - si está disponible - la de referencia de verdad absoluta. Además, permite controlar la actividad del algoritmo, los modos de funcionamiento del componente y el bloque de generación de gráficas.
- *Generador de gráficas.* Bloque de activación opcional, recoge los datos devueltos por el adaptador de sensores y el algoritmo para generar, según el modo de funcionamiento, gráficas representativas de la respuesta del algoritmo. Estas gráficas se devuelven en forma de comandos para que un programa que entienda la sintaxis de `gnuplot`[1] lo interprete.

Para apoyar esta modularidad y permitir depurar o comparar distintos algoritmos mediante el mismo componente, el módulo de algoritmo se implementa como una parte externa al mismo cargada en tiempo de ejecución (*plugin*). Otras ventajas de este esquema modular es la facilidad a la hora de modificar el componente permitiendo tanto su reutilización en distintos ámbitos y condiciones, como por ejemplo la aplicación de nuevos sensores no utilizados en el ámbito de este proyecto o prueba del algoritmo en condiciones no consideradas por el componente.

El componente se apoya también en un fichero de configuración que sigue el formato reconocible por la biblioteca ICE[2]. De esta forma, el componente permite especificar el funcionamiento del sistema de comunicaciones entre componentes a la vez que define elementos propios de configuración.

También se ofrecen *funciones accesorias* como pueden ser la capacidad de *reiniciar* el modo de funcionamiento, *pausar* el bucle de ejecución principal sin interrumpir el sensado, y mostrar o exportar una *gráfica* con los datos más representativos según el modo.

Cabe notar que, aunque tanto el diseño del algoritmo de localización como el componente soportan el tratamiento de un mundo tridimensional y un robot con desplazamiento libre por el mismo, dado que las pruebas experimentales se han basado en robots con ruedas, se han establecido restricciones 2D con vistas a la simplificación de las operaciones del componente.

A continuación se describen las distintas funcionalidades del componente implementado.

[1] `http://www.gnuplot.info`

[2] Para una descripción más detallada, consultar la sección de ficheros de configuración del manual de ICE (`http://www.zeroc.com/doc/Ice-3.4.0/manual/Properties.31.3.html`)

4.3.1. Control interactivo: Interfaz Gráfica de Usuario

La *interfaz gráfica de usuario* o GUI es el principal método de control y visualización de datos relevantes del algoritmo de localización. La ventana principal muestra todos las operaciones básicas que puede efectuar el componente una vez inicializado. Se encuentra dividida en regiones dedicadas cada una a una parte de los cometidos de la aplicación.

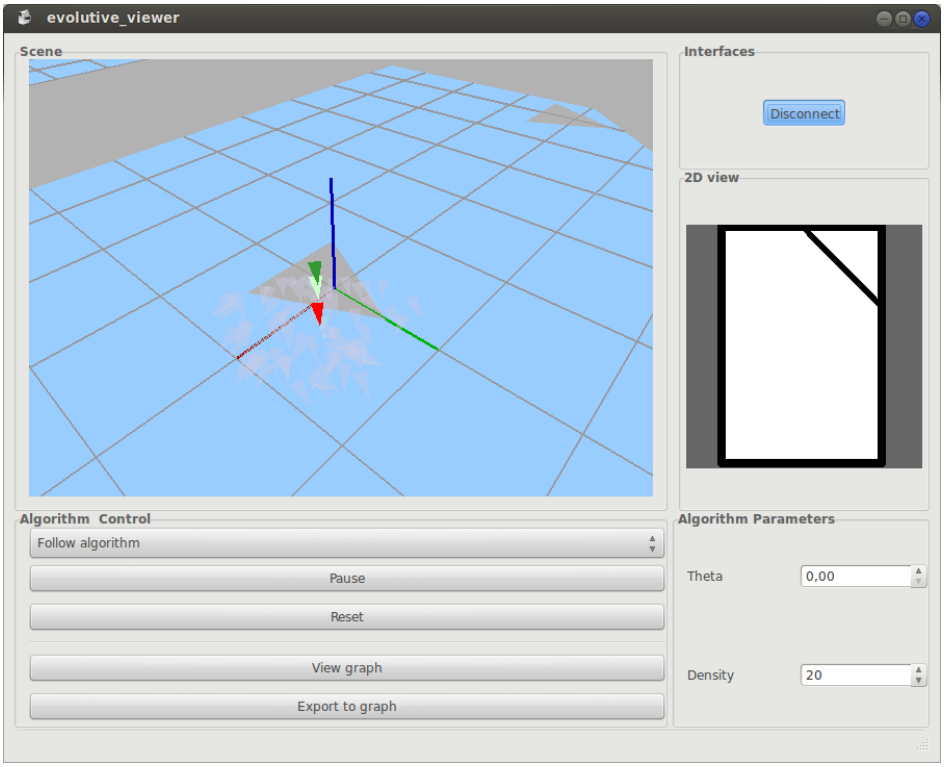


Figura 4.9: Vista de la interfaz de usuario

La región **escena** (*scene*) permite navegar a través una representación tridimensional del mundo visto desde el punto de vista del componente. En esta representación se pueden observar, además del mapa de la zona en consideración, una malla y base de coordenadas imaginarios que facilitan la localización del resto de elementos por parte del usuario.

También representa en distintos colores y degradados los siguientes elementos:

- La **localización del robot**, dada por el sensor de posición, en color **verde claro**.
- La **localización verdadera**, dada por el sensor de verdad absoluta o *ground-truth*, en color **verde oscuro**.
- **Otros elementos** dependiendo del modo de operación. Se indicarán en cada sección.

La región de **interfaces** controla todas las operaciones directamente relacionadas con la comunicación con otros componentes. En este caso las únicas operaciones permitidas son desconectar o conectar las interfaces definidas en la configuración.

La región de **vista 2D** (*2D view*) muestra una vista de la planta del mundo visible en la región de escena. Representa en color negro los obstáculos definidos en el mundo y en gris las zonas de acceso restringido para el algoritmo, siendo blanca cualquier región libre de obstáculos y navegable por el robot. En esta región de la ventana también se pueden definir el perímetro a cubrir para el modo de *mallado*, el punto central del modo de *orientación* y la corrección de posición de arranque del algoritmo en el modo de *seguimiento*.

La región de **parámetros del algoritmo** (*algorithm parameters*) controla los valores configurables de parametrización. Las opciones disponibles son las siguientes:

- Valor de ángulo θ (**theta**) en radianes[3]. Selecciona el ángulo respecto al eje de referencia $\vec{X}$ para los individuos en el modo de *mallado* y la inicialización para el modo de *seguimiento*.
- Valor de **densidad** de individuos. Selecciona cómo de densas serán la malla en el modo de *mallado* o la esfera en el modo de *orientación*.

La región de **control del algoritmo** (*algorithm control*) permite seleccionar, pausar la evaluación y reiniciar el modo de funcionamiento. Adicionalmente contiene las funciones ofrecidas por el módulo *generador de gráficas* que son mostrar la gráfica en tiempo real o exportar la información necesaria para generar una gráfica instantánea con datos de funcionamiento del modo actual.

[3]Los valores de ángulo respecto a los ejes $\vec{Y}$ y $\vec{Z}$ están inicializados a 0 y no son configurables desde la interfaz, como parte de las restricciones comentadas en el apartado 4.3

4.3.2. Adquisición de datos sensoriales

El algoritmo utiliza como datos sensoriales de entrada una medida de posición y orientación y un vector de medidas de distancia por cada iteración; tomando de referencia el formato ofrecido por las interfaces *Pose3D* y *láser*.

El módulo de *adaptación de sensores* recibe la información de conexión para las interfaces del sensor de posición, distancia y - si lo hubiera - verdad absoluta. Mediante esta información, busca qué tipo de interfaz de las soportadas para cada tipo de sensor es la que se está utilizando. Para este proyecto, el módulo dispone de soporte de las interfaces *Pose3D* y *Encoders* para los sensores de posición y verdad absoluta, y de la interfaz *laser* para el sensor de distancia.

Tras establecer la conexión, se realiza un tratamiento previo de los datos de entrada si es necesario para poder enviar al módulo del algoritmo información en formato *Pose3D* y *láser* de forma síncrona. En caso de fallo en la conexión, se asegura que a la salida del módulo de adaptación no aparezca ningún dato que pueda producir un fallo irrecuperable en el resto de módulos y la posibilidad de reconectar ante una petición externa al módulo.

Este comportamiento permite que el funcionamiento ante distintos *drivers* de la plataforma sea el mismo, facilitando el uso y la realización de experimentos con distintos robots tanto reales como simulados. También permite el uso de distintos sensores de distancia como láseres y sensores basados en tecnología PrimeSense (por ejemplo los sensores Kinect de Microsoft y Xtion de Asus), puesto que homogeneiza sus datos antes de dárselos como entrada al algoritmo de autolocalización.

4.3.3. Depuración del algoritmo: mallado de partículas

Como parte de la funcionalidad de depuración y estudio de resultados del algoritmo, el componente permite modos de funcionamiento específicos para estos casos distintos de la función principal de localización. El primero de estos modos es el de *mallado de partículas*.

En el modo de *mallado de partículas* el usuario debe indicar un valor de densidad de malla, orientación de partícula en el espacio y perímetro de la malla. Una vez establecidos estos parámetros e iniciado el modo, se genera dentro del perímetro definido[4] y con la separación determinada por el valor de densidad una serie de partículas con la orientación fija al valor establecido, utilizando la implementación dada por el módulo de algoritmo cargado. Tras la generación, en cada iteración del bucle principal de funcionamiento se aplica el modelo de observación y la función de salud, dadas también por el módulo de algoritmo, a cada una de las partículas.

[4]Por las restricciones descritas en el apartado 4.3, el espacio queda limitado a la superficie del plano XY.

La utilidad de este modo radica en poder observar la respuesta de la función salud para una implementación concreta de algoritmo en una serie de posiciones determinadas, facilitando el ajuste y estudio de la función salud o la comparación con otra implementación distinta.

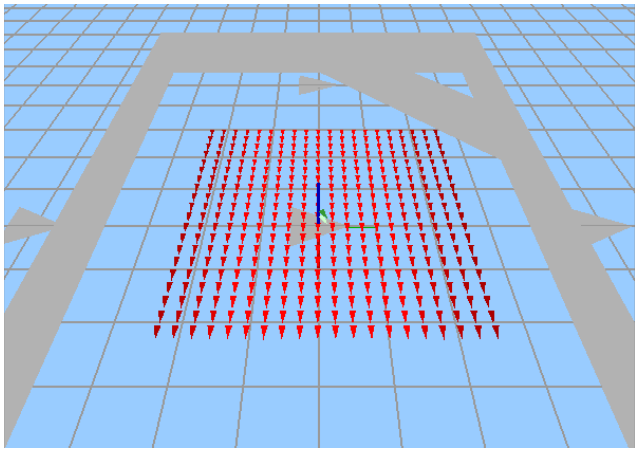


Figura 4.10: Ejemplo gráfico del modo *mallado de partículas*.

De cara a la visualización de resultados, se ofrecen dos tipos de presentación de cara al usuario. La principal consiste en una representación de la malla en la *Interfaz Gráfica de Usuario*, en la que se puede visualizar la malla de partículas (Figura 4.10) en relación al mapa y en la que cada partícula cambia el nivel de brillo en función de su salud; de esta forma, una partícula *rojo claro* tendrá más salud que una *rojo oscuro*. La segunda forma es una representación dada por el *generador de gráficas*, en la que la malla se expresa en un esquema de degradado similar a la representación principal (Figura 4.11) y con capacidad de ser exportada a un fichero para su uso posterior.

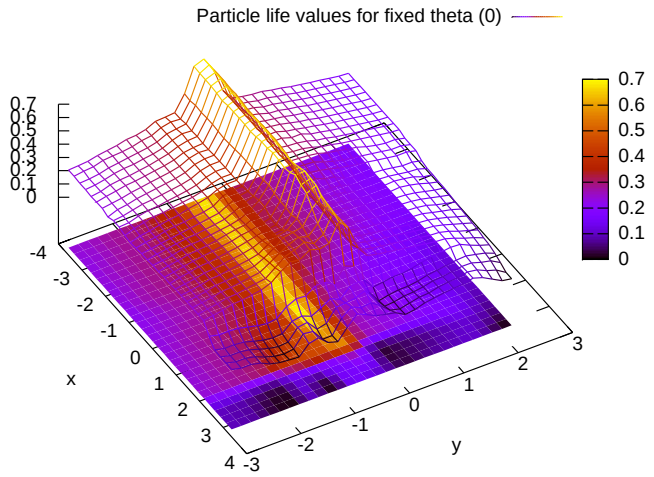


Figura 4.11: Gráfica representativa para el modo *mallado de partículas*.

4.3.4. Depuración del algoritmo: esfera de orientaciones

Otro de los modos ofrecidos dentro de la funcionalidad de depuración y estudio de resultados del algoritmo es el modo de *esfera de orientaciones*.

En el modo de *esfera de orientaciones* el usuario debe indicar un valor de densidad de la esfera y la posición de su centro. Una vez establecidos estos parámetros e iniciado el modo, se genera una serie partículas con posición el centro de la esfera y con distintas orientaciones[5], separadas en función del parámetro de densidad, utilizando la implementación dada por el módulo de algoritmo cargado. Tras la generación, en cada iteración del bucle principal de funcionamiento se aplica el modelo de observación y la función de salud, dadas también por el módulo de algoritmo, a cada una de las partículas.

La utilidad de este modo radica en poder observar la respuesta de la función salud para una implementación concreta de algoritmo en una posición determinada para cualquier orientación, facilitando la depuración del mismo o la comparación con otra implementación distinta.

