Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

ÁREA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, DE COMPUTADORES Y SISTEMAS

TRABAJO FIN DE GRADO Nº 1601_043

ROBOT GUIA PARA EL ACOMPAÑAMIENTO DE PERSONAS EN INTERIORES

Dña. Iria María AYARZA MIRA
TUTOR: D. Rafael Corsino González de los Reyes

FECHA: Julio 2016

Tabla de contenido

1. Hipótesis de partida y alcance.

El ámbito de la robótica se está desarrollando velozmente en los últimos años, por lo que cada vez es posible una mayor variedad de aplicaciones. En un principio los robots fueron desarrollados para usarse en procesos industriales, pero en la actualidad tienen multitud de aplicaciones, destacando aquellas cuyo objetivo es el de ayudar o asistir al ser humano.

Los robots de asistencia ayudan a completar tareas de las cuales no se quiere depender o que no se pueden desarrollar adecuadamente sin ayuda externa. Las aplicaciones de este tipo de robots pueden cubrir campos tan diversos como: domótica (seguridad, gestión energética, bienestar y comunicaciones), tareas del hogar (limpieza, cocina, jardinería, cuidado de niños), sanidad (rehabilitación, intervenciones quirúrgicas, traslación de enfermos, dispensación de medicamentos, investigación en laboratorios), guiado de personas (para invidentes, en museos), entre otros. Aunque la mayoría de estas aplicaciones para robots se encuentran aún en estado de investigación y desarrollo, en la actualidad ya se pueden encontrar robots funcionales que las desempeñen.

En espacios interiores de grandes dimensiones (como podría ser un hospital o una facultad) puede resultar difícil orientarse y llegar hasta la ubicación deseada (según el ejemplo anterior: una consulta, un despacho o aula), es por ello, que tener la posibilidad de disponer de un robot guía particular que se desplace a través del lugar evitando los obstáculos (otras personas u objetos) y acompañando al usuario hasta dicha ubicación es un concepto interesante para investigar.

En base a lo anteriormente mencionado, en este documento se desarrolla el diseño de una aplicación robótica para el guiado de personas en interiores. Esto se logrará mediante la simulación de un robot móvil, que no solo guiará a la persona hasta el lugar deseado, sino que, usando técnicas basadas en visión por computador y una cámara Kinect, adaptará su desplazamiento al de la persona.

2. Objetivos concretos y relación con el estado actual.

El objetivo principal de este proyecto consiste en desarrollar una aplicación mediante simulación, usando el entorno de trabajo conocido como ROS (Robotic Operating System), que permita al robot realizar un correcto desplazamiento por el entorno, evitando los obstáculos que surjan en tiempo real, hasta la ubicación que se le haya indicado sin perder de vista a la persona que está acompañando, lo cual, se consigue determinando la posición relativa de la persona respecto al robot y manteniendo este valor dentro de unos márgenes admisibles.

Para poder desarrollar una aplicación propia, ha sido necesario investigar el estado del arte actual de los robots guía y así estudiar las soluciones propuestas hasta el momento.

2.1. Estado actual de los robots guía.

Un robot guía se puede definir como aquel cuya función es la de guiar a través de un entorno, a una persona o grupo de personas hasta una cierta ubicación. Esta ubicación o destino puede ser de dos tipos: puede pertenecer a una ruta preestablecida con varios destinos, o puede ser única, es decir, ir del punto A al punto B (figura 2.1.).

En la actualidad existen diversos modelos de robots de asistencia al ser humano cuya aplicación principal es el guiado de personas, siendo los que más se aproximan a la solución que se busca en este proyecto, los que se encuentran, principalmente guiando en museos. A continuación, se expondrán brevemente los más relevantes para esta investigación.

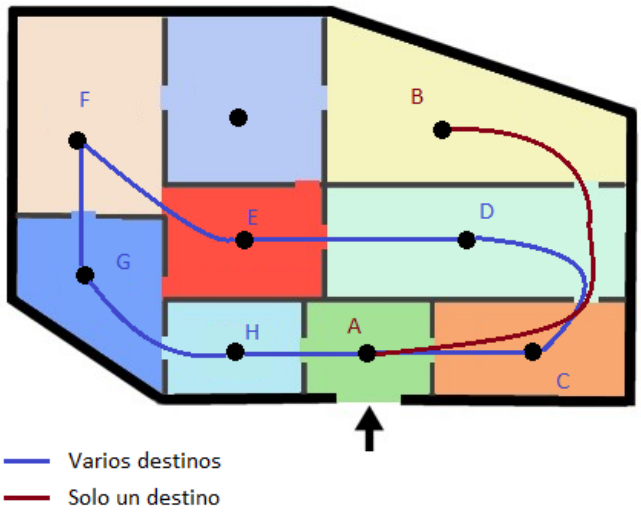


Figura 2.1. Tipos de rutas posibles para un robot guía.

Los modelos conocidos como RHINO [1] (modelo de 1ª generación) y MINERVA [2] (modelo de 2ª generación, versión mejorada del anterior) son robots guía diseñados para museos, han sido desarrollados por la Universidad de Bonn en Alemania y la Universidad Carnegie Mellon en Pittsburg, Estados Unidos.

Ambos son robots guía autónomos que realizan distintos tours[1] por el museo en el que se encuentran, evitando la colisión con los objetos y la gente y que, además, permiten la interactuación con las personas mediante una interfaz compuesta por una pantalla y varios botones. Tienen programados varios tours con rutas preestablecidas que el usuario puede elegir gracias a esta interfaz.

En la figura 2.2. se pueden ver los modelos RHINO y MINERVA (el primero en el Museo de Bonn y el segundo en el Museo Nacional de Historia Estadounidense, en Washington DC), realizando la acción de guía turístico e interactuando con la gente.

[1] Proviene del francés y significa gira o vuelta.

Figura 2.2. Modelos RHINO (izquierda) y MINERVA (derecha).

Se pueden encontrar modelos españoles (aun en desarrollo) como FROG [3], fruto de un proyecto internacional financiado por la Unión Europea, en el que ha participado la Universidad de Sevilla.

Este robot también realiza tours guiados, pero en este caso está pensado para hacerlo incluso en espacios abiertos (figura 2.3.). Es un robot interactivo, ya que es capaz, mediante un sistema de reconocimiento facial, de detectar si la persona se aburre o no está atendiendo, de modo que puede cambiar el tono de su discurso o, simplemente, cortar y cambiar de tema. Como interfaz con el usuario dispone de una pantalla táctil.

Figura 2.3. Modelo FROG en el Real Alcázar de Sevilla (derecha) y en el zoo de Lisboa (izquierda).

El kTBot [4] es un modelo funcional, desde 2012, de robot de servicio desarrollado por la Fundación Tekniker, en Guipúzcoa.

Es más genérico que los anteriores, que estaban diseñados para realizar tours en museos (o similares), este modelo está pensado para interiores con una gran presencia y tránsito de personas como: hogares, residencias, hospitales, museos, oficinas, lugares de trabajo, etc. Es capaz de desplazarse autónomamente evitando los obstáculos y de interactuar con el usuario mediante el reconocimiento de voz y de gestos.

Figura 2.4. Modelo funcional kTBot.

Por último, se ha estudiado un prototipo de robot de servicio para el guiado de personas [5] desarrollado por varios estudiantes de la Universidad Politécnica de Valencia. Este prototipo propone una solución en la que el robot además de navegar autónomamente por el entorno, en este caso un hospital, mediante técnicas de visión por computador reconoce a la persona que debe acompañar a su correspondiente consulta (con cita previa). La detección del paciente la realiza buscando coincidencias en una base de datos con las fotografías de los pacientes que tienen cita. Incluye un módulo que permite al robot disminuir la velocidad o detenerse si se aleja del paciente.

Una vez conocidos los modelos existentes que más se acercan a la solución buscada, se puede apreciar que todos ellos tienen en común la navegación autónoma, por tanto, es el primer objetivo que es necesario solucionar. Así mismo, solo se ha encontrado un modelo (prototipo) que adapte sus desplazamientos a los de la persona, como se pretende en este proyecto. Aunque en dicho prototipo la detección de las personas a las que puede acompañar se reduce a los pacientes registrados, en el caso de este proyecto se pretende hacer una

aplicación más genérica, que pueda detectar a toda persona que necesite del robot guía, no solo aquellos que estén registrados en una base de datos.

Tras analizar el objetivo principal del proyecto, es evidente la existencia de varios sub-objetivos que se deben solucionar para que sea posible el desarrollo la aplicación deseada. Se plantea a continuación cada objetivo concreto por separado:

- La **navegación autónoma** del robot en el entorno (interior de un edificio) en el que se encuentre.
- La **adaptación del movimiento** del robot al de la persona a la que va a acompañar.

Por tanto, el siguiente paso es el estudio de las soluciones a los puntos anteriormente mencionados.

