INGENIERÍA INFORMÁTICA

Escuela Técnica Superior de Ingeniería Informática

Curso académico 2008-2009

Proyecto Fin de Carrera

Autolocalización probabilística visual de un robot en el simulador Gazebo

Tutor: José María Cañas Plaza
Autor: Jose Manuel Domínguez Arroyo

A mi familia y amigos.

Agradecimientos

Quisiera dedicar este proyecto a muchísima gente así que si hay alguien que no se encuentre encuadrado en la gente que voy a decir, que me perdone, pero que también va por ellos.

En primer lugar quisiera agradecerle todo el apoyo que me ha prestado Jose Maria Cañas, sin él no hubiera sido capaz de realizar este proyecto y sobretodo por la paciencia que ha tenido conmigo, ha demostrado ser una gran tutor y una mejor persona.

También darle las gracias a todo el grupo de Robótica por haber aguantado todo este tiempo metidos en laboratorio. Muy especialmente me gustaría acordarme de Julio por haber echado una mano cuando lo necesite, Gonzalo y Pablo por haberme echo pasar unos buenos ratos, Victor H. por haberme apoyado en todo momento y en definitiva a todos con los que estado este largo tiempo.

A mi familia sobretodo los quiero mucho, gracias por haberme permitido poder obtener esta educación y que sin ellos yo no seria lo que soy en estos momentos.

A mis amigos de Fuenla les doy las gracias por estar siempre ahí cuando lo he necesitado. No me puedo poner todos los nombres por que seguro que se me olvidaría alguno.

A mis dos pueblos que siempre os llevare en el corazón Lagunilla y Daimiel. Toda esa gente buena que hay y que es mucha. Y para terminar no podía olvidarme de esos *Jockers* que este año hacemos 10 añitos y como siempre esas fiestas fueron inolvidables.

Que no somos de aquí...

Resumen

La autolocalización constituye una parte muy importante dentro del mundo de la robótica. Existen varios formas de conseguir que un robot se localice en un escenario. Una de ellas es la visual, mediante una cámara y un mapa un robot es capaz de observar una escena y conseguir localizarse en ella.

En el presente proyecto se pretende conseguir una localización visual mediante una cámara. Tenemos un mapa visual de color donde coloreamos los diferentes objetos que podemos observar, en este mapa visual es donde nos localizamos. Tenemos también sensores que nos dan la odometría que ha experimentado el robot llamados *encoders*. Devolvemos la posición en (x,y) y la orientación (θ) del robot, donde suponemos que se encuentra el robot.

El algoritmo que hemos usado para realizar este proyecto, es un algoritmo con muestreo llamado *filtro de particulas*. Se divide en tres partes. El modelo de observación, devuelve una probabilidad que tiene cada una de las particulas que tenemos repartidas por el mapa visual, de que estén en la posición en la que se encuentra el robot. El modelo de movimiento, dota a las particulas de movimiento, hace que las particulas se muevan de la misma manera que lo hace el robot. El remuestreo, actualiza la población de particulas, haciendo que en cada iteración del algoritmo se queden las que tienen más probabilidades de ser la posición donde se encuentra el robot.

Hemos realizado números experimentos para conseguir ajustar el algoritmo de filtro de particulas para que sea robusto y ágil. Todos los experimentos se han realizado en el entorno simulado *Gazebo* en un mudo que simula el departamental II de esta universidad.

Índice general

Índice de figuras

Capítulo 1
Introducción

En este primer capítulo vamos a dar una visión global del contexto del presente proyecto. Comenzaremos con una reseña a lo que significa la palabra robótica, la historia que ella conlleva y algunas aplicaciones que se han realizado basadas en ella. Después explicaremos la localización en robot móviles que es el tema principal de este proyecto, mostraremos ejemplos de cómo abarcar este problema y diferentes soluciones que se han dado. Por último, describiremos proyectos llevados acabo por el grupo de Robótica y que son los precursores de este, en un contexto inmediato.

1.1. Robótica

El significado de la palabra *Robótica* nos dice que es la ciencia y tecnología de los robots. Se encarga del diseño, manufacturación y realización de aplicaciones de los robots. La robótica se compone de varias ciencias entre ellas la electrónica, mecánica, informática, inteligencia artificial e ingeniería del control. El termino robótica fue acuñado por Isaac Asimov (1920 - 1992) figura 1.1 a) en su relato breve ¡Mentiroso! de 1941. Es considerado uno de los *Tres Grandes* escritores de ciencia ficción, fue también el encargado de crear las *Tres leyes de la Robótica*. Una curiosidad aunque no esta confirmada es que el ASIMO de Honda lleva ese nombre por este escritor.

Otra palabra que tenemos que explicar es la palabra *Robot*, su significado nos dice que un robot es una entidad virtual o mecánica artificial. En la práctica, esto es por lo general un sistema electro-mecánico que, por su apariencia o sus movimientos, ofrece la sensación de tener un propósito propio. La palabra robot puede referirse tanto a mecanismos físicos como a sistemas virtuales de software, aunque suele aludir se a los segundos con el término de bots. Un robot para ser considerado como tal debe de contener sensores, actuadores y procesadores. La primera aparición de esta palabra fue en el año 1921 en la obra Rossum's Universal Robot (R.U.R.) realizada por el escritor checo Karel Capek (1890 - 1938) figura 1.1 b). Su origen es la palabra eslava robota, que se refiere al trabajo realizado de manera forzada.

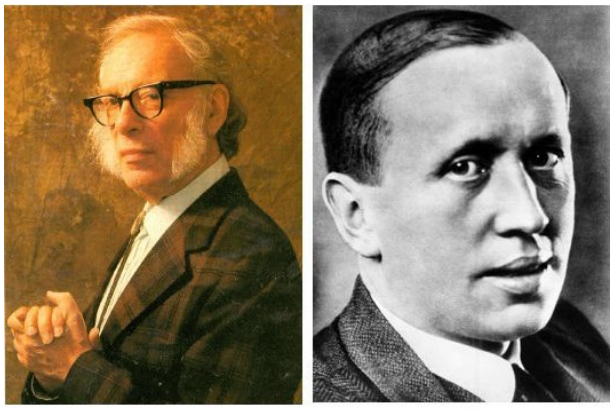

Figura 1.1: a) Foto del escritor Isaac Asimov b) Foto del escritor Karel Capek

Los robot se han introducido en nuestras vidas aunque nosotros no nos hayamos dado cuenta. Existen robots en nuestro entorno doméstico que se dedican por ejemplo a aspirar, barrer, cortar el césped, etc. Uno de ellos es la famosa aspiradora *Roomba* figura 1.2 a) se encarga de limpiar el polvo solo con apretar un botón, ha sido creada por la empresa estadounidense *iRobot* y cuenta con sensores que son capaces de detectar si esta cerca de algún escalón o es capaz de saber sobre que superficie se encuentra y dinámicamente es capaz de adaptarse a esta. Sus actuadores son una ruedas que se encuentran en la base del robot, en esta base también se encuentran unos rodillos que son los que encargan de aspirar todo el polvo, es capaz de cargarse automáticamente y los recorridos que realiza son aleatorios.
Otro robot doméstico que podemos encontrar en el mercado es el *Automower* figura 1.2 b) que es un robot cortacesped autónomo de la empresa *Husqvarna*, su forma de funcionamiento es muy sencilla se coloca en el césped que queremos cortar y se le deja actuar de forma autónoma, puede chocar contra obstáculos como por ejemplo árboles y no pasara nada y si tenemos piscina se colocar un hilo periférico en la zona y cuando el robot quiera pasar por ahí no podrá por este hilo no se lo permitirá, también se puede mojar con la lluvia ya que es estanco y el será el encargado de cortar el césped todos los días sin que tú tengas que preocuparte.

Figura 1.2: a) aspiradora Roomba b) Cortacesped Automower

Otra rama donde los robot tienen una alta presencia es en los robots industriales. Se considera un robot industrial según la Organización Internacional de Estándares (ISO) a un manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas. Una empresa que se encarga de realizar robot industriales es *KUKA*, esta empresa tiene un gran número de robots industriales los cuales los clasifican en pequeños robots, carga ligera, carga mediana, carga pesada y modelos constructivos especiales. Esta clasificación se baso sobre todo por lo que son capaces de manipular. Su uso suele estar orientado al sector de la automoción o el embalaje. Un ejemplo de robot de carga pesada es el *KR 150-2 (Serie 2000)* figura 1.3, tiene un carga de 150/130/110 kg, una carga adicional de 100 kg, el max. alcance de trabajo es de 2700/2900/3100 mm, su numero de ejes es 6, tiene una repetibilidad menor que mas menos 0,12 mm, un peso de 1245/1255/1263 kg y la posición de montajes en el suelo o en el techo.

Figura 1.3: El robot industrial KR 150-2 (Serie 2000)

Los seres humanos siempre hemos querido que nuestras creaciones nos resultaran familiares y los robots no son la excepción, por ese motivo aparecieron los robots humanoides, estos robots no solo realizan tareas que son nuestras si no que también se nos parecen sus actuadores son brazos y piernas y sus sensores pueden ser cámaras que simulan visión, etc. Entre estos robots podemos encontrar el famoso ASIMO de Honda figura 1.4 a), NAO de aldebaran figura 1.4 b), etc.

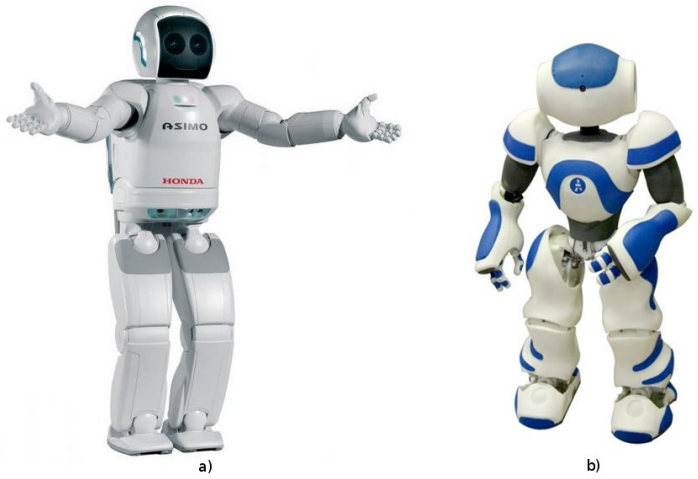

Figura 1.4: a) robot ASIMO de Honda b) robot NAO de Aldebaran

Introduciéndonos en el mundo de las competiciones, la robótica también tiene sus propias competiciones muy importantes y de ámbito mundial.
La Robot Cup es una competición internacional integrada por robots autónomos, su principal objetivo es la investigación y la educación sobre la inteligencia artificial. Su primera edición fue en 1997 en Nagoya (Japón), en 2009 se ha realizado en Graz (Austria). La Robot Cup esta compuesta de varias categorías, la RobotCup Soccer, RobotCup Rescue, RobotCup@Home y RobotCup Junior.
La RobotCup Soccer se juegan partidos de fútbol con sus reglas, esta categoría se divide a su vez en varias subcategorías y en la subcategoría Standard League participa el grupo de investigación de robótica de esta universidad. Algunas reglas importantes que tenemos en esta subcategoría, es que todos los equipos juegan con el NAO de Aldebaran, el numero de robots es tres (un portero y dos jugadores) figura 1.5, una vez que comience el juego no se puede tocar a los robots, por lo tanto deben ser robustos en los movimientos para no caerse y si se caen saber levantarse, tener una buena localización para saber donde están y donde tienen la portería contraria.
La RobotCup Rescue consiste en poner a prueba a robots en tareas de búsqueda y salvamento de víctimas en terrenos desfavorables.
La RobotCup@Home se basa en colocar robot en la vida cotidiana de un ser humano y ver la relación humano-robot que existe.
La RobotCup Junior es una categoría diseñada para los más pequeños, consiste en que muchachos de primaria y secundaria creen robots capaces de jugar al fútbol, al rescate, o bailen.

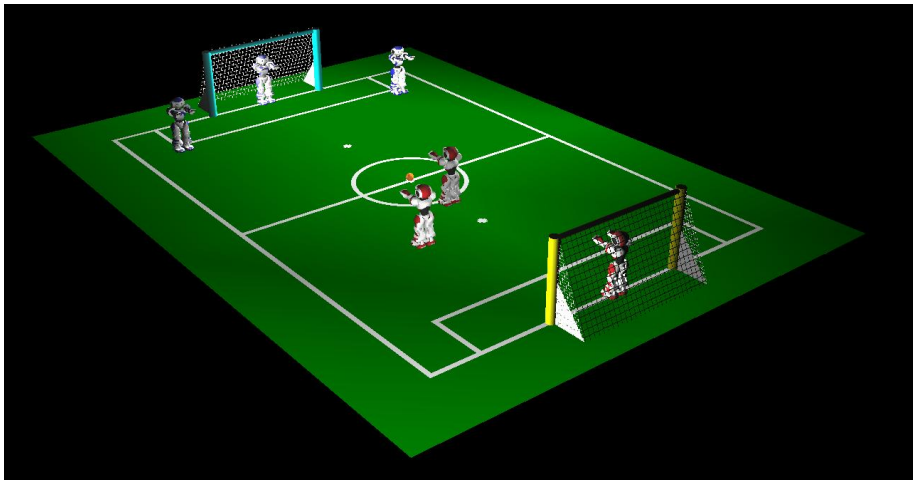

Figura 1.5: Foto que muestra la colocación de los robots en un partido de Standar League.

La DARPA Grand Challenge es otra competición robótica diferente a la anterior. Esta consiste en una carrera de coches autónomos, que deben de llegar desde un punto inicial a otro final pasando por puntos intermedios sin la intervención humana. DARPA es una agencia dependiente del ministerio de defensa de los EEUU que incentiva el desarrollo de proyectos tecnológicos, es por este motivo por el cual creo esta competición, para conseguir vehículos capaces de moverse por zonas de difícil acceso que no necesiten de un control humano. Esta competición ha tenido tres ediciones. En 2004 se celebro la primera edición, su recorrido fue por el desierto del Mojave (EEUU) y no consiguió ningún participantes completar los 240 km que tenia el recorrido, el que más distancia consiguió completar fue de 11.78 km. En 2005 tuvo lugar la segunda edición en mismo lugar pero esta vez hubo cinco vehículos que consiguieron llegar a la meta , el ganador de esta edición fue el vehículo Stanley figura 1.6 construido por la universidad de Stanford que se llevo un millón de dolares. En 2007 tuvo lugar la ultima competición, se le cambio el nombre por Urban Challege y también se cambio el recorrido, en vez de hacerlo en un desierto se decidió ponerlo más difícil y realizarlo en la ciudad, el sitio el elegido fue la base aérea de George en California. Ha esta edición se le quiso añadir algo de dificultad, así que se realizo en un circuito urbano, esto provoco que además de estar atentos a otros vehículos participantes también se tuviera que estar atento al trafico provocado por la ciudad y a las señales de trafico ya que había que respetarlas. El ganador de esta edición fue Tartan Racing construido por la universidad de Carnegie que se llevo dos millones de dolares.
En este tipo de competiciones hay que tener muy en cuenta la localización del vehículo ya que si estamos mal localizado podemos provocar que el vehículo se dirigirá por un camino equivocado y tardemos más en llegar a nuestro objetivo o peor que tenga un accidente y quedemos eliminados. Para esta prueba se usaron sistemas GPS y GIS (localización geográfica).

Otro campo interesante en el mundo de la robótica son los llamados robots guía. Son robots que están programados para ayudar a las personas a llegar a un sitio que ellos desconocen. Este tipo de robots debe tener bien implementados un sistema de

Figura 1.6: Vehículo Stanley que gano el Gand Challenge en 2005

navegación local que le permita esquivar obstáculos cercanos y dinámicos, un sistema de navegación global que averigüe cual es la mejor ruta para llegar del punto en el que se encuentra al punto requerido y un sistema de localización que le diga en todo momento donde se encuentra para saber que camino tomar.

Uno de estos robots guía es el llamado Minerva figura 1.7. Minerva es un robot creado para educar y entretener a las personas en lugares públicos. Minerva es un robot guía que tiene como propósito servir de guía en un museo llevando a la persona que lo requiera de un lugar a otro explicándole todo lo que se encuentra por el camino. Fue creado de forma conjunta por el School of Computer Science(Carnegie Mellon University, U.S.A) y Computer Science Departament III (Bonn, University, Alemania). Fue instalado en 1998 en el Museo Nacional de Historia Americana de Smithsonian. Estuvo dos semanas, interactuó con miles de personas y recorrió mas de 44 km a una velocidad media de 5.8 km/h.

Figura 1.7: Robot Minerva

Su sistema de localización figura 1.8 emplea dos tipos de mapas para orientarse así mismo. Uno de ocupación y otro con la textura del techo del museo. Ambos son incorporados manualmente teleoperandolo a través de sus sensores cámaras, láser, y lecturas odométricas.

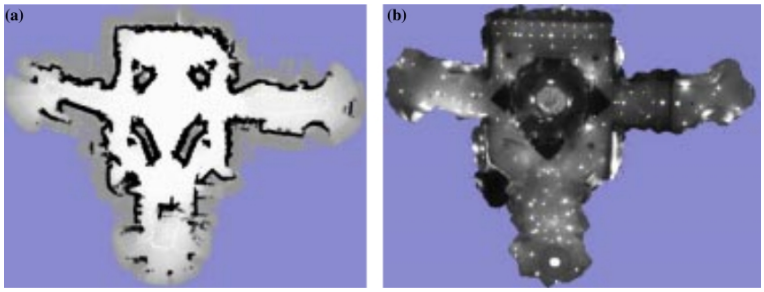

Figura 1.8: Mapas de localización de Minerva

La última rama es la de los robot espaciales. Son usados para explorar territorios que los seres humanos no son capaces de alcanzar todavía. Los podemos encontrar en la Luna o en Marte y su principal función es recoger información de la zona en la que se encuentra, puede buscar agua o vida microscópica, o puede buscar una zona donde en un futuro se pueda asentar un emplazamiento humano o una zona de aterrizaje de una nave tripulada por seres humanos. En 1966 se inicio el programa Surveyor figura 1.9 a) que constó de 7 misiones en las cuales se mandaron a la Luna naves robotizadas para el estudio de la zona donde aterrizaría la expedición Apolo. En 1970 la URSS mando el "tractor"Lunakhod que recorrió la superficie selenita y tomo muestras de ella. A Marte también se han mandado misiones, 1976 aterrizaba la nave viking y las más novedosas que hay ahora mismo tomando muestras del suelo de Marte en busca de agua o vida son la Spirit figura 1.9 b) y la Oportunity.

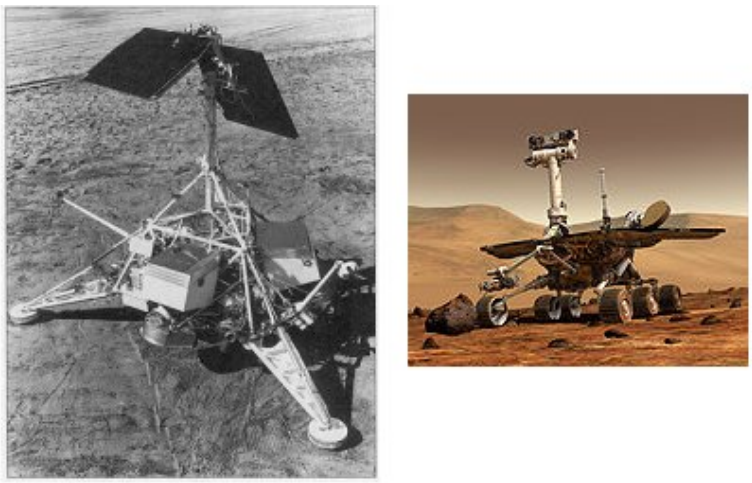

Figura 1.9: a) robot Surveyor en la Luna b) robot Sprit en Marte

1.2. Localización en robots móviles

La localización de robots moviles es un problema clásico de la robótica. ¿El por que es tan importante saber donde nos encontramos? Por que si queremos llegar a sitio lo primero que debemos saber es donde nos encontramos para después poder

trazar un camino que nos lleve a la meta y sea el mejor, también es importante porque mientras realizamos el camino debemos saber donde estamos para verificar que vamos por el camino correcto. Para conseguir localizarnos necesitamos de sensores que tiene el robot como son láser, cámara, sensores odométricos, etc. También se necesitan mapas de la zona a explorar para poder situar el robot en la posición correcta. Con todo ello conseguimos estimar la posición del robot en un entorno y aunque se mueva el robot seguimos sabiendo dónde esta situado.

Para resolver la localización existen varias formas de abordar el problema. El primero es cuando sabemos la posición inicial del robot,es la más sencilla, la solución se consigue calculando la posición final estimando el error odométrico que se va dando por cada movimiento. El segundo es la localización de un robot cuando no se sabe cual es la posición inicial de éste, esta forma es en la que se encuadra este proyecto, es más difícil de averiguar que la primera ya que no sabemos a priori dónde se encuentra el robot y tenemos que hacer un muestreo por todo el entorno para averiguar donde nos encontramos. La tercera forma es la localización de varios robots, es muy interesante porque podemos localizarlos gracias a relación que existe entre ellos.

Los sensores son la parte más importante en la localización, tenemos dos tipos. Los primeros son los encoders figura 1.10 b) que son los encargados de devolver la posición del robot, por si solos son capaces de darnos una localización pero tienen un error producido por el propio robot llamados sistemáticos que es acumulativo,el cual hace que la localización sea cada vez menos precisa, también tenemos un error que es producido por agentes externos llamado no sistemático que también tiene que ser tenido en cuenta.
Los segundos sensores son las cámaras, los lasers, los sonars, que por si solos no sirven para localizarnos pero que gracias a ellos podemos hacer los muestreos que que nos eliminen el error producido por los encoders.

Otro sensor importante el GPS figura 1.10 a) al igual que los encoders no necesita la ayuda de otros sensores para la localización. GPS significa Global Position System y consta de una serie de 27 satelites repartidos por todo el planeta a 20200 km con orbitas sincronizadas para oder abarcar todo el planeta y que sirve para localizar la posición de un objeto, vehículo, persona etc en el globo terráqueo. Para saber la posición lo primero que se obtiene es la posición de al menos tres satélites y mediante triangulación calculamos la posición en la cual nos encontramos. Es una buena forma de localización cuando el robot móvil es por ejemplo vehículo y queremos localizarnos en un entorno exterior como por ejemplo en el Grand Challenge, pero en nuestro caso con sistemas más sencillos nos valdrá para localizarnos.

Las técnicas de localización es otro punto que hay que tener en cuenta, existen

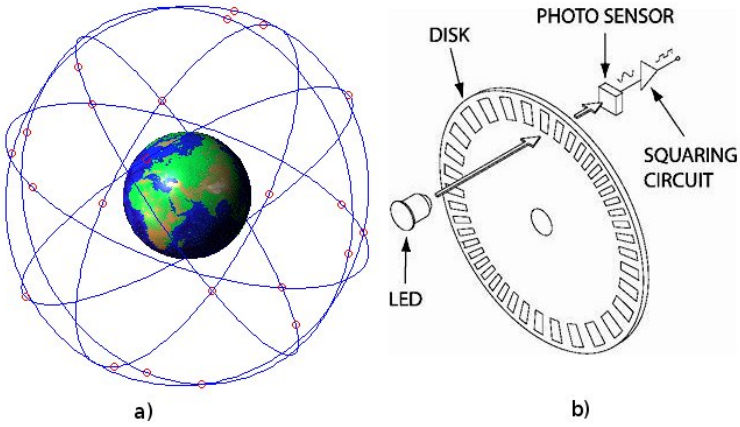

Figura 1.10: a) Sistema GPS de localización b) Encoders de un robot

muchas e intentan resolver el problema de localización sin sensores específicos:

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

- Los *filtros de Kalman*: [Isard y Blake, 1998] estas técnica trata de resolver recursiva y periódicamente la posición de mínima varianza fusionando información parcial e indirecta sobre la localización. Aplica una técnica unimodal y gaussiana. Se comporta mal con el ruido de las lecturas sensoriales, entornos dinámicos y no es capaz de manejar varias hipótesis.

- el *scan matching*: esta técnica utiliza mapa locales para compararlos con lecturas sensoriales alineado estas lecturas con posiciones cercanas a la creemos que se encuentra el robot en el mapa. Primeramente necesitamos estimar la posición inicial del robot con una distribución gaussiana que se va actualizando con la lecturas de los sensores.

- *localización probabilística*: [Thrun, 2000] esta técnica es adecuada para interiores y calcula la posición del robot teniendo en cuenta la lectura de los sensores y el movimiento que este va realizando. Al principio no se sabe donde esta situado el robot y según pasa el tiempo se van introduciendo los datos de las lecturas de los sensores y el movimiento que va realizando se va calculando una probabilidad de que se encuentre en ese sitio. La eficiencia de esta técnica depende del tamaño del mapa.

1.3. Localización probabilística visual

Como hemos dicho anteriormente el problema que intenta resolver este proyecto es localización probabilística mediante visión. Uno de los motivos por el cual hemos

elegido la localización probabilística visual, es porque los sensores que usa son cámaras y estos sensores suelen ser baratos comparados con los GPS por ejemplo, y dan mucha información de la escena. Se han abarcado una posibilidad dentro de la localización probabilística. Se llama localización probabilística visual basada en filtro de partículas. Es posible obtener una localización correcta del robot en poco tiempo porque no es exhaustiva, no se miran todas las posiciones posibles si las mas prometedoras. Se explicara esta técnica con más detalle en capítulos posteriores. También explicaremos la técnica basada en mallas de probabilidad pero en menos detalle ya que solo la hemos usado para comprobar que nuestro algoritmo converge a la posición correcta.

Vamos a ver un ejemplo intuitivo de como funciona la localización probabilística visual en la figura 1.11.

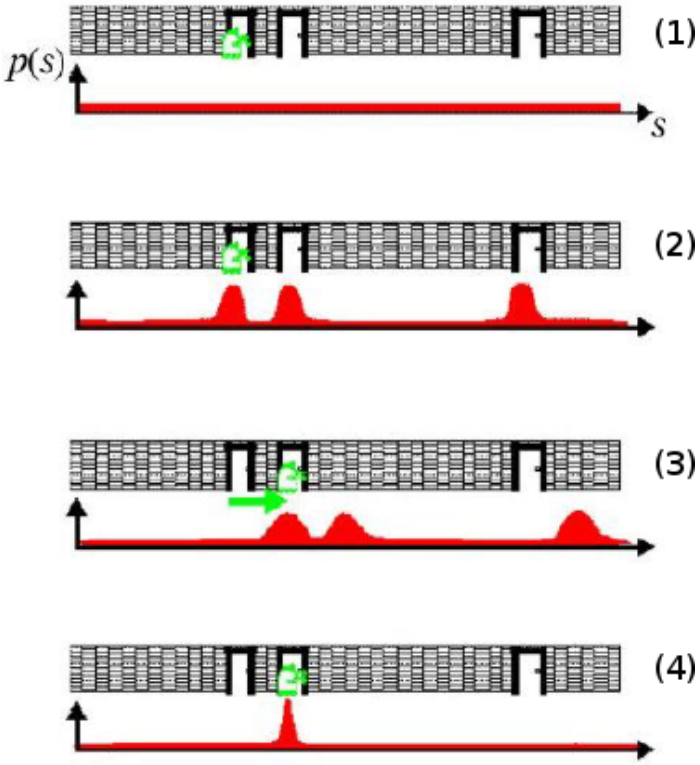

Figura 1.11: Localización probabilística

En el primer paso se dice que cualquier punto tiene la misma probabilidad porque todavía no hemos hecho ninguna lectura sensorial (en este caso sera una observación con la cámara).En el segundo paso ya hemos realizado una observación y hemos llegado a la conclusión de que estamos delante de una puerta, así que todos los puntos que están en frente de una puerta tienen mucha probabilidad. En el tercer paso realizamos un movimiento a la derecha así que desplazamos la probabilidad de todos los puntos a la derecha. En el cuarto paso calculamos una lectura sensorial y obtenemos que estamos delante de una puerta. Como se puede ver a cada paso que vamos dando obtenemos menos sitios probables hasta que al final en el cuarto paso tenemos que solo podemos estar en un sitio.