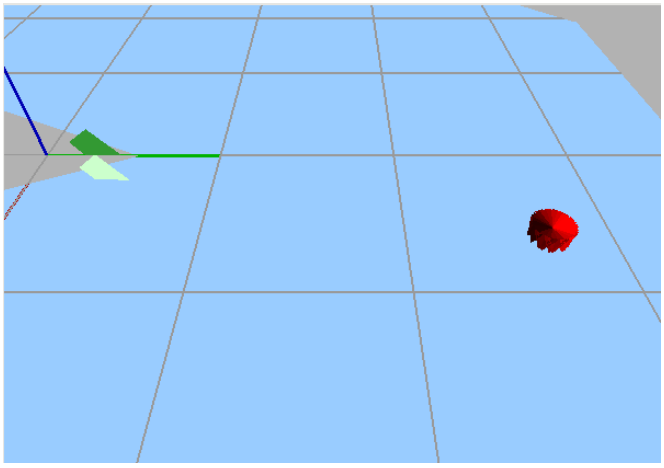


Figura 4.12: Ejemplo gráfico del modo *esfera de orientaciones*.

De cara a la visualización de resultados, también se ofrecen dos tipos de presentación de cara al usuario. La principal consiste en una representación de la esfera en la *Interfaz Gráfica de Usuario*, en la que se puede visualizar la esfera en relación al mapa (parte derecha de la figura 4.12) y en la que cada partícula cambia el nivel de brillo en función de su salud de la misma forma que en el modo de *mallado de partículas*. La segunda forma es una representación dada por el *generador de gráficas*, en la que la esfera se muestra de forma que el radio representa el valor de salud de cada punto (Figura 4.13). Al igual que el resto de gráficas dadas por el generador, tiene la capacidad de ser exportada a un fichero para su uso posterior.

[5]Por las restricciones descritas en el apartado 4.3 se ignorará la coordenada z del punto y se evaluará la esfera en el corte con XY.

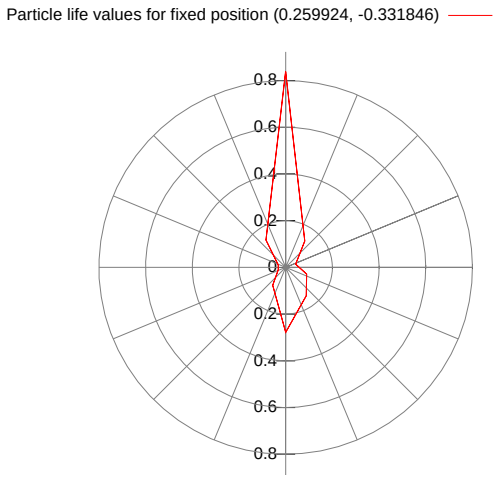


Figura 4.13: Gráfica representativa para el modo *esfera de orientaciones*.

4.3.5. Modo de seguimiento del algoritmo

De los modos de funcionamiento ofrecidos por el componente, el modo de *seguimiento del algoritmo* es el que utiliza la implementación completa del módulo cargado. En este modo el usuario debe indicar una posición y orientación de arranque. Una vez establecidos estos parámetros e iniciado el modo, se ejecutan continuamente los pasos del algoritmo indicados en el esquema de la sección 4.2 (figura 4.3). Este es el modo de operación principal del componente y es capaz de funcionar sin necesidad de los módulos opcionales de *interfaz gráfica* y *generador de gráficas*, sólo requiere de una configuración inicial adecuada.

En cuanto a la visualización de resultados, obviando la salida para que otro componente haga uso de ella, también se ofrecen dos tipos de presentación de cara al usuario. La utilidad de estas presentaciones adicionales es principalmente de retroalimentación para depuración de una implementación y para comparación entre implementaciones.

La principal consiste en una representación de los distintos actores del algoritmo en *Interfaz Gráfica de Usuario* (Figura 4.14), a saber:

- La **localización estimada** o respuesta del algoritmo, en color rojo.
- Los **exploradores**, en niveles de azul en función de la salud instantánea de cada uno. Un azul más vivo y menos transparente, indica mayor salud que uno más tenue y transparente.
- Los representantes de cada **raza**, en verde claro.

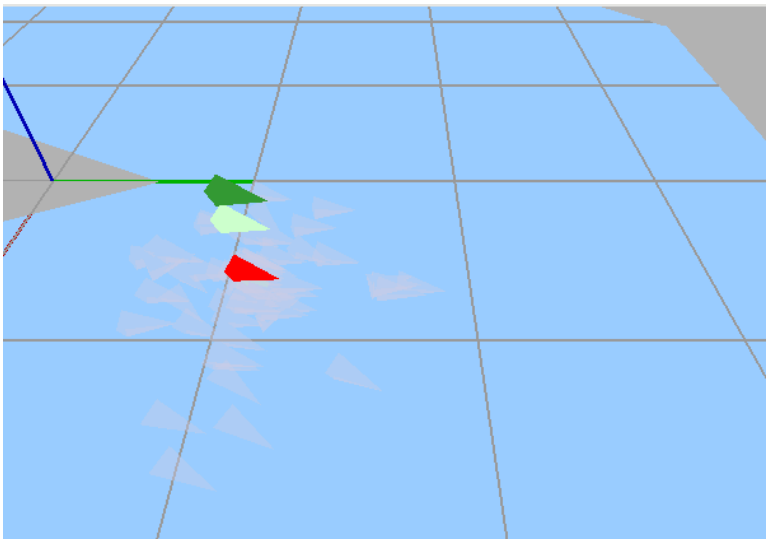


Figura 4.14: Ejemplo gráfico del modo *seguimiento del algoritmo*.

Esta visualización también permite de forma opcional mostrar las rutas dadas por la sucesión de estimaciones del algoritmo y la dada por el sensor del robot.

La segunda forma consiste en dos gráficas del *generador de gráficas* de representación de error (Figura 4.15): una para posición y otra para orientación. En ambos casos se muestran el error del sensor de posición frente a la verdad absoluta, el error de estimación frente al sensor y el error de estimación frente a la verdad absoluta. De no disponer de medida de verdad absoluta, las gráficas que hacen uso de ella se omiten.

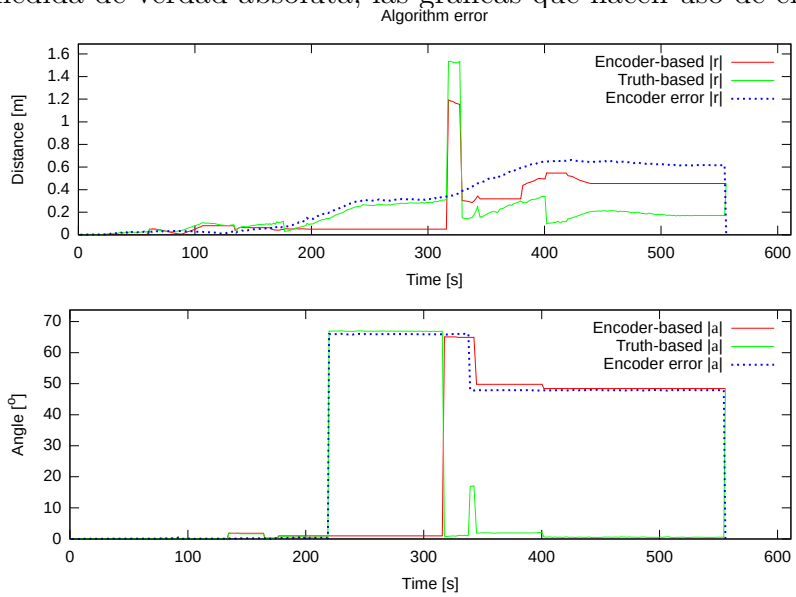


Figura 4.15: Gráfica representativa para el modo *seguimiento del algoritmo*.

4.3.6. Gestión del módulo del algoritmo: gestión de *plugins*

En el componente desarrollado para el proyecto, la implementación del algoritmo se encuentra en un módulo externo o *plugin*. Este módulo externo debe ser gestionado adecuadamente para que el hecho de no ser parte interna del componente sea transparente para todos los demás módulos del programa. El módulo responsable de esta función es el *gestor de plugins*.

La motivación de implementar un módulo dedicado a la gestión de los *plugins* viene principalmente del problema que supone que en C++, el lenguaje utilizado para implementar el componente, la *interfaz binaria de aplicación* (ABI) es dependiente del compilador utilizado, pudiendo diferir incluso entre versiones utilizadas; y dado que estos módulos externos pueden ser compilados de forma independiente al resto del componente, no se puede asegurar que se vaya a utilizar exactamente el mismo compilador. Por ello, se ha decidido que el único requisito de estos módulos estén obligados a ofrecer un ABI estándar del lenguaje C con ciertas interfaces y puntos de entrada predefinidos; siendo el método más común de implementación el de generar una biblioteca dinámica. De esta forma, se deja el lenguaje utilizado para la implementación a discreción del diseñador de cada módulo, siempre que se respete este requisito.

Por tanto, el *gestor de plugins* se encarga de incorporar en tiempo de ejecución estos módulos externos al componente y servir de enlace para acceder a sus funcionalidades.

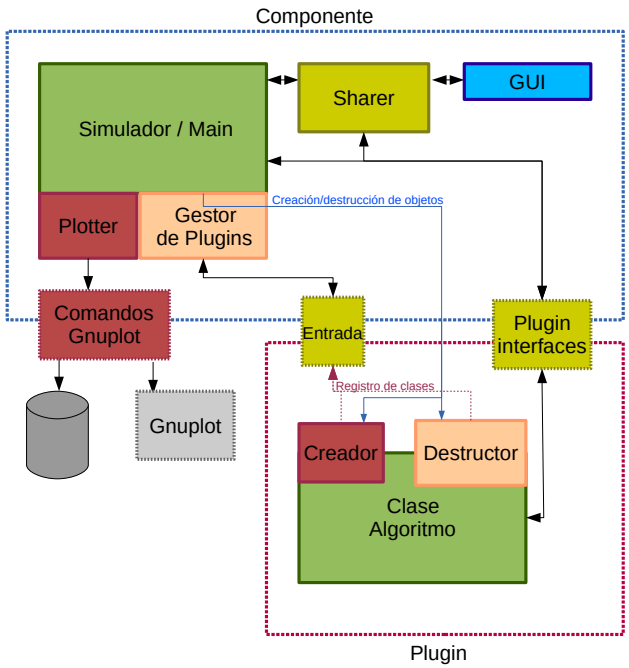


Figura 4.16: Diagrama de bloques del componente enfocado al sisgema de *plugins*. Destacan el bloque *principal (Main)*, la *Interfaz (GUI)* y el *plugin*; el bloque de *sensado* se considera dentro del principal en el diagrama.

Cuando el gestor recibe una petición de carga, localiza el módulo externo y su punto de entrada. El punto de entrada debe registrar todas las clases exportadas junto con sus métodos de creación y destrucción en el gestor. Entonces, cuando algún módulo interno del componente necesite un objeto de una de las clases ofrecidas por un módulo externo, pide el método de creación al gestor y lo utiliza para obtener un puntero a un nuevo objeto. Cuando se deja de necesitar un objeto, se llama al destructor registrado pasándole el puntero devuelto por el creador. Finalmente, cuando el sistema de *plugins* ya no es necesario, se solicita al gestor la finalización y descarga de los módulos externos abiertos. Se puede observar un diagrama de bloques de conexiones entre un *plugin* y el componente en la figura 4.16.

Este sistema de funcionamiento también permite que los *plugins* definan clases adicionales para funciones internas sin que el resto del componente sepa de su existencia. También permite, mediante el registro de servicios desde el componente, que un módulo externo utilice alguna función interna del componente.

4.3.7. Implementación del algoritmo en un *plugin*

Ya se han explicado las motivaciones y las bases de la implementación del módulo *gestor de plugins* en la sección 4.3.6, pero es necesario también desarrollar cómo está constituido internamente uno de estos módulos, y en especial, el que contiene la implementación del algoritmo de autolocalización final utilizado en este proyecto.

Como ya se ha comentado, para que un componente pueda hacer uso de él, un módulo externo debe ofrecer una o varias de las interfaces predefinidas por el gestor. Dentro de las posibles interfaces de clases que pueden implementar los *plugins*, las que están relacionadas con la implementación de un módulo de algoritmo son:

Partícula Representa un individuo de la población. Contiene información principalmente de su localización en forma $(\vec{p}, \vec{q})$ - o $(\vec{p}, \theta)$ si la implementación no soporta espacios en tres dimensiones de forma explicita - e información sobre su salud l. Permite acoplarle un modelo de observación para actualizar el valor de salud.

Colisionador Implementa un modelo de observación basado en cálculo de colisiones de rayos (Figura 4.17). Admite como entrada el mapa de la zona a considerar y tiene un método para solicitar el cálculo de una colisión.

Algoritmo Clase principal de la implementación del algoritmo. Hace uso de las anteriores y, si es necesario, de otras implementadas internamente por el módulo. Admite como entrada el objeto encargado del sensado, el mapa de la zona a considerar, una matriz de áreas restringidas, un *colisionador* y opcionalmente una localización de inicio. Ofrece métodos para reiniciar el algoritmo, ejecutar una iteración y recuperar la lista de exploradores, representantes de razas y localización estimada.