2.2. Estudio de la navegación autónoma.

Se define la navegación autónoma [6] como la metodología que permite guiar la trayectoria de un robot móvil a través de un entorno con obstáculos. Las tareas involucradas en la navegación son: la percepción del entorno a través de los sensores del robot, de modo que le permita crear una abstracción del mundo (mapa); la planificación de una trayectoria libre de obstáculos para alcanzar el punto de destino y el guiado del robot a través del mapa construido. Por tanto, el problema de la navegación se puede dividir en las siguientes etapas:

- Percepción del mundo: Mediante el uso de sensores externos (proporcionan información del entorno del robot) se crea un mapa del entorno donde se desarrollará la tarea de navegación.
- Planificación de la ruta (o trayectoria): Crea una secuencia ordenada de puntos que deben ser alcanzados por el robot.
- Generación del camino: En primer lugar, define una función continua que interpola la secuencia de puntos construida por el planificador. Posteriormente procede a la discretización de la misma a fin de generar un camino.

- Seguimiento del camino: Efectúa el desplazamiento del robot, según el camino generado, mediante el adecuado control de los actuadores del robot.

En la figura 2.5. se puede observar el algoritmo de navegación básico, el cual parte de un mapa del entorno y de las especificaciones del punto de destino de la navegación. Con estos datos se realiza la planificación de un conjunto de puntos que definen la ruta que debe seguir el robot.

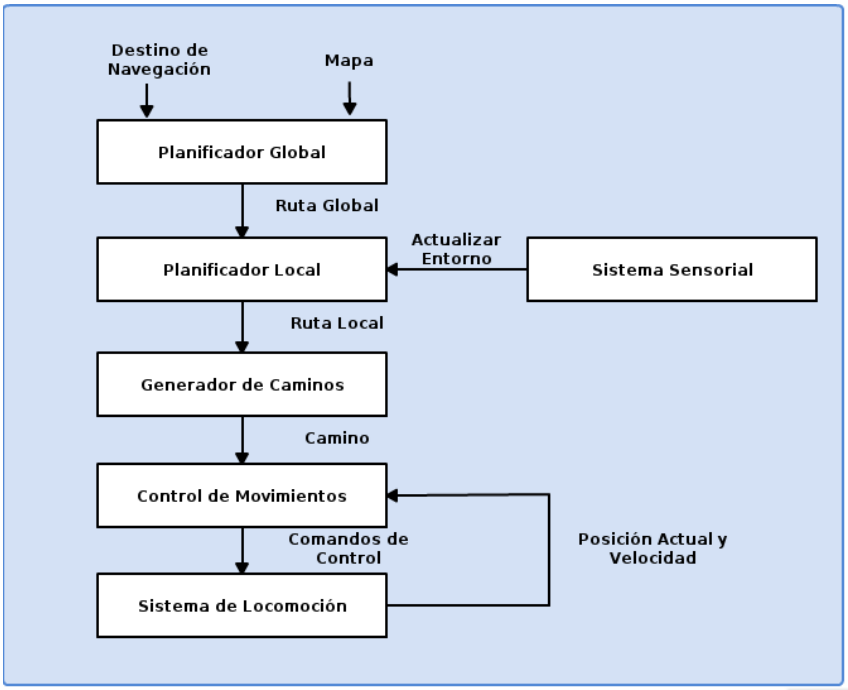


Figura 2.5. Algoritmo de navegación del robot móvil Blanche de AT&T [7].

La tarea de planificación del movimiento se desdobla en dos subtareas: la *planificación global* y la *planificación local*:

- La planificación global construye una ruta sobre la cual se puede definir un camino libre de obstáculos según la información que a priori se posee del entorno (mapa). Si la descripción del entorno introducida fuese perfecta, la ruta calculada sería de forma directa la que pasase al generador de caminos. Sin embargo, al no serlo, puede dar lugar a la construcción de un camino que no esté completamente libre de obstáculos, con el consiguiente peligro de que el robot impacte con algún elemento del entorno.

- La planificación local recibe información del sistema sensorial sobre el entorno local del robot. Mediante el análisis de estos datos se actualiza el modelo preliminar del entorno y decide si se precisa replanificar la ruta local del robot.

Estas tareas en conjunto se aseguran de que la *ruta esté libre de obstáculos* y envían esta información al generador de caminos (figura 2.5.), que genera la referencia que usará el controlador de movimientos para enviar los comandos de control (dirección y velocidad) que actuarán sobre el sistema de locomoción del robot.

Mediante el uso de los sensores internos (integrados en la propia estructura mecánica del robot y que dan información del estado del robot (posición y velocidad) en conjunción con técnicas de odometría, se realiza una estimación de la posición y velocidad actual, que serán realimentadas al controlador de movimientos. Para entender esto, es necesario saber que la odometría es el estudio de la estimación de la posición de un robot móvil durante su navegación, lo que se consigue usando información sobre la rotación de las ruedas del robot para estimar cambios en la posición a lo largo del tiempo.

Tras analizar esto se observa que, para lograr una correcta navegación hay que resolver varios aspectos: el *mapeo*, creación de un mapa, ya que inicialmente no se dispone de uno; la *localización*, posición y orientación del robot en cada instante de tiempo; la *planificación del movimiento*, determinar la ruta que permita desplazar al robot desde su posición inicial a otra final sin colisionar con los obstáculos.

2.2.1. Mapeo.

La navegación autónoma más eficiente es aquella que parte de un mapa previo del entorno en el que se va a desempeñar, por este motivo es tan importarte resolver el problema del mapeo. Pero a la hora de construir este mapa es necesario que el robot sepa cuál es su posición y orientación en cada instante, es decir, su localización. Por ello el problema del mapeo y la localización van de la mano y deben de ser resueltos simultáneamente.

Para solucionar este propósito existe una técnica conocida como SLAM [8] (Simultaneous Localization And Mapping, en español Localización y Mapeo Simultáneos).

Esta técnica usada por el robot para construir un mapa del entorno desconocido en el que se encuentra, a la vez que estima su trayectoria al desplazarse dentro del mismo. Para ello emplea como herramienta el robot móvil y el conjunto de sensores y actuadores que éste posee.

El ruido presente en los sistemas sensoriales, los inevitables errores y aproximaciones cometidos en los modelos empleados, y la dificultad representativa de los entornos a medida que éstos aumentan en complejidad, hacen que la tarea de resolver el mencionado problema sea ardua.

Las soluciones que mejores resultados han proporcionado a la hora de abordar el problema del SLAM son aquellas basadas en técnicas probabilísticas, cuyos algoritmos tienen su base en el teorema de Bayes.

El principal obstáculo a la hora de modelar un entorno desconocido, utilizando un robot móvil para su exploración es la inevitable incertidumbre que se deriva de la imperfección de los sensores y modelos empleados. Si los sensores fueran perfectos, proporcionarían medidas absolutamente precisas de los objetos detectados, que podrían ser insertados en un mapa en su posición exacta respecto a un sistema de referencia ligado al robot. Del mismo modo, al desplazarse el robot, una medida exacta del giro de sus ruedas (o del avance de sus “patas”), combinada con un modelo exacto de su movimiento, permitirían determinar exactamente la posición del robot cuando este realizase una nueva medida. Así sería posible construir incrementalmente un modelo perfecto del entorno.

Desafortunadamente esta perfección no puede ser conseguida con la tecnología de la que se dispone a día de hoy. Se impone por tanto la necesidad de acomodar estas incertidumbres en las soluciones. La forma más evidente de hacerlo es considerar que tanto la posición del robot, como los elementos que modelan su entorno, son variables aleatorias. Así, los algoritmos existentes modelan ambos de manera probabilística, y utilizan métodos de inferencia para determinar aquella configuración que es más probable teniendo en cuenta las medidas que se van obteniendo.

Para dos variables aleatorias, x y d, el Teorema de Bayes establece de manera muy compacta lo siguiente:

$$p(x|d) = \frac{p(d|x)\ p(x)}{p(d)} \tag{2.1}$$

La ecuación 2.1 constituye un mecanismo básico de inferencia: suponiendo que se quiere obtener información acerca de la variable x (por ejemplo, el estado de un sistema compuesto por un mapa y un robot) basándose en la información contenida en otra variable d (un conjunto de medidas adquiridas por un sensor). La anterior regla indica que este problema se puede resolver simplemente multiplicando dos términos:

- El modelo generativo $p(x|d)$, que expresa la probabilidad de obtener la medida d bajo la hipótesis expresada por el estado x.

- El grado de confianza que damos a que x sea precisamente el caso antes de recibir los datos, $p(x)$.

Es importante el hecho de que el denominador de la ecuación anterior no depende de la variable que pretendemos estimar, $p(d)$, por lo que se suele escribir como un factor de normalización.

Las principales soluciones que se usan en el SLAM son: el filtro extendido de Kalman y los mapas de ocupación de celdillas. Estos métodos se revisarán brevemente a continuación:

- **Filtro Extendido de Kalman.**

Es una de las soluciones más extendidas al problema de la localización y mapeo simultáneos, y también es una de las que mejores resultados suele proporcionar en la práctica. A esta categoría de soluciones, cuyo fundamento enraíza en los trabajos introductorios realizados por Randall Smith, Matthew Self y Peter Cheeseman a finales de la década de 1980, se la conoce generalmente como SLAM-EKF (SLAM - Extended Kalman Filter).