En el grupo de Robótica de Universidad Rey Juan Carlos se han realizado algunos trabajos de localización los cuales son precursores de este. El grupo de Robótica lo componemos una serie de profesores y alumnos que estamos interesados en la investigación de esta rama.

Como proyectos de localización del grupo de Robótica que son los predecesores a este podemos encontrar el de María Ángeles Crespo [Crespo, 2003] es un proyecto de localización probabilística con visión implementando el algoritmo de MonteCarlo para el perrito Aibo y el Eyebot, todo ello probado en entorno simulado ,el de Redouane Kachach [Kachach, 2005] es un proyecto de localización probabilística con láser para el robot Pioneer en el departamental,el de Alberto Lopez [López, 2005] es un proyecto de localización probabilística con visión para el Pioneer en el departamental que compara las técnicas de malla de probabilidad y el filtro de partículas y el de Angel Cortes [Cortés, 2007] es un proyecto de localización probabilística y construcción de mapas en el departamental también conocido como método SLAM.

Nuestro proyecto se basa en el proyecto de Alberto Lopez [López, 2005], Alberto comparo dos métodos de localización probabilística visual, mallas de probabilidad y filtro de partículas, llego a la conclusión que el filtro de partículas era mejor porque el de mallas de probabilidad tardaba mucho y nosotros necesitamos un algoritmo que pueda ser usado en tiempo real. Yo en este proyecto he intentado darle una vuelta de tuerca más. He usado el método de filtro de partículas pero he cambiado la función de distancia en el algoritmo de filtro de partículas he añadido algunos conceptos. Todo esto lo he hecho para conseguir que la localización sea más robusta y se mantenga mientras el robot esta en movimiento.

En siguiente capítulo 2 hablaremos de los objetivos que se pretenden cumplir con este proyecto, así como de los requisitos que este tiene. En el capítulo 3 explicaremos la plataforma software JDEROBOT que hemos usado, así como el simulador Gazebo que hemos utilizado para hacer pruebas, también mencionaremos librerías externas que hemos usado como son GTK y Open GL. En el capítulo 4 expondremos los fundamentos y diseño del algoritmo de localización probabilística con filtro de partículas. En el capítulo 5 explicaremos los resultados obtenidos de la localización probabilística con filtro de partículas. Y para terminar en el capítulo 6 sacaremos unas conclusiones y expondremos los siguientes pasos a realizar.

Capítulo 2
Objetivos

Una vez presentado en el capítulo anterior el contexto general en el que se encuadra nuestro proyecto, vamos a describir el problema que se plantea dando unos objetivos, así como, los requisitos que nuestra solución ha de satisfacer.

El presente proyecto tiene como objetivo conseguir una localización del robot robusta por el departamental II simulado mediante visión aplicando técnicas de visión probabilística.

2.1. Descripción del problema

Estamos interesados en conseguir una localización robusta en un entorno simulado. Para conseguir la localización disponemos de un mapa de color en el que tendremos los diferentes objetos que podemos ver (puertas,papeleras, extintores y sillas), también disponemos de una imagen que obtenemos a través de una cámara, que se encuentra en un entorno simulado del departamental II y para terminar también disponemos de la odómetría que nos proporciona el robot. Devolveremos la posición en la que creemos que se encuentra el robot.

El objetivo principal es poner el robot simulado en cualquier posición del mapa, el robot simulado se ira moviendo por el departamental, nos proporcionara imágenes de lo que esta viendo y nos mandara lo que se ha desplazado en ese movimiento, al cabo de algunas iteraciones el robot simulado se encontrara localizado en una única zona. Según se vaya moviendo seguirá localizado en todo momento.

El objetivo principal se puede articular en varios subobjetivos concretos que se desarrollan a lo largo de este proyecto. Estos subobjetivos son los siguientes:

- *Cálculo de las imágenes real y teórica.* Se estudiará cuál es la mejor forma de sacar la información relevante de la imagen real y de la imagen teórica, así mismo las dos información tendrán que ser comparables.
- *Cálculo de la función distancia.* Se estudiaran varios tipos de funciones de distancia entre imágenes, para poder medir el parecido entre la imagen real y la imagen teórica y se elegirá la que que sea más correcta para su uso en la localización probabilística.

- *Experimentos.* Estos experimentos servirán para estudiar el comportamiento del robot aplicando la técnica de localización probabilística, para validar la solución desarrollada y comprobar bien su comportamiento. Se realizarán con simulador, y para tener un aspecto amigable y ayudar a depurar las técnicas y mostrar los resultados se realizará una interfaz gráfica.

En todo momento se realizará el estudio en un entorno simulado con la herramienta *Gazebo*.

2.2. Requisitos

A continuación explicaremos los requisitos que son necesarios para poder realizar este proyecto.
Como requisitos básicos tenemos el lenguaje que se ha utilizado es C para la programación bajo la plataforma JDEROBOT. Como sistema operativo hemos usado Linux con la distribución Ubuntu 8.04.

Un requisito muy importante es que el algoritmo funcione rápido, aunque sea en entorno simulado queremos que la respuesta se parezca la respuesta que se debe de dar en entorno real, ya que una próxima fase en otro proyecto sera usarlo con el robot Pioneer real.

Por último un requisito que nos marca un punto importante del proyecto, es que el algoritmo de localización ha de ser robusto. Tenemos que ser capaces de localizarnos empezando en diferentes posiciones, o si dejamos de localizarnos cuando volvamos a intentar localizarnos tenemos que ser capaces de localizarnos otra vez correctamente.

2.3. Metodología y Plan de Trabajo

Para el desarrollo de este proyecto se ha utilizado una metodología iterativa incremental. Para este desarrollo se han planificado reuniones periódicas, aproximadamente cada dos semanas, en estas reuniones se marcaban unos objetivos y se llevaba a cabo un incremento. En la figura 2.1 se muestra un enfoque global del proyecto.

Atendiendo a la metodología seguida se han definido fases del plan de trabajo divididos en los siguientes incrementos:

- Formación inicial y familiarización con la infraestructura software: En esta fase se pretende adquirir los conocimientos básicos de la infraestructura software desarrollando pequeñas aplicaciones todas ellas realizadas en la plataforma jderobot (véase capítulo 3). También se pretende adquirir la suficiente soltura en el uso del simulador Gazebo, ya que se tiene que realizar un departamental en 3 dimensiones con puertas, extintores y papeleras.

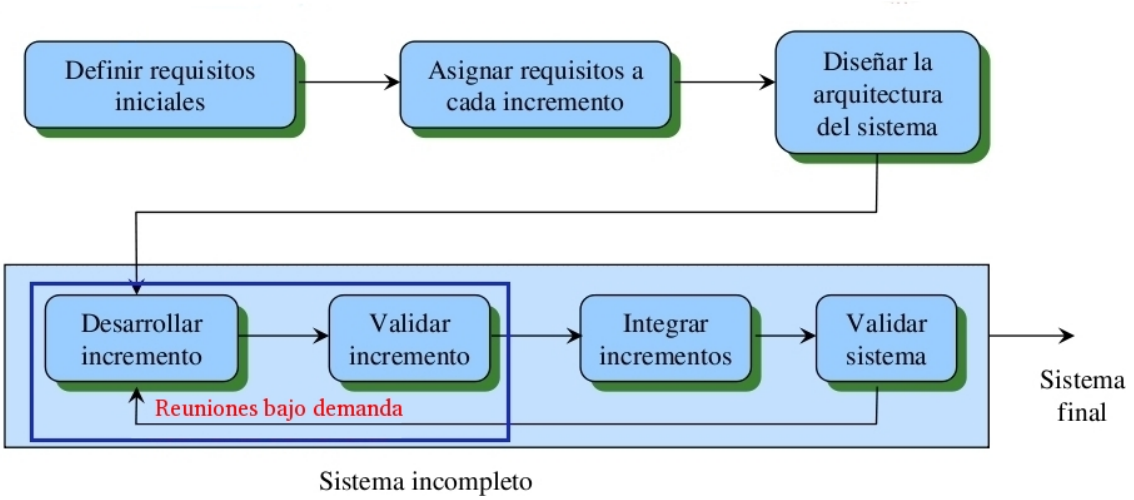

Figura 2.1: Modelo incremental de desarrollo software.

- Calculo de la imagen teórica y imagen real: En esta fase se pretende obtener una imagen real y una imagen teórica que puedan ser comparables y que en un mismo lugar en el mapa la imagen real y la imagen teórica sean iguales.
- Estudio de la función de distancia: En esta fase se calcularan varias funciones de distancia tanto basadas en píxeles como en objetos. Se obtendrán gráficas de esas funciones y la que tenga una función discriminante aceptable, que no sea muy discriminante ni poco discriminante, sera la que usaremos.
- Implementación con partículas: En esta fase se implementara un algoritmo probabilístico con filtro de partículas, el cual se dividirá en modelo de observación, modelo de movimiento y remuestreo.
- Pruebas y Ajustes En esta fase se pretende comprobar que el algoritmo de localización basado en filtro de partículas cumple los objetivos marcados, es decir, que es robusto y ágil. Además se realizaran los ajustes necesarios a los parámetros del algoritmo para que su funcionamiento sea el correcto.

Capítulo 3
Plataforma de Desarrollo

En este capítulo vamos a explicar la plataforma software (jderobot, gazebo) que hemos usado para el desarrollo de este proyecto. Así mismo también explicaremos el uso de algunas bibliotecas externas, OpenGL para la visualización de los algoritmos implementados, GridsLib para la creación de una rejilla de ocupación,GTK para la realización de un interfaz gráfica y progeo para el calculo de geometría proyectiva.

3.1. Gazebo

Gazebo es un simulador para mundos en 3D. *Gazebo* pertenece al proyecto *The Player Proyect* que tiene detrás de él una comunidad muy activa de desarrolladores que han conseguido que sus herramientas sean las más usadas por este tipo de robots. El proyecto The Player Proyect aparte de este simulador también tienen el simulador Stage para entornos en 2D y el servidor Player para manejar el robot ya que es capaz de controlar los dispositivos de este.
Gazebo como he dicho antes es capaz de simular entornos 3D, esta realizado en C++, la visualización del simulador esta implementada en OpenGL, esta perfectamente integrado en la simulación del Pioneer ya que tiene componentes pertenecientes a este robot y es capaz de generar entornos de simulación realizados por nosotros.

3.1.1. Creación de mundos Simulados

Gazebo para generar una escena necesita un fichero de configuración, este fichero viene dado por la extensión .world, el formato que usa es etiquetado como XML, se inicia siempre con la etiqueta <gz:world> y termina con la etiqueta < /gz:world>, todas las etiquetas siguen el mismo formato y se inician por <etiqueta> y terminan por < /etiqueta>, vamos a ver un ejemplo:
<model:ObserverCam>
<id>userCam0< /id>
<xyz>5.637 93.859 3.053< /xyz>
<rpy>-0 13 -39< /rpy>
<imageSize>640 480< /imageSize>
<updateRate>10< /updateRate>
<renderMethod>SGIX< /renderMethod>
< /model:ObserverCam>
La etiqueta <model:ObserverCam> es el objeto de la cámara que visualiza la escena,

el resto de etiquetas son propiedades de esa cámara, <xyz> dice la posición en la que se encuentra la cámara, <rpy> dice respecto a los ejes x,y,z el ángulo que forman, etc. Esta es la forma de definir los objetos que aparecen en la escena. Gazebo tiene un parseador que se encarga de coger este fichero y convertir las etiquetas que hay en el definidas en objetos con sus propiedades en OpenGL. Existen varias formas de crear mundos en gazebo.

Una forma consiste en construir un terrain, un terrain es un dibujo en 2D, se le hace el alzado con una herramienta que posee gazebo, esta forma es buena si queremos construir un laberinto o algo aproximado, ya que suele redondear las esquinas y trata el dibujo como un objeto único.

La segunda forma es construir el mundo en vez que con un objeto si no con muchos objetos. Gazebo proporciona una buena variedad de objetos básicos, tenemos cubos, esferas etc. Además proporciona de algunos objetos complejos como el robot Pioneer, cámaras, etc. Como sabemos que nuestro mundo esta compuesto por paredes, papeleras, extintores, puertas, sillas, podemos crear estos objetos a través de objetos básicos. Las paredes son cubos de color blancos con una altura de dos metros y una anchura de diez centímetros, la longitud lo marcan las esquinas. las puertas son cubos de color ocre con la misma altura y la misma anchura pero con una longitud que varia de ochenta centímetros a un metro. La papelera es un cubo de color azul de un metro de altura y una anchura y una longitud de cuarenta centímetros. los extintores son cubos de color rojo de sesenta centímetros de altura y veinte centímetros de anchura y longitud. La silla son un cubo de color negro con una anchura y una longitud de diez centímetros y una altura de cincuenta centímetros, esto representa la pata de la silla, otro cubo de color verde de anchura y longitud de cuarenta centímetros y una altura de diez centímetros, esto representa el asiento y por ultimo otro cubo de color verde con una anchura de diez centímetros y una altura y una longitud de cuarenta centímetros, esto es el respaldo. Es más costosos de realizar que con el terrain ya que tienes que saber las medidas de cada objeto y saber donde van. Les hemos añadidos a los objetos una propiedad de inmmovible que sirve para que cuando nos choquemos no se caigan o se muevan los objetos. Esta es la forma que hemos utilizado para crear nuestro mundo simulado.

En este proyecto se ha creado un mundo con el departamental 2 en el cuál se han introducido las paredes, las puertas, una papelera , una silla y extintores.

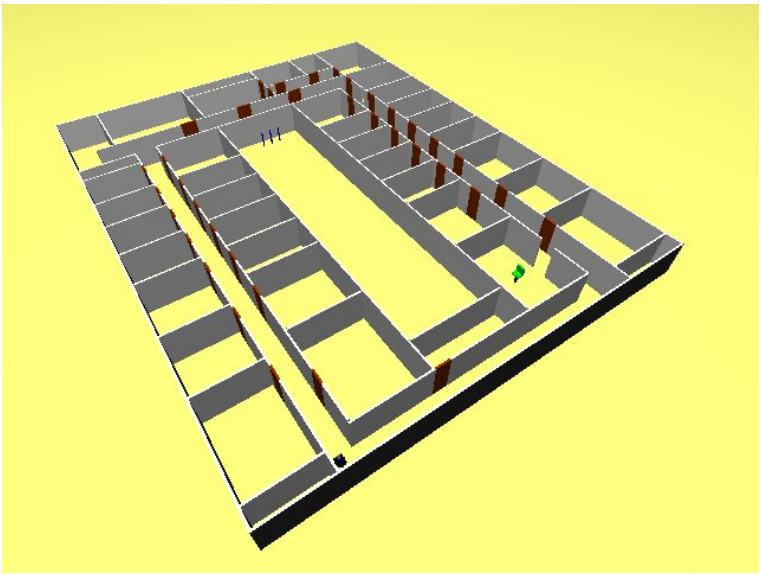

Figura 3.1: Departamental II de la *URJC*, modelado con *Gazebo*.

3.1.2. Robot Pioneer Simulado

Como dijimos anteriormente Gazebo proporciona de una buena simulación del robot Pioneer. Es capaz de simular todos los componentes que tiene el Pioneer, como son, los motores, los encoders, el láser, la cámara y el pantilt.

Tenemos un objeto que simula el robot, lo que tenemos que hacer es el fichero de configuración .world es definirnos ese objeto y añadir las propiedades que queremos que tenga. La etiqueta que usamos es <model:Pioneer2DX>, ponemos a continuación un ejemplo del objeto:

<model:Pioneer2DX>
<id>robot1< /id>
<xyz>11.9 24.1 0.200< /xyz>
<rpy>0.0 0.0 90.0< /rpy>
<model:SickLMS200>
<id>laser1< /id>
<xyz>0.0 0.0 0.00< /xyz>
<model:SonyVID30>
<id>camera1< /id>
<xyz>0.0 0. 0.2< /xyz>
<imageSize>320 240< //imageSize>
<nearClip>0.05< /nearClip>
< /model:SonyVID30>
<rayCount>180¡< /rayCount>
<rangeCount>180< /rangeCount>
< /model:SickLMS200>
< /model:Pioneer2DX>

<id>: define el identificador del objeto.
<xyz>: define la posición del objeto en el mundo. El primer <xyz> dice la posición del robot respecto el mapa, el segundo y tercero dicen las posiciones del láser y la cámara respecto al robot.
<rpy>: define la orientación del robot.
<model:SickLMS200>: define que tenemos un objeto laser llamado SickLMS200 que simula el láser real con ese mismo nombre.
<model:SonyVID30>: define que tenemos un objeto cámara llamado SonyVID30 que simula la cámara real con ese mismo nombre.
<imageSize>: define el tamaño de la imagen de la cámara.
<nearClip>: define la distancia near de la cámara.
<rayCount>: define el numero de rayos del láser.
<rangeCount>: define el ángulo de amplitud del láser.

En la figura 3.2 podemos ver el robot Pioneer simulado en la imagen a) y en la imagen b) podemos ver varias ventanas que tienen como propósito mostrar los valores de los sensores y poder utilizar los actuadores.

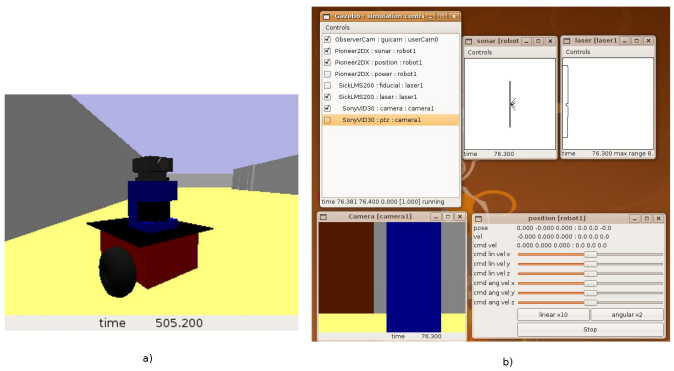

Figura 3.2: a) robot Pioneer en gazebo b) controles del Pioneer en gazebo

3.2. Plataforma Software JDEROBOT

Jde[1] (Jerarquía Dinámica de Esquemas) es una plataforma software libre con las licencias GPL y LGPL, desarrollada íntegramente en la URJC ([Cañas Plaza *et al.*, 2007]) para facilitar la programación de robots y de aplicaciones relacionadas con la visión artificial, así como de domótica. Su origen se remonta al año 1997 como fruto de una tesis doctoral ([Cañas Plaza, 2003]). Desde entonces ha ido creciendo año tras año, con nuevas funcionalidades.

Jderobot se compone de diferentes componentes, cada uno con unas funciones definidas. Estos componentes son los esquemas, que son las aplicaciones que determinan

[1] http://jde.gsyc.es

el comportamiento del robot, los drivers, que son los encargados de comunicarse con los sensores y los actuadores a bajo nivel, los servicios, que son los encargados de realizar trabajos especiales, como son el manejo de las interfaces gráficas o el próxima de imágenes y las librerías auxiliares, que dan funcionalidad de un tipo de tecnología, como son la lógica borrosa o la geometría proyectiva.

Los esquemas sirven para construir aplicaciones robóticas de modo asíncrono. Existen dos tipos de comportamiento de estos esquemas. Unos son llamados *esquemas perceptivos* porque reciben información de los sensores, la procesan y si es necesario la ponen en disposición para que la utilicen otro esquemas. Otro tipo de esquema son llamados *esquemas actuadores* y lo que hacen es a partir de la información que han recibido actúan siguiendo unas reglas.

Los drivers sirven comunicarse con componentes hardware. En jderobot usamos una serie de sensores llamados variables perceptivas y actuadores llamados variables actuadoras que son las que conocen nuestros esquemas, los drivers sirven para independizar la variable que usamos en nuestro esquema con el componente hardware, para ello estos esquemas son virtuales. En este conjunto podemos encontrar drivers para gazebo, imagefile, player, plantilt, evi, video4linux, firewire, networkclient, naobody y simulated 3D.

Los servicios nos permiten realizar trabajos especiales. Podemos crearnos un servidor TCP/IP de imágenes que funcione localmente o por internet, también podemos crear interfaces de usuario, para esto tenemos dos opciones o utilizar XForms o GTK. Estos servicios son networkserver, graphics_gtk y graphics_xforms.

Las librerías auxiliares que se han implementado la librería fuzzylib, que usa tecnología de lógica borrosa, la librería colorspaces, que sirve para transformar el espacio de color de RGB a HSV, la librería progeo, que usa tecnología de geometría proyectiva, la librería gridlib, que proporciona funciones para el uso de mallas y la librería Pioneer, que proporciona la manera de comunicarnos con el robot Pioneer.

Una aplicación de JDEROBOT se puede ver como un conjunto de esquemas que se comunican entre sí. Los esquemas pueden ser activados o desactivados a voluntad y se ejecutan de forma iterativa. El ritmo del esquema es ajustable gracias a una variable, en la cual indicamos la duración de cada ciclo. Esto nos permite tener un mayor control de la respuesta de los actuadores y sensores y poder ajustar esa respuesta como nos interese.

Otro detalle que tenemos que tener en cuenta es que como pasamos valores de variables de unos esquemas a otros podemos tener condiciones de carrera, JDEROBOT propone como solución la naturaleza iterativa del esquema, en la siguiente iteración se obtendrá el valor correcto. Si esto no nos vale podemos acudir al uso de semáforos para controlar que no existan condiciones de carrera.

Como hemos dicho anteriormente JDEROBOT es un conjunto de componentes, en

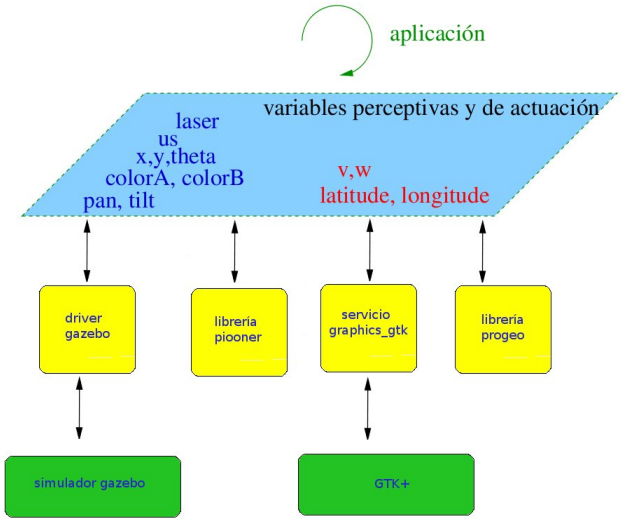

Figura 3.3: Comportamiento de la arquitectura con un esquema en ejecución.

nuestro caso hemos creado un único esquema que puede ser encuadrado en los esquemas perceptivos, como hemos tenido que hacer una simulación en gazebo hemos usado el driver de gazebo, la interfaz la hemos implementado en GTK por lo tanto hemos usado el servicio graphics_gtk y hemos usado las librerías Pioneer,gridlib y progeo.

Este proyecto se ha desarrollado sobre la versión 4.3.0 de *jdec*. Las mejoras más destacables de esta versión han sido la incorporación de nuevos drivers para distintos dispositivos y la creación de un repositorio de paquetes Debian para hacer más fácil su instalación.

A continuación se explicaran algunas de las librerías auxiliares que se han utilizado.

3.2.1. Progeo

La biblioteca Progeo es utilizada para realizar cálculos de geometría proyectiva. Estos cálculos permiten procesar la información de puntos geométricos en el espacio 3D para obtener sus proyecciones en un plano imagen determinado. Mediante estas proyecciones, los puntos 3D pueden ser visualizados en un monitor, empleando para ello algún gestor gráfico. De igual forma, los cálculos pueden ser realizados a la inversa, de manera que a partir de varias proyecciones de un punto, podamos obtener las coordenadas 3D del mismo.

La biblioteca Progeo se basa en un modelo de cámara llamado modelo PinHole el cual podemos observar en la figura 3.4 Nosotros hemos utilizado únicamente el mecanismo de retroproyección, que nos permite obtener la recta de proyección que une el centro óptico de una cámara (la del mundo virtual, en nuestro caso) con el punto 3D que representa un punto 2D de su plano imagen.

Así, hemos podido establecer un sistema muy intuitivo de cara al usuario para

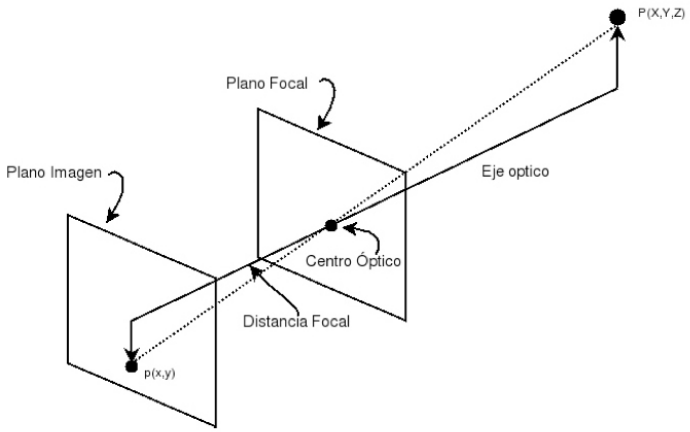

Figura 3.4: Modelo de cámara Pin-Hole.

interactuar con el mundo simulado en 3D con OpenGL, y obtener por ejemplo la imagen visual teórica percibida por el robot en esa posición, es decir, que vería el robot si estuviera en tal posición. Lo cual nos ha resultado muy útil para depurar el algoritmo de localización, además el ángulo de visión de esa imagen teórica también la hemos calculado con esta técnica, haciendo la retroproyección de un punto en una esquina de la imagen y otro punto en la otra esquina, y calculando el ángulo que forman los dos puntos. explicar las variables.

3.2.2. Gridslib

La librería Gridslib nos permite crear y manejar rejillas de ocupación. Esta biblioteca implementa diversas técnicas de construcción de mapas como la regla de Bayes, la regla Dempster-Shafer, rejillas histogramícas o rejillas borrosas. Esta biblioteca genera una rejilla cuadrada con celdas regulares de cierto tamaño. Estos tamaños son especificados en un archivo de configuración que la biblioteca se encarga de leer.

En nuestro proyecto esta biblioteca nos va servir para representar el mapa local de ocupación del escenario. Cada celda de la rejilla almacenará un valor que representar un punto del espacio. Estos valores pueden representar espacios vacíos, paredes, obstáculos.