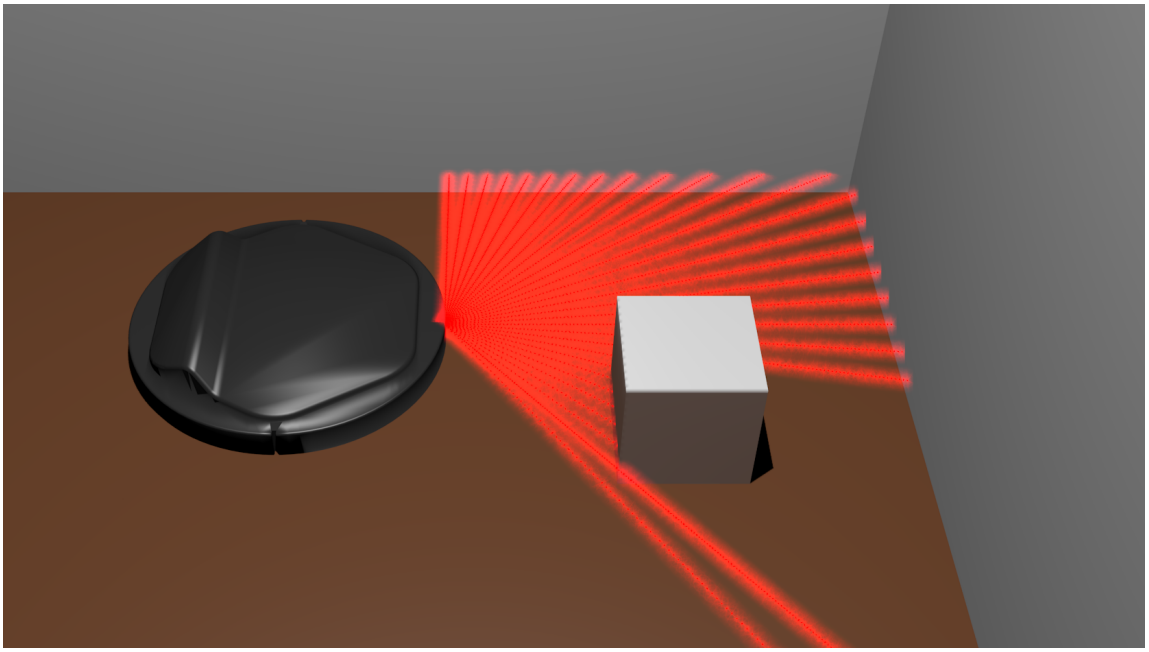


Figura 4.17: Modelo conceptual de funcionamiento de un *colisionador*. En este modelo se observa la forma de operar de un colisionador basado en rayos en un plano. El colisionador calcula sendos rayos desde el punto inicial (robot) hasta el primer punto del mapa donde choque; la distancia recorrida por cada rayo es la distancia teórica que debería devolver un sensor de posición para la posición origen.

Para que el componente funcione correctamente deben estar disponibles las implementaciones de estas tres interfaces, siendo indistinto si las implementa todas el mismo *plugin* o cada una uno distinto. Este esquema permitiría, por ejemplo, acoplar una implementación de modelo de observación distinta a la planeada para una partícula y utilizar ambos en una implementación de algoritmo que se ha diseñado posteriormente.

Comentando la implementación concreta para este proyecto, en las primeras fases de desarrollo se ha utilizado una versión inicial existente del algoritmo embebida en un componente y se ha portado al sistema de carga modular aquí descrito. Posteriormente se han ido depurando y añadiendo características para corregir su funcionamiento hasta alcanzar la implementación final. Entre las mejoras añadidas a esta base se encuentran los siguientes aspectos clave para que el algoritmo de localización funcione correctamente: mejora de la función de salud, descrita en la sección 4.2.2; sistema de dinámica entre razas basado en crédito, explicado en la sección 4.2.5; correcciones sobre el sistema de generación y extinción de exploradores y explotadores, desarrollado en las secciones 4.2.1 y 4.2.3; y capacidad de funcionamiento en espacio tridimensional completo.

La implementación final descrita en este proyecto define, además de las clases ya presentadas, una serie de clases auxiliares internas basadas en el modelo original de algoritmo evolutivo:

Exploradores almacena y gestiona las partículas que cumplen la función de explorador. Recibe del algoritmo el modelo de observación, la matriz de zonas restringidas y la información necesaria para inicializar y actualizar los exploradores. A su vez, se encarga de seleccionar, si existe, el explorador candidato a generar una nueva raza.

Raza almacena y gestiona las partículas con función de explotador para una raza. Recibe los mismos datos que la clase anterior, y devuelve cuando procede el explotador representante y la viabilidad de la misma.

Razas almacena y gestiona todas las razas activas en cada iteración del algoritmo. Se encarga de generar, actualizar y extinguir las razas cuando procede. Devuelve al algoritmo principal el explotador representante de la raza dominante en cada iteración.

5 Experimentos

En anteriores capítulos se han explicado las bases teóricas, el diseño y la implementación del algoritmo de autolocalización desarrollado en este proyecto. En este capítulo se explican los experimentos realizados con el mismo. Estos experimentos tienen el objetivo de validar la respuesta del algoritmo en distintos escenarios y han permitido depurarlo y ajustarlo durante el desarrollo de este proyecto.

Para realizar estos experimentos se han utilizado tres escenarios distintos: un escenario artificial consistente en un rectángulo con una esquina en chaflán distinta al resto (Figura 5.1a), parte del sótano de la biblioteca del campus de Fuenlabrada de la URJC (Figura 5.1b) y un área del Departamental II del campus de Móstoles de la URJC (Figura 5.1c). De estos escenarios se dispone tanto la localización verdadera como una definición de mundo para el simulador *Gazebo*, por lo que es posible medir la precisión con bastante exactitud.

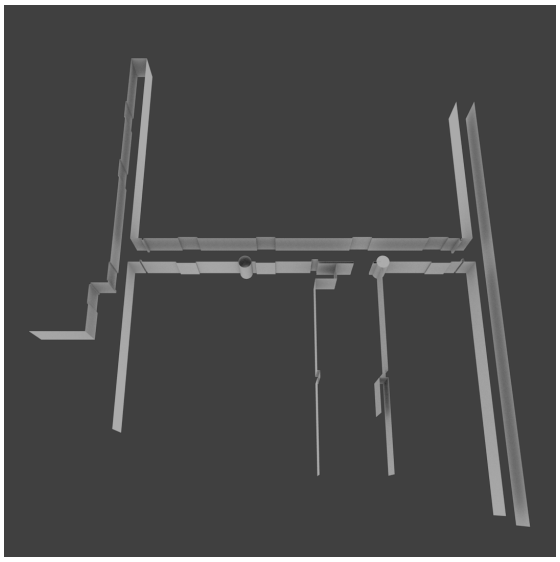
5.1. Validación de la función salud

El primer experimento realizado consiste en comprobar la validez de la función salud utilizado en los distintos escenarios planteados, variando las condiciones y puntos de validación. Debido a la vital importancia de que la salud sea correcta para el funcionamiento del resto del algoritmo, este experimento no sólo es el primero realizado, sino que se ha repetido para cada cambio en la depuración del algoritmo.

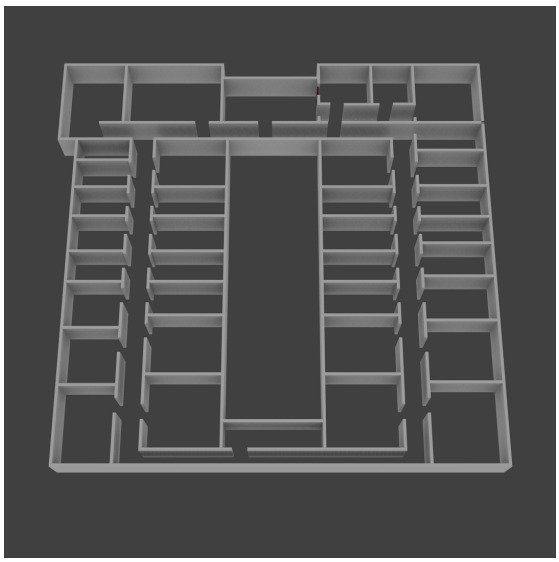
Para este experimento, se ha hecho uso de los modos de depuración de *mallado de partículas* y *esfera de orientaciones* que ofrece el componente desarrollado, tal y como se explicó en los apartados 4.3.3 y 4.3.4.

Tras efectuar este experimento en los tres escenarios en varias posiciones clave, se verifica que el modelo de observación devuelve valores máximos para las posiciones y orientaciones correctas y aquellas en las que el sensor de distancia devuelve exactamente los mismos datos. También se comprueba que se devuelven valores menores en las cercanías a dichas posiciones y mucho menores en puntos lejanos a esas zonas.

(a)



(b)



(c)

Figura 5.1: Escenarios utilizados para los experimentos.

Respecto a la interpretación de las gráficas, notar que las posiciones (x, y) y la orientación respecto al eje x θ están referidas al eje de coordenadas de cada mapa - visible en las figuras de *posición física* de cada experimento - donde el color *rojo* corresponde al eje x, el *verde* al eje y y el *azul* al eje z. Debajo de cada figura se especifica el modo utilizado y las posiciones y/o orientaciones que abarcan. Las gráficas de *mallado* representan la salud en el eje z y en gradiente de color frente a la posición; las de *orientación* representan el valor de la salud como el radio frente a la orientación θ en grados.

Se comentan a continuación los resultados más notables obtenidos en estos experimentos.

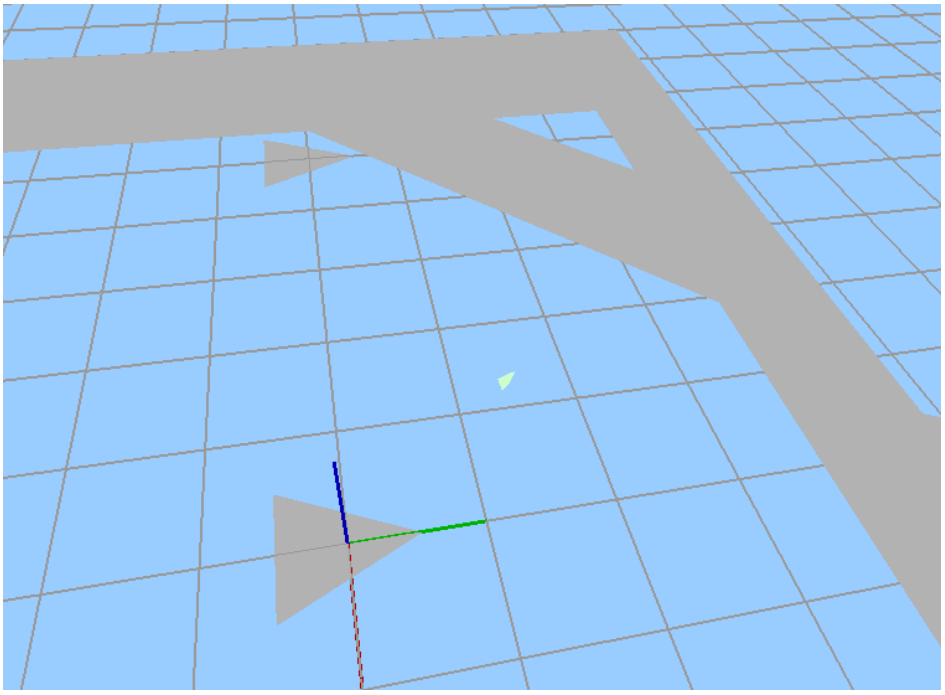
Escenario artificial

El primer escenario donde se ha realizado este experimento es el de la construcción artificial, por ser el más sencillo y más intuitivo ante lo que debería devolver la función salud ante cada caso. Las particularidades de este escenario son varias. Primero, que los robots utilizados no son capaces de detectar con el sensor de distancia las paredes desde la zona central (están lejos, fuera del alcance de los sensores); segundo, las paredes dan impresión de simetría hasta acercarse a las esquinas; y tercero, sólo hay una esquina totalmente diferente en todo el mapa.

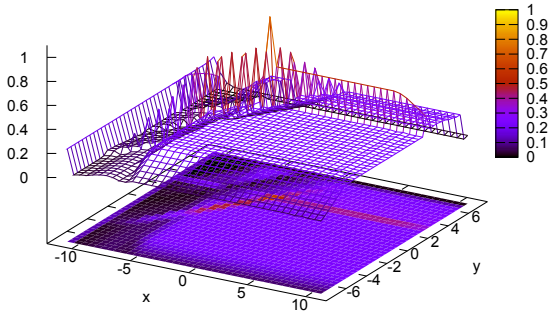
Un resultado experimental representativo es el que se puede observar en la figura 5.2. En este experimento se ha colocado el robot enfrentado a la esquina recortada del escenario (indicador verde de la figura 5.2a) y se ha procedido a obtener las medidas del modo de *mallado de partículas* con distintas orientaciones, de las que destacan la orientación correcta (figura 5.2b) y la de 0° (figura 5.2c). En el primer caso se obtiene totalmente lo esperado; valores bajos de salud para toda la malla excepto para la zona donde se encuentra el robot. En el segundo caso se observa que hay otras zonas del mapa que pueden dar valores mayores de lo normal cuando se comprueban otras orientaciones, no obstante entra dentro de lo aceptable.

Para comprobar hasta qué punto podría afectar el problema de la orientación, se comprueban los valores del modo de *esfera de orientaciones* para la posición del robot (figura 5.2d) y para una de las posiciones que han devuelto un valor interesante de salud (figura 5.2e). En la primera gráfica se observa que se obtiene un pico reconocible de valor cercano a 1 en la función salud sólo en la orientación adecuada; mientras que en la segunda se observa un aumento de valor más difuso que sólo alcanza valores de 0.6.

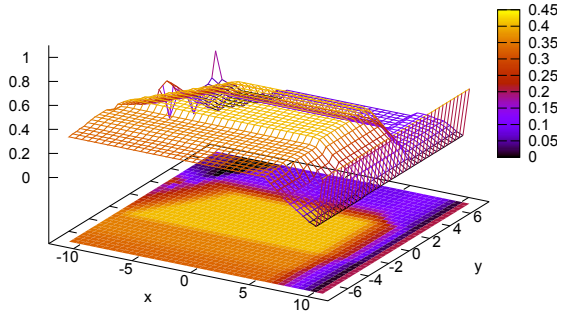
Estos resultados además de validar la función salud, arrojan indicaciones aproximadas de sobre qué valor habría que utilizar como límite para considerar una salud buena.