Por su propia naturaleza, el SLAM-EKF requiere disponer de un mapa en el cual las entidades que lo componen sean fácilmente parametrizables. Esto es, los elementos que componen el mapa deben poder ser descritos utilizando un conjunto de parámetros que encajen de forma sencilla en el estado del sistema.

A la vista de las hipótesis realizadas, cabe enumerar las siguientes desventajas de esta clase de algoritmos:

- La premisa de partida del algoritmo, que supone una distribución Gaussiana para el estado del sistema, puede no corresponderse con la realidad. En el mejor de los casos, las linealizaciones introducidas en los modelos harán que las estimaciones de los momentos de esta distribución degeneren a lo largo del tiempo no correspondiéndose con sus auténticos valores.

- El punto anterior indica además un grave problema de consistencia del algoritmo, que hace que el nivel de confianza de la estimación obtenida no se corresponda con el auténtico error cometido. Esto origina que la exactitud de los resultados obtenidos por el filtro sea a menudo impredecible, observándose saltos bruscos en la estimación sin causa aparente alguna.

- El coste computacional crece al cuadrado con el número de objetos contenidos en el mapa. Este hecho limita su aplicación en tiempo real a mapas formados por unos pocos objetos.

- No siempre es sencillo o inmediato extraer características del entorno asimilables a una clase particular de objeto. En ocasiones ni siquiera es preciso extraer información que pueda describirse empleando formas geométricas simples como puntos, segmentos, arcos de circunferencia o planos.

- Es preciso disponer de un método de asociación de datos robusto que permita emparejar las observaciones realizadas con los elementos contenidos en el mapa. Una asociación de datos errónea provocará casi con total seguridad la divergencia irrecuperable de la estimación del filtro.

A pesar de los inconvenientes que surgen al emplear esta solución, se trata de la más extendida en la literatura y presenta interesantes propiedades que resultan atractivas:

- El hecho de describir el entorno a partir de entidades geométricas descriptibles de manera compacta, se corresponden más con una visión antropomórfica del mundo, que representa este último a través del concepto del objeto y sus relaciones.

- Estas soluciones cuentan con una larga tradición, lo cual hace que su estructura y sus ventajas e inconvenientes sean bien conocidos. Históricamente se han planteado múltiples soluciones que intentan paliar algunos de los problemas anteriormente mencionados.

- Al mantenerse la matriz de covarianzas del sistema completa, es capaz de cerrar bucles exitosamente.

- **Mapas de ocupación de celdillas.**

Los mapas de ocupación de celdillas (Occupancy Grid Mapping) fueron introducidos por Hans Moravec y Alberto Elfes a mediados de la década de 1980. El método se basa en discretizar el espacio, dividiéndolo en unidades de tamaño predefinido, que se clasifican como ocupadas o vacías con un determinado nivel de confianza o probabilidad.

Estas soluciones parten de la hipótesis de que la posición del robot es conocida En la práctica, se necesita de algún método de localización que estime la posición del robot en cada instante que, en este caso, no es considerada una variable estocástica[2]. La precisión que alcanzan estos mapas en la descripción del entorno (mayor cuanto más fina es la división del espacio), permite que el algoritmo de localización empleado acumule errores reducidos a lo largo de intervalos prolongados de tiempo.

La mayor desventaja de estos métodos es la pérdida de potencia que se deriva de no tener en cuenta la incertidumbre asociada a la posición del robot, lo cual origina que su capacidad para cerrar bucles correctamente se vea mermada.

[2] Variable aleatoria.

Por otra parte, cuenta con numerables ventajas, cabe destacar las siguientes:

- El algoritmo es robusto y su implementación sencilla.
- No hace suposiciones acerca de la naturaleza geométrica de los elementos presentes en el entorno.
- Distingue entre zonas ocupadas y vacías, consiguiendo una partición y descripción completa del espacio explorado. Por este motivo *es popular en tareas de navegación*, al facilitar la planificación y generación de trayectorias empleando métodos convencionales.
- Permite descripciones arbitrariamente densas o precisas del mundo, simplemente aumentando la resolución de la rejilla que lo divide (i.e. disminuyendo el tamaño de las celdillas individuales). Como es lógico, esto va en detrimento del rendimiento computacional del algoritmo.
- Permite una extensión conceptualmente simple al espacio tridimensional.

2.2.2. Localización.

Una vez se tiene el mapa por el cual va a navegar el robot, es necesario que mientras navega tenga información de su posición. Por ello en este apartado se describirá un método de localización (además de los métodos combinados mencionados anteriormente) de robots muy extendido en este ámbito y generalmente con buenos resultados:

- **Localización de Monte Carlo.**

La Localización de Monte Carlo o *Monte Carlo Localization* (MCL), utiliza un filtro de partículas para representar la distribución de estados posibles, siendo cada partícula del filtro un posible estado, es decir, una hipótesis de la posición del robot.

El algoritmo comienza con una distribución aleatoria y uniforme de las partículas sobre el espacio de configuración, lo que significa que el robot no tiene información sobre dónde se encuentra y asume que hay la misma probabilidad de estar en cualquier punto del

mapa. Cada vez que el robot se mueve, las partículas cambian para predecir su nueva posición tras el movimiento. Cuando el robot detecta algo, las partículas se vuelven a muestrear basándose en la *estimación bayesiana recursiva*, es decir, miden cuánto se ajustan los datos detectados por los sensores (datos reales) con la posición del robot predicha. En última instancia, las partículas deberían converger hacia la posición real del robot.

2.2.3. Planificación de trayectorias.

La planificación de trayectorias es la tarea mediante la cual el robot determina el camino que va a recorrer evitando los obstáculos presentes (figura 2.6.).

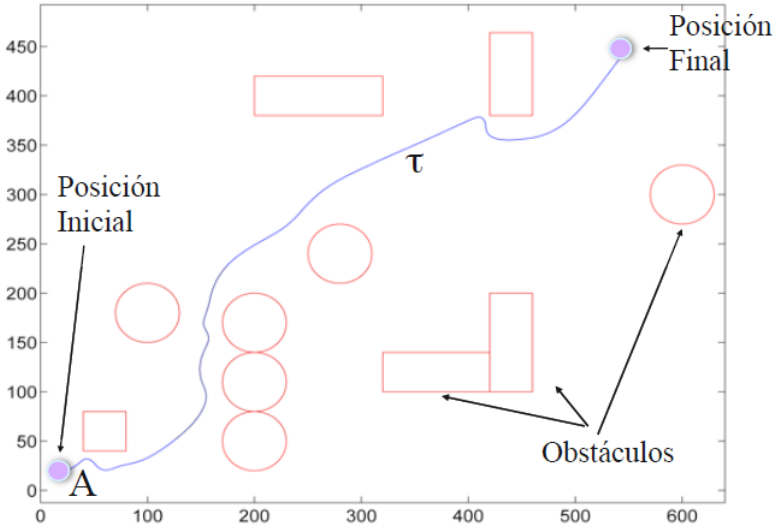


Figura 2.6. Ruta planificada libre de obstáculos [9].

Como ya se ha mencionado anteriormente, esta tarea se compone de dos sub-tareas: la *planificación global*, que es la que se encarga de establecer un camino libre de obstáculos basándose en el mapa y la *planificación local*, que se encarga de la evitación de obstáculos no previstos en el mapa.

La planificación de trayectorias global se puede resolver mediante distintas técnicas, las más relevantes se expondrán a continuación:

- **Descomposición en celdas.**

Los métodos de descomposición en celdas dividen el espacio en muchas regiones. A cada celda se le asigna una etiqueta para identificar si está libre u ocupada. Entonces, un algoritmo de planificación local encuentra una secuencia de celdas vecinas libres para asegurar un camino realizable.

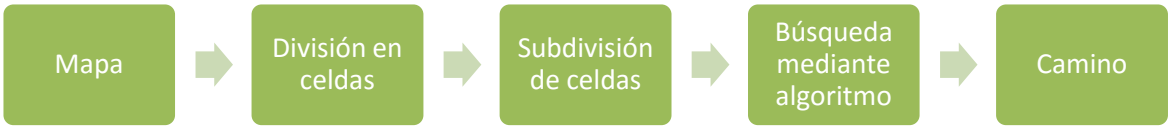


Figura 2.7. Esquematización de la planificación de trayectorias mediante descomposición en celdas

La figura 2.8. muestra cómo se divide el espacio en este método. Una vez separado el espacio en las distintas regiones, se construye un grafo y se realiza la búsqueda del camino sobre él.

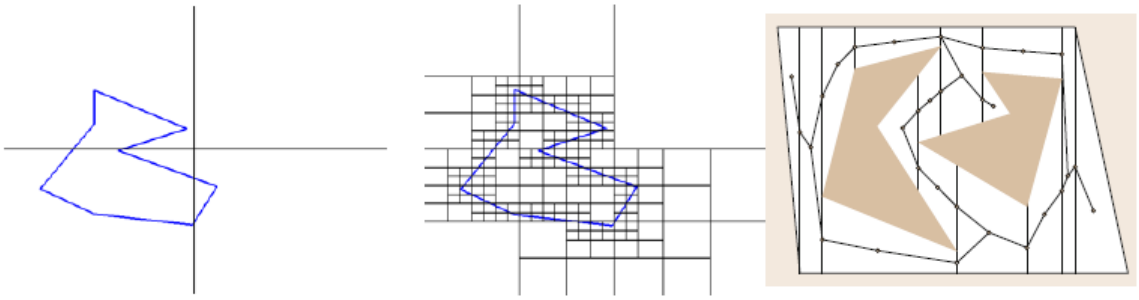


Figura 2.8. Ejemplos de descomposición en celdas [9].