3.3. OpenGL

La librería gráfica OpenGL es una especificación estándar que define un API multilenguaje y multiplataforma. Es capaz de realizar escenas en dos dimensiones y en tres. Se trata de crear escenas tridimensionales complejas a partir de primitivas geométricas simples. Se usa mucho en aplicaciones tipo CAD, simuladores de vuelo figura 3.5, desarrollo de videojuegos etc.

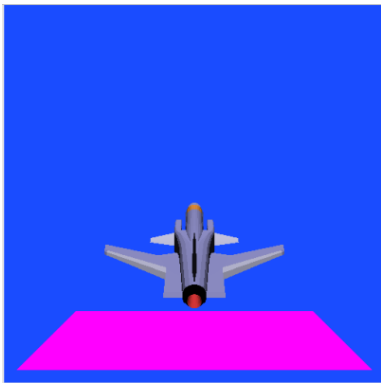

Figura 3.5: Ejemplo de un simulador de vuelo hecho en OpenGL

La librería gráfica OpenGL nos va a permitir visualizar de forma elegante y eficiente escenas 3D o las estructuras internas de los algoritmos, por lo que nos sera muy útil como método de depuración. Además, la carga computacional que teníamos con la librería progeo (que también se puede usar para renderizar escenas 3D) ahora se trasladara al procesador gráfico (GPU), ya que esta biblioteca trabaja directamente con el procesador gráfico y no con la CPU, como lo hace progeo.

Concretamente hemos utilizado esta librería para crear el entorno virtual donde realizaremos la localización del robot, en nuestro caso el entorno es el departamental, con puertas, papelera, extintores. En este entorno colocaremos las partículas y calcularemos las imágenes teóricas.

3.4. GTK+

GTK+ es un conjunto de bibliotecas y rutinas para desarrollar interfaces gráficas. Inicialmente fue creada para desarrollar aplicaciones de edición GIMP. Sin embargo, en estos momentos es una de las bibliotecas de interfaces gráficas de los sistemas GNU/Linux más usada. Hemos escogido esta biblioteca porque utiliza como entorno de escritorio por defecto Gnome, que es el que tenemos en los puestos del laboratorio.

3.4.1. GTK

La biblioteca GTK (o Gimp Toolkit) es una de las muchas que compone el conjunto GTK+ y nos permitirá crear la interfaz gráfica de usuario gracias a las funciones y objetos que ofrece. Esta escrita en lenguaje C Nos permite crear una interfaz gráfica con la que visualizar y depurar los resultados que vamos obteniendo con los distintos algoritmos.

Esta biblioteca proporciona una herramienta denominada glade con la que poder crear visualmente objetos gráficos en sistemas de ventanas, denominados widgets, como botones, menús, etiquetas, diales, barras de desplazamiento, canvas para visualización de OpenGL y muchos otros.

El mecanismo de eventos GTK es tal que asociamos señales a manejadores. El diseñador de interfaces Glade permite asociar las señales de determinado widget a sus funciones manejadoras o callbacks. Este tipo de funciones reciben siempre como parámetro una referencia al widget del que procede la señal, pudiendo reconocer la fuente del evento.

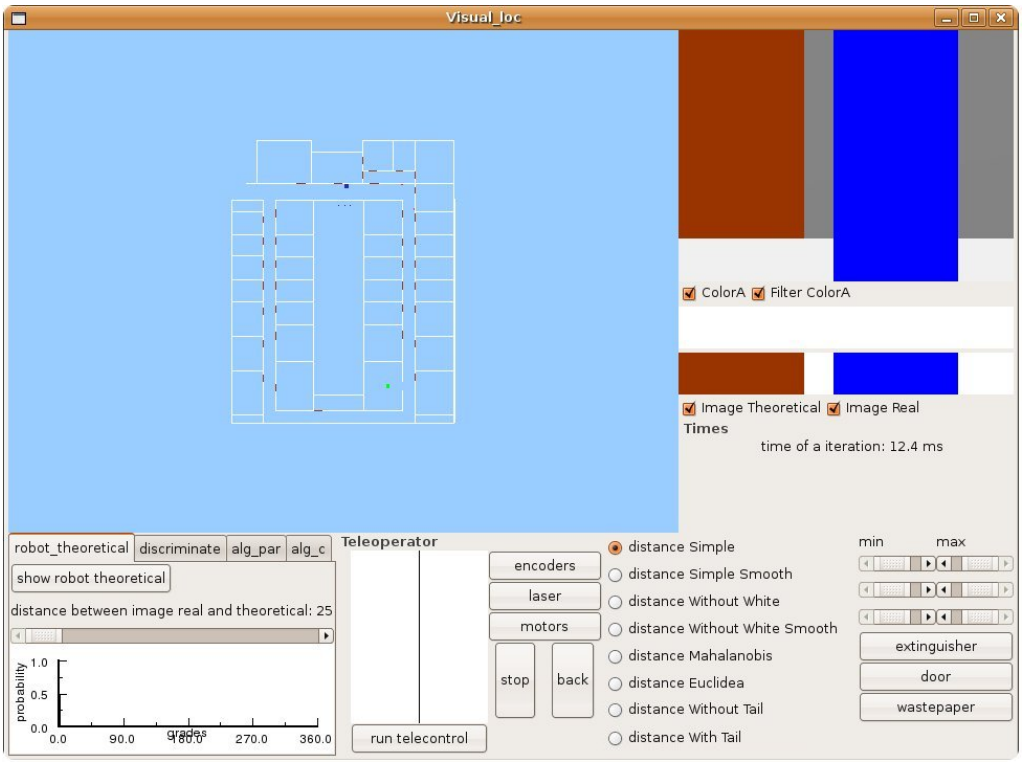

Figura 3.6: Interfaz implementada en GTK

3.4.2. GtkPlot

GtkPlot es un widget que pertenece a GTK+ igual que la biblioteca GTK, este widget se encuentra en el paquete GtkExtra y sirve para crear tanto curvas en 2D como superficies en 3D, todo ello para interfaces en GTK.

En este proyecto la hemos usado para crear la gráfica con una curva 2D que muestra la probabilidad que tiene un robot teórico en una posición (x,y) y en todo el espacio de θ.

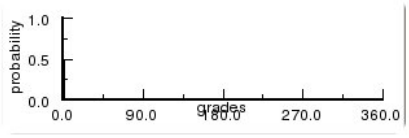

Figura 3.7: gráfica en 2D implementada en Gtkplot

Capítulo 4
Localización probabilística con filtro de particulas

Como hemos explicado en capítulos anteriores la localización es uno de los problemas principales en la robótica móvil, y existen muchas formas de abordar este problema.

En este capítulo vamos a explicar el filtro de partículas que es la técnica que se ha desarrollado para solucionar el problema de autolocalización. Mostraremos sus fundamentos teóricos, también enseñaremos el diseño general y del algoritmo usado, expondremos el modelo de observación usado, el modelo de movimiento y el remuestreo, por separado y en conjunto, también explicaremos otra técnica de localización probabilística, llamada mallas de probabilidad que hemos usado para comprobar que nuestros modelos de observación y movimiento convergen a una posición correcta.

4.1. Fundamentos del filtro de partículas

Dependiendo del tipo de robot y escenario que tengamos tendremos que optar por un tipo de localización. En nuestro caso tenemos como premisas que no sabemos la posición inicial del robot, tenemos una cámara (modelo de observación) y tenemos unos *encoders* que nos dicen cuanto nos hemos movido (modelo de movimiento). Con todo esto hemos decidido usar un algoritmo probabilístico ya que en un primer momento tratara todo el espacio como equiprobable y han dado buenos resultados en otros trabajos como son el de Alberto Lopez [López, 2005] o el de Angel Cortés [Cortés, 2007] entre otros.

La idea fundamental de la localización probabilística es la estimación y representación de las densidades de probabilidad asociadas a las posibles posiciones del robot en el mundo real. En nuestro caso la posible posición del robot se encuentra en el espacio (x,y,Θ), que define una posición bidimensional en el mundo plano y una orientación. Se le asocia una probabilidad a esa posición que representa la similitud con la posición real del robot.

Para conseguir la probabilidad en una posición se cuenta con un mapa visual, este mapa visual se define en metros cuadrados y representa el departamental II. Colocamos el robot real en una posición (x,y,Θ) después ponemos por todo el mapa partículas en

diferentes posiciones elegidas aleatoriamente y comparamos la imagen calculada de cada una de las partículas con la imagen proporcionada por la cámara. Si estamos delante de una puerta todas las partículas que miren a una puerta tendrán alta probabilidad y las que no tendrán baja probabilidad. Si el robot se mueve, los *encoders* nos dirán cuanto se ha movido y tendremos que mover las partículas esa distancia.

El algoritmo probabilístico de filtro de partículas contiene una población de partículas y se divide en tres partes que son independientes y se ejecutan iterativamente:

- Modelo de movimiento: Obtenemos el desplazamiento y/o giro realizado por el robot desde la última vez que se actualizo este modelo y añadimos este desplazamiento y/o giro a cada una de las partículas.
- Modelo de observación: Obtenemos la información de la observación real. Calculamos las nuevas densidades de probabilidad para cada una de las partículas que tenemos.
- Remuestreo: Generamos la nueva población de partículas para la siguiente iteración. Teniendo mayor posibilidades de pertenece a la siguiente población a las partículas con mayor probabilidad.

A medida que va avanzando la ejecución y el robot se va moviendo, las partículas van convergiendo hacia una localización, que es donde se encuentra el robot real. Al final tenemos una nube de partículas encima del robot real, diciendo que el robot real esta situado en esa zona del mapa visual.

4.2. Diseño general

Hemos realizado el algoritmo programado y diseñado los componentes software dentro de la plataforma JDEROBOT, para ello lo hemos estructurado como un esquema llamado visual_loc. Este esquema se divide en dos partes, que son dos threads diferenciados en ejecución.
El primero es el que ejecute el algoritmo, inicializa el mapa, obtiene la imagen real, obtiene las imágenes teóricas de cada partícula, calcula la probabilidad y la almacena, realiza el movimiento de las partículas por el mapa y genera la siguiente población de partículas.
El segundo se encarga de mostrar la interfaz gráfica, manejar los eventos de esta e ir refrescando lo visualizado cada cierto tiempo, esta GUI se encarga de depurar el algoritmo de localización y mostrar sus resultados.

Como se puede observar en la figura 4.1 tenemos como entrada el mapa visual en forma de fichero de imagen, la posición y orientación del robot que se recibe de los encoders y la imagen de la cámara. Como salida tenemos la posición que estimamos que es donde se encuentra el robot después de ejecutar el algoritmo de localización.
Estructuralmente tenemos que en el nivel más bajo esta el robot real pioneer o el simulador gazebo, proporcionan los valores de los sensores que se usan para la

localización. En el segundo nivel tenemos la plataforma JDEROBOT que recoge el valor de los sensores y los convierte en variables que entienden todos los esquemas de JDEROBOT. En el tercer y último nivel esta nuestro esquema visual_loc, recibe las variables de JDEROBOT que necesita, el mapa visual en forma de fichero de imagen, dentro de nuestro esquema hay dos estructuras que se comparten entre los dos threads y son una estructura con todas las partículas que guarda su posición y su probabilidad y otra estructura con el mapa visual pero guardado en memoria para agilizar su uso. Al final se obtiene una localización probabilística del donde se encuentra el robot.

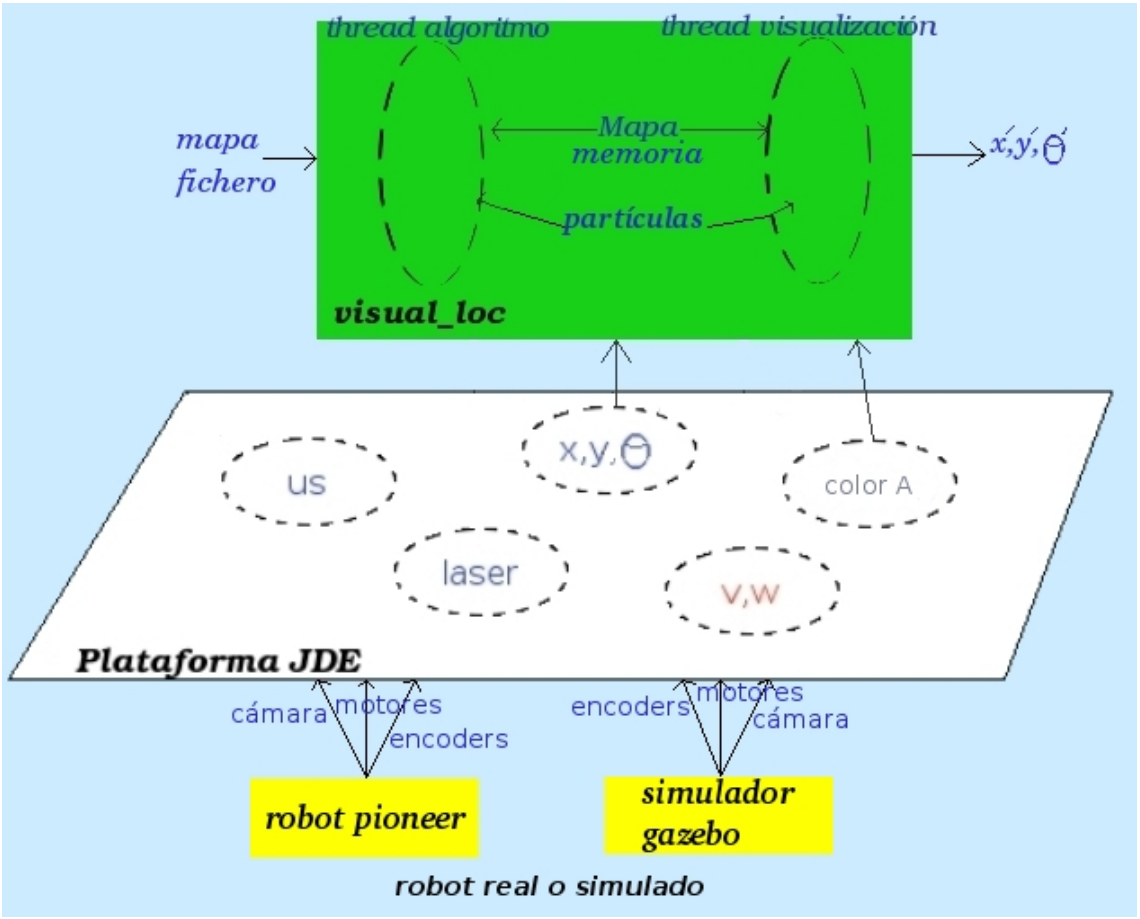

Figura 4.1: Diseño del algoritmo de localización probabilística con partículas

El algoritmo que se encarga de calcular la localización del robot con filtro de partículas se divide en:

1. Leer el archivo del mapa de balizas y guardarlo en una estructura.
2. Creación e inicialización de las partículas.
3. Localización.
   - Modelo de movimiento.
   - Modelo de observación

- Remuestreo

La figura 4.2 muestra el flujo de control de este esquema.

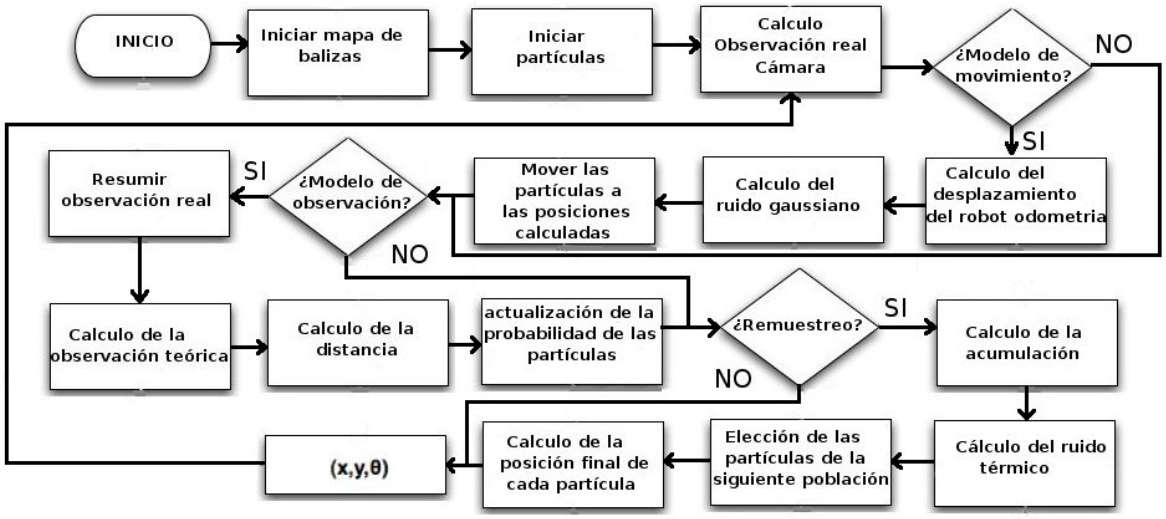

Figura 4.2: Estructura del algoritmo de localización probabilística con partículas.

Ahora vamos a explicar las dos estructuras que tenemos dentro del esquema visual_loc.

El mapa visual es un mapa de color que nos va a servir para saber donde se encuentra el robot en el mundo y nos da una idea de lo que tiene a su alrededor. El mapa nos entra en el algoritmo como un fichero de imagen en formato ppm, hemos elegido este formato porque cada color se representa en el fichero como un número y por lo tanto es muy fácil de leer y si se quiere modificar se trata como una imagen con píxeles. Nosotros en nuestra imagen tenemos puertas, papelera, extintores, paredes, y una silla, cada cosa de un color y por tanto cada uno con un número distinto. El resto de números se tratan como color transparente (no existe ningún objeto).

Para guardar el mapa visual internamente en memoria hemos usado una estructura que es una rejilla de la librería LibGrid explicada en capítulo 3.2.2. El tamaño del rejilla es de 30000x23500 mm, el tamaño de la imagen ppm es de 300x235 píxeles, así que el tamaño de cada celda de la rejillas es de 100 mm y en cada celdilla se guardara el color de un píxel.

El motivo porque hemos decidido guardar el mapa visual en memoria es porque se usara muchísimo para calcular la imagen teórica como veremos en apartados posteriores, por este motivo nos interesa que sea ágil y la mejor forma es tenerlo guardado en memoria.

Para visualizar el mapa visual hemos utilizado OpenGL, lo que hacemos es ir leyendo la estructura que antes hemos rellenado, calculamos la posición (x,y) del píxel

y lo pintamos con el valor que nos da el estado. Para añadirle dinamismo tenemos una cámara de openGL colocada encima del mapa, esta cámara puede moverse con los botones del ratón, figura 4.3.

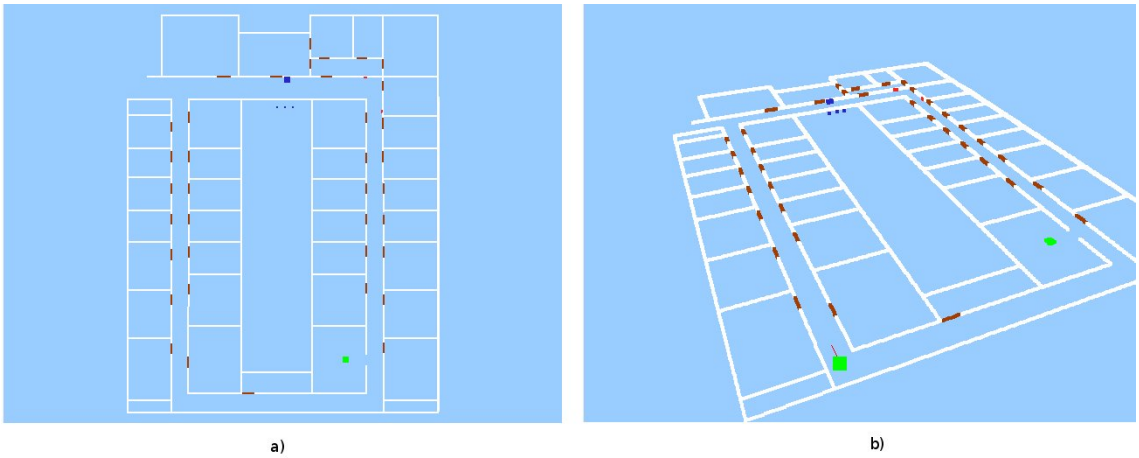

Figura 4.3: a) mapa visual en OpenGL, b) mapa visual en openGL con otra perspectiva.

Las partículas son la parte más importante de este algoritmo, ya que son las encargadas de conseguir la localización. La idea es que cada partícula representa un robot colocado en nuestro mapa visual. Tenemos que saber cuál es la posición de esa partícula, para ello nuestra partícula tiene una posición x y una posición y, además también tenemos que saber cual es la orientación que tiene para ello tenemos un θ. Una vez que sabemos dónde esta la partícula tenemos que saber cuál es la probabilidad de que se encuentre cerca de la posición real del robot, para ello tenemos una valor con esa probabilidad.

En código esto se representa mediante una estructura que guarda la posición x, y, θ y la probabilidad de que se encuentre en la posición donde esta el robot real.

Con una sola partícula no hacemos nada así que tenemos un conjunto de ellas todas ellas tienen los mismos componentes pero diferentes valores así que se ha usado un array para guardar este conjunto.

4.3. Modelo de Movimiento

Hasta ahora hemos explicado en que consisten los fundamentos del filtro de particulas y el diseño general del algoritmo, ahora vamos a mostrar en detalle cada una de las partes en las que se divide el algoritmo del filtro de partículas. Primero hablaremos del modelo de movimiento, en el siguiente capitulo explicaremos el modelo de observación y para finalizar detallaremos el remuestreo de partículas.

El modelo depende de las lecturas sensoriales y no de las órdenes que se le envían a los motores. Esto significa que puedes decirle a los motores que se muevan diez metros

y cuando llevas cinco metros pedirle que calcule el modelo de movimiento, aunque el robot se va mover hasta los diez metros las partículas solo se moverán hasta los cinco metros actuales, que es lo que hemos leído hasta ahora, la siguiente vez que pidamos una lectura del modelo de movimiento se actualizaran los cinco metros restantes.

La incorporación de movimiento consiste en calcular el desplazamiento que se ha realizado desde la última vez que preguntamos a los sensores de odometría hasta ahora, y después actualizar cada una de las partículas con ese desplazamiento relativo realizado.
Existen dos tipos de desplazamiento, el desplazamiento lineal que es lo que se mueve el robot en línea recta, y el desplazamiento angular que es lo que gira el robot. La incorporación del movimiento en las partículas es directa ya que cada partícula se considera como si fuera el robot, así que se actualizarán sumándoles el desplazamiento lineal relativo y el desplazamiento angular relativo a cada una. Como usamos encoders para saber cuanto nos hemos desplazado y estos sensores no son cien por cien correctos, hemos añadido un ruido tanto lineal como angular el cual sumamos al desplazamiento realizado, este ruido lo hemos modelado con la función gaussiana tomando como valores limites cero y un valor proporcional al desplazamiento .

Para calcular el desplazamiento tenemos que realizar las siguientes ecuaciones:

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

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

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

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

$$Ruido_{lineal} = funcion\_gauss(Mov_{lineal} * \%_{Ruidolineal}, 0) \tag{4.5}$$

$$Ruido_{angular} = funcion\_gauss(Mov_{angular} * \%_{Ruidoangular}, 0) \tag{4.6}$$

Como se puede ver en las ecuaciones, nuestro objetivo es saber el incremento de la posición del robot tanto lineal como angular desde la ultima vez que se leyó el modelo de movimiento y esta vez. Lo primero que calculamos es el movimiento lineal y el angular, para calcular el movimiento lineal lo que se hace es calcular la distancia lineal que hay desde la ultima posición del robot leída y la que hay ahora, para calcular el movimiento angular restamos el ángulo que tenemos ahora del que teníamos en la anterior lectura del modelo de movimiento. El segundo paso es calcular el ruido lineal y el ruido angular, para esto hemos utilizado una función gaussiana, esta función gaussiana calcula un valor aleatorio entre dos valores aplicándole la función gaussiana, los valores entre los que esta definida la función gaussiana son cero y un porcentaje del movimiento tanto lineal como angular que nos hemos movido, eso quiere decir que, si nos hemos movido seis mil milímetros y le aplicamos un 10 % el valor que pasamos a la función gaussiana es de seiscientos, en este proyectos se han elegido como porcentaje para el ruido lineal el diez por ciento para el ruido lineal y para el ruido angular la misma cantidad. Lo ultimo que se calcula son los incrementos lineal y angular y para ello sumamos los movimientos lineal y angular con los ruidos lineal y angular.

En la figura 4.4 podemos ver dos partículas a las cuales les aplicamos una lectura del modelo de movimiento. Las partículas en rojo se encuentran en la posición anterior a aplicar el modelo de movimiento, las partículas en azul claro son las partículas sin aplicarles el ruido gaussiano y las partículas en azul oscuro son las partículas en su posición final.

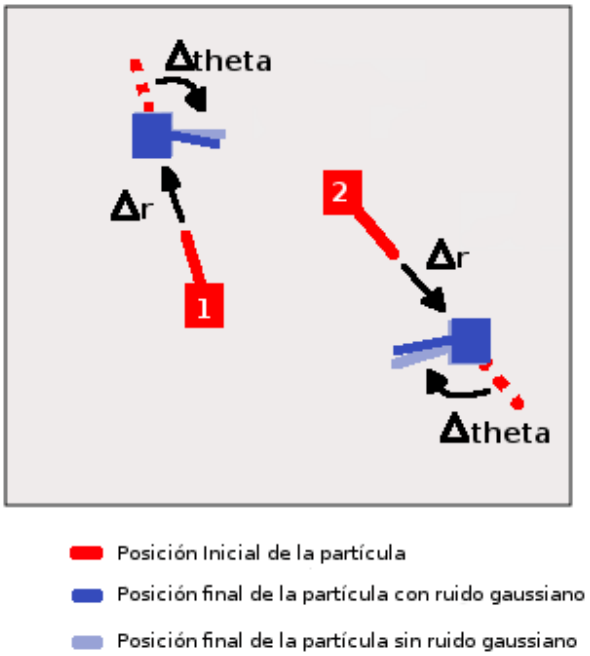

Figura 4.4: Ejemplo del movimiento de dos partículas

Existe un detalle que hay que tener en cuenta, por la forma en la que se ha hecho el algoritmo del calculo del modelo de movimiento, tenemos que realizar el movimiento del robot real respetando el modelo de movimiento, el primer movimiento que se le aplica a cada una de las partículas es el incremento lineal y después el incremento angular, por tanto si queremos que las partículas represente el movimiento que ha realizado el robot real, primero tenemos que mover el robot real linealmente y después girarlo, si no lo hacemos así cuando se muevan las partículas no lo harán representado el movimiento del robot real. Esto afecta cuando la distancia que se ha movido el robot es grande, si el intervalo entre un modelo y otro es muy corto este problema no afecta demasiado.