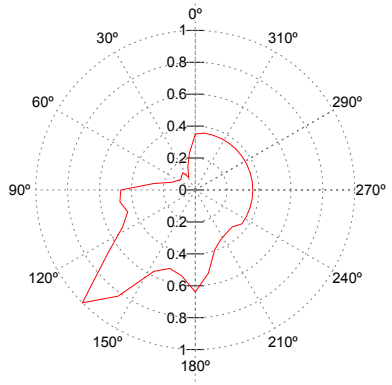
(a) Posición física



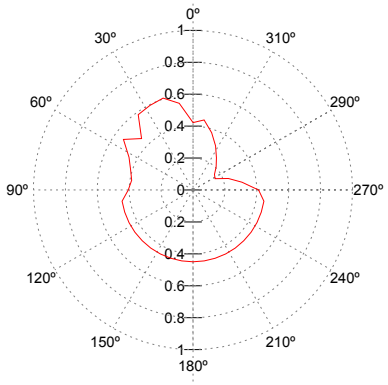
(b) Mallado: $x \in [-12, 12]; y \in [-8, 8]$; $\theta = 135°$



(c) Mallado: $x \in [-12, 12]; y \in [-8, 8]$; $\theta = 0°$



(d) Orientación: $\vec{p} = (-2.67, 3.08)$



(e) Orientación: $\vec{p} = (-5.20, -2.03)$

Figura 5.2: Validación de la función salud: Escenario Artificial.

Sótano de la Biblioteca

El segundo escenario elegido para realizar este experimento es el del sótano de la Biblioteca, al ser el segundo en complejidad y el más sencillo de los basados en localizaciones reales. Este escenario se centra en el laboratorio de robótica del sótano de la Biblioteca y contiene los tres pasillos más cercanos a esta sala. También asume que las puertas de las demás salas se encuentran cerradas, representándose en el mapa como si fueran una extensión de las paredes. También se establecen como regiones prohibidas cualquier área que se salga de estos pasillos y el laboratorio.

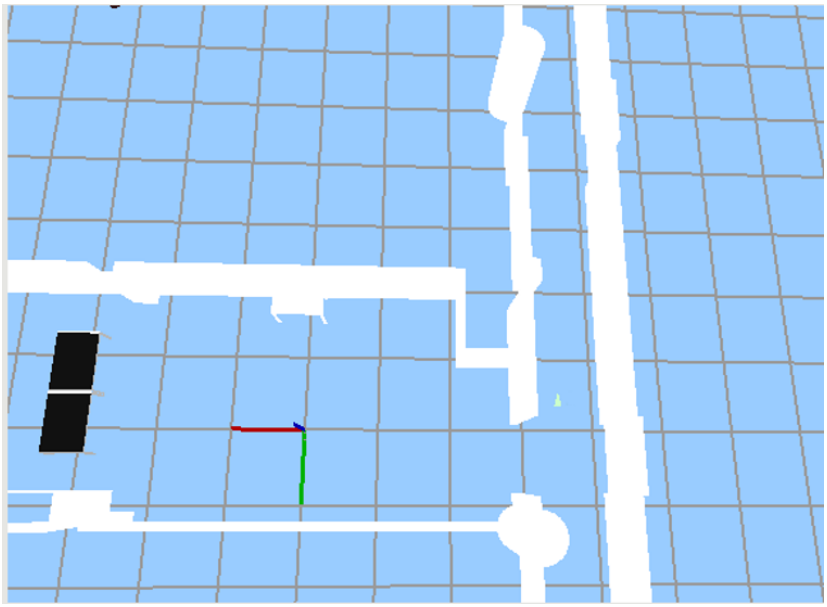
La principal característica de este escenario está directamente relacionada con el hecho de que hay tres pasillos similares, con sólo algunas diferencias cada cierta distancia. Esto provoca un efecto de simetría en el robot mayor que en el caso anterior.

Se ha elegido como resultados representativos los correspondientes a la figura 5.3. En este experimento, se ha colocado el robot a la salida del laboratorio mirando hacia uno de los extremos del pasillo (indicador verde de la figura 5.3a) y se ha procedido como en el caso anterior a obtener las medidas del modo de *mallado de partículas* con distintas orientaciones y el modo de *esfera de orientaciones* en distintos puntos. Se han escogido dos casos representativos.

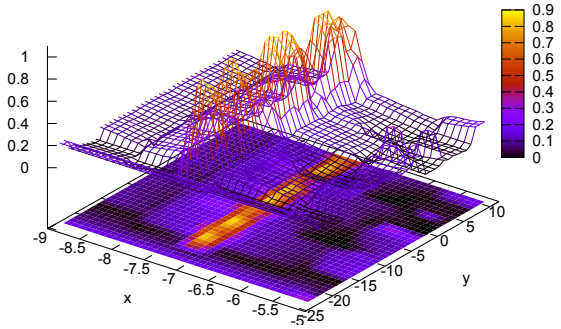
El primero, corresponde al pasillo donde se encuentra el robot y utilizando la orientación correcta (figura 5.3b). En este caso se observa que el pasillo al ser simétrico respecto al eje central provoca que la función salud tome el mismo valor alto en prácticamente toda la línea del pasillo que corresponde con la posición del robot, reflejando que todas esas posiciones podrían ser igual de probables. Las secciones donde se reduce este valor corresponden a zonas donde el pasillo presenta elementos diferenciadores como columnas y rebajes de puertas, que permitirían a la función determinar en qué punto de este pasillo se encuentra. El factor de simetría también se refleja en las medidas de orientación (figura 5.3d y figura 5.3e), tomadas en dos puntos del mismo pasillo donde se encuentra el robot. En las gráficas se observa que hay una respuesta en orientación similar para ambas correspondiente a las dos orientaciones plausibles (robot mirando hacia un lado u otro del pasillo), lo que confirma la respuesta simétrica esperada de la función salud.

El segundo caso corresponde a uno de los dos pasillos accesorios al que se encuentra el robot. En su gráfica de malla (figura 5.3c) se observa una respuesta importante de la función salud, debido al parecido entre ambos pasillos y la simetría que esto presenta en el mapa.

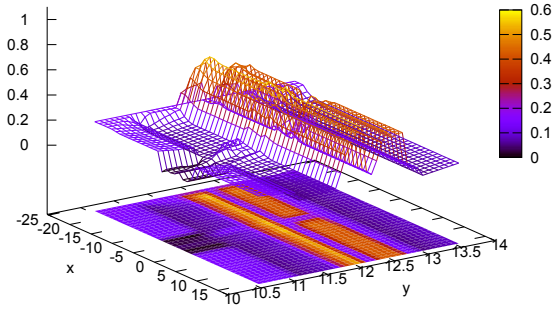
En base a estos resultados se observan la existencia de dos tipos de simetrías que hay que resolver: una *simetría local*, correspondiente a simetrías en zonas cercanas y visible en la respuesta de la función salud dentro del mismo pasillo; y una *simetría global*, correspondiente a zonas separadas del mismo escenario y visible en la comprobación contra el segundo pasillo. El problema de la simetría se resuelve en otros pasos del algoritmo y se expondrá junto al experimento correspondiente y sus resultados en la sección 5.7.



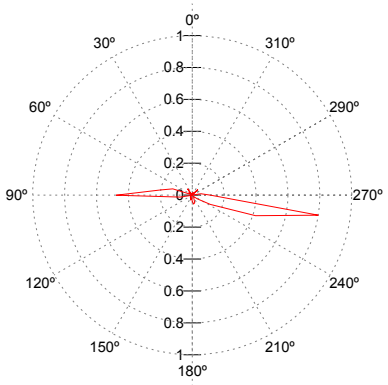
(a) Posición física



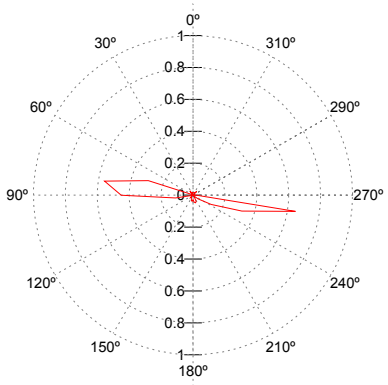
(b) Mallado: $x \in [-9, -5]; y \in [-25, 11]$; $\theta = 270°$



(c) Mallado: $x \in [-20, 20]; y \in [10.5, 13.5]$; $\theta = 180°$



(d) Orientación: $\vec{p} = (-6.95, -1.14)$



(e) Orientación: $\vec{p} = (-6.95, -17.45)$

Figura 5.3: Validación de la función salud: Sótano de la Biblioteca.

Departamental II

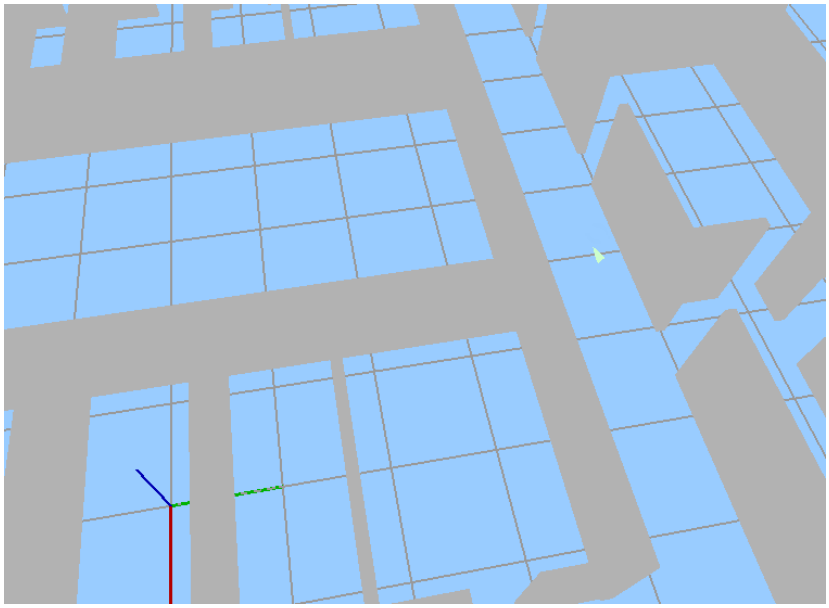
El último escenario utilizado para realizar este experimento es el del Departamental II por ser el que más complejidad presenta. Este escenario está restringido a una de las alas de la primera planta del edificio Departamental II del campus de Móstoles de la URJC. Dentro de este área se asume que todas las puertas están abiertas y las salas prácticamente vacías o con elementos que pueden ser interpretados como paredes por el robot. Cualquier área fuera de estas salas y los pasillos circundantes queda establecida como región prohibida.

Este escenario resulta el más complejo de navegar por el alto nivel de simetría que contiene y la densidad de los obstáculos que presenta. No sólo ofrece lo que se ha denominado *simetría global* en el escenario anterior, a través de los pasillos circundantes y las salas similares entre sí; sino que también tiene un alto nivel de *simetría local* en cuanto a que partes del propio pasillo pueden ser interpretadas como partes de salas circundantes en ciertas posiciones.

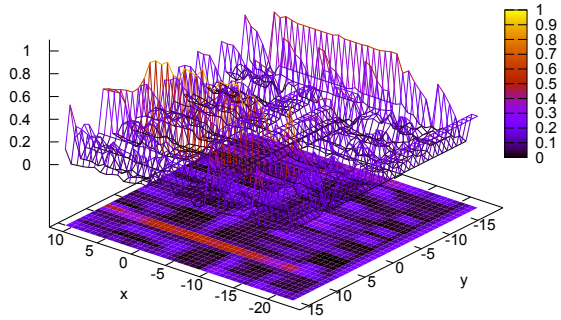
Se ha elegido como resultados representativos los correspondientes a la figura 5.4. En este experimento, se ha colocado el robot en el pasillo más al norte en relación al eje de coordenadas del mapa (indicador verde de la figura 5.4a) para tomar las medidas experimentales. Para ver el alto nivel de simetría, se han tomado como base para las medidas de *mallado* el propio pasillo donde se encuentra el robot (figura 5.4b) y el pasillo situado más hacia el oeste (figura 5.4c), ambos con parte de las salas cercanas.

En estos resultados se observa que, al igual que en el caso anterior, el efecto de simetría del pasillo ofrece valores altos de salud en todo el eje del pasillo donde se encuentra el robot, pero en cambio también aparecen réplicas en otros ejes paralelos. Estas réplicas se explican con dos motivos: las más cercanas y desplazadas corresponden a posiciones del pasillo en el que se puede observar la puerta de manera similar pero no exacta a la de la posición verdadera, dando estas valores relativamente bajos en comparación con el eje principal; la más alejada y de valor cercano al eje principal corresponde a un eje que pasa por fuera del pasillo y desde donde se observa la pared de forma similar a la de la posición verdadera. En el caso del segundo pasillo, se observa el efecto de simetría con el pasillo original del robot, acompañado de réplicas explicables de la misma forma que los de la gráfica anterior.