- **Métodos de "roadmaps".**

Los métodos de roadmaps construyen una red de posibles caminos libres de colisión a partir de nodos generados en el espacio. Entre estos métodos destacan: los grafos de visibilidad y los diagramas de Voronoi.

Los *grafos de visibilidad* constituyen un método puramente geométrico. Dado un conjunto de obstáculos poligonales definidos por sus vértices, los grafos de visibilidad relacionan los nodos (vértices de los obstáculos) entre sí, de manera que sólo estén conectados los vértices que geométricamente sean visibles entre sí (figura 2.10.). La

visibilidad entre dos vértices se comprueba verificando si el segmento que los une intersecta a algún obstáculo. Una vez construido el grafo de visibilidad incluyendo los nodos correspondientes al inicio, al objetivo y a los vértices de los obstáculos, se ejecuta un algoritmo de búsqueda en grafos para encontrar el camino más corto entre los nodos inicio y objetivo. Definido así, un algoritmo de grafos de visibilidad sólo está pensado para robots puntuales y supondría navegar por el borde de las paredes de los obstáculos.

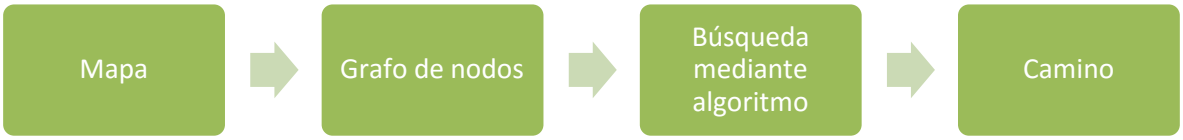


Figura 2.9. Esquematización de la planificación de trayectorias mediante grafos de visibilidad.

A fin de subsanar estos dos inconvenientes, se realiza una expansión de los obstáculos. Esta expansión provoca que el obstáculo que se tiene en cuenta sea de tamaño superior al que existe realmente. Para el algoritmo subyacente, todo sigue igual, sin embargo, se ha aportado seguridad al método.

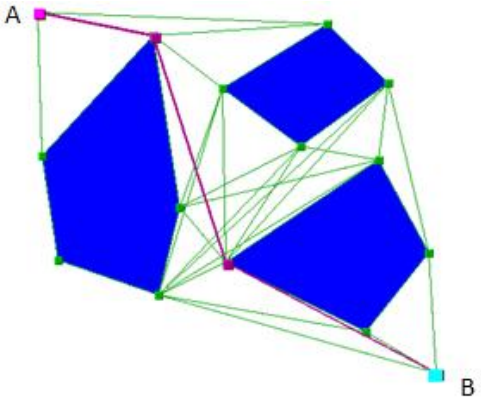


Figura 2.10. Planificación de trayectorias mediante grafos de visibilidad.

Los *diagramas de Voronoi* ofrecen una forma interesante de dividir el espacio cartesiano. Partiendo de la base de que los obstáculos son puntuales, el objetivo es generar un grafo cuyos nodos estén situados espacialmente lo más separados posible de cada obstáculo. Gráficamente, esto se logra trazando las mediatrices entre los segmentos que unen los obstáculos y comprobando las intersecciones. Las regiones de Voronoi son polígonos convexos cuyos lados tienen la propiedad de estar a la misma distancia entre dos obstáculos.

Al contrario que el método de los grafos de visibilidad, que propicia la navegación cercana a los obstáculos, este método produce caminos que están lo más alejados posible de los obstáculos. Una vez se ha calculado el diagrama de Voronoi y se ha construido el grafo correspondiente, se ejecuta el algoritmo de búsqueda en grafos que se desee.

Figura 2.11. Esquematización de la planificación de trayectorias mediante diagramas de Voronoi.

La figura 2.12. muestra un ejemplo de generación del grafo mediante la división del mapa en regiones. Los obstáculos han sido posicionados aleatoriamente y representados como círculos rojos. Las aristas verdes conforman visualmente el grafo generado a partir de sus extremos, que son los nodos del grafo. Pese a la aparente simplicidad de este método, existen algunos problemas de ahí derivados. Así, es necesario tener en cuenta de alguna forma la no puntualidad de los obstáculos y evitar utilizar aristas del grafo que pasen muy cerca de dos obstáculos.

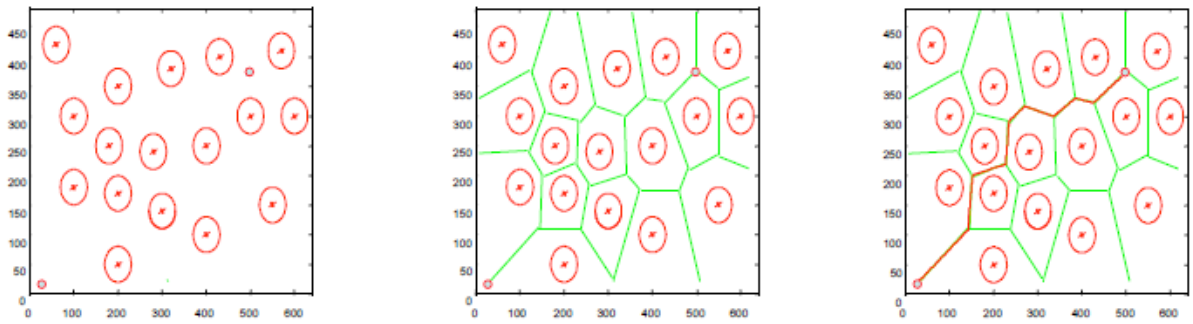


Figura 2.12. Ejemplo de diagrama de Voronoi.

Tanto los algoritmos de roadmaps como los de descomposición en celdas necesitan *hacer uso de un algoritmo de búsqueda en grafos* para determinar el camino más corto. Algunos de estos métodos son:

- *Dijkstra* [10]: Es el algoritmo clásico de búsqueda en grafos para encontrar el camino más corto entre dos nodos. Tiene la desventaja de no permitir pesos negativos de las aristas.

- *Bellman-Ford* [11]: Permite que los pesos de las aristas sean negativos, pero es más lento que el algoritmo de Dijkstra.

- *A** [12]: Hace uso de una función heurística que le hace converger más rápidamente a la solución que el algoritmo de Dijkstra. Su uso está muy extendido.

- *Lifelong Planning A* (LPA*)* [13]: Es una variante incremental del algoritmo A*. Inicialmente actúa de igual forma que un algoritmo A*, pero si se realizan pequeños cambios en el grafo y se ejecuta de nuevo el algoritmo, la solución se encuentra en un tiempo inferior al conseguido por A*.

- *D* y D* lite* [14]: Son algoritmos que combinan una búsqueda incremental y heurística que se centran en resolver de forma eficiente problemas de planificación similares a los resueltos en iteraciones anteriores. Esto implica que pueden ser usados en planificación local de forma eficiente, ya que la replanificación de trayectorias en cada iteración se adapta al objetivo de estos algoritmos.

- *Búsqueda en amplitud (en inglés Breadth First Search - BFS)* [15]: Asume que todos los pesos de las aristas son iguales. Comienza en el nodo raíz y explora los nodos vecinos. A partir de ahí, explora los nodos sucesores de dichos vecinos. Este algoritmo siempre visita los nodos avanzando en escalones de complejidad. Es decir, siempre visitará primero todos los nodos que estén a una distancia de dos aristas antes que los que estén a tres aristas de distancia.

- *Búsqueda en profundidad (en inglés Depth First Search - DFS)* [16]: También asume que los pesos de las aristas son todos iguales. En este caso, se explora en una dirección del grafo hasta llegar al máximo nivel de profundidad posible y después se continua desde atrás (backtracking).

- **Campos de potencial.**

Los campos de potencial se utilizan para decidir cuál es la dirección más prometedora en cada instante para alcanzar la meta. Para ello, utilizan la teoría de los campos eléctricos, teniendo el robot, la meta y los obstáculos del entorno asociado un potencial. Lo que se pretende conseguir es que el robot sea atraído por la meta, a la vez que es repelido por los obstáculos (figura 2.14.).

Figura 2.137. Esquematización de la planificación de trayectorias mediante campos potenciales.

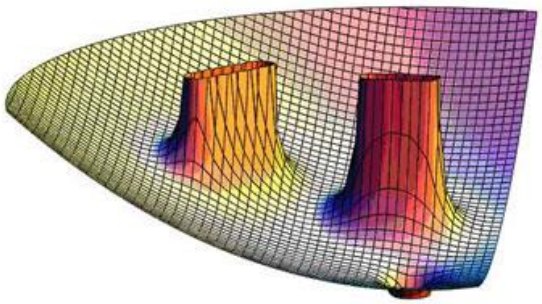


Figura 2.14. Representación potencial de dos obstáculos y la meta [17].