Como últimos detalles del modelo de movimiento podemos decir que es independiente del modelo de observación y por tanto se pueden realizar todos las lecturas del modelo de movimiento que queramos antes de realizar una lectura del modelo de observación. Se coloca como primer paso en la localización con el filtro de partículas, porque ante de realizar una observación tenemos que estar seguros que las partículas están bien situadas en el mapa visual y si hacemos la observación justo

después de haber actualizado la posición de las partículas, estaremos seguros de que lo hemos hecho bien.
El modelo de movimiento se incorpora de forma automática. Como hemos dicho antes realizamos el modelo de observación justo después del modelo de movimiento, nos interesa que las observaciones sean independientes, esto quiere decir que dos observaciones contiguas no se sean la misma, por lo tanto antes de realizar una observación el robot real se tiene que mover una cierta distancia para que las observaciones sean distintas. Hemos decidido realizarlo con umbrales, tenemos dos umbrales, uno para la distancia lineal y otro para la distancia angular, el robot se ira moviendo y nosotros iremos recogiendo la información que nos aportan los sensores en unas variables, cuando se llegue a uno de los dos umbrales se realizara un modelo de movimiento, se reiniciaran las variables que han recogido la información de los sensores y realizara seguidamente el modelo de observación.
El motivo por el cual se ha decidido hacerlo así, es porque realizar observaciones de imágenes iguales no aporta información nueva y además lleva consigo un gasto computacional innecesario y sabemos que las imágenes serán diferentes cuando el robot se haya movido.

4.4. Modelo de Observación Visual

Un modelo de observación visual en algoritmos probabilísticos consiste en sacar información de la cámara, sacar información del mapa visual que hayamos construido, comparar estas dos informaciones y calcular una probabilidad de similitud. Cuanto más se parezcan las dos informaciones mayor será la probabilidad y cuanto menos se parezcan menor será la probabilidad.

En nuestro caso, primero sacamos la imagen que nos proporcione la cámara del robot, que llamamos imagen real, y la tratamos convenientemente para que pueda ser comparada. Segundo, para cada partícula que tenemos en el mapa visual sacamos una imagen, que llamamos imagen teórica, y la tratamos para que pueda ser comparada. Tercero, comparamos las dos imágenes sacadas para saber cuanto se parecen y damos el resultado como una probabilidad, esto lo llamamos calculo de la distancia entre las imágenes.

4.4.1. Obtención de la imagen real resumida

La cámara nos proporciona una imagen cruda de 320x240 píxeles en RGB que la plataforma JDEROBOT nos devuelve en forma de array. Una vez que tenemos esta imagen tenemos que aplicarle un filtro para sacar la información relevante, el filtro que hemos elegido es un filtro de color en HSV. El motivo por el que hemos elegido un filtro de color HSV es porque aunque la imagen es RGB el filtro de color HSV ([Baena, 2003], [Barrera *et al.*, 2005]) es más robusto y podemos obviar o no la iluminación de la escena ya que separa esta de la información del color. Para aplicar este filtro hemos utilizado

rango máximo y mínimo (Hmax, Hmin, Smax, Smin, Imax y Imin) que se define antes de la ejecución aunque puede ser modificado durante la ejecución, calculamos el H S e I de cada píxel y comparamos que se encuentra entre el rango antes descrito, para calcular los valores H S e I se utilizan las siguientes formulas:

$$H = cos^{-1}\left(\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right) \tag{4.7}$$

$$S = 1 - \frac{3}{(R+G+B)} min\,(R,G,B) \tag{4.8}$$

$$I = \frac{1}{3}(R+G+B) \tag{4.9}$$

Una vez que tenemos filtrado todos los píxeles obtenemos la imagen real filtrada figura 4.5 y la almacenamos en una estructura para que sea tratada. Como se puede ver en la imagen el filtro es perfecto, el motivo es que estamos trabajando con un simulador y los colores son puros, el filtro se ha pensado más para cuando haya que realizar el tratamiento de la imagen real con el robot pioneer real.

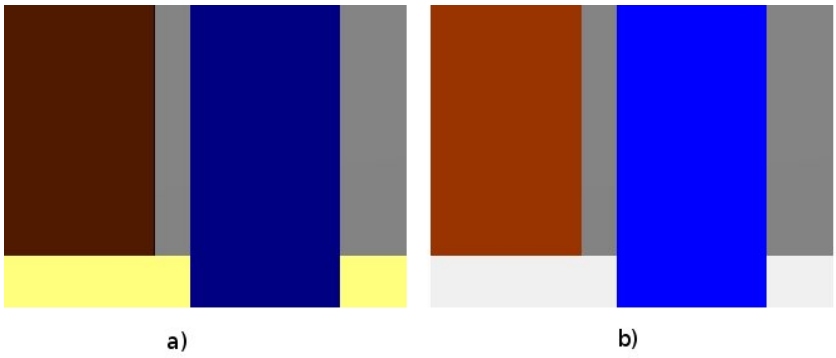

Figura 4.5: a) imagen real proporcionada por la cámara b) la misma imagen filtrada

Lo siguiente que tenemos que hacer es coger la imagen real filtrada que es de 320x240 y resumirla en una imagen de 320x1. Para ello vamos leyendo columna a columna, nos quedamos con el número de píxeles que hay de un mismo color en cada columna y decidimos si son suficientes para representar un objeto coloreado. Esta información se almacena en una imagen de 320x1. El motivo por el que se ha decido reducir la imagen es porque tratar una imagen de 320x1 es más ágil y menos costoso computacionalmente que tratar una imagen de 320x240 y con la de 320x1 nos quedamos los datos relevantes de la imagen.

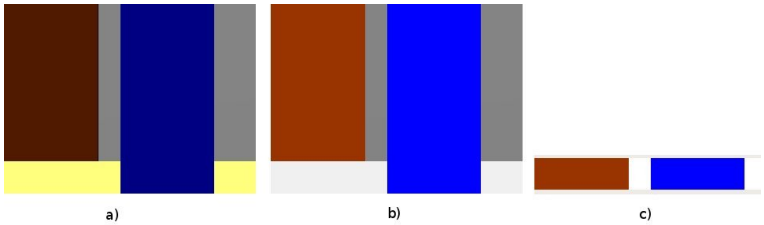

Figura 4.6: a) imagen real proporcionada por la cámara b) la misma imagen filtrada c) imagen resumida

4.4.2. Obtención de la imagen teórica

Una vez que ya hemos obtenido la imagen real resumida que vamos a comparar, el siguiente paso que es obtener la imagen teórica.
Lo primero que calculamos es el ángulo de visión, para esto hemos usado *progeo*. Como se explicó en el capítulo 3.2.1 para definir una cámara en progeo tenemos que dar valor a dos tipos de parámetros, los parámetros intrínsecos y los parámetros extrínsecos. Para crear esta cámara los parámetros extrínsecos son la posición de la cámara, para el calculo del ángulo puede estar en cualquier punto, nosotros la hemos colocado en la posición (0,0,0), y el foco de atención(foa, punto al que mira nuestra cámara), que puede estar en cualquier sitio excepto en el mismo lugar que la posición de la cámara, nosotros lo hemos colocado en la posición (10,0,0), otro atributo extrínseco es el *roll* (dice si la cámara esta girada respecto a los ejes x e y), en nuestro caso la cámara está sin girar. Los parámetros intrínsecos de la cámara son u0 y v0 (definen centro óptico de la imagen), en este caso es 160 y 120 que es justo el centro de la imagen que proporciona gazebo que es de 320x240, el fovy (ángulo de visión) es el ángulo de visión que tiene la cámara de gazebo que no es lo mismo que el ángulo de visión que estamos calculando,en nuestro caso es 57, el ultimo valor es el fdist (distancia focal) es una función que en nuestro caso es $fdist = \frac{altura\_imagen}{2*\tan(\frac{fovy}{2})}$. Con esto hemos calibrado la cámara de gazebo en progeo, ahora cogemos el punto que se encuentra en medio de la imagen y que esta más a la derecha de la imagen (0,120) y el punto que se encuentra en medio de la imagen y que esta más a izquierda (320,120). Realizamos una retroproyección de esos dos puntos y obtenemos esos dos puntos en 3D. El ultimo paso es obtener los vectores que forman cada punto con la posición de la cámara y calcular el ángulo que forman esos dos vectores con el producto escalar. Esto se puede ver en la figura 4.7

El segundo paso para calcular la imagen teórica es obtener la imagen que ve la partícula con el ángulo de visión calculado. Tenemos el mapa visual y el ángulo de visión y la posición donde se encuentra la partícula. Ponemos el vértice del ángulo que vamos a formar en la posición en la que se encuentra la partícula, decimos que la orientación de la partícula θ,es el centro del ángulo de visión y así tenemos que las lineas de visión se dividen igualmente por la derecha y por la izquierda. Proyectamos por cada grado del ángulo de visión una línea de visión de x mm,y recorremos esa línea en intervalos de cierta distancia, si obtenemos un color que puede ser el color

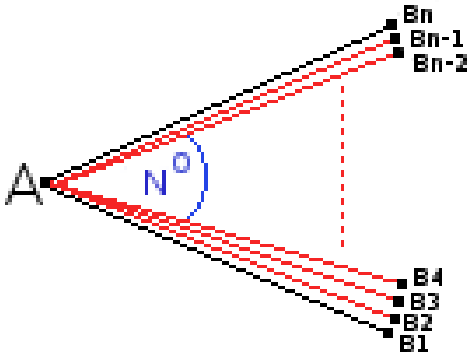

Figura 4.7:

de la pared, la papelera, los extintores, las puertas o la silla dejamos de mirar en esa línea de visión y decimos que esa línea tiene ese color, si no obtenemos ningún color de los mencionados, decimos que es transparente y seguimos mirando en esa linea, al final obtenemos que cada línea representa un color o si no se ha encontrado ninguno el color es blanco. Se puede ver en la figura 4.8 que el vértice del ángulo de visión es justo donde esta la partícula (cuadrado rojo) el centro del ángulo es justo donde mira la partículas y que las lineas cuando llegan a ver un objeto paran su búsqueda.

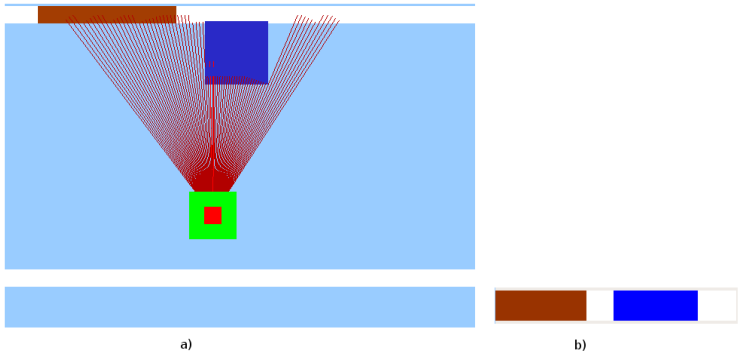

Figura 4.8: a) ángulo de visión de la imagen teórica b) imagen teórica

Una vez que tenemos una estructura con los colores que se ven con la imagen teórica tenemos que obtener una imagen teórica del mismo tamaño que la imagen real resumida para poder compararlas. La imagen real resumida es de 320x1 y como la estructura con colores que se ven con el ángulo de visión que obtenemos, es menor, lo que hacemos es clonar la estructura para que llegue a ser del tamaño de 320x1. Por ejemplo tenemos que el ángulo de visión de 64, la estructura que obtenemos tiene un tamaño de 64x1 si dividimos 320 entre 64 nos da 5 eso quiere decir que cada linea de visión de la estructura representa 5 píxeles de la imagen teórica, lo que hacemos es crearnos una imagen de 320x1 y guardar el color de cada linea de visión cinco veces consecutivas. Este es un ejemplo ideal ya que no tenemos decimales,normalmente el ángulo de visión es un valor que no es divisible por 320 por lo que lo que hemos hecho es truncar al valor entero que nos da la división e ir sumando lo que sobra a la siguiente división, por ejemplo el

ángulo de visión es de 71 la división de 320 entre 71 es 4.507, haciendo lo que hemos dicho antes, habrá algunas linea de visión que serán 4 píxeles y otras serán 5 píxeles pero al final tendremos una imagen de 320x1.
El motivo por el que se ha elegido que cada linea sea un grado y después se tenga que clonar las lineas para poder llegar a formar una imagen de 320x1 es por eficiencia. Tenemos que realizar la operación antes descrita para cada partícula si tenemos 320 lineas tardaremos más en obtener la imagen teórica, nos interesa que sea buena la imagen teórica pero que no se tarde demasiado en calcular, clonando las lineas de visión obtenemos una buena imagen teórica y es menos costosa computacionalmente.

4.4.3. Calculo de la distancia y probabilidad

Ahora que tenemos la imagen real resumida y la imagen teórica las tenemos que comparar para saber cuánto se parecen, esto lo hemos llamado calculo de la distancia entre las imágenes. Para realizar esta parte hemos usado varias técnicas de calculo de distancia las cuales explicaremos a continuación. Las hemos divido en dos grupos en unas la unidad básica es el píxel y lo que comparamos son si los píxeles se parecen, y en la otra la unidad básica son objetos y lo que comparamos son el tamaño y el color de los distintos objetos que tenemos tanto en la imagen real resumida como en la imagen teórica.

La figura 4.9 mostramos el robot real en frente de la papelera y una puerta, el robot teórico colocado un poco a izquierda, la imagen real, la imagen real resumida y la imagen teórica. A continuación mostramos la diferentes funciones de distancia que hemos implementado explicándolas y diremos cual es la probabilidad que se da en la situación de la figura.

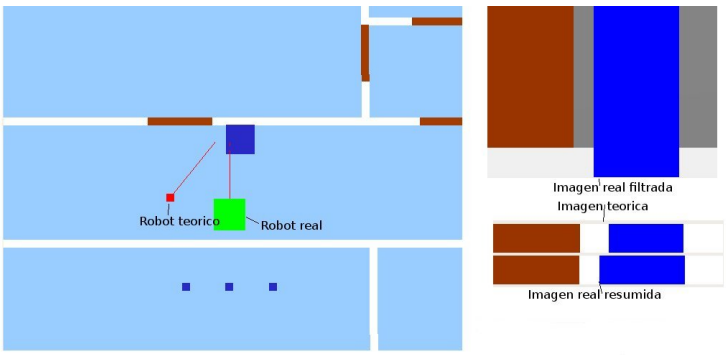

Figura 4.9: Un ejemplo de calculo de la función distancia.

Las funciones de distancia basadas en píxeles tienen todas como similitud que usan el color del píxel para saber lo que se parecen la imagen real resumida y la imagen teórica. Las técnicas usadas son:

1. Distancia simple: Como su propio nombre dice es la más sencilla de todas, y consiste en ir comparando píxel a píxel si son exactos o no lo son. Al final si las dos imágenes son exactamente iguales su parecido sera del cien por cien. Aplicamos la siguiente formula porque al final tenemos que obtener un valor entre 0 y 1

$$probabilidad_{dist\_simple} = \frac{numero\_pixeles\_iguales}{320} \tag{4.10}$$

   La probabilidad que obtenemos entre la imagen teórica y la imagen real resumida es del 95 % el motivo es porque las imágenes son parecidas.

2. Distancia simple con suavizado: Consiste en lo mismo que la anterior pero aplicamos un suavizado solo a la imagen teórica, la imagen real resumida no se suaviza. Este suavizado consiste en coger un píxel, los dos de su izquierda y los dos de su derecha y aplicar la formula:

$$suavi_{pixel\_i} = \frac{(pixel_{i-2} * 1) + (pixel_{i-1} * 2) + (pixel_i * 3) + (pixel_{i+1} * 2) + (pixel_{i+2} * 1)}{9} \tag{4.11}$$

   Conseguimos que los cambios de color no sean tan bruscos, ya que contamos también con los de su alrededor.
   La función de probabilidad es:

$$probabilidad_{dist\_simple\_suavizada} = \frac{numero\_pixeles\_iguales_{imagen\_teorica\_suavizada}}{320} \tag{4.12}$$

   La probabilidad que obtenemos entre la imagen teórica y la imagen real resumida es del 91 %.

3. Distancia sin blancos: Consiste en ir comparando píxel a píxel la imagen teórica con la imagen real, pero sólo se comparan si el píxel de la imagen teórica no es blanco.La formula que se aplica es:

$$probabilidad_{dist\_sin\_blancos} = \frac{numero\_pixeles\_color\_iguales}{numero\_pixeles\_color_{imagen\_teorica}} \tag{4.13}$$

   La probabilidad que obtenemos entre la imagen teórica y la imagen real resumida es del 99 %.

4. Distancia sin blancos con suavizado: Consiste en lo mismo que la anterior pero aplicamos a la imagen teórica mismo suavizado que aplicado en la distancia simple.
   La función de probabilidad es:

$$probabilidad_{dist\_sin\_blancos\_suavizada} = \frac{numero\_pixeles\_color\_iguales_{imagen\_teorica\_suavizada}}{numero\_pixeles\_color_{imagen\_teorica\_suavizada}} \tag{4.14}$$

   La probabilidad que obtenemos entre la imagen teórica y la imagen real resumida es del 99 %.

5. Distancia de Mahalanobis: Consiste en aplicar la función de distancia de Mahalanobis. Esta función determina la similitud entre dos variables aleatorias multidimensionales. Para el caso que nos ocupa lo que hacemos es:

$$dMa_{pixel} = \sqrt{\left(\frac{R_{real} - R_{teorica}}{\sigma_1}\right)^2 + \left(\frac{G_{real} - G_{teorica}}{\sigma_2}\right)^2 + \left(\frac{B_{real} - B_{teorica}}{\sigma_3}\right)^2} \quad (4.15)$$

Donde R,G,B son los valores de cada color del píxel y σ_i es la varianza de la componente i de los datos por los que se divide.
Debemos normalizar para que el valor sea entre 0 y 1, dividimos la distancia que nos ha dado entre la máxima posible.

$$probabilidad\_dMa = \frac{\sum_{i=0}^{320} dMa_{pixel}}{(\sqrt{3} * 4) * 320} \quad (4.16)$$

La probabilidad que obtenemos entre la imagen teórica y la imagen real resumida es del 97 %.

6. Distancia euclídea: Consiste en aplicar la función de distancia euclídea. Esta función determina la similitud aplicando la distancia ordinaria en un espacio euclidiano. La formula que hemos aplicado es la siguiente:

$$dEu_{pixel} = \sqrt{(R_{real} - R_{teorica})^2 + (G_{real} - G_{teorica})^2 + (B_{real} - B_{teorica})^2} \quad (4.17)$$

Debemos normalizar para que el valor sea entre 0 y 1, dividimos la distancia que nos ha dado entre la máxima posible.

$$probabilidad\_de = \frac{\sum_{i=0}^{320} dEu_{pixel}}{(\sqrt{3} * 255) * 320} \quad (4.18)$$

La probabilidad que obtenemos entre la imagen teórica y la imagen real resumida es del 99 %.

El otro tipo de funciones de distancia se basa en objetos, en vez de comparar la imagen teórica y la imagen real por píxeles lo que hacemos es compararla por objetos. Un objeto es un conjunto de píxeles contiguos que tienen el mismo color, en cuanto se encuentra un píxel con un color que no es el mismo que el del anterior decimos que el píxel anterior es el final de un objeto y el píxel que estamos mirando ahora es el principio de otro objeto. Este tipo de funciones de distancia es más laboriosa ya que hay que calcular los objetos. Hemos implementado dos técnicas que explicamos a continuación:

1. Distancia con cola: Esta técnica es la que implementó Alberto López en su proyecto [López, 2005] para el filtro de partículas. Lo primero que se calcula son los objetos que contiene la imagen real resumida. El segundo paso es comparar los objetos de la imagen real resumida con la imagen teórica, sabemos cual es el primer píxel de un objeto y el ultimo, así que vamos mirando si los píxeles del objeto de la imagen real resumida tienen el mismo color que los píxeles de la imagen teórica, esto quiere decir que por ejemplo un objeto empieza en el píxel

20 y termina en el 40 de la imagen real resumida, lo que hacemos es ir recorriendo del píxel 20 al 40 de la imagen teórica y guardamos el número de píxeles que son iguales.Tenemos que tener en cuenta las colas no sólo miramos del 20 al 40 si no del 15 al 45 de la imagen teórica. Otra cosa que hay tener en cuenta es que hay objetos que tienen más peso que otros. El ultimo paso es saber cual es la distancia para ello tenemos los casos especiales de los blancos y el caso normal. Los casos especiales son cuando se solapan los objetos de color un cien por cien pero se ve que son del mismo tamaño, lo que se hace es penalizar los objetos blanco para que no den una probabilidad del cien por cien. Para el caso normal se aplicar la formula 4.19.

$$p_{x,y,\Theta/imagen} = \frac{1}{N}\sum_{i=0}^{N-1} p_i \cdot Peso_i \tag{4.19}$$

La probabilidad que obtenemos entre la imagen teórica y la imagen real resumida es del 93 %.

2. Distancia sin cola: El calculo de la distancia sin cola se puede dividir en tres fases.
   En la primera fase lo que hacemos es obtener los objetos que tienen tanto la imagen teórica como la imagen real resumida. Solo nos quedamos con los objetos que no son blancos, ya que los blancos no dan ninguna información. De cada objeto nos quedaremos como información con el píxel en el que empieza el objeto, el píxel en el que termina el objeto y el color del objeto, figura 4.10.

Figura 4.10: Objetos en los que se compone una imagen teórica o imagen real resumida.

En segunda fase calculamos cuanto es de igual y de diferente la imagen teórica y la imagen real resumida. En esta función de distancia se han diferenciado tres casos especiales en esta fase donde obtienen.
El primer caso es si tenemos que la imagen teórica no tiene ningún objeto y la imagen real resumida tampoco, en este caso para el algoritmo de partículas diremos que la partícula tiene una probabilidad del 100 %.
El segundo caso es cuando tenemos que la imagen real resumida no tiene objetos y la imagen teórica si, en este caso para el algoritmo de partículas decimos que la probabilidad es del 0 %.
El tercer caso especial y ultimo es cuando tenemos que la imagen teórica no tiene objetos y la imagen teórica si los tiene, en este caso para el algoritmo de

partículas decimos que la probabilidad es del 0 %.
Si no estamos en ninguno de los tres casos especiales recorremos la imagen teórica y la imagen real resumida de dos formas, de izquierda a derecha y de derecha a izquierda. Comparamos que el objeto teórico tiene el mismo color que el objeto real resumido, si es así calculamos el número de píxeles que son iguales y el número de píxeles que son diferentes. Si no son iguales y estamos comparando el primer objeto real pasamos al siguiente objeto teórico, si no, en el siguiente recorrido comparamos el mismo objeto teórico con el primer objeto real. Puede ser que cuando terminamos de recorrer la imagen teórica queden objetos sin comparar estos objetos tendrán como píxeles iguales cero y píxeles diferentes todos. El motivo por el que recorremos los objetos de izquierda a derecha y de derecha a izquierda es porque en un principio no sabemos si la partícula esta a la derecha del robot, en el centro o a la izquierda, si hacemos los dos recorrido al final habrá uno que dará mayor probabilidad, será con el que nos quedemos y tendremos una mejor aproximación.

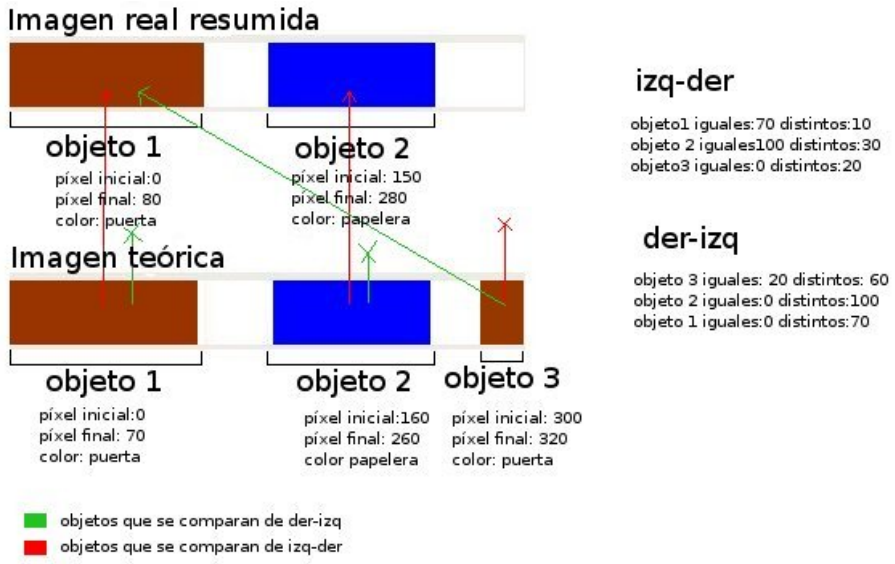

Figura 4.11: Comparación que hacemos de izquierda a derecha y de derecha a izquierda.

Como podemos observar en la figura 4.11 en rojo comparamos los objetos de izquierda a derecha y en verde comparamos los objetos de derecha a izquierda.
De izquierda a derecha primero comparamos los dos objetos de la izquierda de las dos imágenes, son del mismo color, por lo tanto, sacamos el tamaño de píxeles que son iguales y el que son diferentes, por ejemplo, en la imagen real resumida tenemos que tiene 70 píxeles y la imagen teórica tiene 80 píxeles, así que por tamaños tenemos que son iguales 70 píxeles y diferentes 10 píxeles. Después miramos los dos segundos objetos que tienen el mismo color, así que repetimos la operación del calculo de los píxeles iguales y diferentes.Para terminar miramos el tercer objeto de la imagen teórica, como la imagen real ya no tiene más objetos, todos los píxeles del objeto de la imagen teórica son diferentes.

De derecha a izquierda primero comparamos los dos objetos de la derecha de las dos imágenes, son de diferentes color, por lo tanto, pasamos a comparar el primer objeto de la imagen teórica con el segundo objeto de la imagen real, estos dos si son del mismo color, así que sacamos los tamaños de los píxeles que son iguales y los que son diferentes. Ahora tenemos que comparar el segundo objeto de la imagen teórica con la imagen real pero la imagen real ya no tiene más objetos, así que todos son diferentes. Con el tercer objeto de la imagen teórica pasa lo mismo.
Finalmente tenemos dos conjuntos de datos diciendo cual es el tamaño de píxeles iguales y cual es el tamaño de píxeles diferentes, habiendo recorrido las dos imágenes de izquierda a derecha y de derecha a izquierda.

A la tercera fase llegamos solo cuando no estamos en un caso especial, si estamos en un caso especial ya habremos devuelto una probabilidad. Para cada comparación entre un objeto de la imagen teórica y un objeto de la imagen real, si quedan, tenemos sus tamaños de píxeles iguales y tamaños de píxeles diferentes, lo que hacemos es aplicar la formula 4.20 pasa saber lo parecidos que son.

$$peso\_objeto = \frac{iguales - diferentes}{iguales + diferentes} \tag{4.20}$$

Si observamos el rango posibles de valores que podemos tener estamos en [-1,1] y esto no es lo que queremos nosotros necesitamos que el rango de valores sea [0,1], por lo tanto, tenemos que cambiarlo. La forma de hacerlo es sencilla si aplicamos la siguiente formula al peso de cada objeto obtenido obtendremos el valor en el intervalo [0,1].

$$peso\_objeto[0,1] = \frac{peso\_objeto + 1}{2} \tag{4.21}$$

Para obtener la probabilidad entre la imagen real resumida y la imagen teórica aplicando este modelo de observación tenemos que sumar el valor de todos los pesos de todos los objetos calculados y dividirlo entre el numero de objetos.

$$probabilidad\_funcion\_sin\_colas = \frac{\sum_{i=0}^{num\_objetos} peso\_objeto[0,1]}{num\_objetos} \tag{4.22}$$

La probabilidad que obtenemos entre la imagen teórica y la imagen real resumida es del 93 %.