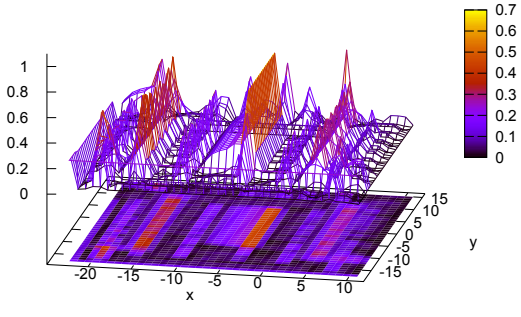
Para observar mejor el efecto de simetría entre ambos pasillos, se han tomado un punto representativo de cada uno para el análisis de *orientación*. El primer punto (figura 5.4d) se ha colocado en la posición del robot y refleja la naturaleza lineal de esta simetría, es decir, ésta sólo existe en la dirección del pasillo, no en otros ángulos. El segundo se ha colocado en el otro pasillo cerca de uno de los ejes que respondían con valores altos de salud en el análisis anterior (figura 5.4e). Se observa que al no haberse colocado exactamente en uno de los ejes observados en el análisis anterior, ninguna de las orientaciones alcanza valores altos de salud, aunque sigue distribuyéndose en la dirección del pasillo de referencia. Estos resultados no sólo confirman la buena respuesta de la función salud, sino que también sostienen las observaciones de los casos anteriores.



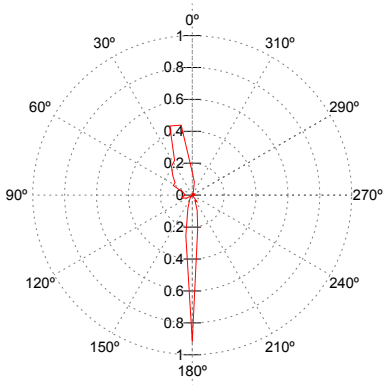
(a) Posición física



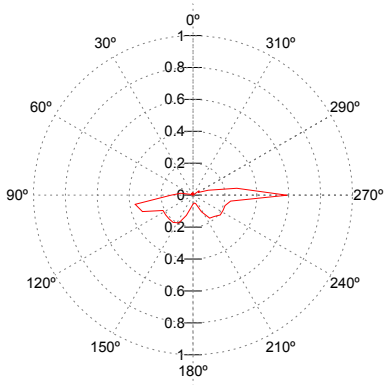
(b) Mallado: $x \in [-24, 11]; y \in [-16, 16]$; $\theta = 180°$



(c) Mallado: $x \in [-22, 12]; y \in [-17, 15]$; $\theta = 270°$



(d) Orientación: $\vec{p} = (-3.28, 8.35)$



(e) Orientación: $\vec{p} = (-3.72, -2.63)$

Figura 5.4: Validación de la función salud: Departamental II.

5.2. Precisión sin ruido sensorial

La finalidad de este experimento es comprobar la precisión espacial de la respuesta del algoritmo durante el recorrido de un circuito sin ningún incidente y el tiempo necesario para alcanzar la precisión máxima. Al realizar el experimento en distintos escenarios, se puede observar también en qué condiciones de una ejecución habitual se pierde precisión o una respuesta válida y cuanto se tarda en recuperar de nuevo.

Debido a que las respuestas de precisión con ruido sensorial (explicadas en el apartado 5.3) dieron resultados prácticamente similares a los experimentos de esta sección, se ha decidido comentar sólo las conclusiones en este apartado, expresando de forma más exhaustiva los detalles experimentales en el otro.

Algunas de las conclusiones extraídas durante esta sesión de experimentos fueron:

- La precisión a la hora una posición y orientación válidas, viene determinada por la cantidad de elementos diferenciadores en el escenario y la capacidad de los sensores de expresar esa diversidad.
- Generalmente, al algoritmo le resulta más sencillo ser preciso en posición más que en orientación. Se observa que este hecho está relacionado directamente con los detalles de los obstáculos observados, ya que con menor cantidad de detalles, la función de salud tiende a dar también valores altos en orientaciones más alejadas de la original.
- Un escenario con alto nivel de *simetrías* (se desarrollará este concepto secciones posteriores) tiende a dificultar la convergencia del algoritmo de autolocalización, llegando en casos extremos a devolver una localización completamente errónea por simetría.
- En condiciones y escenarios típicos el algoritmo devuelve una posición y orientación bastante fieles a las verdaderas y estables en tiempo; se ha tenido de media errores aproximados de 0.15 m y 0.4°.

5.3. Precisión con ruido sensorial

Este experimento añade el factor del ruido al problema de autolocalización. La diferencia entre el experimento anterior y éste radica en que, en este caso, se considera el ruido de los sensores del robot. Para este experimento, de cara a las simulaciones se han implementado unos *drivers* para simulación que introducen ruido tanto a la distancia recorrida medida como al giro, lo menos correlado posible entre sí.

Las representaciones tridimensionales correspondientes a este experimento muestran los siguientes elementos:

- La posición de *verdad absoluta* como una flecha orientada de color verde oscuro; el recorrido correspondiente será una línea del mismo color.
- La posición dada por el *sensor de posición* como una flecha orientada de color verde claro; el recorrido correspondiente será una línea del mismo color.
- La posición dada por el *algoritmo de localización* como una flecha orientada de color rojo; el recorrido correspondiente será una línea del mismo color.

Las gráficas mostradas representarán los siguientes datos:

- El eje de abscisas indica el *tiempo transcurrido* en segundos desde que se inició el algoritmo.
- El eje de ordenadas superior indica el *error de distancia* en metros.
- El eje de ordenadas inferior indica el *error de orientación* en grados.
- La línea punteada azul representa el *error cometido por el sensor de posición* respecto a la verdad absoluta. Será la referencia comparativa.
- La línea roja representa el *error cometido por el algoritmo respecto del sensor.*
- La línea verde representa el *error cometido por el algoritmo* respecto de la verdad absoluta. Será el valor de error de referencia.

Escenario artificial

Como en el caso anterior, se ha decidido comenzar por el escenario más simple por dar la intuición de que al algoritmo le resultará más sencilla la tarea de autolocalización. Para simplificar también el recorrido el robot se dirigirá a una de las paredes, la seguirá hasta encontrarse dos esquinas y viajará a la esquina diferenciada para repetir la operación con la pared restante y volver al centro.

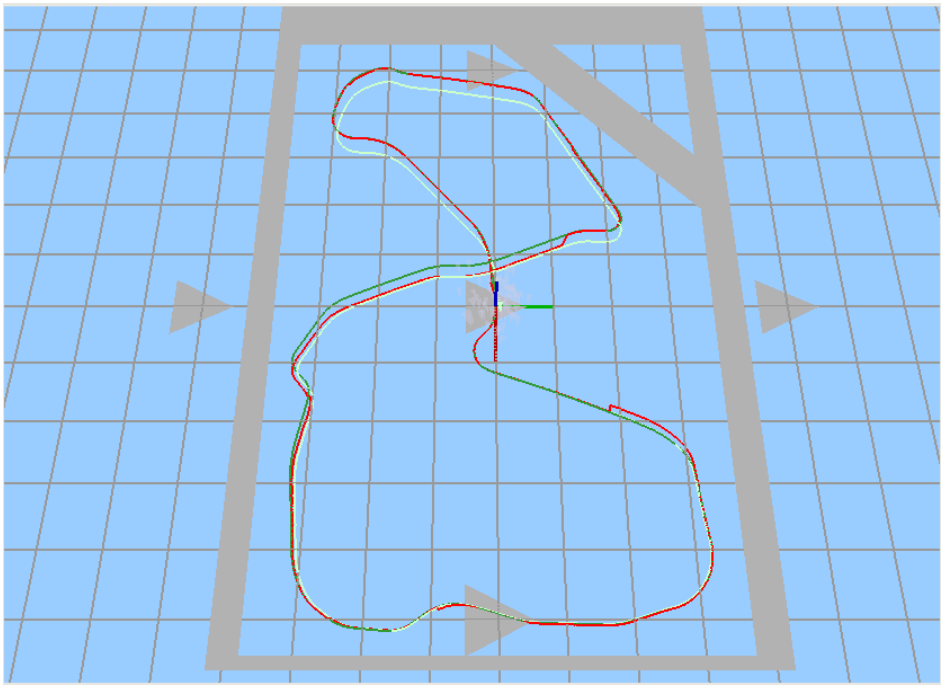
Para comentar este experimento se ha escogido una realización que ha resultado en el recorrido que aparece en la figura 5.5a.

Analizando primero el recorrido efectuado, primero se observa que todas las líneas se solapan hasta que el sensor de distancia localiza la pared. Esto se debe a que el algoritmo carece de otras referencias distintas al encoder, por lo que lo más que puede hacer es seguir los movimientos que éste le indica. A continuación, la línea del *algoritmo* se separa debido a que la simetría de la pared produce varias posibles respuestas válidas y ha escogido una que, aunque cercana, no es la correcta. Más tarde se observa que conforme va avanzando el robot el algoritmo es capaz de refinar la posición, hecho visible sobre todo cuando se alcanzan visuales con las esquinas. En la vuelta hacia la esquina diferenciada se observa que al perder referencias el algoritmo de nuevo tiende a seguir al encoder. Este hecho es visible sobre todo en el momento en el que el error del encoder es tal que la línea de *verdad absoluta* difiere de las otras dos. No obstante, al alcanzar la visual con la esquina, el algoritmo es capaz de volver a corregir la posición hasta el final del recorrido.

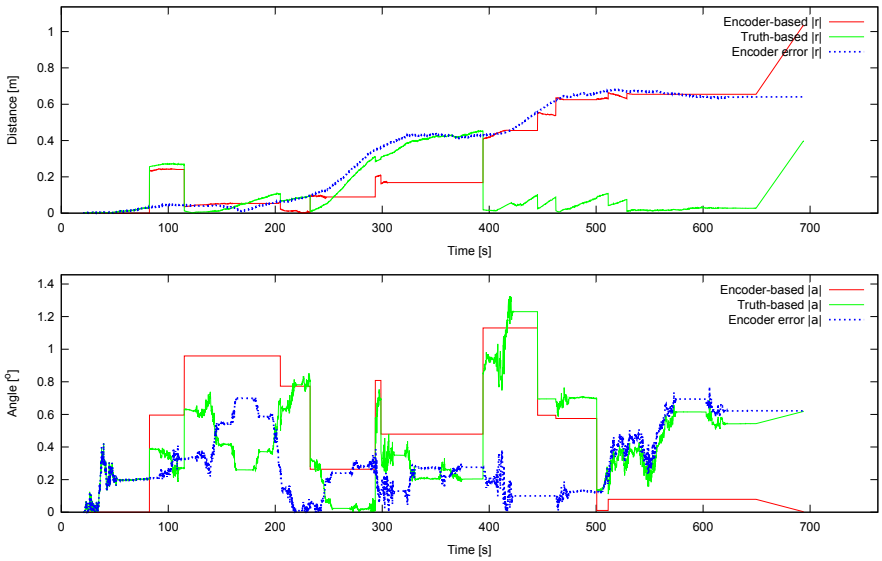
Analizando la gráfica del *error de distancia* (figura 5.5b), se observa que a pesar del primer error cerca del instante 100 s, donde se ha superado el error cometido por el propio sensor, la mayor parte del tiempo el algoritmo responde bastante mejor que el sensor de odometría. Este hecho también se refleja en que el error máximo cometido en este experimento no sobrepasa los 0.5 m.

En el caso de la gráfica del *error de orientación* (figura 5.5b) se observa que al algoritmo le cuesta determinar la orientación dentro del margen de 1°. Esto se debe principalmente a la falta de referencias y a la simetría local de las paredes, dicho de otro modo, al encajar el modelo de observación de una pared desde determinada distancia con distintos ángulos de inclinación, una pared no resulta una referencia suficiente para determinar la orientación concreta del robot.

De los resultados obtenidos se extrae la conclusión de que un escenario con muy pocas referencias en un área obligará al algoritmo a utilizar prácticamente sólo los sensores de abordo del robot como guía.



(a) Recorrido efectuado en la realización



(b) Gráfica de resultados

Figura 5.5: Experimento de precisión con ruido sensorial: Escenario artificial

Sótano de la Biblioteca

Siguiendo el orden de escenarios, el siguiente a considerar es el del sótano de la Biblioteca. En vista de los resultados del escenario anterior, a primera vista el nivel de complejidad que presenta este juega a favor del algoritmo de autolocalización. En este caso, el recorrido consistirá en salir del laboratorio de robótica, recorrer el pasillo hasta el final y girar en el pasillo de la izquierda.

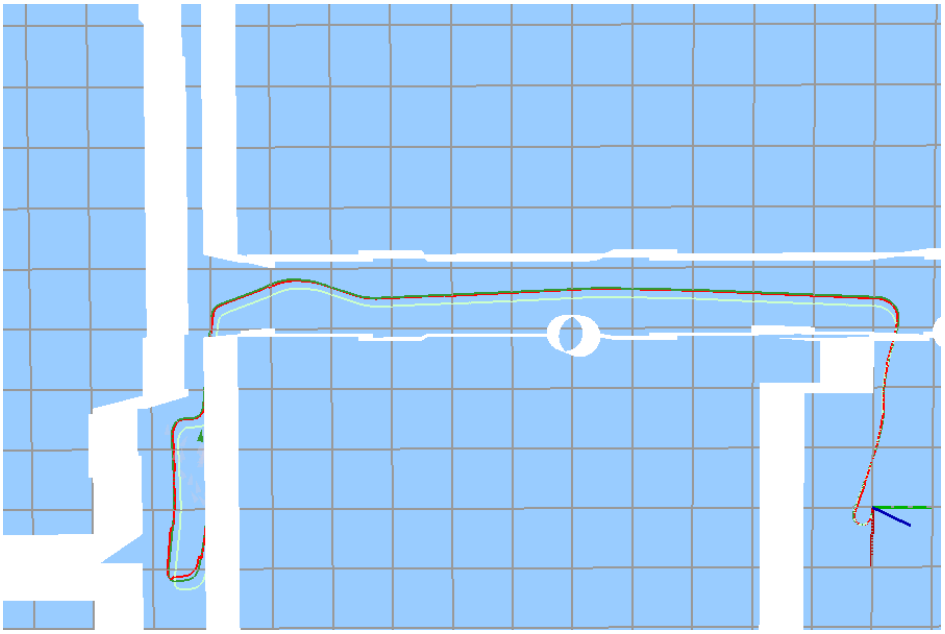
Para comentar este experimento se ha escogido una realización que ha resultado en el recorrido que aparece en la figura 5.6a.