Una de las principales ventajas de estos métodos es que consiguen resolver problemas en entornos simples utilizando muy pocos recursos computacionales. Por contrapartida, no son métodos muy recomendables para abordar la resolución de problemas complejos, ya que siguen una estrategia que: en cada momento estudia la influencia de los potenciales emitidos por los diferentes objetos del entorno, para decidir cuál es la dirección de avance más prometedora.

- **Métodos probabilísticos.**

Los métodos probabilistas o “sampling-based” se basan en la generación de una estructura (roadmap) que se construye de forma aleatoria para rellenar el espacio de configuraciones. Una vez construida dicha estructura, se realiza una búsqueda en ella para calcular el camino más corto entre dos puntos pertenecientes a la estructura, de forma

análoga a como exploraríamos un grafo. Los algoritmos basados en Rapidly-Exploring Random Tree (RRT) [18] se basan en una exploración pseudo-aleatoria del espacio de configuraciones. Dicho espacio de configuraciones puede ser bidimensional, tridimensional, o de dimensiones superiores. La ventaja del uso de RRT es su tendencia a explorar regiones inexploradas primero, y después centrarse en el detalle. Es decir, la distribución estadística utilizada está sesgada para favorecer la exploración de todo el mapa y no explorar sucesivamente las mismas zonas.

En la planificación local uno de los métodos más usados para la evitación de obstáculos es el DWA (en inglés Dynamic Window Approach) [19]. A diferencia de otros métodos de evitación de obstáculos, el DWA deriva directamente de la dinámica del robot, y está especialmente diseñado para hacer frente a las limitaciones impuestas por las velocidades y aceleraciones del robot. Primero genera un espacio de búsqueda válido, y después selecciona la solución óptima en este espacio.

2.3. Estudio de la adaptación de movimientos.

Para que sea posible adaptar los movimientos del robot a los de la persona, éste debe de recibir la información de la posición de la persona en cada instante. Es por ello que, mediante técnicas de visión por computador se debe lograr, primero, la *detección de la persona* que va a guiar para estimar su posición (y distancia) inicial y, a continuación, realizar un *seguimiento* (o tracking) *de la posición* de la misma respecto al robot durante toda la navegación.

2.3.1. Detección de la persona.

Este problema se puede abordar mediante la detección de movimiento entre frames[3] consecutivos en una secuencia de imágenes. Esto se puede lograr a través de distintos métodos: detección de esquinas, sustracción del fondo y segmentación.

[3] Término inglés que se traduce por fotograma(s), usado habitualmente en la jerga de visión por computador.

- **Detección de esquinas.**

Una solución posible a este problema es mediante la detección de esquinas, que son puntos característicos de la imagen o puntos de interés. Esta técnica se utiliza para extraer ciertos tipos de rasgos dentro de la imagen (como pueden ser, en este caso, los rasgos faciales de una persona).

Aunque esta técnica no es siempre robusta, debido a varios factores: la localización de las esquinas puede variar significativamente de un frame a otro y en algunos casos se puede producir la aparición o desaparición simultánea de alguna de ellas en frames consecutivos.

Figura 2.15. Detección de esquinas mediante el método de Kanade-Tomasi [25].

- **Sustracción del fondo.**

El proceso de sustracción del fondo se basa en separar los objetos en movimiento o de primer plano, de los objetos estáticos o de fondo en una secuencia de frames. Si el modelo de referencia o "modelo de fondo" presenta cualquier cambio significativo, esto significa un objeto en movimiento. Los píxeles que constituyen las regiones en proceso de cambio se marcan para su posterior procesamiento. En general, en esta técnica se aplica un algoritmo que determina los puntos que están conectados entre sí para obtener regiones que corresponden a los objetos en movimiento.

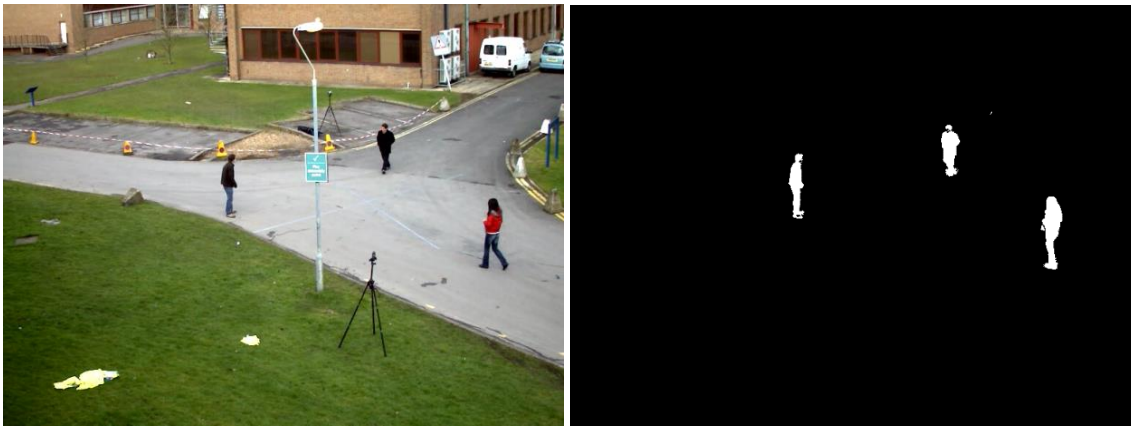


Figura 2.168. Imagen antes y después de aplicar la sustracción del fondo.

- **Segmentación.**

El objetivo de los algoritmos de segmentación es dividir la imagen en regiones de tamaño similar. Los algoritmos de este tipo abarcan dos problemas: establecer los criterios para una buena división y elegir un método para conseguir una división eficiente. Existen diferentes técnicas de segmentación de objetos en movimiento, se pueden agrupar en dos grandes grupos:

➢ Técnicas basadas en movimiento: Estas técnicas hacen uso principalmente de la información de movimiento. Dentro de este grupo, a su vez, se pueden diferenciar dos tipos: los que trabajan con el movimiento en dos dimensiones (son los más usados) y los que lo hacen en tres (3D). Las técnicas en dos dimensiones se clasifican en:

- Técnicas basadas en las discontinuidades del flujo óptico: realizan la segmentación basándose en el desplazamiento o en el flujo óptico. El flujo óptico es un vector que representa el movimiento entre un píxel en un frame y el píxel correspondiente en el siguiente frame.

- Técnicas basadas en la detección de cambios: el principal objetivo de estos algoritmos es la detección de los píxeles del objeto y los píxeles de fondo. Los algoritmos de detección asumen que el fondo es normalmente estacionario y tiene un movimiento simple global y que además los cambios entre imágenes consecutivas son debidos al movimiento.

- Técnicas espaciotemporales: los métodos de segmentación basados únicamente en movimiento son sensibles a las inexactitudes de la valoración de movimiento, por eso, en los métodos espaciotemporales se propone complementar el movimiento mediante el uso de la información espacial.

2.3.2. Seguimiento de la persona.

Una vez conseguida la detección de la persona, el seguimiento robusto de la misma es un problema cuyos enfoques cambian constantemente, debido a la complejidad y no rigidez de los movimientos del cuerpo humano, a los cambios de iluminación, al ruido en la imagen, etc.

El objetivo principal de las técnicas de seguimiento es generar la trayectoria de un objeto a través del tiempo, posicionándolo dentro de la imagen. Se puede hacer una clasificación de las técnicas existentes en tres grandes grupos: técnicas de seguimiento de puntos, técnicas de seguimiento del núcleo (kernel) y técnicas de seguimiento de siluetas.

- **Técnicas de seguimiento de puntos.**

Los objetos detectados en imágenes consecutivas están representados cada uno, por uno o varios puntos y la asociación de éstos está basada en el estado del objeto en la imagen anterior, esto puede incluir posición y movimiento. Esta técnica puede presentar problemas en escenarios donde el objeto presenta oclusiones y en las entradas y salidas de éstos. Las técnicas de seguimiento de puntos se pueden clasificar, también, en dos grandes categorías:

- Deterministas: determinan el coste de correspondencia a través de una predicción futura del comportamiento del objeto a partir del anterior.

- Estadísticos: estos métodos solucionan los problemas de seguimiento considerando las observaciones y las incertidumbres del modelo para la valoración del estado del objeto que se está siguiendo. Utilizan el espacio de estados para modelar las propiedades del objeto, tales como la posición, la velocidad y la aceleración.

- **Técnicas de seguimiento del núcleo.**

Las técnicas de seguimiento del núcleo realizan un cálculo del movimiento del objeto de una imagen a la siguiente. El movimiento del objeto se expresa, generalmente, en forma de movimiento paramétrico (translación, rotación...) o mediante el campo de flujo. Se pueden distinguir dos categorías:

- Seguimiento utilizando plantillas y modelos de apariencia basados en densidad de probabilidad: el método que más se utiliza en esta categoría es el llamado "template matching".

- Seguimiento basado en modelos multivista: se utiliza cuando el aspecto del objeto cambia drásticamente y como consecuencia se puede llegar a perder el seguimiento del mismo.