Finalmente podemos decir que nos hemos quedado con que la función de distancia que hemos elegido para el algoritmo de filtro de partículas es la función de distancia sin colas. Aunque en el capítulo 5 se dirá en más detalle el porque se ha elegido esta función podemos anticipar que es la que mejores resultados experimentales ha obtenido, aunque es más costosa computacionalmente, pero ese coste nos lo podemos permitir porque los resultados son buenos.

4.5. Remuestreo

Una vez que hemos desplazado las partículas la distancia que se ha desplazado el robot real, hemos observado la escena y hemos dicho cuales son las partículas que

tienen más probabilidad de encontrarse en la posición correcta, tenemos que decidir que partículas van a componer la siguiente población, para esta tarea usamos el remuestreo. El remuestreo dota de inteligencia nuestro algoritmo ya que provocara que poco a poco las partículas con mucha probabilidad se mantengan en la población y las partículas con poca probabilidad se vayan de la población. Conseguiremos que las partículas converjan hacia la zona en la cuál se encuentra el robot real.

El remuestreo se basa en el algoritmo de la ruleta, este algoritmo consiste en tener una ruleta que esta divida en fragmentos, cada uno de un tamaño. Cada partícula que tenemos representa un fragmento y el tamaño del fragmento lo da la probabilidad que tiene la partícula, a mayor probabilidad mayor será el tamaño y a menor probabilidad menor será el tamaño. una vez que tenemos la ruleta se elige un fragmento aleatoriamente teniendo mayor posibilidad de que toque los que tienen más tamaño y menor posibilidad los que tienen menor tamaño. La partícula que corresponde con el fragmento que ha sido elegido sera una partícula de la siguiente población. Se elegirán tantos fragmentos como partículas queramos que tenga la siguiente población y podrán repetirse fragmentos.

A nivel de código la ruleta se representa como un array con un tamaño igual al número de partículas. Cada elemento del array es una estructura llamada acumulation_type que tiene como atributos, la posición de la partícula (x,y,θ),valor inicial del fragmento y valor final del fragmento. El valor inicial del fragmento es el valor final del fragmento anterior mas uno, y el valor final del fragmento es el valor inicial mas la probabilidad de esa partícula. Por ejemplo la primera partícula tiene una probabilidad de 0.5, su valor inicial es 0 y su valor final es 0.5, la segunda partícula tiene una probabilidad de 0.6, su valor inicial es de 0.6 y su valor final es 1.1, y así sucesivamente.
Para sacar el fragmento que queremos lo que se hace es calcular un valor aleatorio entre cero y el máximo valor acumulado. En el ejemplo anterior el máximo es 1.1. Este valor calculado pertenecerá a un fragmento, por ejemplo si el valor que da es 0.7 quiere decir que la partícula seleccionada para la siguiente población a sido la segunda partícula, ya que 0.7 esta entre 0.5 y 1.1. Para saber sacar cual es la partícula realizamos una búsqueda binaria.

Que puedan repetirse fragmentos produce un problema, ya que vamos a tener varias partículas en la misma posición, para resolverlo hemos usado la técnica de ruido térmico, que hace que las partículas no se sitúen en la posición exacta donde tienen que ponerse si no que se sitúan una posición un poco desplazada. Esto lo hacemos para que cuando obtengamos la nueva población, no tengamos por ejemplo tres partículas en una misma posición ya que se moverán igual y darán la misma probabilidad en la observación, si no que tengamos tres partículas que estén relativamente cerca unas de las otras y podamos observar nuevas posiciones dentro de la zona donde estaba la partícula anterior.

Para calcular la nueva posición de una partícula aplicando ruido térmico se usan

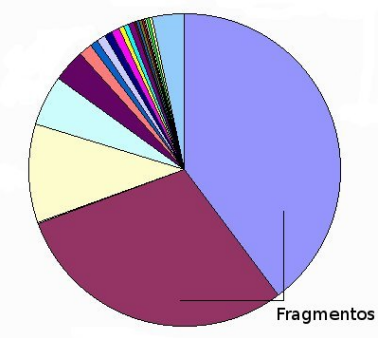

Figura 4.12: Representación de la ruleta, los fragmentos más grandes son los que tienen mayor probabilidad de salir.

las siguientes funciones:

$$Ruido\_termico_{lineal} = Randon(0, Max\_Ruido_{lineal}) \tag{4.23}$$

$$Ruido\_termico_{angular} = Randon(0, Max\_Ruido_{angular}) \tag{4.24}$$

$$nueva\_particula_x = aumulacion_x + Ruido\_termico_{lineal} \tag{4.25}$$

$$nueva\_particula_y = acumulacion_y + Ruido\_termico_{lineal} \tag{4.26}$$

$$nueva\_particula_\theta = acumulacion_\theta + Ruido\_termico_{angular} \tag{4.27}$$

Donde Max_Ruido_{lineal} y $Max\_Ruido_{angular}$ son dos constantes que hemos definido, hemos elegido que sean 150mm y 15 grados ya que con esos valores obtenemos una nube de partículas aceptable. $acumulacion_x$, $acumulacion_y$ y $acumulacion_\theta$ es la posición de la partícula del fragmento elegido.

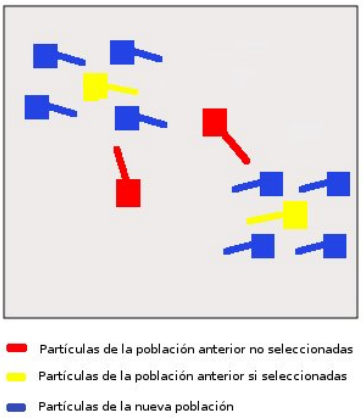

Figura 4.13: Representación del ruido térmico

4.6. Localización probabilística con mallas de probabilidad

Esta es otra técnica de localización que de manera complementaria al filtro de partículas también hemos desarrollado. No entraremos mucho en detalle, solo explicaremos en qué consiste, como se ha implementado y para que la hemos usado. Si se quiere ver esta técnica con más detalle leer el proyecto fin de carrera de Alberto Lopez [López, 2005].

Esta técnica es similar a la anterior pero sin muestreo. Como información de entrada tenemos lecturas sensoriales recibidas por el ojo del robot, la cámara. También tenemos las lecturas sensoriales del movimiento del robot producido por los los encoders que hay en las ruedas.

También nos tenemos que apoyar de una mapa visual que es donde se pone la malla de probabilidad, representa un mapa del departamental. Al disponer de este mapa, si la cámara visualiza una puerta, todas las posiciones que tengan una puerta delate tendrán una probabilidad alta.

El algoritmo de localización por mallas de probabilidad se dividen de dos partes que se ejecutan independientemente cuando se les pide.

- Modelo de movimiento: Se captura la información medida por los encoders del robot.
- Modelo de observación: Se capturan toda la información que proporcionan las nuevas lecturas sensoriales proveniente de la cámara del robot y se calculan las nuevas densidades de probabilidad de cada una de las posiciones de la malla de probabilidad. Las evidencias que se parezcan a la observación de la cámara aumentaran su probabilidad y las que no se aparezcan disminuirán su probabilidad.

Para la actualización de las evidencias se utiliza el teorema de Bayes que nos permitirá efectuar la fusión de la información sensorial recibida con la información acumulada en las anteriores evidencias.

Los motivos por los que hemos decidido implementar esta otra técnica son:

- Para ver que la función distancia que hemos elegido converge a la posición concreta que queremos, al cabo de algunos movimientos del robot y unas observaciones visuales.
- Estudiar los modelos de observación y movimiento en toda la extensión espacial.

El mapa visual que usa en esta técnica es el mismo que usamos en la técnica de filtro de partículas así que no lo explicaremos porque ya lo hemos hecho.

4.6.1. Cubo de probabilidad

El cubo de probabilidad es un cubo con todas las posibles posiciones donde se pueden encontrar el robot. La posición de un robot se define por (x,y, Θ), al tener tres variables se obtiene un cubo. El tamaño del cubo es de 235x300x60, siendo 235 la x máxima, 300 la y máxima y 60 el número de rejillas, en vez de tener 360 rejillas una por grado lo que hacemos es tener una por cada 6 grados, el motivo de esto es para no tener un número excesivo de posiciones posibles y tarde una barbaridad en calcular una observación. Para implementar el cubo lo que hemos hecho es crear un arrray de 60 valores y cada valor es un puntero a un Tgrid de tamaño 235x300, en cada posición del Tgrid se guardara entre otras cosas la probabilidad que se tiene en esa posición.

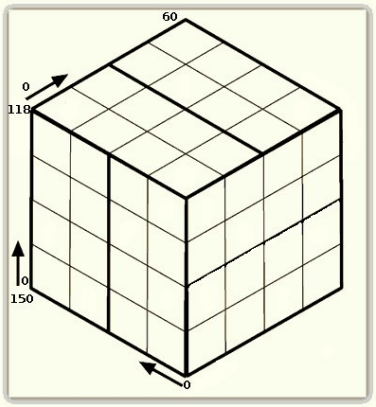

Figura 4.14: Representación del cubo de probabilidad

4.6.2. Modelo de movimiento

El modelo de movimiento de las mallas de probabilidad es diferente al modelo de movimiento de las partículas. Mientras que en el modelo de movimiento de las partículas son las partículas las que se mueven y la probabilidad asociada a ellas se queda siempre fija a ellas, en el modelo de movimiento de las mallas de probabilidad es la probabilidad la que se va desplazando por las posiciones del cubo.

La implementación ha sido fácil. Calculamos el movimiento que ha realizado el robot de la misma forma que con el algoritmo del filtro de partículas. Cuando lo tenemos calculado, cogemos de cada posición la probabilidad que tiene asociada, y se la asociamos a una nueva posición que es la posición que teníamos más el desplazamiento que hemos calculado. Si la nueva posición esta fuera de los rangos de movimiento que son [0-235] [0-300] la probabilidad no se almacena. Para que no se pierdan datos mientras estamos actualizando el cubo de probabilidad hemos usado *double-buffering*. También hay que tener en cuenta que con esta técnica no hemos usado ruido gaussiano, el motivo es porque como solo usamos la técnica para comprobar si es converge nuestro modelo no hace falta que sea tan real.

4.6.3. Modelo de Observación

El modelo de observación que hemos utilizado en las mallas de probabilidad es el mismo que tenemos el modelo de observación del filtro de partículas. Este requisito es necesario, ya que lo queremos ver con esta técnica es si el modelo de observación que hemos escogido es discriminante y por tanto valido para la localización.

4.6.4. Acumulación Bayesiana de Evidencias

Sólo con una observación no es posible averiguar cual es la posición del robot, necesitamos de la acumulación de evidencias. El motivo es que con una única observación obtendremos muchas posibles posiciones y necesitamos realizar más observaciones en otros lugares para ir eliminando posiciones erróneas y a final quedarnos con la correcta.

El paso de acumulación lo hemos realizado siguiendo la regla de Bayes que se comporta de forma robusta y sirve para ir fusionado todas las evidencias que se van obteniendo de la observación con la cámara y el movimiento de los encoders. Esas evidencias las hemos descrito como probabilidades siguiendo el modelo descrito. En las posiciones compatibles con las observaciones se ira aumentando la probabilidad y en las posiciones no compatibles con la observación se ira disminuyendo la probabilidad.

Inicialmente todas las posiciones del cubo de probabilidad tendrán una probabilidad igual a cero, porque todavía no se ha hecho ninguna observación y no sabemos donde esta. Por este motivo lo primero que se hace es realizar una observación y rellenar el cubo con la probabilidad que tiene cada posición respecto de la observación realizada. A medida que se realicen nuevas observaciones se irán desechando las posiciones que no sean compatibles.

Para realizar la fusión de evidencias se sigue el desarrollo matemático completo de la regla de Bayes [Thrun, 1997]. A través de este desarrollo se llega a una ecuación incremental (ecuación: 4.28) que nos permite manejar la probabilidad acumulada a través de la ultima observación sensorial y de una forma muy sencilla. Para calcular la nueva probabilidad acumulada en el instante (t), fusionamos los ratios de probabilidad en el instante (t-1) y de la nueva probabilidad calculada de la observación en el instante (t), mediante la relación 4.31 que se deduce del desarrollo de la regla de Bayes.

$$P(x, y, \Theta/obs_1, obs_2 ... obs_N) \sim P(x, y, \Theta/obs_1, obs_2 ... obs_{N-1}) \cdot P(x, y, \Theta/obs_N) \quad (4.28)$$

$$Pac(x_t) \sim Pac_{t-1} \cdot Pobs_t \quad (4.29)$$

$$\rho ac(x_t) = \rho ac_{t-1} \cdot \rho obs_t \quad (4.30)$$

$$\ln \rho ac(x_t) = \ln \rho ac_{t-1} + \ln \rho obs_t \quad (4.31)$$

Donde:

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

Nos interesa manejar los logaritmos de los ratios porque si la probabilidad es mayor de 0.5, su ratio es mayor de 1, su logaritmo es positivo y por tanto la acumulada sube con esta incorporación. Mientras que si la probabilidad es menor de 0.5, su ratio es menor de 1, su logaritmo es negativo y por tanto la acumulada baja con esta incorporación. Una vez aplicada la regla de Bayes podemos decir que tras varias iteraciones del algoritmo se llega a obtener una buena localización o lo que es en nuestro caso comprobamos que el modelo de observación es convergente.

4.7. Interfaz de Usuario

En la figura 4.15 podemos observar la interfaz de usuario inicial realizada con la librería de interfaces de usuario *GTK*.

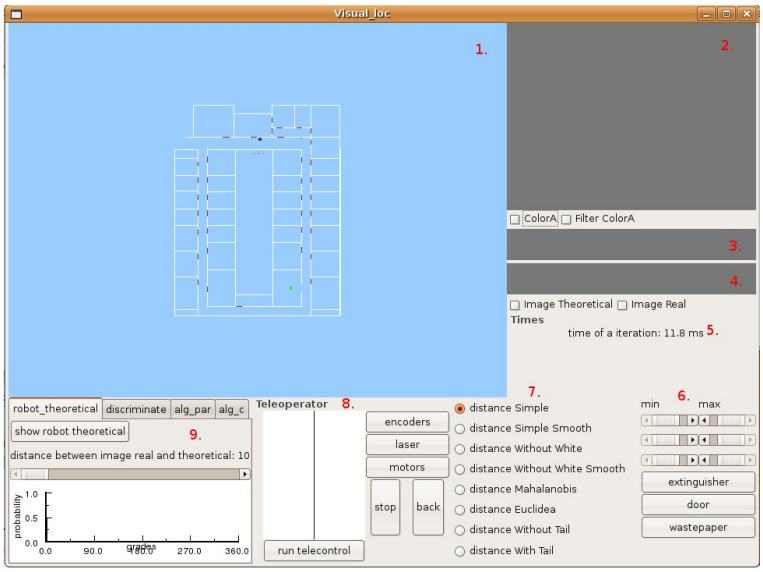

Figura 4.15: Interfaz que se muestra según arrancamos la aplicación.

A continuación vamos a explicar cada una de las partes en las que se divide esta interfaz:

1. *Canvas* en OpenGL donde se muestran el mapa visual de color, las partículas y el robot real.
2. *Image* donde se muestra lo que nos manda el sensor de la cámara a partir de la variable ColorA. Tenemos dos *checks* uno llamado ColorA para activar el sensor de la cámara y recibir datos y otro llamado Filter ColorA que activa el filtro de color de la imagen ColorA, el cual se mostrara en ese *Image*.

3. *Image* donde se muestra la imagen teórica que este seleccionada en ese momento. Tenemos un *check* que activa esta visualización llamado Image Theorical.

4. *Image* donde se muestra la imagen real resumida. Tenemos un *check* que activa esta visualización llamado Image Real.

5. Muestra el tiempo que se tarda en realizar una iteración completa.

6. Sirve para modificar el filtro de color. Tenemos 6 *scrolls* cada uno sirve para modificar el Hmin, Hmax, Smin, Smax, Imin y Imax y tenemos 3 botones cada uno con el nombre de un objeto relevante en el mapa visual, papelera, extintor y puerta. Modificamos los scrolls con los valores que queramos y pulsamos uno de los botones, automáticamente el filtro de color de ese objeto se modifica por nuevos valores.

7. Tenemos 8 *radio buttons* que sirven para seleccionar que función de distancia queremos aplicar.

8. Zona de la interfaz con la que podemos teleoperar el robot real y así poder movernos por el mundo gazebo y obtener nuevas instantáneas de la cámara. Se divide en un *canvas*, que dependiendo en el que pinchemos calculara la velocidad lineal y angular que mandara al robot. Tiene tres *buttons* que sirven para activar los encoders, el laser, y los motores. También tiene un *button* para que se pare el robot y otro para que vaya marcha atrás. El ultimo *button* sirve para activar la teleoperación del robot o no.

9. Menú con los diferentes pestañas en los que podemos hacer experimentos. Se explicaran a continuación.

El menú se ha divido en cuatro pestañas, en cada pestaña se realizan diferentes funciones, tenemos cuatro pestañas que son *robot_teorical, disciminate, alg_par y alg_c.*

- *Pestaña robot_theorical.* En esta pestaña si pinchamos en el canvas donde se encuentra el mapa visual, en punto justo donde hemos pinchado se colocara un robot que nos devolverá una imagen teórica de lo que esta visualizando. Gracias a esto podremos realizar algunos experimentos que explicaremos en el capítulo 5. Todos los elementos de esta pestaña se pueden ver en la figura 4.16.

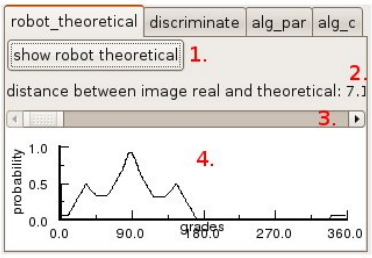

Figura 4.16: Pestaña del robot teórico.

1. *Button* que visualiza el robot teórico.
2. Muestra el porcentaje de igualdad de la imagen teórica y la imagen real resumida atendiendo a la función de distancia elegida.
3. *Scroll* que sirve para decir cual es el ángulo θ del robot teorico.
4. Gráfica que muestra la probabilidad del robot teórico con una posición x e y fijas y en todo θ.

- *Pestaña discriminate.* En esta pestaña en vez de colocar un robot teórico en una posición colocamos robot teóricos por todo el mapa de forma uniforme hasta rellenarlo entero. Todos tienen la misma orientación. Calculamos la probabilidad de cada uno de ellos con una fucnión de distancia previamente elegida y podemos mover el ángulo para ver como se comporta con otras orientaciones. Todos los elementos de esta pestaña se pueden ver en la figura 4.17.

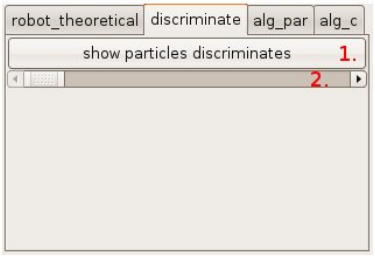

Figura 4.17: Pestaña para saber la discriminación de una función de distancia en todo el mapa visual.

1. *Button* que muestra todos los robots teóricos colocados en el mapa visual.
2. *Scroll* que selecciona cual es la orientación de todos los robots teórico.

- *Pestaña alg_par.* En esta pestaña colocamos el algoritmo de filtro de partículas. Todos los elementos de esta pestaña se pueden ver en la figura 4.18

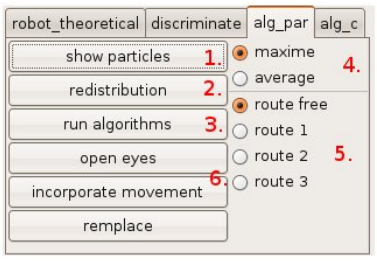

Figura 4.18: Pestaña del algoritmo de filtro de partículas.

1. *Button* que si se pulsa muestra en el *canvas* del mapa visual las partículas.
2. *Button* que reinicia el algoritmo, redistribuye las partículas y las pone con probabilidad igual a 0.

3. *Button* que inicia el algoritmo de filtro de partículas.
4. *Radio buttons* que sirven para mostrar en el canvas del mapa visual, la partículas con mayor probabilidad de la población que en ese momento se esta mirando o el centro de la nube de puntos que se ha formado.
5. *Radio buttons* que eligen el recorrido que vamos a hacer, existen 4 recorridos.*Route_free* es un recorrido en el que el robot se teleopera y el usuario elige cuando se añaden el modelo de observación, el modelo de movimiento y el remuestreo pulsando unos botones. *Route_1* hace el mismo recorrido que el recorrido 1 del experimento con mallas de probabilidades. *Route_2* hace el mismo recorrido que el recorrido 2 del experimento con mallas de probabilidades. *Route_3* es con el que hemos hecho los experimentos del filtro de partículas, el recorrido inserta de forma automática el modelo de observación el modelo de movimiento y el remuestreo dependiendo de los parámetros que hemos definido.
6. *Buttons* que si se les pulsa insertan el modelo de observación, el modelo de movimiento o el remuestreo si estamos *route_free*.

- *Pestaña alg_c.* En esta pestaña colocamos el algoritmo de mallas de probabilidad. Todos los elementos de esta pestaña se pueden ver en la figura 4.19.

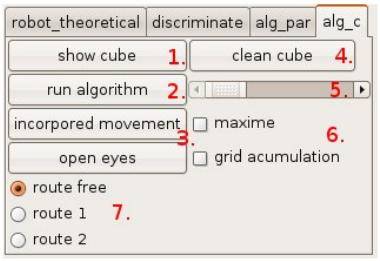

Figura 4.19: Pestaña del algoritmo de mallas de probabilidad.

1. *Button* que si se le pulsa muestra la rejilla del cubo correspondiente al ángulo que hayamos seleccionado.
2. *Button* que si se le pulsa inicia el algoritmo de mallas de probabilidad.
3. *Buttons* que si estamos en *route_free*, incorporan el modelo de observación o el modelo de movimiento.
4. *Button* que reinicia el cubo de probabilidad.
5. *Scroll* que selecciona el ángulo por el cual obtendremos la rejilla que se mostrara en el *canvas* del mapa visual.
6. *Radio buttons* que muestran la posición del cubo con mayor probabilidad de todas o una rejilla con la acumulada del todo el cubo.
7. *Radio buttons* que selecciona el recorrido que vamos a hacer con el algoritmo de mallas de probabilidad. Tenemos tres recorridos. *Route_free* es un

recorrido en el que el robot se teleopera y el usuario elige cuando se añaden el modelo de observación y el modelo de movimiento pulsando unos botones. *Route_1* hace el recorrido 1 de mallas de probabilidad que se explica en el capítulo 5. *Route_2* hace el recorrido 2 de mallas de probabilidad que se explica en el capítulo 5.

Capítulo 5
Resultados Experimentales

En el capítulo 4 hemos explicado en que consiste el algoritmo de localización basado en el filtro de partículas, así como hemos implementado cada una de sus partes. En este capítulo vamos a explicar los experimentos que hemos realizado para ajustar cada una de las partes del algoritmo de localización con filtro de partículas y así obtener una localización ágil y robusta.

Primero, mostraremos los experimentos que han servido para ajustar algunos parámetros como son el tamaño del ruido gaussiano, el número de partículas o o el tamaño del ruido térmico. Segundo, expondremos todos los los experimentos que hemos hecho para que el modelo de observación sea el que queríamos. Tercero y último, haremos pruebas de convergencia del algoritmo de mallas de probabilidad y el algoritmo de filtro de partículas y mostraremos algunos ejemplos de ejecuciones típicas de nuestro algoritmo de filtro de partículas.

5.1. Ajustes de Parámetros

Definir bien los parámetros de nuestro algoritmo es muy importante, ya que queremos que el algoritmo sea robusto y ágil y si los tenemos bien definidos conseguiremos ese objetivo.
En este apartado mostraremos experimentos que hemos realizado para saber cual era el mejor valor para esos parámetros y diremos porque elegimos esos valores.

5.1.1. Ruido del Modelo de Movimiento

El modelo de movimiento debe ser correcto, como dijimos en el capítulo 4 el robot tiene una posición y una orientación y las partículas también tienen su posición y orientación, según se vaya moviendo el robot las partículas se moverán esa distancia sobre su posición, es decir, nuestras partículas tienen un movimiento relativo respecto al movimiento del robot.

A la pregunta que hemos tenido que contestar en el modelo de movimientos han sido.¿Que valor de ruido gausiano tenemos que poner?

En un primer momento pensamos en poner valor absoluto al ruido gausiano, decidimos que el ruido gaussiano lineal seria 20 cm y el ruido gausiano angular fuese 15

grados, esto tiene un inconveniente bastante claro, si por ejemplo nos movemos 10 cm y añadimos el modelo de movimiento, nuestro ruido gausiano es de 20 cm eso provoca que una partícula se pueda mover 20 cm y eso no es correcto. Por este motivo pensamos en utilizar como ruido gausiano un porcentaje de lo que nos hemos desplazado. Para este mismo ejemplo si nos movemos 10 cm y tenemos un porcentaje del 10 % el valor del ruido gaussiano lineal es de 1 cm.

En segundo lugar, tuvimos que decidir que cuál sería ese porcentaje. En la figura 5.1 a) tenemos el robot y partículas que están en posiciones iniciales, en la figura 5.1 b) hemos movido el robot una distancia y hemos añadido el modelo de movimiento a las partículas, podemos observar en color azul las partículas sin aplicarles ruido gaussiano y en color blanco las partículas aplicándoles un ruido gaussiano, en la figura 5.1 c) hemos realizado la misma operación. La diferencia entre 5.1 b) y 5.1 c) es que en 5.1 b) hemos usado como porcentaje de ruido gaussiano el 10 % y en 5.1 c) hemos usado como porcentaje de ruido gaussiano el 50 %.

Observando las dos figuras, podemos ver que en la que tiene como ruido gaussiano el 10 % la posición final de las partículas esta más cerca de la posición final de las partículas sin aplicar el ruido y en la que tiene como ruido gausiano el 50 % la posición final de las partículas esta más lejos de la posición final de las partículas sin aplicar el ruido. Con el ruido gaussiano simulamos que los sensores del robot tienen un error, normalmente el error de los sensores no es muy grande, por lo tanto nos interesa que el ruido gaussiano no sea elevado. Hemos decidido quedarnos con un ruido gaussiano del 10 % porque hace que las partículas finales no se encuentren en la posición en la que deberían estar idealmente sino que están un poco desplazadas y además este desplazamiento no es exagerado.