Sobre el recorrido efectuado, no se observa una separación notable entre ninguno de los indicadores hasta que el del sensor de posición se separa de la detección algorítmica y la verdad absoluta. Esta observación confirma la idea de que con suficientes elementos de referencia, el algoritmo comete menor error en la posición.

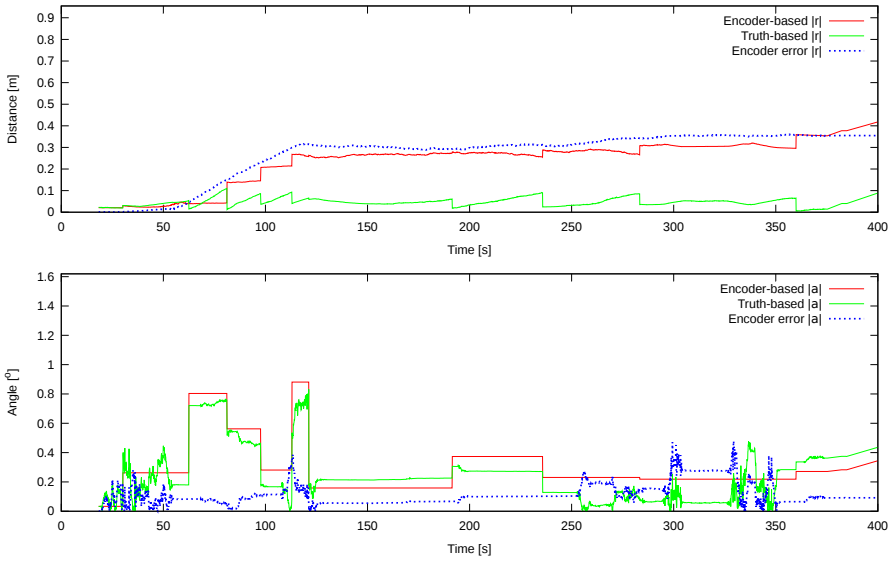
Analizando la gráfica del *error de distancia* (figura 5.6b) se observa un error inferior a los 0.15 m durante toda la prueba, siendo este el resultado más estable de los observados hasta el momento.

En el caso de la gráfica del *error de orientación* (figura 5.5b) se observa que, a pesar de que la estructura del entorno es bastante rica en referencias para la autolocalización, la falta de elementos diferenciadores en las paredes provoca que al algoritmo le siga costando determinar la orientación; no obstante el margen de error se ha reducido a $0.4°$.

Como conclusión de este experimento, se han determinado valores de error medio para la detección del algoritmo y se comprueba que para que la orientación estimada sea más exacta hacen falta referencias fácilmente diferenciables según el ángulo de observación.



(a) Recorrido efectuado en la realización



(b) Gráfica de resultados

Figura 5.6: Experimento de precisión con ruido sensorial: Sótano de la Biblioteca

Departamental II

En este experimento, el escenario del Departamental II además de ofrecer las complejidades que se han comentado en experimentos anteriores, añade el hecho de que el ruido de los sensores de odometría podría inducir al algoritmo a decidir que el robot se encuentra en una sala distinta a la correcta o el pasillo. La ruta principal para este escenario será salir de una de las salas y dar la vuelta completa al escenario.

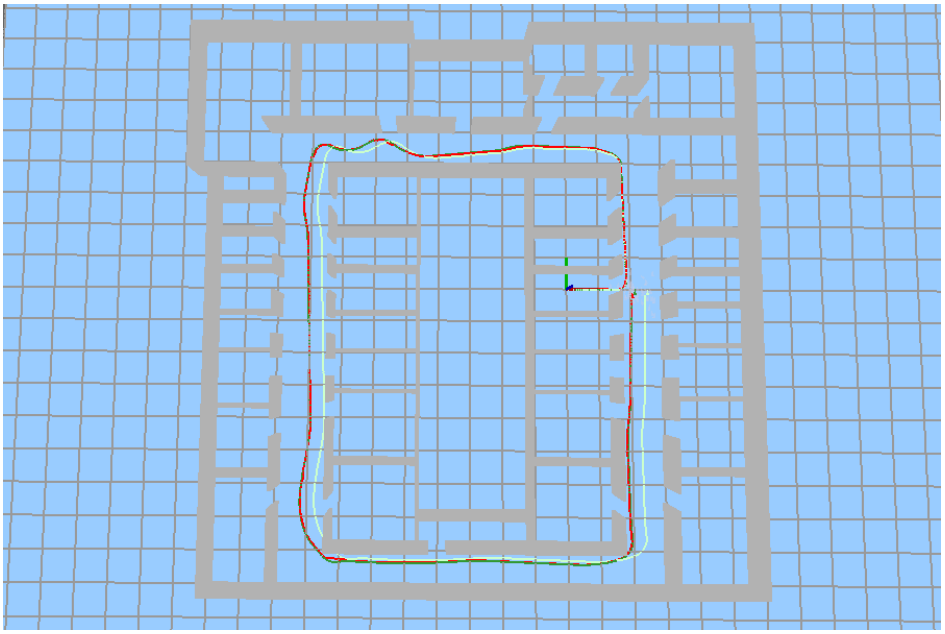
Para comentar este experimento se ha escogido una realización que ha resultado en el recorrido que aparece en la figura 5.7a.

En esta realización se observa que, al contrario de las expectativas basadas en la complejidad conocida de este escenario, apenas ha cometido errores importantes durante el recorrido. Analizando primero el recorrido efectuado, se observa que durante prácticamente todo el trayecto, las líneas del *algoritmo* y la de *verdad absoluta* se solapan o están muy cerca una de la otra; en cambio, el *sensor de posición* se desvía conforme pasa el tiempo. Este resultado apoya la validez del funcionamiento del algoritmo en un entorno relativamente complejo.

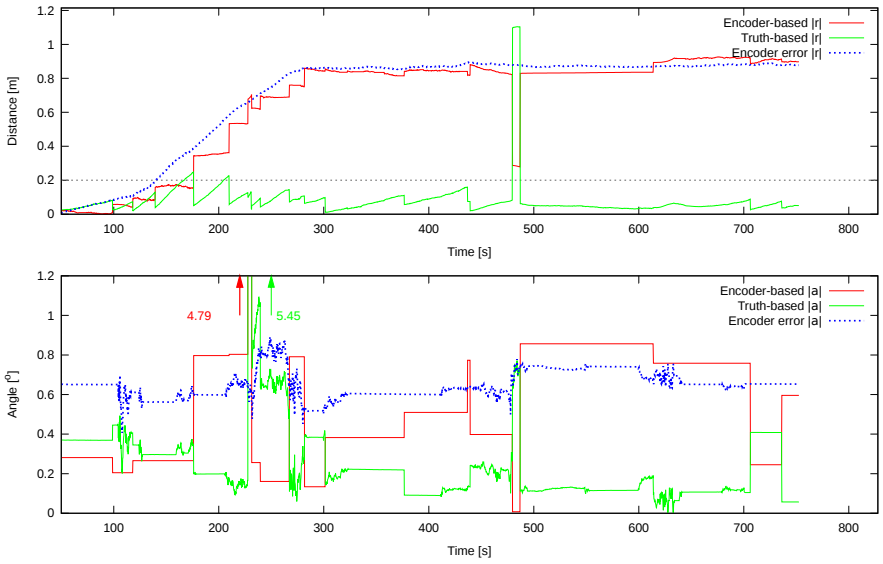
Analizando la gráfica del *error de distancia* (figura 5.7b), se observa también que el error cometido por el sensor de posición aumenta de forma progresiva hasta alcanzar prácticamente el metro de error. Por otro lado, el algoritmo ha ido oscilando en error entre prácticamente nulo y unos 0.2 m exceptuando en el intervalo temporal cercano a 490 s, donde se ha superado el error cometido por el propio sensor. Dado que este tipo de error no se ha producido ante las mismas condiciones en repeticiones distintas a la presentada, es seguro asumir que se trata de un fallo espúreo en la localización producido por el carácter aleatorio de ciertas fases del algoritmo.

En el caso de la gráfica del *error de orientación* (figura 5.7b) se observa que el error del *algoritmo de localización* presenta oscilaciones entre 0.1° y 0.45°, frente a los 0.65° y 0.8° del *sensor de posición*. En este caso se ve un pico que alcanza los 5.45° pasados los 210 s, lo que no coincide en tiempo con el valor pico del error de distancia. Esto refuerza la idea de que tanto este como aquel son fallos espúreos del algoritmo.

En vista de los resultados obtenidos, podemos afirmar que el algoritmo responde con suficiente precisión en presencia de ruido sensorial en entornos de relativa complejidad.



(a) Recorrido efectuado en la realización



(b) Gráfica de resultados

Figura 5.7: Experimento de precisión con ruido sensorial: Departamental II

5.4. Coste temporal

De cara a comprobar que el algoritmo cumple el requisito de *tiempo real*, es necesario observar el tiempo mínimo, máximo y medio por iteración que requiere el algoritmo para devolver una respuesta.

Este experimento consiste en evaluar el tiempo transcurrido desde que inicia una iteración de actualización del algoritmo hasta que dicha iteración termina, ignorando posibles instrucciones posteriores dedicadas a cuestiones ajenas al algoritmo. Para comprobar la dependencia de los resultados respecto al mapa se evalúan el mejor coste, el peor y el caso medio para los tres escenarios mostrados en la Figura 5.1. También se comprueba la dependencia al estado de movimiento del robot, dejándolo en reposo en una sala u obligándole a hacer un circuito mientras se realizan las mediciones.

El hardwade utilizado para realizar este experimento ha sido un PC de sobremesa Intel Core i7-3770 @ 3.40 GHz, 16GB de RAM y sistema operativo Ubuntu 13.10 x64.

En una ejecución típica del componente y tomando de referencia como mínimo 3000 iteraciones para cada caso, se han alcanzado los siguientes valores:

	Entorno artificial		**Sótano de Biblioteca**		**Departamental II**	
	Reposo (ms)	Movimiento (ms)	Reposo (ms)	Movimiento (ms)	Reposo (ms)	Movimiento (ms)
Mínimo	26.847	24.633	32.740	32.691	39.209	39.212
Máximo	59.576	82.244	61.451	98.882	97.341	99.701
Medio	35.832	42.964	40.171	52.666	59.049	62.529

En base a los resultados del experimento se observa que el coste temporal del algoritmo está directamente relacionado con la complejidad del escenario donde debe localizarse el robot y con el movimiento del mismo.

En el caso de la complejidad se debe a que el tiempo dedicado al cálculo de colisiones para el *modelo de observación* está relacionado con la cantidad de superficies a comprobar para cada posición.Se nota el contraste entre la simplicidad del *Entorno artificial*, donde sólo están las paredes perimetrales como obstáculos y el área restringida es el exterior a ese perímetro, frente a la complejidad del *Departamental II*, en la que al robot se le permite encontrarse en cualquier sala y existe una gran simetría tanto global (pasillos) como local (salas).

El caso del movimiento se explica con que una vez encontrada una posición lo suficientemente buena y sin variaciones, el algoritmo no necesita dedicar tantos recursos a la exploración y explotación de nuevas soluciones.

No obstante, para el caso medio, el coste se encentra en el rango entre 35.83 ms y 59.05 ms para el ordenador más potente, lo que entra en el rango de tiempo real respecto a los dispositivos tratados en el proyecto.

5.5. Robustez ante mala inicialización

El algoritmo parte de la información del sensor de posición para comenzar a trabajar y la utiliza en los casos en los que no observa un resultado lo suficientemente bueno; esto sugiere que una mala inicialización de la posición del algoritmo afecta de alguna forma a su funcionamiento.

Para ilustrar los resultados de este experimento y explicar su desarrollo, se ha escogido una realización en el escenario del *Departamental II* en la que se obliga al algoritmo de autolocalización a inicializar su posición y orientación en una sala distinta de la verdadera localización del robot. Para dificultar más la situación se ha escogido la sala para que, sin información adicional, el algoritmo obtenga buenos valores de salud en dicha sala. Esta situación provoca que en la primera parte del experimento donde no se mueve el robot, el algoritmo devuelva como posición estimada la de inicialización (figura 5.8).

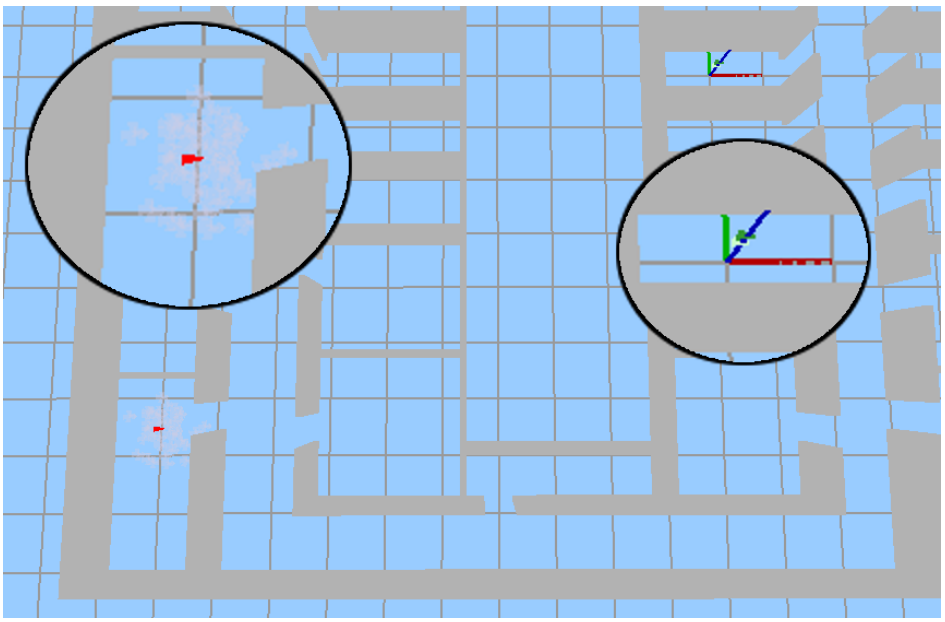


Figura 5.8: Robustez ante mala inicialización: paso de inicialización. Se observa la posición real del robot (derecha) y la inicialización de los valores del algoritmo y sus exploradores en otra sala (izquierda).