- **Técnicas de seguimiento de siluetas.**

Estas técnicas se realizan mediante la valoración de la región del objeto en cada frame y utilizando la información que contiene. Esta información puede ser en forma de densidad de aspecto o de modelos de forma. Dispone de dos métodos:

- Correspondencia de forma: se busca la silueta del objeto y su modelo asociado dentro de la imagen.

- Seguimiento del contorno: evolución de un contorno inicial en un fotograma a la nueva posición del contorno en el fotograma actual.

3. Metodología de trabajo.

Tras el estudio del estado actual de los robots guía y de los distintos objetivos del presente proyecto, en este capítulo se expone el flujo de trabajo seguido para el desarrollo de la solución y los métodos seleccionadas para resolver cada una de las distintas partes.

El método empleado para la resolución del problema que se plantea en este proyecto se divide en cuatro etapas:

1. Mapeo. En primer lugar, se elabora un mapa del entorno por el que va a navegar el robot.

2. Navegación autónoma. En segundo lugar, se implementa un sistema de navegación para el robot.

3. Localización de la persona. Es necesario que el robot sea capaz de detectar y seguir a la persona durante la navegación, por ello se desarrolla una aplicación que consiga este fin.

4. Guiado de la persona. Por último, se crea un módulo que adapte los movimientos del robot a los de la persona, teniendo en cuenta la distancia a la que ésta se encuentre.

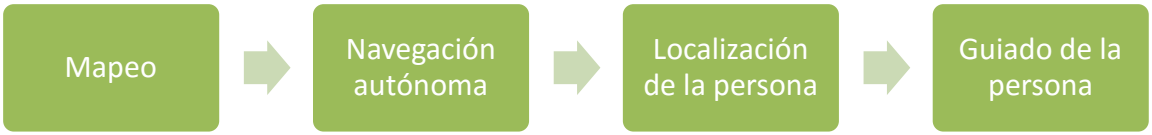


Figura 3.1. Etapas de la solución propuesta.

3.1. Navegación autónoma.

Primero se resolverán los conflictos que abarca la navegación autónoma del robot. Incluyendo en este apartado la técnica usada en la etapa de mapeo, ya que es una parte de la navegación.

Se presentan a continuación las técnicas utilizadas en cada tarea de la navegación y sus respectivas explicaciones.

- **Mapeo: Mapa de ocupación de celdillas.**

Para la parte del mapeo se usará el método de SLAM, mediante mapas de ocupación de celdillas (descrito en el apartado 2.2.1.). Se ha optado por esta opción ya que, en comparación con el método del filtro extendido de Kalman, es menos costoso computacionalmente y no requiere que las entidades del entorno sean fácilmente parametrizables. Es el método más extendido para realizar tareas de navegación ya que facilita la planificación de trayectorias.

Esta técnica intercala los pasos de localización y actualización del mapa, primero registrando el mapa a corto plazo y luego fundiéndolo con el mapa global (localización), para más tarde actualizar la ocupación percibida del mapa global (mapeo).

Figura 3.2. Estructura del algoritmo de mapeo mediante ocupación de celdillas.

El resultado de este método es un mapa de ocupación de celdillas en 2-D por el que el robot va a realizar la navegación.

- **Localización: Monte Carlo.**

El método elegido para resolver el problema de la localización durante la navegación es el método de Monte Carlo (descrito en el apartado 2.2.2.). Este método usa la información del mapa creado previamente, de la odometría y de los sensores del robot para lanzar un filtro de partículas que converja en una estimación de la posición del robot.

Figura 3.3. Inputs y output de la localización de Monte Carlo.

El filtro de partículas se compone de un conjunto de *N* partículas y unos valores, o pesos, asociados a cada una de ellas. Las partículas son estados posibles del proceso, que se pueden representar como puntos en el espacio de estados de dicho proceso. Posee cuatro etapas principales:

- Inicialización: el filtro de partículas "lanza" al azar un conjunto de puntos sobre la imagen, es decir, se crea un conjunto de partículas con un estado aleatorio.
- Actualización: realizando cálculos se le asignará un valor, o valores, a cada uno de esos puntos.
- Estimación: a partir de estos valores, se creará un nuevo conjunto de puntos que reemplazará al anterior. Esta elección también será al azar, pero los valores que se han adjudicado a cada uno de los puntos provocarán que sea más probable de elegir aquellos puntos que hayan capturado al objeto sobre el que quiere realizar el seguimiento.
- Predicción: Una vez que se crea el nuevo conjunto de puntos, se realiza una leve modificación al estado (posición) de cada uno de ellos, con el fin de estimar el estado del objeto en el instante siguiente.

Al terminar la etapa de predicción, se obtiene un nuevo conjunto de puntos al que se le vuelve a aplicar la etapa de actualización, repitiéndose este bucle hasta que termine la secuencia o desaparezca el objeto, caso en el cual se volvería a la etapa de inicialización.

- **Planificación de trayectorias global: Grafo de visibilidad y Dijkstra.**

Para la tarea de planificación de rutas global se usará el método de grafos de visibilidad, junto con el algoritmo de Dijkstra como buscador de la ruta óptima en el grafo. Se ha elegido la técnica de grafos de visibilidad (basado en el método de "roadmaps", explicado en el apartado 2.2.3.), frente a técnicas de descomposición en celdas debido a que estos métodos son más lentos: primero se realizan las sucesivas descomposiciones en celdas, luego se crea el grafo y por último se emplea el algoritmo de búsqueda; mientras que el método elegido genera directamente el grafo. A pesar de que los métodos de campos potenciales utilizan pocos recursos computacionales, cuando se enfrentan a un entorno con muchos obstáculos no se comportan de forma robusta. Por último, los métodos probabilísticos no son los más rápidos y no siempre dan la solución más óptima, ya que se demoran en explorar todos los nodos del mapa, en contraste con los roadmaps, que solo exploran los nodos alrededor del "nodo de origen" (posición inicial del robot).

La forma de construcción de grafos de visibilidad ya se ha explicado con anterioridad (apartado 2.2.3.), básicamente se trata de construir a partir de un mapa, un grafo de nodos, siendo éstos los vértices de los obstáculos reflejados en dicho mapa. Estos nodos están conectados (mediante aristas) solo si son geométricamente visibles entre sí, es decir, si existe un camino despejado entre ellos. Además, para que sea más seguro, es necesario expandir los obstáculos, evitando así posibles colisiones debido a la forma y tamaño del robot.

A la hora de determinar del camino más corto se utiliza el algoritmo de búsqueda de caminos de Dijkstra. A pesar de no permitir pesos negativos en los nodos y de no ser el algoritmo más rápido (el más rápido es el algoritmo A*), es el menos costoso computacionalmente de los expuestos en el apartado 2.2.3 de planificación de trayectorias, además suele dar muy buenos resultados.

El algoritmo de Dijsktra parte de que cada nodo tiene asociado un peso y que existe un nodo origen. Consiste en ir explorando todos los caminos más cortos que parten del origen y que llevan a todos los demás nodos; cuando se obtiene el camino más corto desde el origen, al resto de nodos que componen el grafo, el algoritmo se detiene. Es una especialización de la búsqueda de un camino de costo uniforme, y como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

Por tanto, teniendo un grafo dirigido ponderado de *N* nodos no aislados y siendo *x* el nodo inicial, un vector *D* de tamaño *N* guardará al final del algoritmo las distancias desde *x* al resto de los nodos. El algoritmo se puede dividir en varios pasos:

1. Inicializar todas las distancias en *D* con un valor infinito relativo, ya que son desconocidas al principio, exceptuando la de *x* que se debe ser 0 (ya que la distancia de *x* a *x* sería 0).

2. Sea $a = x$ (se toma *a* como nodo actual).

3. Se recorren todos los nodos adyacentes de *a*, excepto los nodos marcados (se explica a continuación cuándo se marcan), llamaremos a los nodos no marcados v_i.

4. Para el nodo actual, se calcula la "distancia tentativa" desde dicho nodo a sus vecinos con la siguiente fórmula:

$$dt(vi) = Da + d(a, vi) \tag{3.1}$$

   Es decir, la distancia tentativa del nodo 'v_i' es la distancia que actualmente tiene el nodo en el vector *D* más la distancia desde dicho el nodo 'a' (el actual) al nodo v_i. Si la distancia tentativa es menor que la distancia almacenada en el vector, se actualiza el vector con esta distancia tentativa. Es decir: si

$$dt(vi) < Dvi \quad \rightarrow \quad Dvi = dt(vi) \tag{3.2}$$

   se marca como completo el nodo *a*.

5. Tomamos como próximo nodo actual el de menor valor en *D* y se vuelve al paso 3. Se repite mientras existan nodos no marcados.

Una vez terminado al algoritmo, *D* estará completamente lleno, indicando así el camino óptimo.

- **Planificación de trayectorias local: DWA.**

Una vez se tiene la trayectoria global que debe de seguir el robot, es necesario asegurar que ésta es la trayectoria óptima en cada momento y determinar a qué velocidad debe ir el robot. Además, se debe de verificar que no tiene ningún obstáculo que no esté registrado en el mapa. Para ello es necesario hacer uso del planificador local junto con la información de los sensores. En este caso se ha optado por el método de planificación local Dynamic Window Aproach (comentado en el apartado 2.2.3).