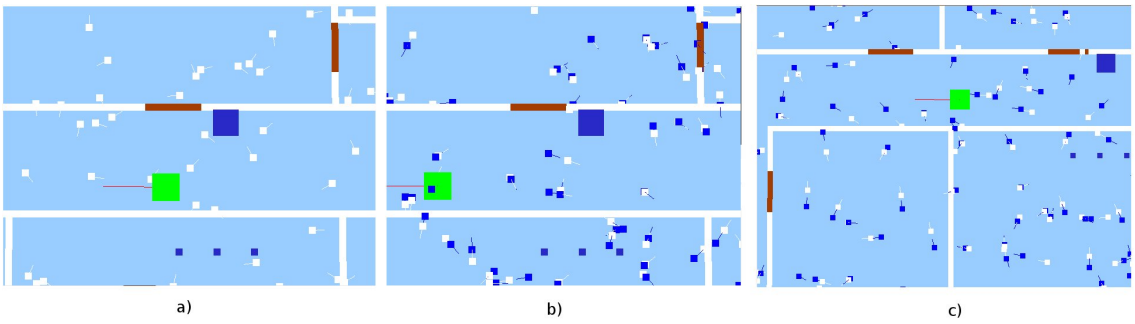

Figura 5.1: a) partículas antes de movimiento b) partículas después de un movimiento con ruido gaussiano del 10 % c) partículas después de un movimiento con ruido gaussiano del 50 %.

5.1.2. Número de Partículas

En el capítulo 4 definimos que era una partícula y para que la utilizamos, en este capítulo diremos cuantas tenemos que poner para conseguir localizar el robot y nuestro

algoritmo sea ágil.

El número de partículas es muy importante porque por cada partícula tenemos que calcular su imagen teórica y después tenemos que compararla con la imagen real resumida para obtener su probabilidad, esto lleva su tiempo. Podemos decir que el número de partículas es nuestro cuello de botella en cuanto al tiempo en que se tarda en realizar una iteración, es verdad que además de calcular la imagen teórica y sacar la probabilidad de la partícula, también tenemos que mover las partículas y obtener las partículas que van a componer la siguiente población, pero estas dos cosas se tardan poco en hacer, unos 20 milisegundos más o menos. El tiempo que se tarda en calcular la imagen teórica y sacar la probabilidad de 1000 partículas es unos 150 milisegundos, el tiempo que se tarda con 2000 partículas es de 300 milisegundos y 9000 partículas es de 1300 milisegundos.

Podemos sacar como conclusión que si doblamos el numero de partículas doblamos el tiempo que se tarda en obtener la probabilidad de todas las partículas. Como dijimos al principio queremos que el algoritmo sea robusto y sea ágil, si ponemos 9000 partículas tardaremos más de 1 segundo en hacer una iteración y eso no nos vale, haciendo experimentos hemos visto que 1000 partículas es un buen número el robot se localiza bastante bien y no se tarda mucho en hacer una iteración aproximadamente 170-180 milisegundos sumando el modelo de observación, el modelo de movimiento y el remuestreo.

5.1.3. Ruido térmico con el Remuestreo

El remuestreo se encarga de calcular la población de partículas que va ser utilizada en la siguiente iteración por medio de la ruleta, según se explicó en el capítulo 4. Interesa conseguir que una partícula con alta probabilidad sea promocionada a la siguiente población pero también nos interesa que las zonas contiguas a esa partícula también sean exploradas.
Para el primer objetivo usamos la ruleta ya que si tiene una probabilidad alta seguramente salga seleccionada.
Para el segundo objetivo a parte de la ruleta usamos el ruido térmico, como hemos dicho antes si la partícula tiene una probabilidad alta seguramente sea elegida para la siguiente población, pero además seguramente sea elegida varias veces, así que tendremos varias partículas en la misma posición, esto no nos interesa , el ruido térmico lo que consigue es poner esas partículas un poco desplazadas entre si, conseguimos una nube de partículas y conseguimos que se exploren zonas contiguas, con el fin de conseguir converger a la posición en la que se encuentra el robot.

Hemos tenido que hacer un estudio para averiguar qué valor de ruido térmico era el mejor, nos interesa que la nube de partículas no sea demasiado grande, porque cuando llegamos a los pasillos existen zonas muy parecidas entre si ya que el pasillo es muy simétrico, si tenemos una nube de partículas muy larga se provoca que las partículas estén en dos zonas simétricas, se mueven de una zona a otra y se pierde la localización.

Tampoco nos interesa que sea demasiado pequeño, porque cuando se pasa de no ver nada a ver algo, puede ocurrir que todas las partículas no vean nada todavía y el robot si vea algo, se provoca una degeneración de partículas y se reinicia el algoritmo, hemos perdido la localización.

La figura 5.2 muestra en la figura a) las partículas justo antes de realizar el remuestreo habiendo hecho el modelo de observación sin colas, en la figura b) tenemos un ruido térmico lineal de 15 cm y un ruido térmico angular de 5 grados y en la figura c) tenemos un ruido térmico lineal de 30 cm y un ruido térmico angular de 10 grados.

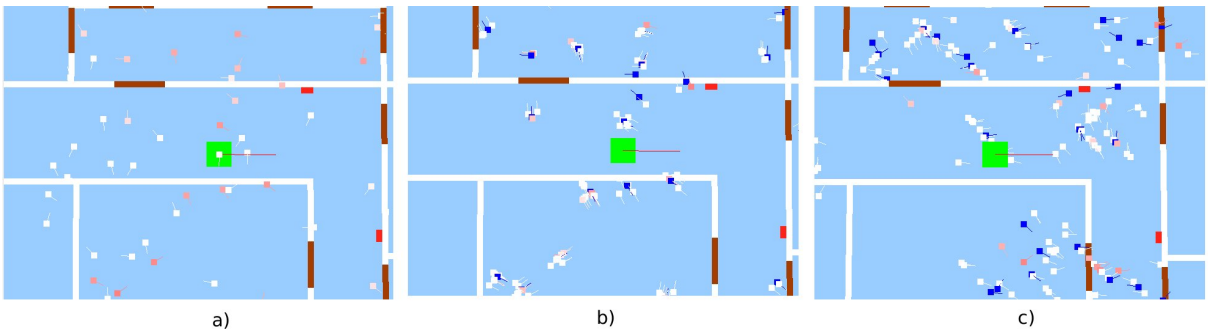

Figura 5.2: remuestreo con diferentes tamaños de ruido térmico

En la figura 5.3 se muestra en la figura a) la nube de partículas que nos queda cuando conseguimos converger a la localización en la que se encuentra el robot con los datos del ruido térmico lineal de 15 cm y ruido térmico angular de 5 grados, en la figura b) podemos ver la nube de partículas con el ruido térmico lineal de 30 cm y el ruido térmico angular de 10 grados.

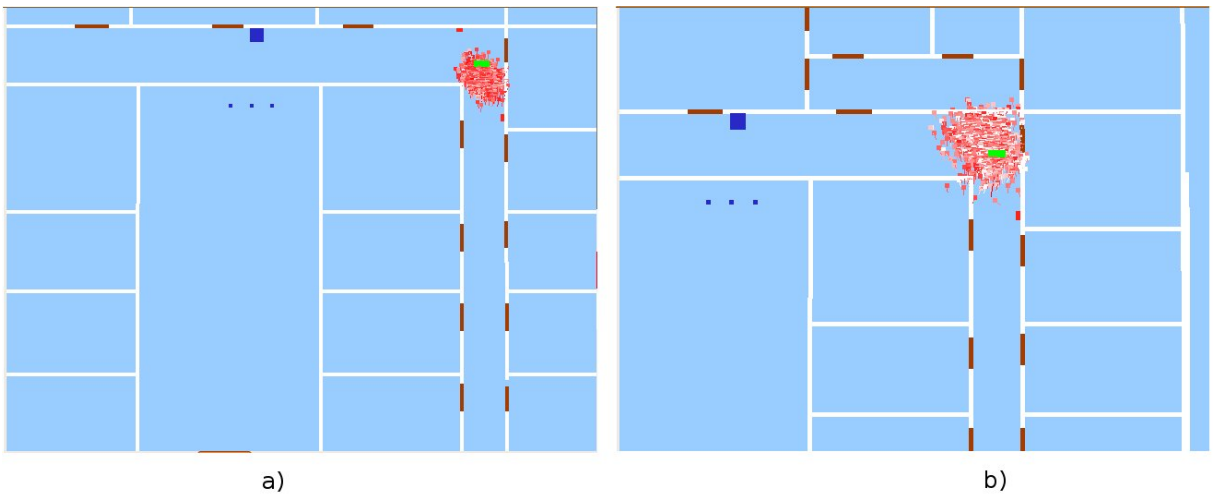

Figura 5.3: Nubes de particulas con diferentes tamaños de ruido térmico

Hemos decido que el ruido térmico lineal sea de 15 cm y el ruido térmico angular sea de 5 grados porque la nube de puntos que tenemos con el ruido térmico lineal de 30 cm y

ruido térmico angular de 10 grados es muy grande y puede que nos provoque problemas de simetría en los pasillos. Con el ruido térmico lineal sea de 15 cm y el ruido térmico angular sea de 5 grados conseguimos una nube de puntos buena y además cuando pasamos de no ver objetos a ver objetos no se produce degeneración.

5.2. Experimentos del modelo de observación

Para obtener un buen modelo de observación hemos tenido en cuenta dos cosas. Primero, tenemos que obtener una buena observación teórica y que sea igual o muy parecida a la observación real, para esto hemos estudiado el ángulo de visión. Segundo, tenemos que conseguir que las partículas con posiciones cercanas a la que nos encontramos tengan una probabilidad alta y el resto baja, para ello hemos hecho un estudio de la función distancia.

5.2.1. Ajuste del ángulo visión

El ángulo de visión como dijimos en el capítulo 4 lo calculamos utilizando la librería *progeo*, influye en la calidad de las imágenes generadas. Vamos a explicar porque llegamos a la conclusión de que esta es la mejor forma calcular el ángulo de visión.

En un primer momento el ángulo de visión era una constante que teníamos en nuestro programa, era el programador el que decidía cual era el ángulo de visión que poníamos. Esta forma no esta mal pero si en vez de utilizar gazebo utilizamos una cámara real, tendremos que volver a hacer un estudio de cual es el ángulo que debemos poner. Usamos *Gazebo* que utiliza OpenGL para simular cámaras y jderobot nos proporciona la librería progeo para expresar cámaras en OpenGL como cámaras pin-hole que la que usa jderobot, es por este motivo es por el cual decidimos calcular el ángulo de visión con progeo

En la figura 5.4 tenemos al robot real colocado en dos posiciones diferentes, los dos comparten que el robot teórico esta exactamente en la misma posición que el robot real. En a) tenemos el robot colocado enfrente de una puerta y la papelera, al estar el robot teórico y el robot real en la misma posición las imágenes real resumida y la imagen teórica son muy parecidas. En b) tenemos el robot colocado en medio de un pasillo en el que se ven puertas, pasa lo mismo que en el caso anterior.

La figura 5.5 muestra dos gráficas. La primera corresponde a la figura a) del anterior dibujo y la segunda a la figura b) del anterior dibujo. En estas dos gráficas se muestra la probabilidad que se obtiene aplicando el modelo de observación con la función de distancia sin colas. La zona de la que se extrae la gráfica es la posición donde se encuentra el robot y todas las posiciones cercanas con el mismo θ que el del robot.

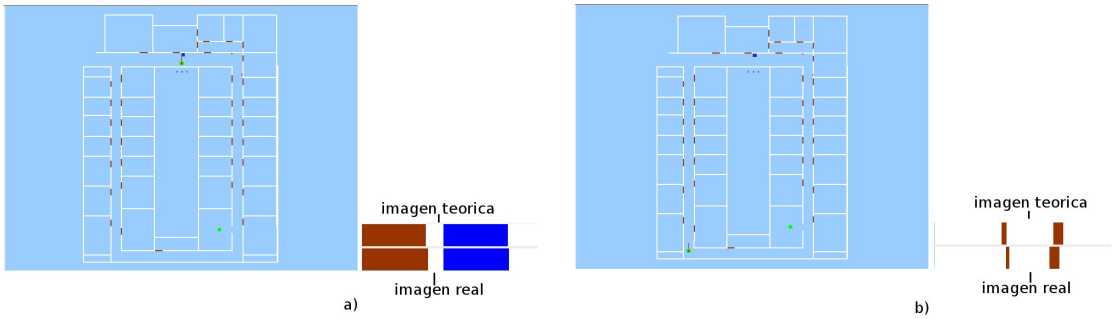

Figura 5.4: a) Robot en frente de la papelera y una puerta b)Robot en medio del pasillo.

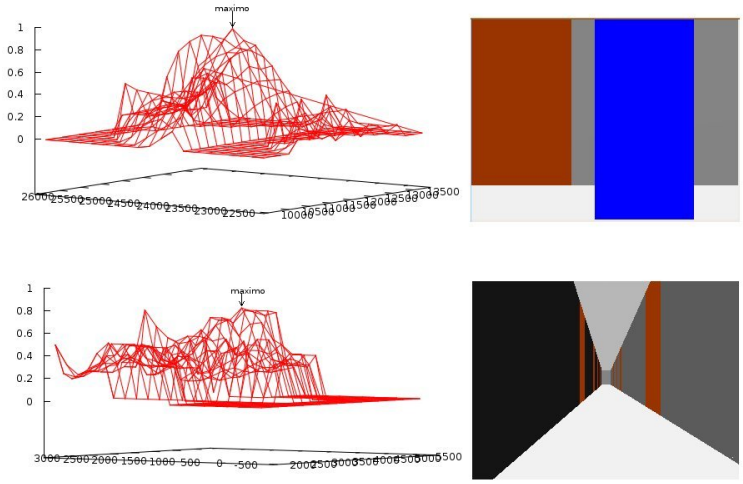

Figura 5.5: a) Robot en frente de la papelera y una puerta b)Robot en medio del pasillo.

Podemos sacar dos conclusiones viendo las dos figuras.
Si nos fijamos en la figura 5.4 podemos ver que la imagen real y la imagen teórica no son exactamente iguales. El motivo por el que ocurre esto es que la imagen real se genera a partir de información continua (mundo simulado), mientras que la imagen teórica se forma a partir de información discreta (mapa visual). Como la diferencia entre la imagen real y la imagen teórica es muy pequeña damos por buena esta aproximación.
Si nos fijamos en la figura 5.5 podemos ver que el máxima probabilidad nos la da en el punto en el que se encuentra el robot. Esto es lo que estamos buscando, que en una zona tengamos el máximo donde se encuentra el robot. También podemos observar que la función de distancia sin colas tiene una gráfica de termino medio, no es muy discriminante ni poco, explicaremos esto más adelante pero podemos decir que esta función cumple con lo que estamos buscando.

5.2.2. Discriminación espacial de corto alcance

La función de distancia es la parte más importante de nuestro algoritmo, devuelve la probabilidad que tiene una partícula de encontrarse en la posición donde se encuentra el robot y marca la evolución de las partículas. Por este motivo a esta parte le hemos dedicado muchos experimentos y un estudio detallado.

Como explicamos en capítulo 4 hemos probado dos tipos de funciones de distancia, las basadas en píxeles y las basadas en objetos, que a su vez se dividen en diferentes funciones que calculan la distancia como son simple, Mahalanobis, sin cola, etc. Primeramente expondremos los experimentos que hemos realizado y al final diremos cuál es la mejor y el porque.

Nuestro robot tiene tres coordenadas de búsqueda (x,y,Θ), esto quiere decir que nuestro espacio es un cubo. Hemos realizado dos tipos de experimentos y que lo que pretenden es recorrer este cubo de diferentes formas y así hacernos una idea de cual es la mejor función de distancia.

La discriminación espacial de corto alcance consiste en colocar el robot real en una posición y obtener su imagen real, colocar el robot teórico en diferentes posiciones x e y, ver cual es el comportamiento de la probabilidad en todo su espacio θ.
Hemos seleccionado tres ejemplos representativos. En la figura 5.6 tenemos el robot real colocado enfrente de la papelera y una puerta, el robot teórico se encuentra en la misma posición. En la figura 5.7 tenemos el robot real colocado enfrente de la papelera y una puerta, el robot teórico se encuentra observando un extintor y una puerta. En la figura 5.8 tenemos el robot real colocado enfrente de la papelera y una puerta, el robot teórico se encuentra observando el pasillo.

Ahora vamos a sacar conclusiones de las figuras. La primera conclusión es que viendo los resultados de las gráficas de distancia simple con distancia simple suavizada y distancia sin blancos con distancia sin blancos suavizada, el suavizar la imagen teórica para compararla con la real no hace que varié mucho el resultado, así que no es interesante suavizar las imágenes porque tiene un coste computacional extra y no se obtiene una mejora.
Si observamos la figura 5.6 el robot real tiene una orientación de 90 grados, como el robot teórico se encuentra en la misma posición que el robot real, en la orientación de 90 grados se tiene que obtener el máximo de probabilidad, tiene que ir subiendo la probabilidad según nos acercamos a 90 grados y bajando según nos vamos alejando de 90 grados. La distancia de Mahalanobis y la distancia euclídea cumplen con esto, pero para el resto de valores se tiene una probabilidad alta y debería ser baja. La distancia con colas cumple el objetivo pero es demasiado discriminante ya que sube y baja muy rápido.
Si observamos la figura 5.7 podemos ver que la que la imagen real tiene una papelera y una puerta y la imagen teórica vera un extintor y dos puertas. La función de distancia puede parecerse pero como mucho debería llegar al 50 % cuando la imagen teórica

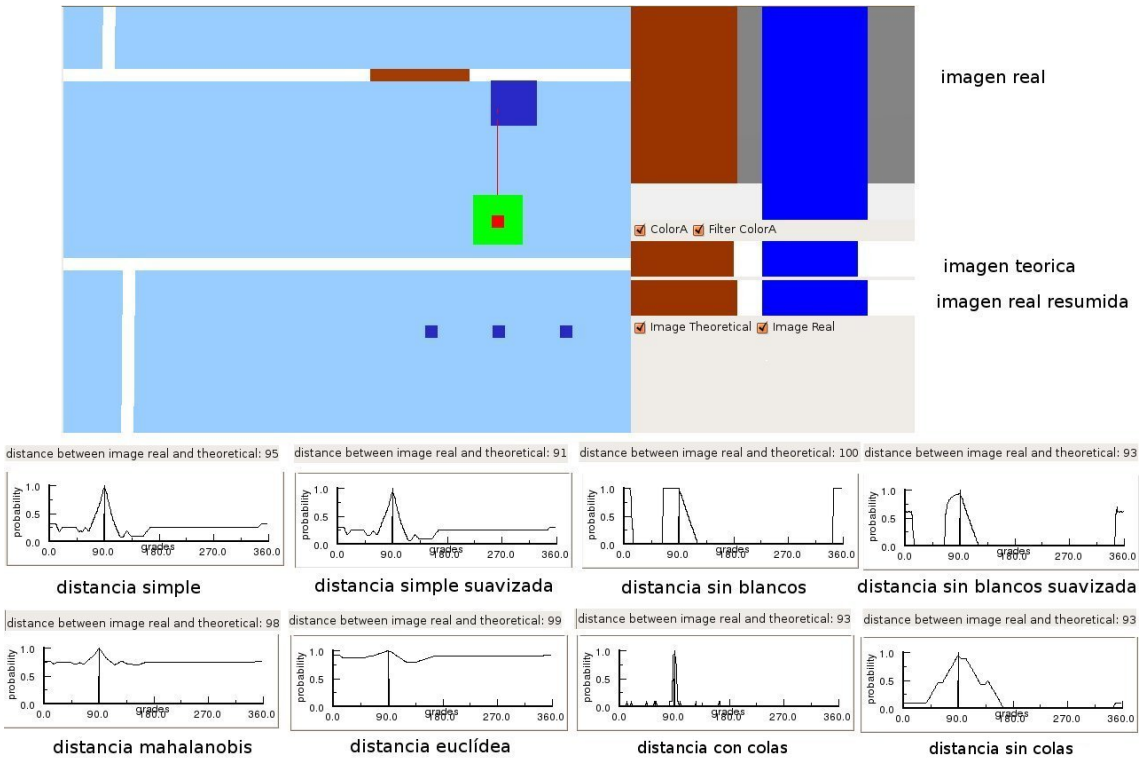

Figura 5.6: Robot colocado delante de la papelera y una puerta y mostramos todas las gráficas de las funciones de distancia.

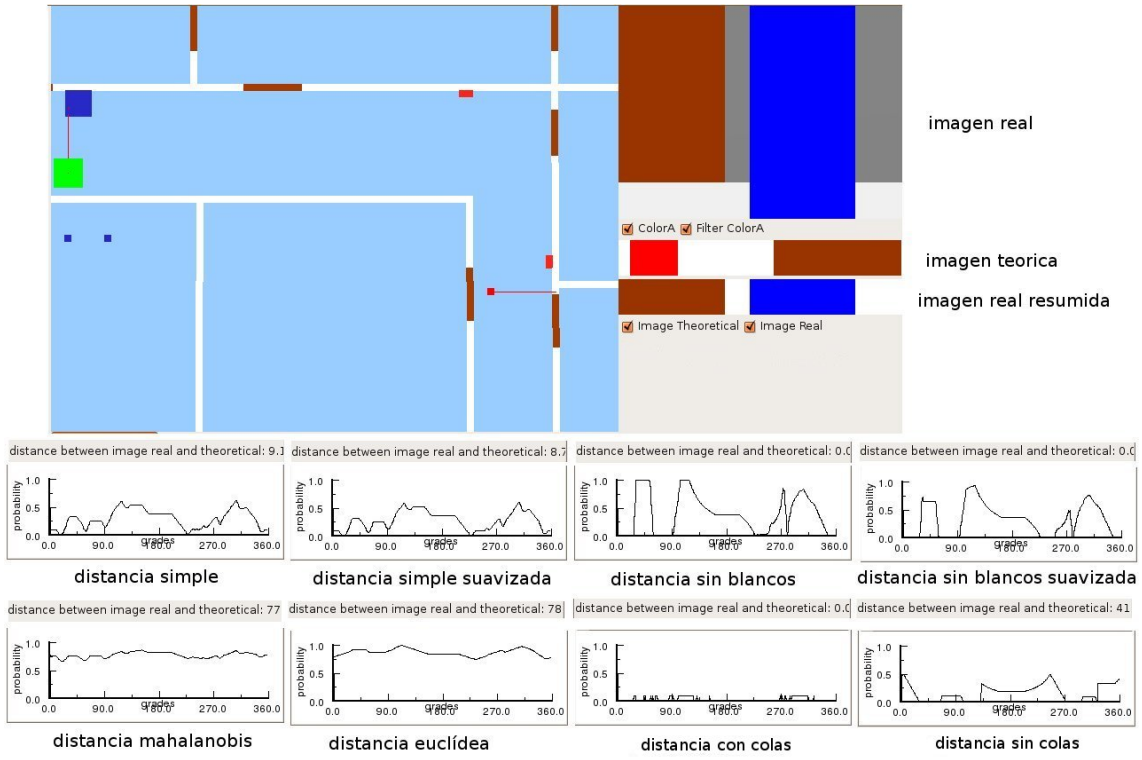

Figura 5.7: Robot colocado delante de una puerta y un extintor y mostramos todas las gráficas de las funciones de distancia.

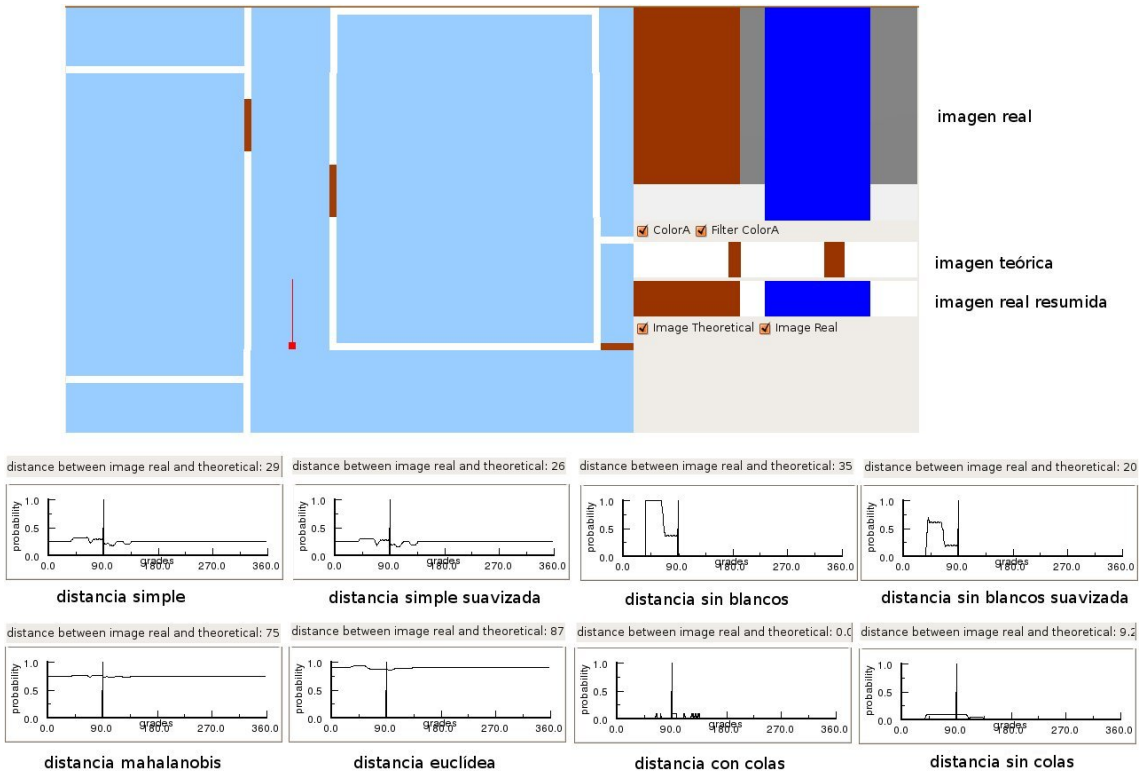

Figura 5.8: Robot colocado en medio del pasillo y mostramos todas las gráficas de las funciones de distancia.

vea una puerta. La función de distancia sin blancos llega a poner la probabilidad al 100 % y esto no es correcto. Las funciones de Mahalanobis y euclídea siguen teniendo probabilidades muy altas y apenas varían en todo θ.
Si Observamos la figura 5.8 la imagen real y las imágenes teórica no deben parecerse mucho porque las dos tienen puertas pero los tamaños de las puertas son muy diferentes, así que las probabilidades tienen que ser bajas. Como en el caso anterior las funciones de distancia sin blancos, de Mahalanobis y euclídea tienen un mal comportamiento.
Analizando los resultados podemos decir que suavizar la imagen teórica no aporta una mejoría, las funciones de distancia sin blancos, de Mahalanobis y euclídea no dan valores aceptables y la función de distancia con colas parece demasiado discriminante. Podemos decir que nuestras posibles funciones de distancia son la distancia simple y la distancia sin colas.

5.2.3. Discriminación espacial de largo alcance

Una vez que hemos visto como se comportan todas las funciones de distancia fijando la x y la y y viendo todos los valores de su orientación. Vamos a ver como se comportan fijando la orientación a un valor fijo y poniendo robots teóricos por todo el mapa visual.

Hemos colocado el robot real delante de una puerta y la papelera y hemos sacado las rebanadas correspondientes a 0 grados, 45 grados, 90 grados, 135 grados, 180 grados, 225 grados, 270 grados y 315 grados. En las figuras siguientes se puede ver los resultados

de cada una de las funciones de distancia.