Eventualmente, algún explorador alcanza la posición del robot original y genera otra raza, esta vez en la posición correcta. Mientras se mueve el robot y pasa el tiempo, la raza inicial comienza a tener cada vez menos salud que la que sigue a la posición correcta que, en cada iteración, va ganando mayor salud. Este proceso se mantiene hasta que, siguiendo los criterios del sistema de *dominancia de razas* (expuesto en la sección 4.2.6) la que sigue la posición verdadera se convierte en la estimación devuelta por el algoritmo de autolocalización (figura 5.9).

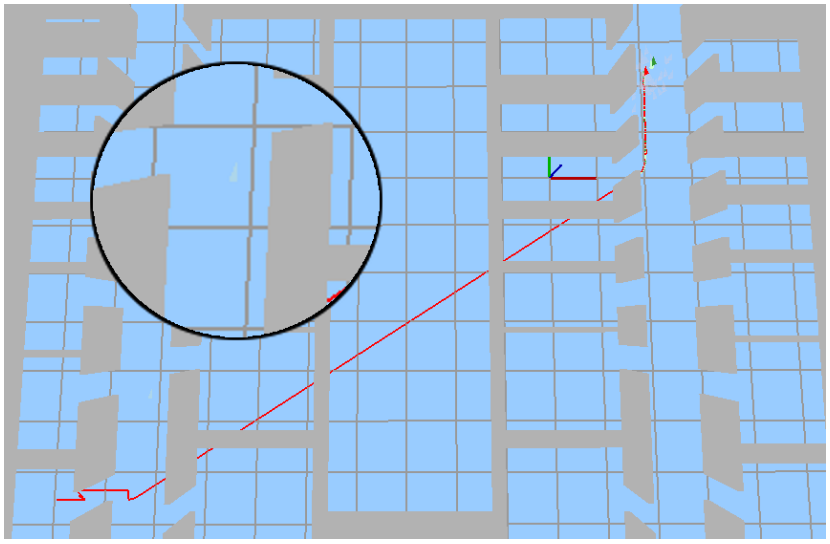


Figura 5.9: Robustez ante mala inicialización: recuperación de posición verdadera. Se observa la secuencia de posiciones estimadas por el algoritmo (línea roja) hasta alcanzar la posición real del robot. En el lado izquierdo y ampliado, el indicador de la raza inicial que ya no es dominante.

Por último, tras transcurrir un tiempo en el que la raza inicial va perdiendo cada vez más salud y crédito por encontrarse en una zona poco prometedora, el algoritmo extingue la raza por falta de explotadores viables (figura 5.10), quedando sólo la que sigue a la posición verdadera.

Los resultados de este experimento tras varias realizaciones indican que el algoritmo es bastante robusto ante fallos de inicialización en condiciones típicas y con entornos con elementos diferenciables. En situaciones de mayor simetría o falta de complejidad del entorno, el algoritmo de autolocalización tiende a necesitar más tiempo e información sensorial del robot para recuperar la posición correcta.

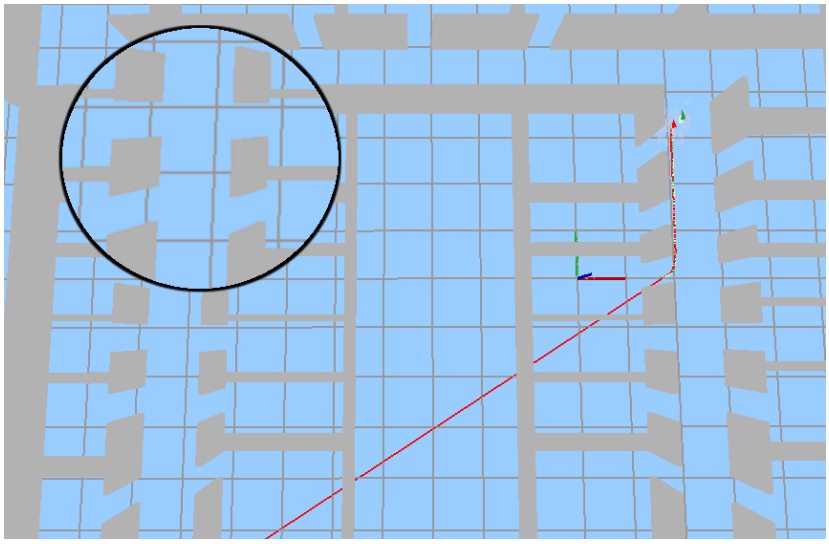


Figura 5.10: Robustez ante mala inicialización: extinción de raza inviable. Se observa la secuencia de posiciones estimadas por el algoritmo (línea roja) y los movimientos del robot (línea verde). En el lado izquierdo y ampliado, se muestra la zona donde debería encontrarse la primera raza, ahora extinta y por tanto inexistente.

5.6. Robustez ante el secuestro

Debido al carácter iterativo del algoritmo, este se apoya en los resultados obtenidos en instantes anteriores para ofrecer los próximos con mayor precisión. Esto presenta la cuestión de cómo reacciona el algoritmo ante el desplazamiento del robot por motivos desconocidos y sin información del sensor de posición, o si el propio bucle principal interrumpe temporalmente su ejecución (pausa del componente, fallo de comunicación, procesador saturado durante unos instantes de tiempo podrían ser posibles causas). Estas situaciones se conocen como *secuestro* del robot.

Para comprobar la robustez del algoritmo de autolocalización ante un secuestro, se han realizado dos tipos de experimentos: en el primero, se ha simulado la interrupción del componente mediante una pausa de tiempo prolongado y su posterior reanudación; en el segundo, se ha desplazado al robot a otra posición totalmente distinta asegurando que los sensores del mismo no detectaran ningún cambio.

A modo ilustrativo, se comenta una de las realizaciones en el escenario del *Departamental II*. Como primera fase se ha desplazado el robot durante un tiempo por el entorno como en experimentos anteriores. A continuación, se ha pausado el componente y se ha seguido moviendo el robot hasta alcanzar una posición relativamente alejada de la original (figura 5.11) y se ha reanudado la operación del algoritmo.

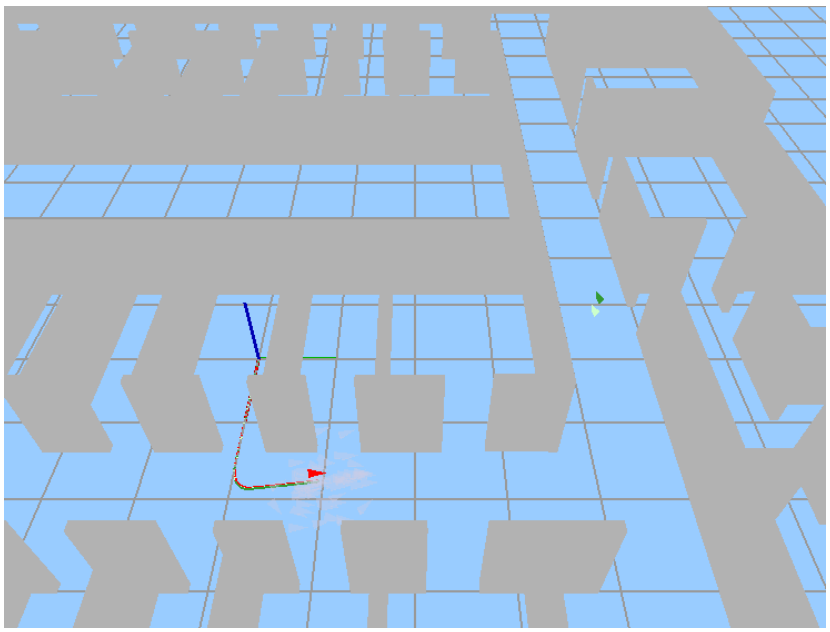


Figura 5.11: Robustez ante secuestro: fase inicial de simulación de interrupción. Se observa la posición estimada del robot (rojo) fija y la posición verdadera en cierto instante de tiempo (verde).

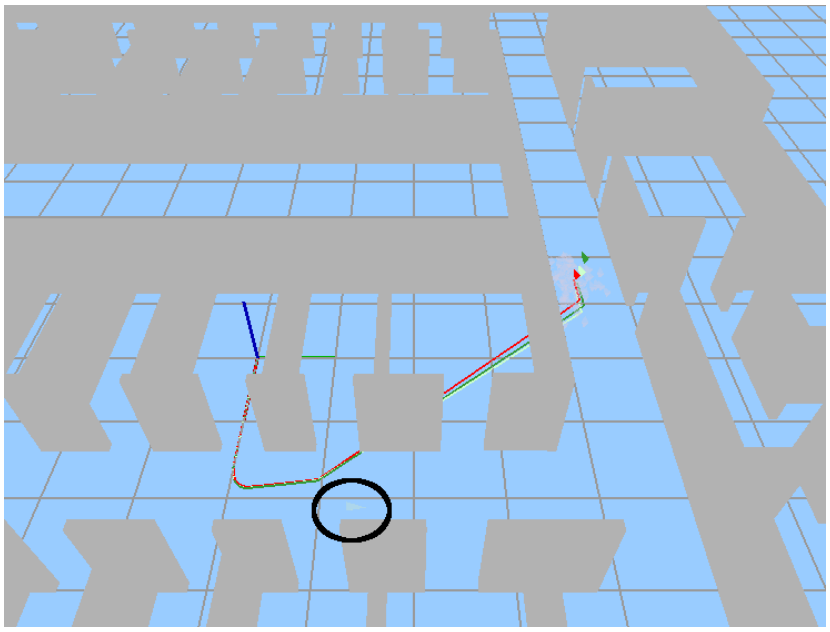


Figura 5.12: Robustez ante secuestro: relocalización de posición. Se observa el salto al relocalizar la posición verdadera tras reanudar el algoritmo. Marcado en negro, se observa aún la raza inicial que desaparecerá en posteriores iteraciones.

Tras su reanudación, el algoritmo ha reaccionado de forma similar a como lo hacía en los experimentos de *robustez ante mala inicialización* (sección 5.5), recuperando la posición correcta tras cierto intervalo de tiempo (figura 5.12). Esto ofrece la idea de que el caso de *mala inicialización* puede verse como un caso particular de secuestro.

Para confirmar esta idea y que la respuesta del algoritmo de autolocalización sigue siendo la esperada, se procede a realizar el experimento de desplazar al robot a una posición totalmente distinta y comprobar si el algoritmo es capaz de localizar la nueva posición (figura 5.13). Tras dejar que el robot se moviese con normalidad por el entorno en su nueva posición, el algoritmo ha reaccionado en todos los casos de forma similar al caso anterior y al de los experimentos de *robustez ante mala inicialización* (figura 5.14).

De estos experimentos se concluye que, el algoritmo de autolocalización trata de forma similar los casos de *secuestro* y *mala inicialización* y que, al igual que con los primeros, el algoritmo resulta ser bastante robusto ante este tipo de problemas.

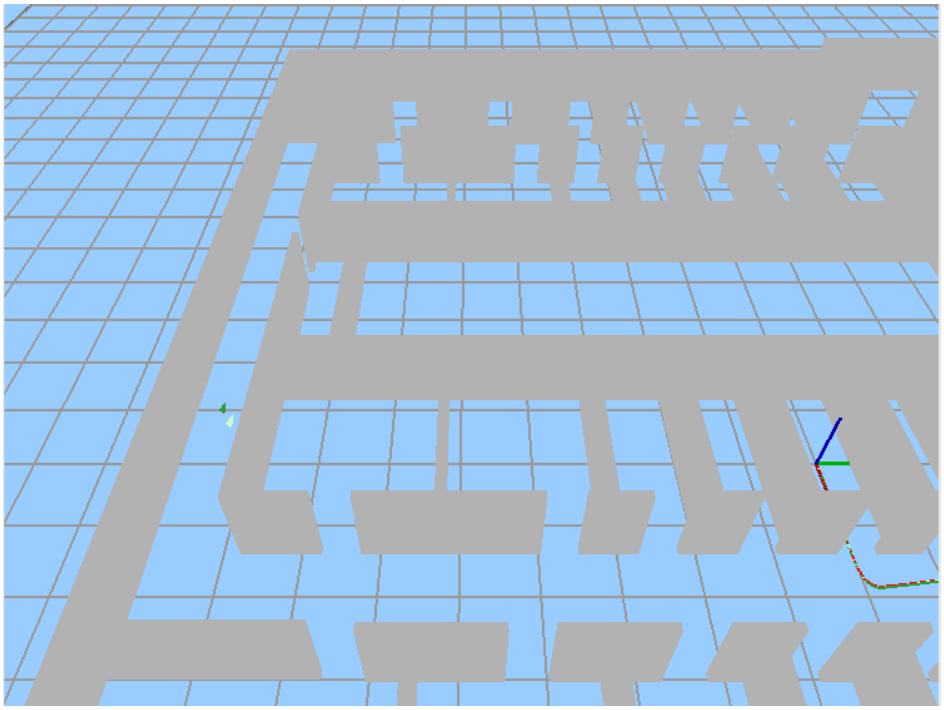


Figura 5.13: Robustez ante secuestro: desplazamiento de robot a nueva posición. Se muestra en verde la nueva posición del robot respecto al caso anterior.