Mediante el uso de un mapa, el planificador DWA crea una trayectoria cinemática que permita al robot llegar desde un punto de partida hasta una ubicación objetivo. Durante el camino, el planificador crea, localmente alrededor del robot, una función de valores, representada como un mapa de celdas. Esta función calcula los costes de recorrer las celdas de la cuadrícula. El controlador de movimientos del robot utiliza esta función para determinar el valor de las velocidades *dx*, *dy*, *dtheta* que debe enviar a los motores del robot.

La idea básica del algoritmo DWA es la siguiente:

1. Muestrea discretamente el espacio de control del robot (dx, dy, dtheta).

2. Para cada velocidad de muestreo, realiza una simulación de avance del estado actual del robot, para predecir lo que sucedería si la velocidad de muestreo se aplicara durante un período (corto) de tiempo.

3. Evalúa puntuando cada trayectoria resultante de la simulación anterior usando una métrica que tiene en cuenta características como: la proximidad a los obstáculos, la proximidad a la meta, la proximidad a la trayectoria global, y la velocidad. Descartando las trayectorias en las que chocaría con obstáculos.

4. Escoge la trayectoria de mayor puntuación y envía la velocidad asociada a ella a la base móvil del robot.

5. Esto se repite hasta llegar a la ubicación de destino.

3.2. Localización de la persona.

Con el fin de conseguir un sistema para localizar la posición de la persona, se divide el problema en tres fases (como se muestra en la figura 3.4.), partiendo de un vídeo grabado de la escena:

1. Detectar la cara de la persona.

2. Identificar los rasgos faciales para realizar un seguimiento.

3. Seguimiento (o tracking) de la cara.

Figura 3.4. Esquema del método utilizado en la localización de la persona.

Se usará una secuencia de imágenes tomadas por una Kinect convertida a vídeo, ya que no se dispone de una cámara de este tipo para realizar el estudio en tiempo real.

- **Detectar la cara: Detección de esquinas.**

Para esta primera fase se ha optado por un método de detección de esquinas (apartado 2.3.1.), en concreto el algoritmo de Viola-Jones [20]. Es un método robusto que se utiliza para la detección de caras, aunque puede ser entrenado para detectar otros objetos. Este método tiene algunas limitaciones, ya que para poder ser detectada la cara debe de estar de frente a la cámara en el instante inicial.

El algoritmo tiene cuatro etapas:

1. Selección de características Haar [21]: Todos los rostros humanos comparten algunos rasgos similares. Estos rasgos pueden ser emparejados usando las Características Haar (figura 3.5.). Usando estas cuatro características se puede detectar la cara.

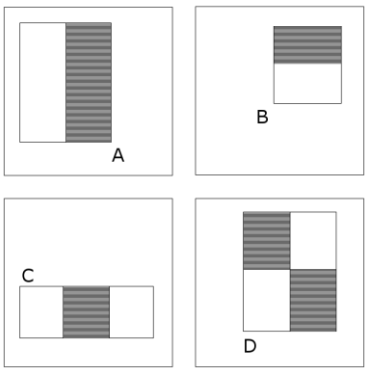


Figura 3.5. Características Haar utilizadas por Viola-Jones.

2. Creación de una imagen integral: es una representación de la imagen que evalúa las características rectangulares "en tiempo constante", lo que le da una ventaja de velocidad considerable sobre otras alternativas más sofisticadas. Debido a que cada característica de un área rectangular está siempre al lado de al menos otro rectángulo, se puede deducir que cualquier característica de dos rectángulos (A,B en la figura 3.5.) se puede calcular con seis arrays de referencia; cualquier característica de tres rectángulos (C) con ocho arrays; y cualquier característica de cuatro rectángulos (D) en nueve. La imagen integral en la posición (x,y) es la suma de los píxeles por encima y a la izquierda de (x,y), éste inclusive.

3. Entrenamiento AdaBoost [22]: La velocidad con que las características pueden ser evaluadas aumenta al aumentar el número de píxeles, por lo que tratar de evaluarlos todos no es una opción viable. Por ello se usa una variante del algoritmo AdaBoost para seleccionar las mejores características y entrenar los clasificadores que las utilizan. Construye un clasificador "fuerte" como combinación lineal de muestras de clasificadores "débiles" ponderados, siendo cada clasificador "débil" una función de umbralización de la característica

seleccionada.

4. Clasificadores en cascada: La forma general del proceso de detección es la de un árbol de decisión degenerada, conocido como "cascada". Tras un resultado positivo del primer clasificador, desencadena la evaluación de un segundo clasificador, que también ha sido ajustado para alcanzar altas tasas de detección. Un resultado positivo del segundo clasificador desencadena un tercer clasificador, y así sucesivamente. Un resultado negativo en cualquier punto conduce al rechazo inmediato de la característica. El tratamiento posterior puede ser de varias formas: puede incluir etapas adicionales de la cascada o un sistema de detección alternativo.

Los siguientes pasos se realizarán mediante el algoritmo KLT (Kanade-Lucas-Tomasi) [23], es un método de seguimiento de conjuntos de puntos. Se basa en el método de Lucas y Kanade [24], que fue posteriormente desarrollado por Tomasi y Kanade [25] y es explicado con mayor detalle en el artículo de Shi y Tomasi [26], aunque estos dos últimos métodos suelen ser usados indistintamente. Este método hace uso de la información de intensidad espacial para dirigir la búsqueda de la posición que produce la mejor coincidencia. Es más rápido que las técnicas tradicionales porque examina muchas menos posibles coincidencias entre imágenes.

- **Identificar rasgos faciales: Detección de esquinas.**

Tras la detección de la cara el siguiente paso es identificar rasgos faciales que se puedan seguir fielmente, para ello se usa el método de Tomasi y Kanade [25] de detección de esquinas. Este método calcula la diferencia ponderada entre una ventana centrada en el punto y esta ventana desplazada en el área *(u, v)*:

$$S(u,v) = \sum_{x,y} w(x,y)\,[I(x+u, y+v) - I(x,y)]^2 \qquad (3.3)$$

Siendo *S(u,v)* la diferencia ponderada, *w(x,y)* la ventana, *I(x+u,y+u)* la imagen desplazada e *I(x,y)* la imagen. Haciendo el desarrollo de Taylor:

$$S(u,v) \approx [u\ v] \begin{bmatrix} \sum_{x,y} w(x,y) I_x^2\,(x,y) & \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y) \\ \sum_{x,y} w(x,y) I_x(x,y)\, I_y(x,y) & \sum_{x,y} w(x,y) I_y^2\,(x,y) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \quad (3.4)$$

Siendo I_x y I_y las derivadas parciales de *I*, respecto a *x* e *y*. Esto se puede expresar también como:

$$S(u,v) \approx [u\ v]\ M \begin{bmatrix} u \\ v \end{bmatrix} \quad (3.5)$$

Siendo *M* la matriz de momentos de segundo orden:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad (3.6)$$

Aproximando localmente *S(u,v)* por una superficie cuadrática y considerando un corte de esta (elipse), se deduce que la *ecuación 3.5* es igual a una constante, por tanto diagonalizando *M* se obtiene:

$$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \quad (3.7)$$

Los semiejes de la elipse vienen determinados por los calores propios (λ_1,λ_2) y sus direcciones por *R*. Una vez hecho esto, calcula el $min(\lambda_1,\lambda_2)$ hallando así las esquinas que son más estables para rastrear.

- **Seguimiento de la cara: Segmentación por discontinuidad de flujo.**

Tras la detección de los rasgos faciales más característicos se usa el método de Lucas y Kanade [24]. Este algoritmo piramidal es un potente algoritmo de discontinuidad de flujo óptico utilizado en el seguimiento de características, está basado en técnicas de segmentación (apartado 2.3.1.).

Considerando un pixel $u=(u_x,u_y)$ de la imagen *I*, el objetivo es encontrar la localización $v=u+d=(u_x+d_x,u_y+d_y)$ en la siguiente imagen (o frame) *J*, esto es debido a que *I(u)* y *J(v)* se consideran "similares". El desplazamiento del vector *d* es la velocidad de

imagen en *u* o flujo óptico en *u*. Siendo ω_x y ω_y dos números enteros, el vector *d* es aquel que minimiza la función residual definida como:

$$\in (d) = \in (d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x-\omega_x} \sum_{y=u_y-\omega_y}^{u_y-\omega_y} \left(I(x,y) - J\left(x+d_x, y+d_y\right)\right)^2 \quad (3.6)$$

La función de similitud se mide en una vecindad de tamaño $(2\omega_x+1)$ x $(2\omega_y+1)$, esta vecindad se llama también "ventana de integración". Los valores típicos para ω_x y ω_y son entre 2 y 7 píxeles, en este caso se usará una ventana de 2x2 y se realizará un máximo de 30 iteraciones.

A continuación se presenta el pseudocódigo de localización de la persona empleado en este proyecto (se puede encontrar completo en el anexo 2).