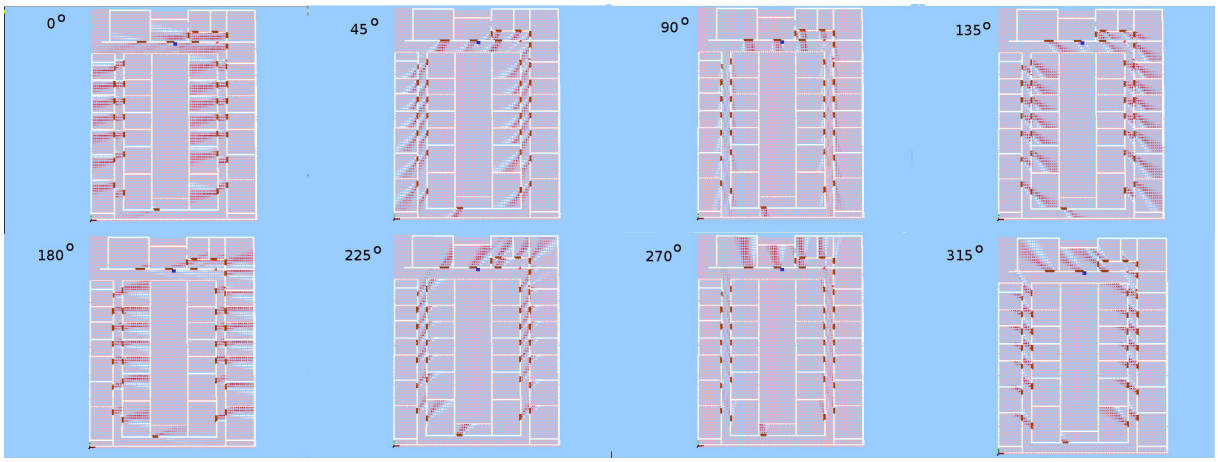

Figura 5.9: distancia simple.

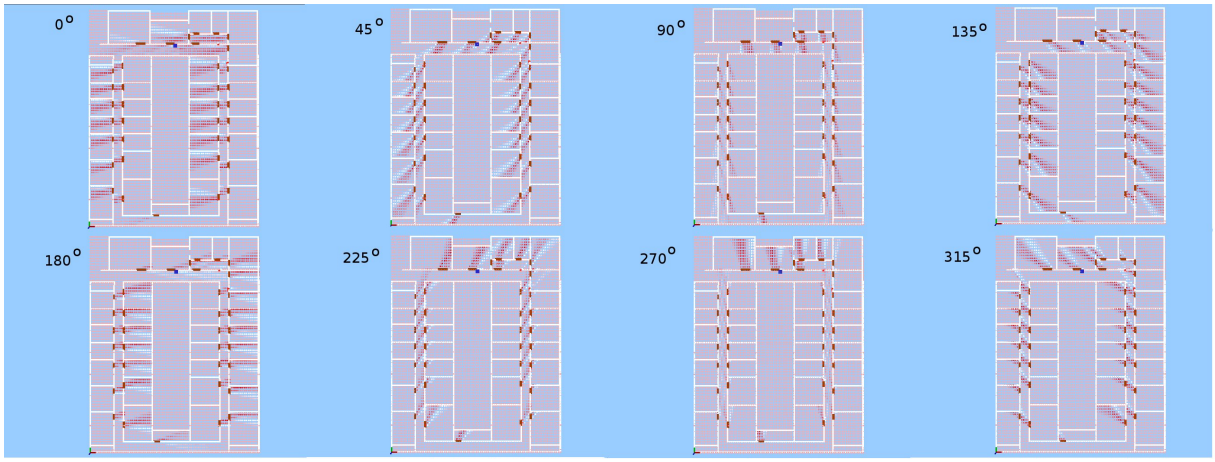

Figura 5.10: distancia simple suavizada.

Como vimos en el experimento anterior el hacer suavizado a las funciones de distancia simple y sin blancos no nos proporciona una mejor solución.
La función de distancia simple en el experimento anterior nos dio buena impresión pero en este no es lo mismo, al comparar también los los píxeles blancos si la imagen real resumida tiene algunos píxeles blancos, la imagen teórica que sea todo blanca tendrá una probabilidad, sera tan grande como píxeles blancos que tengamos en la imagen real resumida, esto no es bueno porque si lo que vemos es muy estrecho las imágenes teóricas totalmente blancas tendrán una probabilidad alta y puede que nos de una localización errónea.
La función de distancia sin blancos da buenos resultados cuando la imagen teórica es toda blanca, pero da valores muy altos cuando solo esta viendo una puerta y esto no es correcto.
Las funciones de distancia euclídea y de Mahalanobis dan como resultado unas

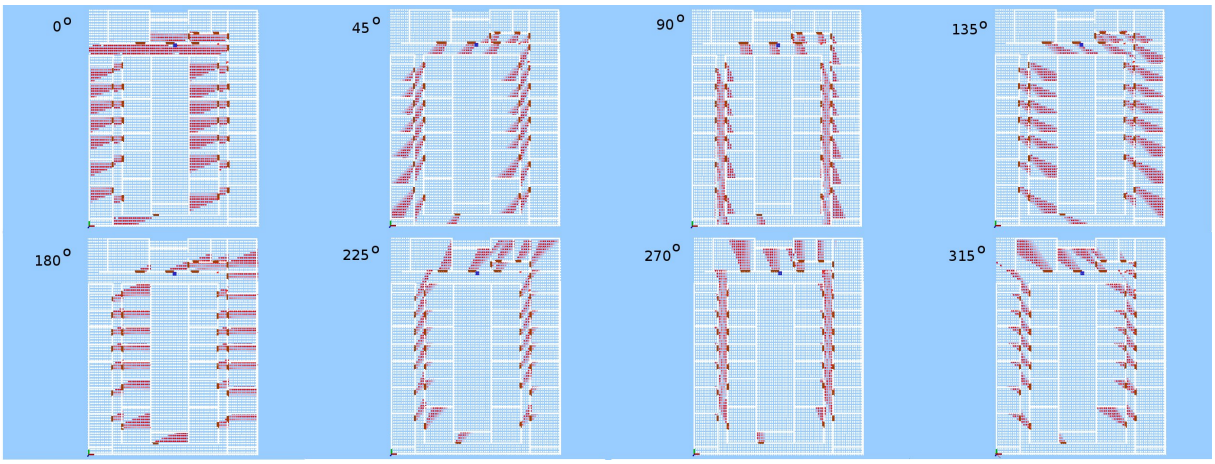

Figura 5.11: distancia sin blancos.

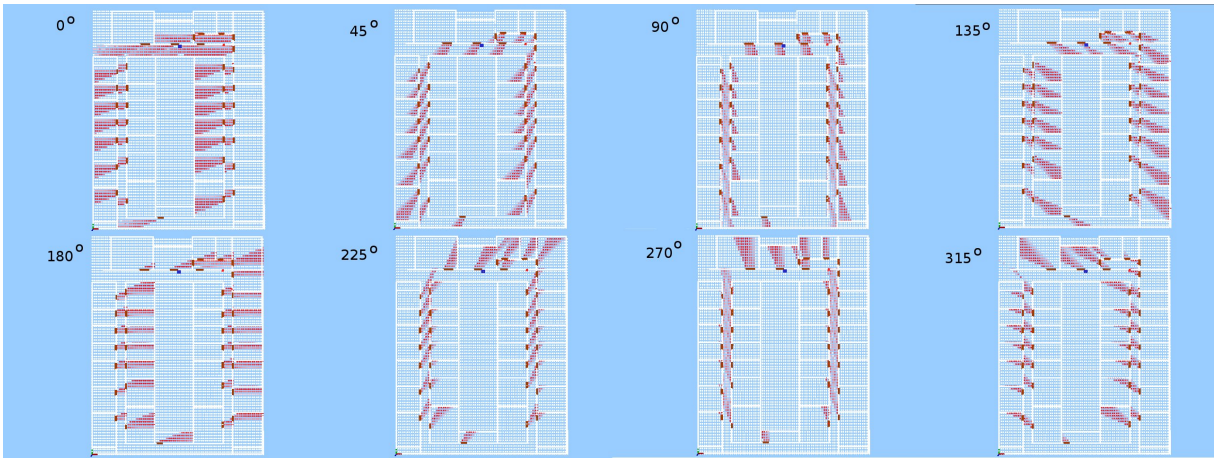

Figura 5.12: distancia sin blancos suavizada.

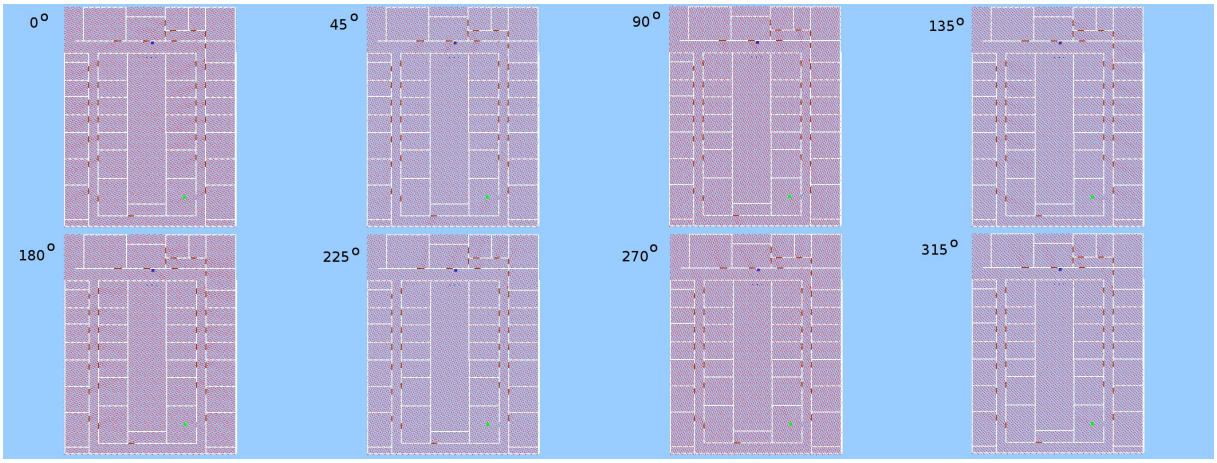

Figura 5.13: distancia Mahalanobis.

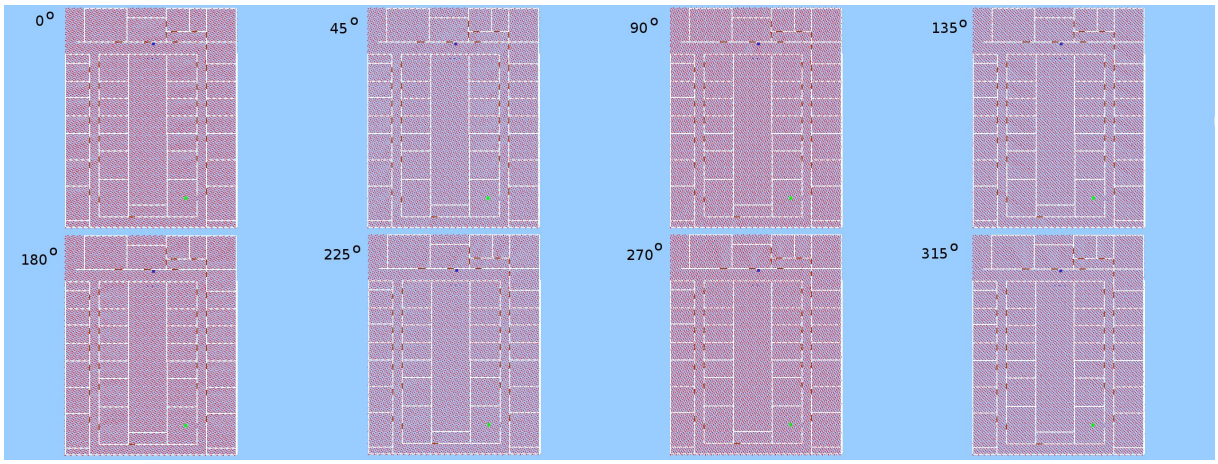

Figura 5.14: distancia euclídea.

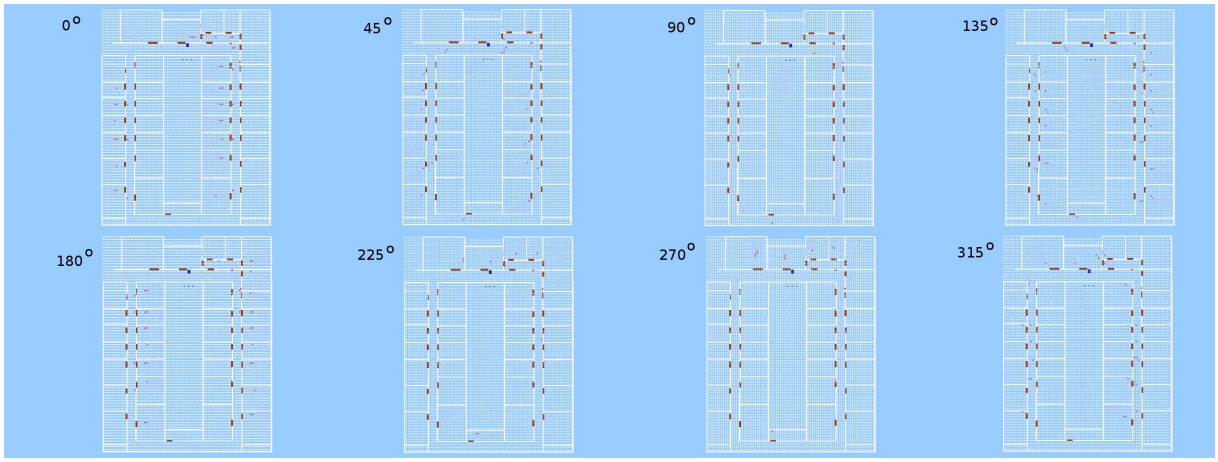

Figura 5.15: distancia con cola.

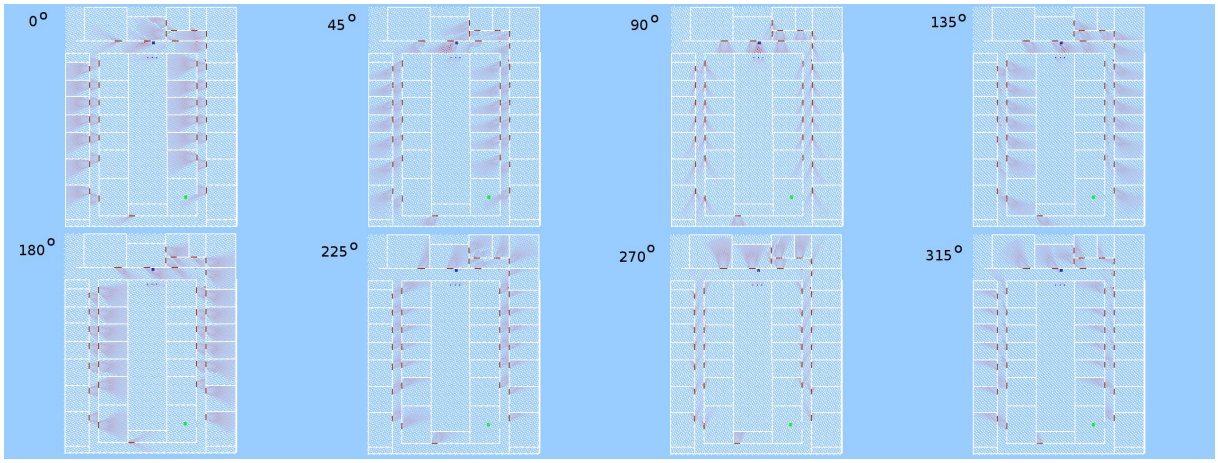

Figura 5.16: distancia sin cola.

probabilidades muy altas y poco diferenciadas así que estas dos distancias no son buenas para una localización
La función de distancia con colas al ser una función muy discriminante solo en muy pocos puntos tiene una probabilidad alta, la localización es posible pero puede perderse con mucha facilidad, además se necesita de muchas partículas ya que es mas difícil que las partículas caigan en posiciones correctas.
La función de distancia sin colas da unos resultados bastante aceptables tiene una discriminación que no es excesiva y a la hora de la localización es más difícil que se pierda.

A continuación mostramos un cuadrado con el tiempo que se tarda en hacer una iteración del modelo de observación con cada una de las funciones de distancia con 1000 partículas.

distancia	simple	simple suavizada	sin blancos	sin blancos suavizada	mahalanobis	euclidea	con cola	sin cola
tiempo	150 ms	165 ms	160 ms	170 ms	180 ms	170 ms	170 ms	200 ms

Figura 5.17: Tabla con los tiempos que se tarda en hacer una iteración del modelo de observación.

Hemos decidido quedarnos como función de distancia la función basadas en objetos llamada sin colas porque lo que buscábamos era que la función de distancia fuera discretizante pero no en exceso, si la imagen teórica y la imagen real resumida no eran parecidas la probabilidad fuera baja y si era parecidas diera una probabilidad alta. También queríamos que fuera ágil, es la que más tarda en ejecutarse pero 200 ms es un tiempo razonable que podemos permitirnos. Creemos que la función de distancia sin colas cumple con los objetivos.

5.3. Convergencia de mallas de probabilidad

El algoritmo de mallas de probabilidad es otro modo de calcular la localización de nuestro robot, hemos explicado su funcionamiento en el capítulo 4 y como dijimos en este, este algoritmo no lo hemos usado para compararlo con el algoritmo de partículas ya que este no es un objetivo de este proyecto. El algoritmo de mallas tiene como objetivo verificar que nuestro modelo de movimiento y nuestro modelo de observación son correctos y al cabo de llevar algunas iteraciones realizando el modelo de movimiento y el modelo de observación mientras el robot se mueve por el entorno, la probabilidad converge y se acumula en la posición real del robot.
El modelo de movimiento se probado sin aplicarle ruido gausiano porque como hemos dicho queremos probar que se converge a la posición correcta y no probar que los *encoders* tienen error. La función de distancia que se ha utilizado es la distancia sin cola, ya que es con la que vamos a probar el algoritmo del filtro de partículas.

Para conseguir este objetivo hemos probado dos recorridos del robot. En el primer recorrido consta de los siguientes pasos:

- Paso 1: Realizamos una observación con el robot real en la posición inicial. Esta posición inicial es en frente de una puerta y la papelera.
- Paso 2: Realizamos un movimiento de 90 grados a la derecha
- Paso 3: Realizamos un desplazamiento de 3500 mm de frente
- Paso 4: Realizamos un movimiento de 90 grados a la derecha
- Paso 5: Realizamos una observación
- Paso 6: Realizamos un movimiento de 90 grados a la derecha
- Paso 7: Realizamos una observación

A continuación mostramos dos figuras. La figura 5.18 muestra la secuencia antes descrita para la rebanada del cubo donde debe de converger el cubo. La figura 5.19 muestra la acumulada, esto quiere decir que se muestra en cada posición la suma de todas las rejillas en esa posición.

Inicialmente obtenemos muchas posibles donde puede estar el robot como se puede ver en el paso 1 y 2 de la figura 5.19. si nos fijamos bien en medio de los pasillos se obtiene más probabilidad acumulada que delante de la papelera que es donde se obtiene el máximo de probabilidad. En medio del pasillo podemos llegar a ver cuatro puertas, aunque la probabilidad de una sola posición no sera mayor del 50 % (por como hemos hecho la función de distancia) tenemos muchas posiciones con probabilidad que se sumaran todas. Delante de la papelera serán probabilidades muy altas pero serán pocas las rejillas que en esa posición tenga probabilidad. Es por este motivo por el que en el pasillo se tiene más probabilidad acumulada que delante de la papelera.En los pasos 3 y 4 de la figura 5.19 se ve que la acumulada en extraña el motivo es que se ha realizado un movimiento en linea recta, hay rejillas que miran para 90 grados, otras para 45 grados, todas tienen una probabilidad, si se mueven con 45 grados y están en el pasillo, la posición final del movimiento ha atravesado paredes y se ha colado en medio del mapa, mientras que la de 90 grados a seguido por el pasillo recta. Cuando se realiza una observación en el siguiente paso todas esas posiciones que han salido por el medio del mapa visual se ponen a 0 y se descartan como posibles posiciones donde puede estar el robot.
Finalmente en el paso 7 podemos ver en la figura 5.19 como llegamos a converger en unas pocas posiciones, y además la posición con la máxima probabilidad es donde se encuentra el robot.

El segundo recorrido tiene los siguientes pasos:

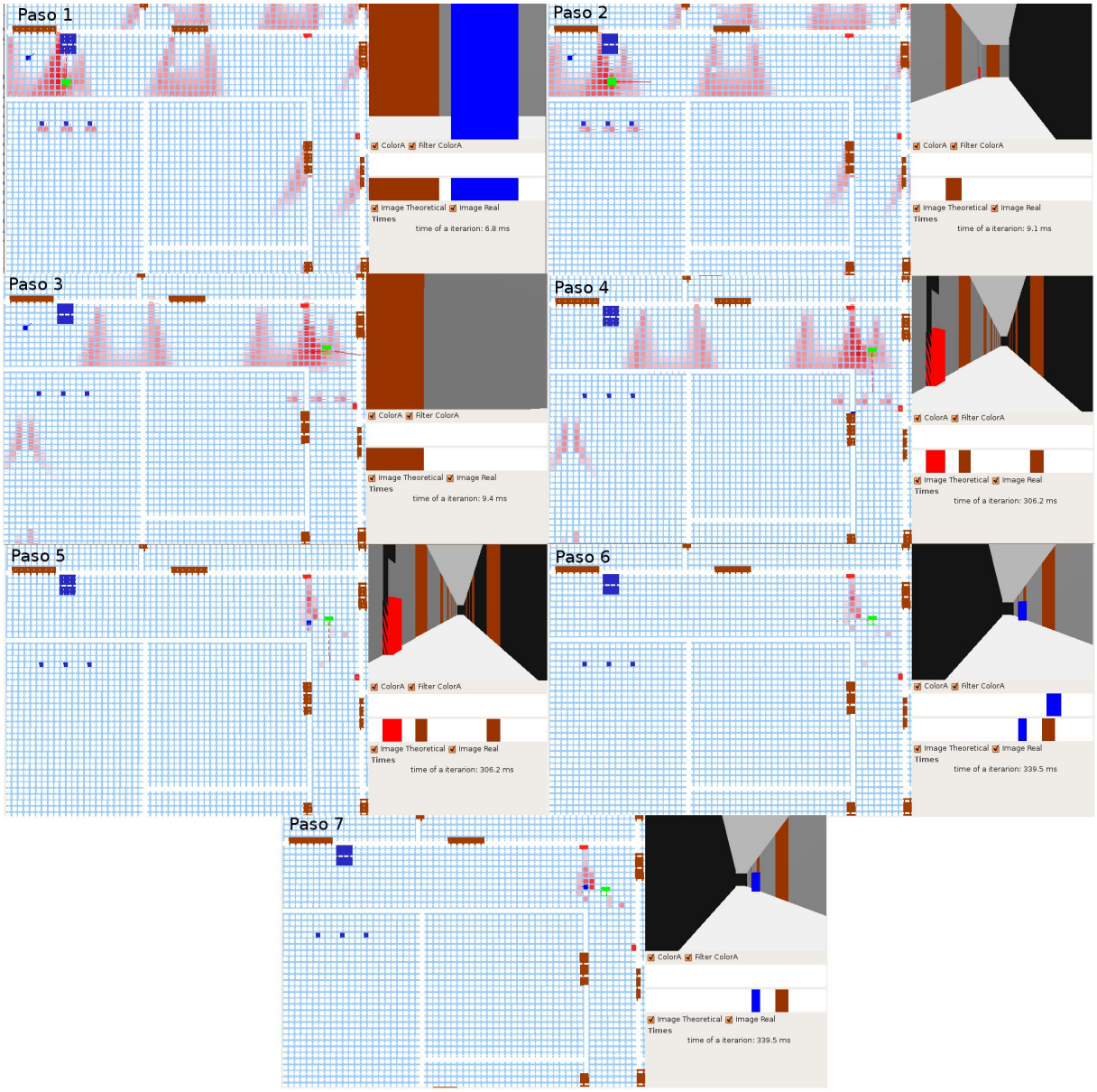

Figura 5.18: ruta 1.

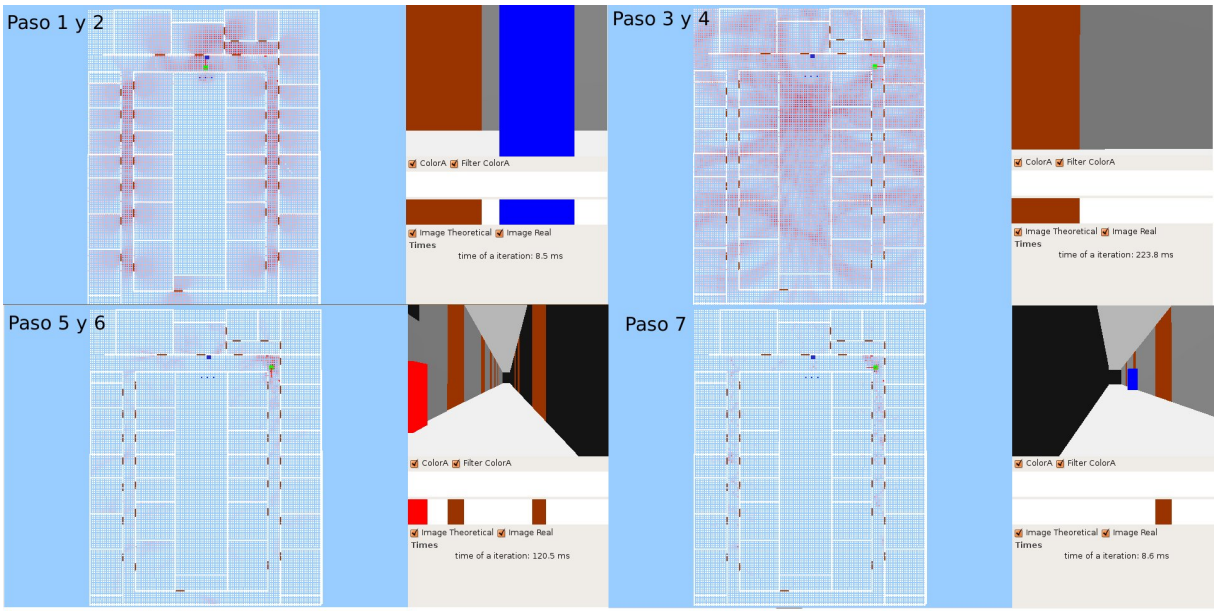

Figura 5.19: ruta 1 acumulada.

- Paso 1: Realizamos una observación con el robot real en la posición inicial. Esta posición inicial es en medio de un pasillo
- Paso 2: Realizamos un desplazamiento de 5000 mm de frente
- Paso 3: Realizamos una observación
- Paso 4: Realizamos un desplazamiento de 5000 mm de frente
- Paso 5: Realizamos un movimiento de 90 grados a las derecha
- Paso 6: Realizamos una observación
- Paso 7: Realizamos un desplazamiento de 3000 mm de frente
- Paso 8: Realizamos una observación

A continuación mostramos dos figuras la figura 5.20 muestra la secuencia antes descrita para la rebanada del cubo donde debe de converger el cubo. La figura 5.21 muestra la acumulada.