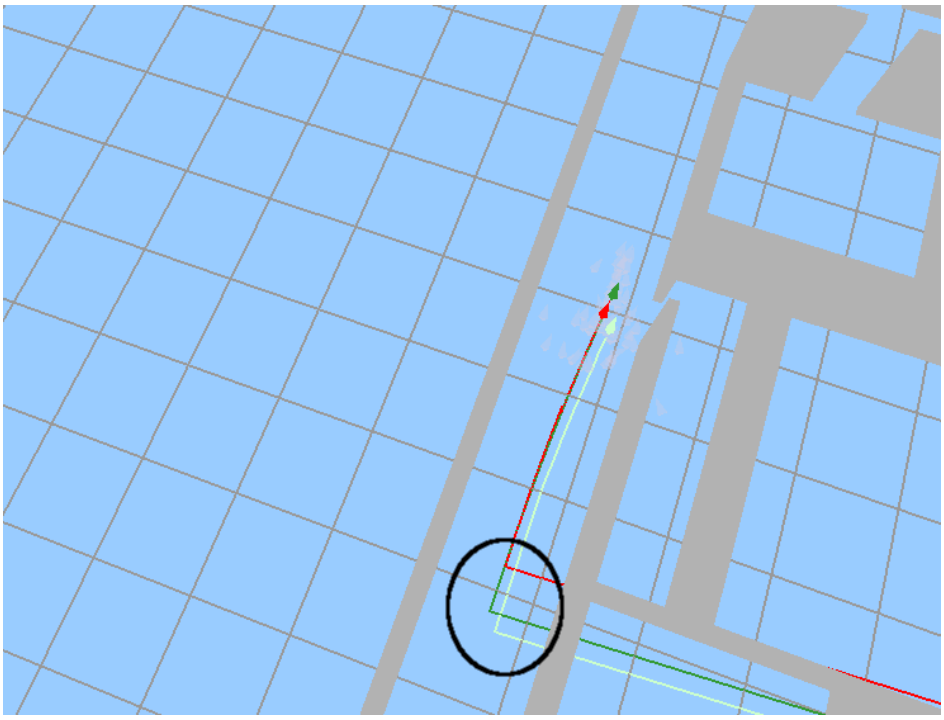


Figura 5.14: Robustez ante secuestro: relocalización tras desplazamiento. Se muestran los saltos en los recorridos producto del desconocimiento del *secuestro* por parte del robot y algoritmo. Marcado en círculo se destaca el recorrido efectuado por el robot antes de que el algoritmo emitiese como posición estimada de nuevo la cercana a la verdadera.

5.7. Robustez ante simetrías

A lo largo de estos experimentos se ha observado la existencia un problema recurrente a la hora de efectuar la autolocalización: la *simetría* en el escenario.

Una región presenta cierta simetría con otra si desde el punto de vista de los sensores de un robot existe un cierto grado de parecido, de forma que a mayor simetría entre dos regiones más difícil resultará diferenciar una de otra. Además, tal y como se introdujo en los experimentos anteriores, las simetrías pueden dividirse según la distancia entre las regiones afectadas en *locales* y *globales*; de esta forma, serán *simetrías locales* las producidas dentro de un mismo pasillo o en salas contiguas y, en cambio, *simetrías globales* las producidas entre dos salas alejadas en el escenario considerado.

Para hablar de la robustez ante este tipo de problema, esta sección se basará en el experimento y escenario que presentan el caso más crítico: el de *precisión con ruido* (sección 5.3) en el escenario del *Departamental II* debido a la cantidad de simetrías a distinto nivel que presenta.

Durante la validación de la función salud en este escenario (ver sección 5.1) se observó la existencia de *simetrías globales* importantes entre los cuatro pasillos principales y *simetrías locales* dentro de cada pasillo; también existen simetrías entre todas las salas a distinto nivel y parte de las salas con los pasillos.

Revisando el experimento de *precisión con ruido* del *Departamental II*, se observa que, a pesar de todo lo comentado, no solo las posiciones obtenidas son bastante cercanas a la verdadera sino que también hay elementos causantes de estas simetrías que juegan a favor de afinar más la localización y orientación; no obstante en algunas realizaciones del experimento se han producido errores espúreos (fallos esporádicos de localización durante intervalos cortos de tiempo) relacionados con el problema de la simetría.

A la vista de los resultados experimentales, se puede concluir que el algoritmo de autolocalización desarrollado es bastante robusto ante problemas de simetría.

6 Conclusiones y trabajos futuros

En estos capítulos se han expuesto los problemas tratados en este proyecto fin de carrera, las soluciones que se han alcanzado y los experimentos que han servido para validar dichas soluciones. En éste se expondrán las conclusiones finales que se pueden extraer de todo lo expuesto y algunas líneas de trabajo que surgen como posible continuación del proyecto.

6.1. Conclusiones

En esta sección se hace un balance general del proyecto en base a los objetivos cumplidos y requisitos cubiertos que se expusieron en el capítulo 2.

En vista de lo expuesto en los capítulos anteriores se observa que en este proyecto se ha cumplido el objetivo principal, es decir, se ha conseguido implementar un algoritmo de autolocalización funcional en un espacio tridimensional completo con características de robustez, rapidez e independiente de señalización en el entorno; sólo necesita para su funcionamiento los sensores de posición y distancia del robot y un mapa del escenario a considerar.

Revisando los subobjetivos, se comprueba que se han cumplido en su totalidad. A continuación se desarrolla cada uno de los subobjetivos:

- *Desarrollo de un entorno para evaluar la respuesta de un algoritmo de localización.*
  El componente desarrollado en este proyecto fin de carrera dispone de funciones para depuración de la función de la salud tal y como se expone en las secciones 4.3.3 y 4.3.4. Además, estas funcionalidades se han utilizado durante las fases experimentales del proyecto, demostrando su utilidad y buen funcionamiento.

- *Diseño e implementación del algoritmo de autolocalización.*
  Tal y como se ha expuesto en el capítulo 4 se ha diseñado e implementado de forma funcional un algoritmo que cumple los requisitos necesarios para solucionar el problema de autolocalización autónoma basada en mapa y sensores de distancia. El algoritmo se basa en el *esquema evolutivo* y define actores para dividir la búsqueda de la solución en distintos niveles: *exploradores* para búsqueda de grano grueso, *explotadores* organizados en *razas* formadas en puntos escogidos por los exploradores para búsquedas a bajo nivel. También define un sistema novedoso de competición entre *explotadores* y entre *razas* basado en un parámetro denominado *crédito* con el fin de decidir el *individuo dominante* en cada caso.
  El componente resultante de estos dos puntos comprende más de 4900 líneas de código C++.

- *Diseño y ejecución de pruebas experimentales de evaluación.*
  Los experimentos utilizados para validar tanto el algoritmo completo como cada una de sus partes y los distintos escenarios que se han descrito en el capítulo 5 han servido para demostrar que el algoritmo cumple los objetivos marcados.

Al igual que se ha hecho con los subobjetivos, se repasan todos los requisitos descritos en el capítulo 2 y cómo se han cumplido:

- *Implementación de todo el sistema basada en la plataforma JdeRobot.*
  Se ha implementado el algoritmo como parte de un componente siguiendo la versión 5.2 de la plataforma *JdeRobot*; además, dicho componente hace uso de las interfaces definidas por la plataforma. Como prueba de que la implementación es totalmente compatible con el esquema general de la plataforma, no sólo los drivers utilizados para comunicar con los robots, sino que otros componentes como *replayer* y *teleoperator*, son capaces de comunicar con el componente.

- *Capacidad de todo el sistema de funcionamiento en tiempo real.*
  En base a las pruebas experimentales sobre caracterización temporal de la sección 5.4, se observa que el sistema es capaz de funcionar de media entre 35.83 y 59.05 ms/iteración dependiendo de la complejidad presentada por el entorno. Por tanto, la implementación presentada en este proyecto puede utilizarse en aplicaciones reales sin necesidad de optimizarlo para mejor respuesta en tiempo.

- *Convergencia del algoritmo dentro de un margen de error determinado.*
  A partir de los resultados experimentales, se ha observado que el algoritmo de autolocalización utilizado en un escenario de complejidad suficiente comete un error máximo aproximado de 0.2 m y 1° (reducible a 0.4° si el entorno presenta suficientes elementos diferenciadores).

- *Implementación como algoritmo genérico en el espacio tridimensional.*
  A pesar de que los experimentos se hayan realizado con robots que sólo son capaces de desplazase a ras de suelo, el hecho de que el algoritmo espere información del robot en formato $(x, y, z, \theta, \psi, \phi)$ le da capacidad de funcionar con un robot que tenga 6 grados de libertad. Además, el hecho de que en los experimentos se haya podido utilizar con éxito robots con 3 grados de libertad (x, y, θ) , implica que el algoritmo es lo suficientemente genérico para funcionar en ambos casos de forma transparente.

- *Modularización de todos los componentes del sistema.*
  La implementación final consta de distintos módulos, tal y como se ha comentado en el apartado 4.3, pudiendo desactivar los módulos no esenciales para el modo de funcionamiento natural del componente. Dando un paso más allá, se ha implementado el algoritmo y sus partes como módulos externos, permitiendo reutilizar el componente con nuevas implementaciones. Por último, el sistema de módulos externos no está limitado a nuevos algoritmos, sino que por diseño puede extenderse para agregar otras funcionalidades.

Durante la realización de este proyecto fin de carrera se han adquirido conocimientos sobre técnicas de autolocalización de robots autónomos y aplicaciones de algoritmos metaheurísticos y estadística a la robótica. Además, se han adquirido conocimientos y profundizado respecto al software de infraestructura utilizado, entre ellos se pueden nombrar los siguientes: C++, C aplicado a bibliotecas compartidas, OpenGL, gnuplot, gtkmm++, LaTeX, ICE, Boost y sistemas de carga de módulos externos a la aplicación.

6.2. Trabajos futuros

Para finalizar, en esta sección se exponen algunas posibles líneas de trabajo futuras que podrían resultar interesantes para ampliar los resultados de este proyecto:

1. **Profundizar en la aplicación sobre robots reales Kobuki y Autonav.**
   Aunque durante este proyecto se han utilizado robots reales en algunas de las pruebas, la mayor parte de la experimentación se ha llevado a cabo sobre entornos simulados. Profundizar en este aspecto no sólo permitiría conocer cómo de bien responde el algoritmo actual en la realidad, sino que también podrían observarse características o condiciones nuevas que el algoritmo pudiera aprovechar para mejorar su respuesta.

2. **Complementar la autolocalización con sistemas de apoyo como marcas visuales**
   En este proyecto, el sistema de autolocalización sólo tiene el apoyo de un sensor de posicionamiento y uno de distancia, lo que limita la robustez que puede alcanzar y limita su uso en entornos de demasiada complejidad. Añadir otra nueva fuente de información de distinta naturaleza como puede ser una marca visual (por ejemplo AprilTags), no sólo aportaría mayor robustez al algoritmo, sino que también permitiría investigar el trasfondo de las marcas visuales.

3. **Probar y mejorar para operación en escenarios muy poblados y con obstáculos móviles.**
   En el diseño del algoritmo no se ha considerado la posibilidad de que aparezca un obstáculo móvil que no se encuentre en el mapa del entorno original; pudiendo ser este obstáculo, por ejemplo, otro robot. Además, se ha observado que en escenarios que disponen de cierta complejidad estructural o están superpoblados, el sistema tiende a necesitar más capacidad de procesamiento o incluso pierde convergencia en la zona problemática. Estas observaciones dan pie a que se realicen experimentos orientados a estas problemáticas para observar hasta qué punto el algoritmo puede responder y si existe la posibilidad de mejorarlo para que lo haga correctamente.

Bibliografía

[1] Imagen tomada del artículo *Robot Industrial* de Wikipedia:
`http://es.wikipedia.org/wiki/Robot_industrial`

[2] Imagen tomada del artículo *Cirugía Robótica* de Wikipedia:
`http://es.wikipedia.org/wiki/Cirugía_robótica`

[3] Imagen tomada del artículo *Robot Doméstico* de Wikipedia:
`http://es.wikipedia.org/wiki/Robot_doméstico`

[4] Repositorio de subversion para este proyecto:
`https://svn.jderobot.org/users/lr.morales/pfc`

[5] Mediawiki para este proyecto:
`http://jderobot.org/Lr.morales-pfc`

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

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

[8] Redouane Kachach. Localización del robot Pioneer basada en láser. *Proyecto fin de carrera Ing. Técnica en Informática de Sistemas, Universidad Rey Juan Carlos*, 2005

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

[10] Julio Manuel Vega Pérez. Navegación y autolocalización de un robot guía de visitantes. *Proyecto fin de carrera Ing. Informática, Universidad Rey Juan Carlos*, 2008

[11] José Manuel Domínguez Arroyo. Autolocalización probabilística visual de un robot en el simulador Gazebo. *Proyecto fin de carrera Ing. Informática, Universidad Rey Juan Carlos*, 2009

[12] Eduardo Perdices García. Autolocalización visual en la RoboCup basada en detección de porterías 3D. *Proyecto fin de carrera Ing. Informática, Universidad Rey Juan Carlos*, 2009

[13] Eduardo Perdices García. Autolocalización visual en la RoboCup con algoritmos basados en muestras. *Trabajo fin de máster en Sistemas Telemáticos e Informáticos, Universidad Rey Juan Carlos*, 2010

[14] Darío Rodríguez de Diego. Algoritmo evolutivo para la autolocalización de un robot móvil con láser y visión. *Proyecto fin de carrera Ing. Técnica en Informática de Sistemas, Universidad Rey Juan Carlos*, 2010