Con este segundo recorrido también conseguimos converger, empezando en otra posición y haciendo un recorrido diferente al primero. Hemos mostrado dos recorridos diferentes para que se vea que conseguimos llegar a converger independientemente de donde empecemos. Habrá veces que se necesiten más pasos para conseguirlo pero al final obtendremos un buen resultado.

Como conclusión sacamos que al final se consigue que el algoritmo converja eso significa que nuestro modelo de movimiento es correcto, igualmente que el modelo de observación, pero este algoritmo es inviable para la localización porque para realizar el

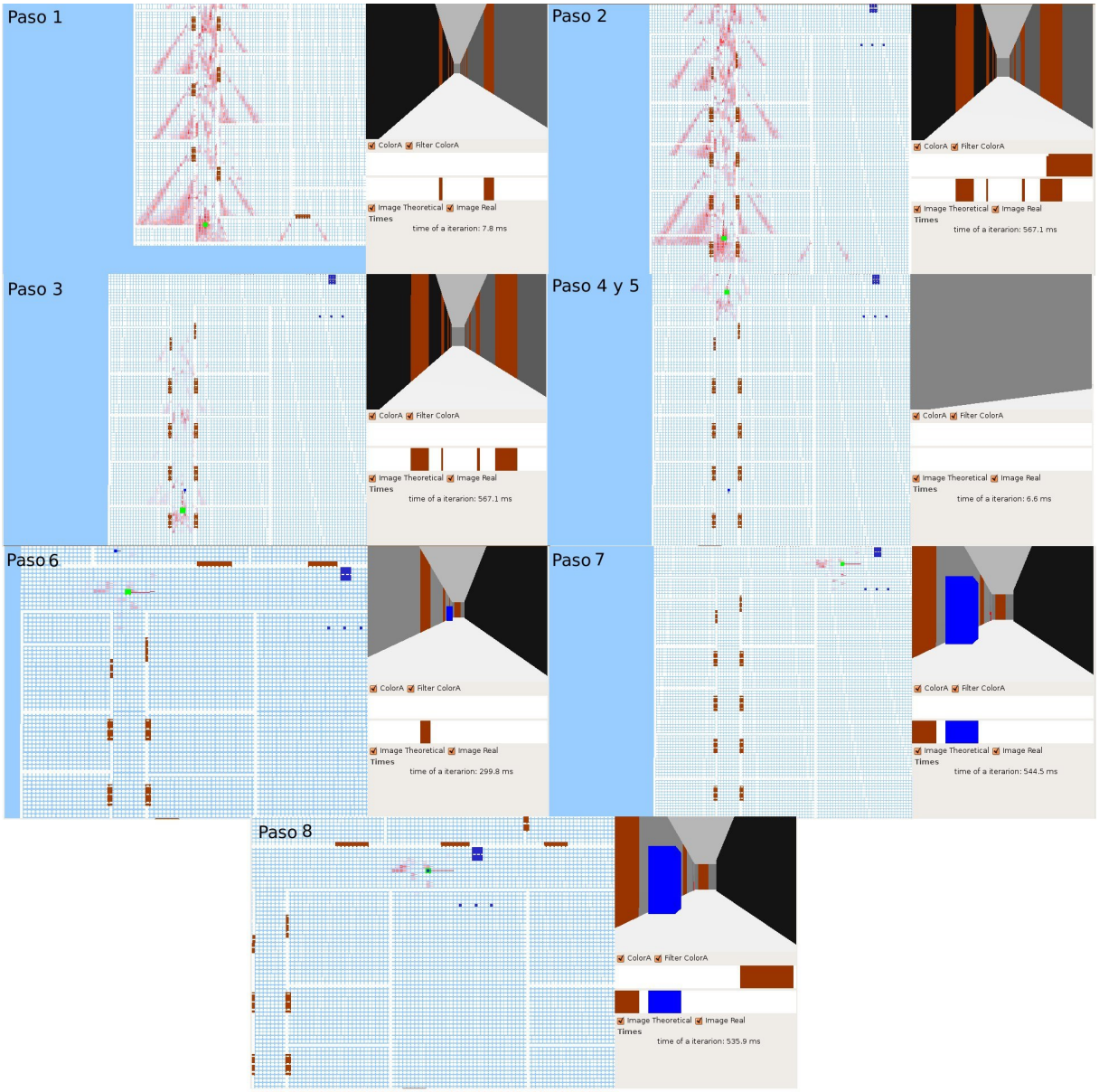

Figura 5.20: ruta 2.

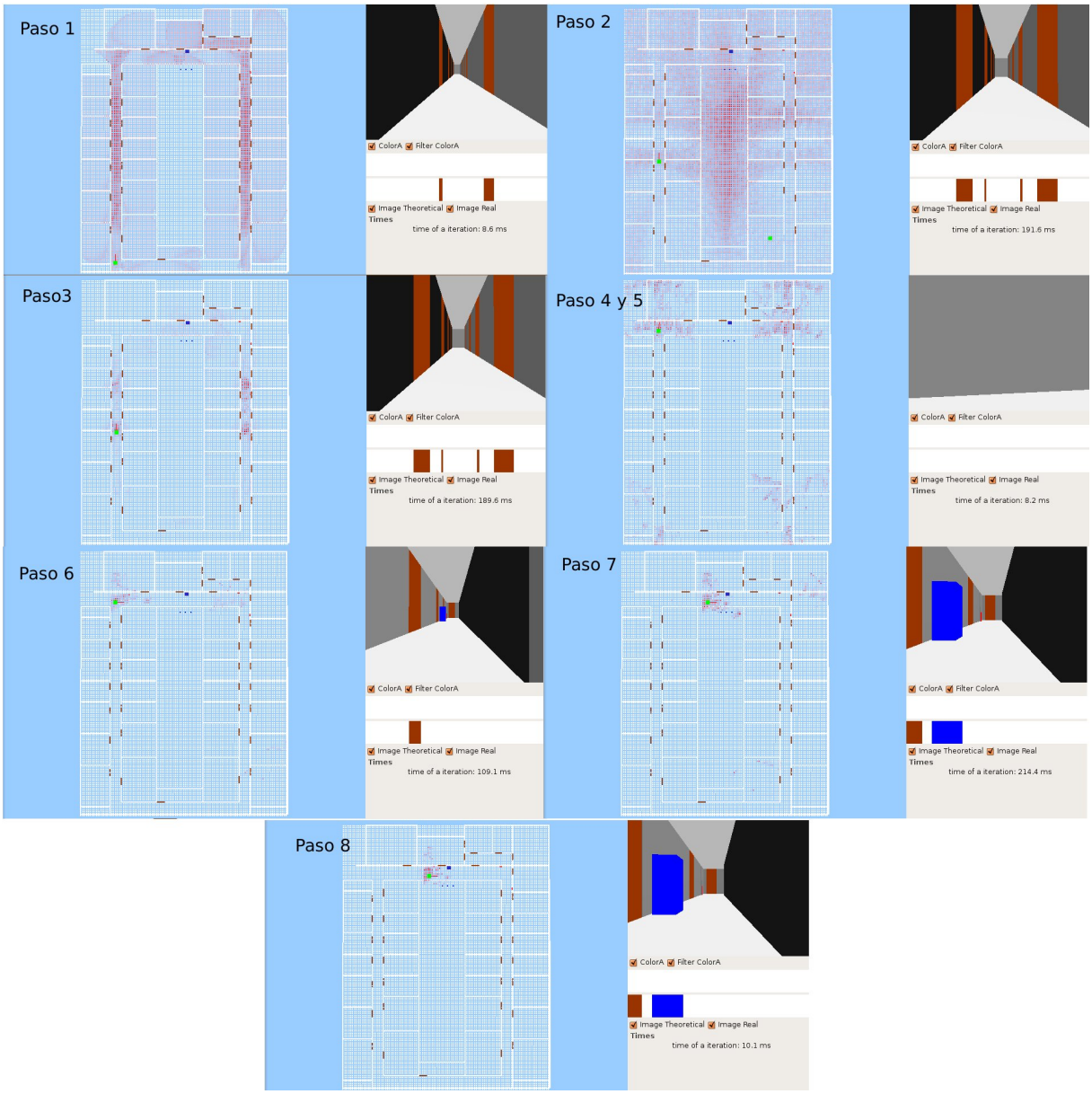

Figura 5.21: ruta 2 acumulada.

primer recorrido se han tardado 17 minutos y para el segundo recorrido se han tardado 22 minutos, y si queremos que nuestro algoritmo sea ágil este no lo cumple.

5.4. Convergencia del algoritmo de partículas

Este es el experimento definitivo una vez que hemos probado el modelo de movimiento, el modelo de observación y el remuestreo por separado ha llegado la hora de explicar cómo funciona el algoritmo de localización basado en partículas.

He seleccionado aquí tres ejemplos de recorridos representativos para que se pueda ver que el algoritmo es robusto, además se han tomado tiempos para ver que es ágil. Todos los ejemplos han sido realizados con 1000 partículas, la función de distancia que usamos es la función sin cola y el robot se lo hemos teleoperado en todo momento, con una velocidad media. Los ejemplos están en formato vídeo junto con el material de este PFC [1].

El primer recorrido es la figura 5.22. Consiste en localizar donde se encuentra el robot mientras nos movemos, se da una vuelta completa al departamental para que se vea que es robusto.
Se inicia desde la papelera y una puerta, al principio se puede ver que todas las partículas están distribuidas uniformemente por todo el mapa y su probabilidad se inicia a cero. El robot se empieza mover, como dijimos en el capítulo 4 se incorpora nueva información cuando el robot se ha movido lo suficiente para que la imagen real anterior y la imagen de ahora sean independientes, hemos decido que sean 300 mm o 15 grados, aquí no se marcan manualmente los instantes en los que se incorporan nuevas observaciones. Es el propio algoritmo el que decide automáticamente cuando hacerlo.
Al principio las partículas se centran en varios puntos del mapa, por ejemplo en la instantánea 3 podemos ver que tenemos dos nubes de partículas, esto muestra que existen dos puntos simétricos. Pero cuando llegamos al medio del pasillo de la derecha, la instantánea 4, ya el algoritmo a convergido en una zona y la nube de partículas sigue al robot.
Un sitio que hay que tener mucho cuidado es cuando llegamos al final del pasillo nos encontramos que la imagen real esta en blanco (no tenemos objetos que vea la cámara, solo ve pared), este es uno de los casos especiales porque estamos usando la función de distancia sin colas y esta tiene tres casos especiales, de los cuales este es uno. En esta caso lo que se hace es que se aplica el modelo de movimiento a cada partículas, se calcula la probabilidad que tiene cada partícula realizando el modelo de observación, si la imagen teórica es todo blanco la probabilidad es uno y si existe algún objeto en la imagen teórica la probabilidad de esa partícula es 0. Para terminar todas las partículas que tenían que su imagen teórica no tenia objetos no se remuestrean, sino que se promocionan para la siguiente población, las partículas que si tenían objetos en su imagen teórica se remuestrean.

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

En los siguientes paso se ve como se sigue moviendo por el mapa localizado el robot y al final llegamos otra vez a la papelera localizado después de a ver dado una vuelta completa al departamental.

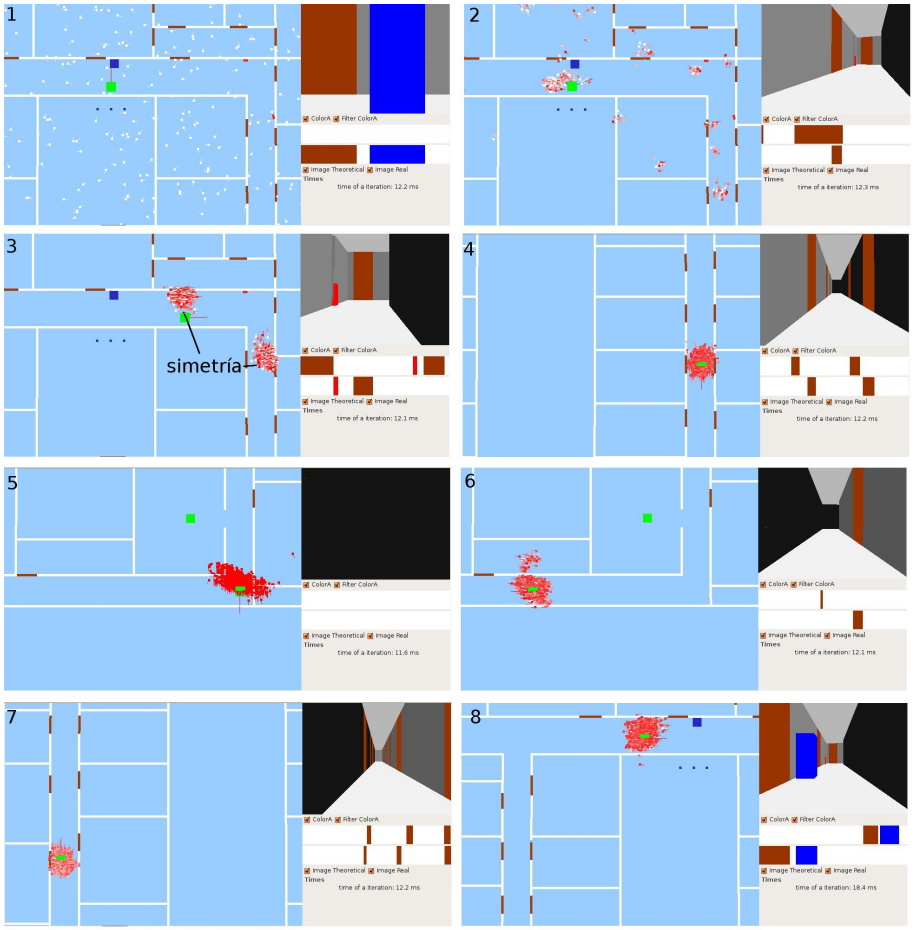

Figura 5.22: algoritmo vuelta completa

El segundo recorrido figura 5.23. Se inicia abajo en el pasillo de la izquierda con todas las partículas distribuidas por el mapa de forma uniforme y con una probabilidad de cero. Seguidamente echamos a andar al robot, tarda un poco más en localizarse porque el punto que que hemos elegido es en medio del pasillo y en el mapa visual existen más punto simétricos, como por ejemplo si observamos la instantánea 3 tenemos tres nubes donde existe simetría.
En este ejemplo también tenemos un punto donde no obtenemos objetos de la imagen real, es la instantánea 6. Como podemos observar conseguimos seguir localizados.
Al final conseguimos que llegue a la papelera localizado.

En el tercer recorrido hemos querido probar el secuestro figura 5.24. Iniciamos en frente de la papelera como en el primer recorrido y conseguimos localizarnos de la misma forma. Llega un punto en medio del pasillo de la derecha y desactivamos el algoritmo de partículas, el robot lo seguimos moviendo por el pasillo, pero sin realizar el algoritmo del filtro de partículas. Cuando llegamos al final del pasillo a la ultima puerta de la

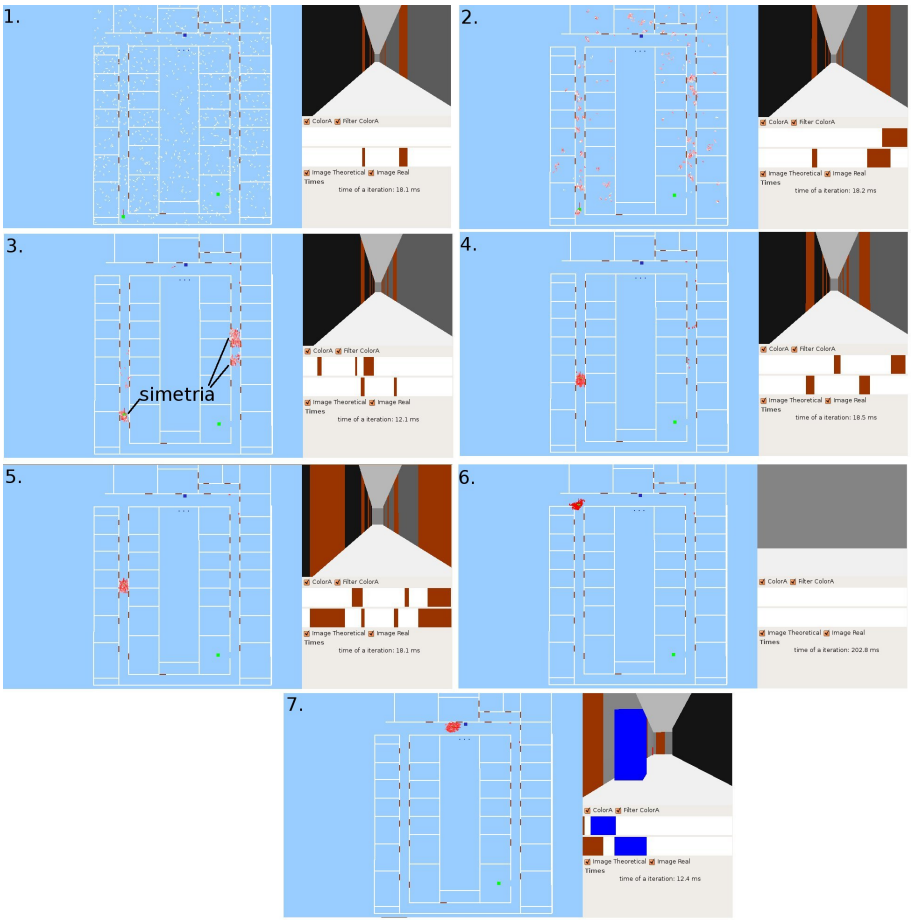

Figura 5.23: algoritmo de pasillo

izquierda nos quedamos mirándola y activamos el algoritmo, al instante el algoritmo se da cuenta que el robot esta mal localizado porque todas las partículas tienen probabilidad cero, así que se produce una degeneración y se reinician las partículas por todo el mapa. El robot se sigue moviendo con el algoritmo de filtro de partículas funcionando, las partículas se vuelven a colocar encima del robot y conseguimos volver a localizarnos.

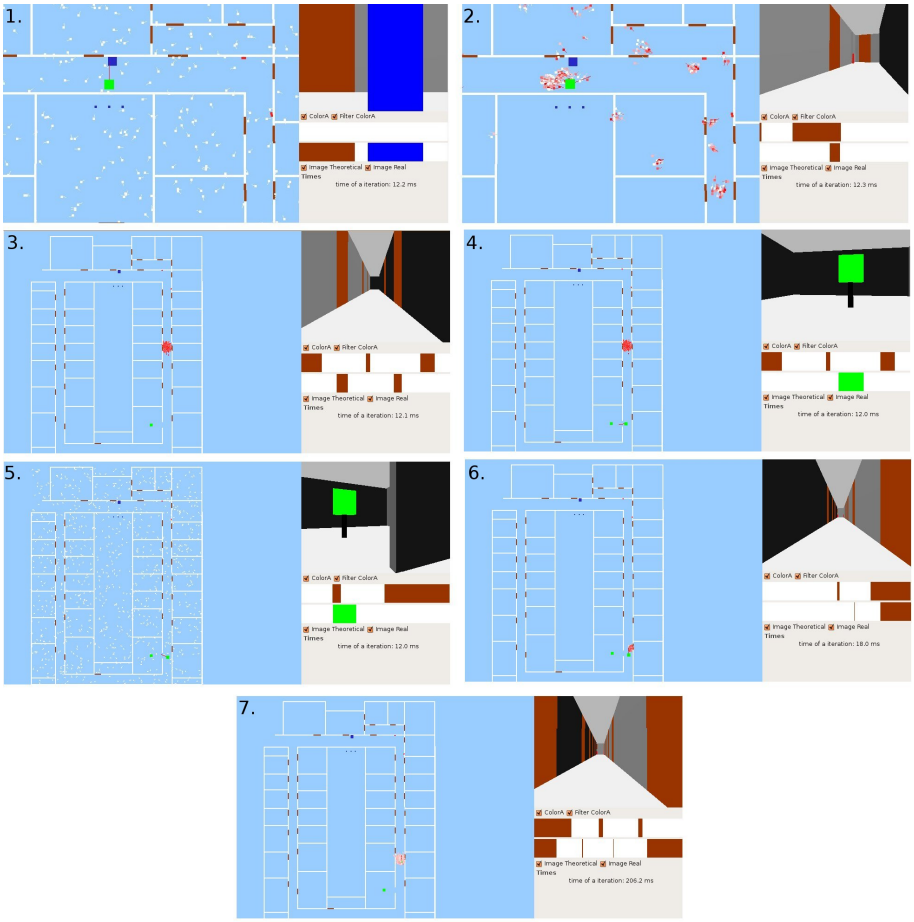

Figura 5.24: secuestro

El tiempo que se tarda en realizar una iteración del algoritmo completo del filtro de partículas con 1000 partículas y con la función de distancia si colas suele variar entre 180 ms y 200 ms. Es razonable y ágil.

En la figura 5.25 se muestra una tabla con lo que se tarda en tener la nube de partículas encima del robot.

recorridos	recorrido 1	recorrido 2	recorrido 3
tiempos	20 s	90 s	23 s

Figura 5.25: Tabla con el tiempo que tardan en converger las partículas.

Como se puede ver los recorrido 1 y 3 son similares es porque el algoritmo se inicia desde el mismo sitio y se hace lo mismo. El recorrido 2 tarda más porque el pasillo es muy simétrico y tienen que avanzar mucho hasta que se queda con la nube correcta.

Capítulo 6
Conclusiones y Trabajos Futuros

En el primer capítulo se dio una visión global de la robótica, se explicó en qué consistía la localización y la localización visual. En el segundo capítulo se definieron los requisitos, el plan de trabajo y los objetivos concretos de nuestro proyecto. En el tercer capítulo se explico la plataforma software y algunas librerías necesarias. En el cuarto se expuso el algoritmo de filtros de partículas que se ha implementado. En el quinto capítulo se explicaron los diferentes experimentos que se han llevado a cabo para la validación del algoritmo de localización implementado en el capítulo anterior. En el sexto y último capítulo se repasaran las conclusiones que se pueden sacar de este proyecto, así como las futuras líneas a seguir.

6.1. Conclusiones

La conclusión general que podemos sacar es que hemos desarrollado un sistema de localización probabilística robusto,validado experimentalmente, en el entorno del departamental II con el simulador gazebo.

De esta conclusión podemos decir que se han cumplido los objetivos expuestos en el capítulo 2 de esta memoria.

- *Cálculo de la imagen real y teórica.* Para la imagen real lo que se ha hecho ha sido aplicar un filtro HSV y después se ha resumido la imagen a 320x1. La imagen teórica se ha calculado, sacando el ángulo de visión mediante progeo y obteniendo una imagen a partir del mapa visual y de la posición de la partícula (x,y,θ), el tamaño de esta imagen es de 320x1. El motivo de que sea de 320x1 las dos imágenes es porque es menos pesado computacionalmente de tratar que las imágenes como 320x240.
- *Cálculo de la función distancia.* Se ha realizado un estudio de diferentes funciones de distancia. Primero mirando su comportamiento para una posición x e y determinada y todo su espacio θ, después mirando para una posición θ determinada y todas las posibles posiciones de x e y. Al final se ha llegado a la conclusión que la función distancia sin colas es la mejor para la localización.
- *Experimentos.* Se han realizado muchos experimentos. Se han realizado experimentos para ver lo parecidas que eran la imagen real y la imagen teórica.

Se han realizado experimentos para ajustar parámetros del algoritmo de filtro de partículas. Se han realizado experimentos para ver la convergencia con el algoritmo de mallas de probabilidad y el algoritmo de filtro de partículas. Para comprobar los robusto que es nuestro algoritmo de filtro de partículas se ha probado el secuestro, que ha funcionado bien, además se han realizado varios recorridos para comprobar la robustez y se ha medido el tiempo que se tardaba en localizar para ver lo ágil que era.

Estos objetivos tenían una serie de requisitos que condicionaban el desarrollo del proyecto. Veamos si se han satisfecho estos requisitos y en que medida.

- *Desarrollar el comportamiento usando la plataforma JDEROBOT.* Este requisito impone que el lenguaje de desarrollo es C o C++, así como la programación es esquemas. El algoritmo de localización ha sido desarrollado en C y se compone de un esquema llamado *visual_loc* que ha ocupado un total de 4000 líneas de código.

- *Desarrollar Bajo el sistema Linux.* Todo el desarrollo se ha llevado a cabo bajo el sistema GNU/Linux, concretamente sobre la distribución Ubuntu 8.04.

- *La solución final ha de funcionar sobre el simulador Gazebo.* Es un requisito importante porque el siguiente paso a dar sera probarlo en robot real y si conseguimos una localización robusta en simulador el paso a robot real sera más sencillo. Se ha desarrollado un mundo del departamental II en Gazebo, de él se ha cogido la imagen de la cámara, los encoders y los lasers, que se han pasado a nuestro esquema y con ellos se ha realizado la localización.

- *El comportamiento del algoritmo ha de ser vivaz.* Este requisito tiene el mismo objetivo que el anterior, que al final se pueda probar este algoritmo en un entorno real. Por este requisito se ha hecho que el algoritmo de localización sea muy pesado y en pocas iteraciones, seamos capaces de dar una localización correcta.

- *La Localización ha de ser robusta.* Este ultimo requisito es muy importante ya que este punto básico al que se quiere llegar con este proyecto. Se han hecho pruebas de localización iniciando esta en varias posiciones, completando circuitos largos (una vuelta completa al departamental) y probando que el secuestro funciona.

Haciendo un balance global podemos decir que con este proyecto hemos aportado un sistema de localización visual, que hemos intentado que sea robusto y rápido, para ello hemos utilizado el algoritmo de filtro de partículas siguiendo el método de MonteCarlo. Para esto nos hemos apoyado sobre todo del proyecto de Alberto Lopez [López, 2005], del cual cogimos su código como inicio del nuestro, y también nos hemos apoyado de algunas ideas del proyecto de Angel Cortés [Cortés, 2007].

Los resultados de este proyecto, documentación, código, vídeos y demás material se puede encontrar en [1]

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

6.2. Trabajos Futuros

El primer camino y el más sencillo e interesante, sería conseguir una localización robusta con este algoritmo en el robot real Pioneer. Tendríamos que hacer un estudio parecido al realizado con el simulador. Primero tendríamos que comprobar que la imagen real y la imagen teórica en una posición del robot son similares. Segundo tendríamos que comprobar como se comporta la función distancia en el mundo real. Y tercero y ultimo tendríamos que probar el algoritmo de filtro de partículas haciendo pruebas por el el departamental II.

Otro camino y que vendría siguiendo al anterior, sería conseguir integrar la localización visual robusta en el proyecto robot guía que se esta desarrollando en el grupo de investigación de Robótica. Con esta localización robusta daríamos un paso muy importante en este proyecto ya que a parte de conseguir una localización correcta tendríamos una buena navegación global por todo el departamental II.

Bibliografía

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

[Baena, 2003] Alfonso Matute Baena. Filtro de color configurable. *Proyecto Fin de Carrera, URJC*, 2003.

[Barrera *et al.*, 2005] Pablo Barrera, Jose María Cañas, y Vicente Matellán. Visual object tracking in 3d with color based particle filter. In *Int. Journal of Information Technology*, 2005.

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

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

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

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

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

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

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

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

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

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

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