Algoritmo de localización de la persona

Entrada: Vídeo de una persona realizando varios movimientos.
Salida: Vídeo del seguimiento facial de esta persona.

```
01: INICIO
02:       LEER vídeo
03:       DETECTAR rostro
04:       DIBUJAR recuadro sobre el rostro detectado
05:       DETECTAR rasgos faciales característicos
06:       MIENTRAS el vídeo este en ejecución
07:            SEGUIR los puntos característicos detectados del
               rostro de la persona
08:                SI hay más de dos puntos característicos
09:                     CALCULAR desplazamiento de los puntos
10:                     ACTUALIZAR recuadro
11:                     ACTUALIZAR puntos característicos
12:                FIN SI
13:            VISUALIZAR recuadro y puntos característicos
14:       FIN MIENTRAS
15: FIN
```

Tabla 3.1. Pseudocódigo del algoritmo de localización de personas.

El algoritmo recibe una secuencia de imágenes de una escena en la que hay una persona realizando distintos movimientos, esta secuencia se convierte a vídeo para poder procesarlo como si fuera una muestra de la captura de la cámara en tiempo real.

El algoritmo lee el vídeo y aplica el detector de rostros de Viola-Jones, a continuación se marca el rostro detectado mediante un recuadro (bounding box). Dentro del recuadro se buscan los puntos característicos del rostro detectado método de Tomasi y Kanade, para poder seguirlo a través de los frames del vídeo. Debe de encontrar al menos dos puntos característicos para poder seguir el algoritmo.

Mientras el vídeo se está reproduciendo se usa el método de Lucas y Kanade para seguir los puntos característicos anteriormente detectados, que en cada nuevo frame están sujetos a aparecer y desaparecer. Se calcula el desplazamiento de dichos puntos y se actualiza la posición de los mismo y del bounding box en cada nuevo frame.

El resultado de este algoritmo es el vídeo procesado con la detección facial de la persona y su seguimiento hasta el final del vídeo.

3.3. Guiado de la persona.

Una vez conseguida la navegación autónoma del robot y la localización de la persona que va a guiar, se procede a desarrollar la aplicación de guiado, implementando la adaptación de movimientos del robot a los de la persona.

Para ello es necesario crear un algoritmo que reciba las posiciones relativas de la persona y del robot en cada instante, las compare calculando la distancia entre ellos y dependiendo del valor resultante mande un comando de velocidad al robot. Si la distancia entre la persona y el robot está comprendida entre 2 y 3 metros el comando que recibe el robot es el de ralentizar su desplazamiento, mientras que si la distancia es mayor de 3 metros recibe la orden de parar completamente. Si la distancia entre ellos es 2 metros o inferior, el robot navegará a la velocidad óptima en cada instante (según el planificador local).

A continuación se puede encontrar el pseudocódigo empleado para implementar el algoritmo de control de velocidades del robot (se puede encontrar completo en el anexo 1).

Algoritmo de control de velocidades del robot

Entrada: Posición de la persona y posición del robot.
Salida: Velocidad del robot.

```
01: REPETIR
02:       SI la posición de la persona cambia
03:               ALMACENAR posición de la persona
04:       FIN SI
05:       SI la posición del robot cambia
06:               ALMACENAR posición del robot
07:       FIN SI
08:       CALCULAR distancia entre posición del robot y posición
          de la persona
09:       SI distancia menor que 3 metros y mayor que 2 metros
10:               ENVIAR velocidad reducida al robot
11:       FIN SI
12:       SI distancia mayor que 3 metros
13:               ENVIAR velocidad vacía al robot
14:       FIN SI
15: FIN REPETIR
```

Tabla 3.2. Pseudocódigo del algoritmo de control de la velocidad del robot.

4. Trabajo realizado y resultados obtenidos.

En esta sección del documento se exponen las plataformas usadas para la implementación de las soluciones propuestas en el apartado anterior, el trabajo desempeñado con ellas para lograr el sistema propuesto y los resultados obtenidos.

4.1. Herramientas.

Para el desarrollo de las distintas partes del proyecto ha sido necesario recurrir a distintas herramientas y plataformas de trabajo: las partes del *mapeo*, la *navegación autónoma* y el *guiado de la persona* se han desarrollado de forma simulada en el entorno de trabajo ROS, mientras que la parte de *localización de la persona* se ha implementado con MATLAB. Se ha creído conveniente realizar un desglose de las mismas.

Figura 4.1. Esquema de las herramientas utilizadas.

4.1.1. ROS.

La plataforma principal para el desarrollo del presente proyecto es ROS (Robotic Operating System). Este entorno de trabajo de código abierto, es un marco flexible para la escritura y desarrollo de software para robots, se compone de una colección de herramientas,

bibliotecas y convenios que simplifican la tarea de crear aplicaciones complejas y robustas, en una amplia variedad de plataformas robóticas.

Dado que el soporte completo para ROS solo se ofrece para el sistema operativo Ubuntu, se trabajará con las siguientes versiones: Ubuntu "14.04.4 LTS (Trusty Thar)" y "ROS Indigo Igloo". Esta última incluye: ROS, librerías genéricas para robots, simuladores 2D y 3D, varios sistemas de percepción 2D y 3D y una librería para la navegación de robots móviles.

Es un entorno de trabajo bastante complejo, por tanto, una parte importante del proyecto ha sido el aprendizaje del mismo. A continuación, se hará una breve introducción de los conceptos básicos para entender este sistema, así como el posterior desarrollo del documento.

ROS proporciona los servicios estándares de un sistema operativo como son: abstracciones hardware, drivers, librerías, envío de mensajes entre distintos procesos, gestión de paquetes y más. Se divide en dos espacios de trabajo diferentes: el sistema de archivos y el sistema computacional.

- El sistema de archivos es donde se crean los diferentes ficheros que se encargarán de generar más tarde todas las tareas. Lo que para la mayoría de los programas significa tener una carpeta, en ROS, ese concepto se sustituye por una variable (ROS_PACKAGE_PATH), la cual contiene la ruta de los diferentes sistemas de archivos que se definen dentro del entorno de trabajo. Se diferencian dos tipos: *paquetes* (unidades de archivo mínimas, que contienen los ficheros necesarios para poder ejecutar un programa) y *stacks* (conjuntos de paquetes).

- El sistema computacional es la parte donde se ejecutan los diferentes sistemas de archivos. El papel de ROS dentro del mismo es prácticamente equivalente al de un Sistema Operativo. Sus funciones principales son las de ejecutar los distintos procesos, establecer las comunicaciones entre ellos, ejecutar los drivers necesarios y realizar las conexiones adecuadas. Para definir un sistema se utilizan una serie de herramientas: *mensajes,* estructuras simples de datos que se encargan de almacenar

la información; *tópicos,* buses de transmisión de datos que se encargan de transmitir los diferentes mensajes; *nodos,* ficheros con código escrito en C++ o Python, que se encargan de generar los distintos procesos, y *servicios,* establecen comunicación de doble sentido entre nodos.

La arquitectura de funcionamiento interna, es la de un grafo donde existen diferentes *nodos* que se encargan de realizar distintas tareas y que, además, están conectados a otros *nodos* mediante *tópicos*, que se encargan de transmitir los *mensajes* producidos en los distintos *nodos*. Formándose de esta manera una red en la que los *nodos* pueden: publicar en un *tópico*, es decir, transmitirle información (*mensajes*), o bien suscribirse a él, recibiendo así la información (*mensajes*) que transmite éste (figura 4.2.).

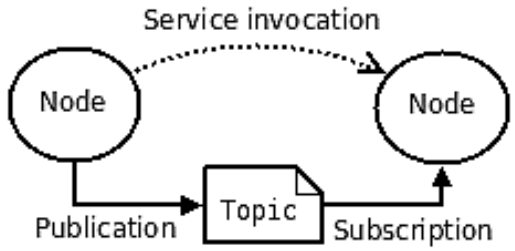


Figura 4.2. Ejemplo de comunicación entre nodos mediante un tópico.

Para comprender mejor todos estos conceptos y el funcionamiento de ROS es recomendable seguir la explicación del libro Erle Robotics GitBook [27] y los tutoriales oficiales de ROS [28].

Como se ha mencionado en la introducción del documento, este proyecto se realizará en modo de simulación. Para ello se usarán las herramientas que proporciona ROS conocidas como: Gazebo (para simular en 3D el robot y su entorno) y RVIZ (para poder visualizar la lectura de los sensores del robot, la creación del mapa, la planificación de trayectorias de la navegación y el guiado de la persona).

4.1.2. MATLAB.

Se ha decidido que la plataforma de desarrollo de la parte de visión por computador sea MATLAB (MATrix LABoratory). Este programa ofrece un Entorno de Desarrollo

Integrado (IDE) con un lenguaje de programación propio (lenguaje M). Entre sus prestaciones básicas se encuentran: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes.

Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes). En este caso se ha trabajado con la versión: Matlab R2015a y la extensión de procesamiento de imágenes "Image Processing Toolbox".

4.2. Creación del entorno de trabajo en ROS.

En lo que respecta a la organización de los distintos paquetes de simulación, estarán contenidos dentro del entorno de trabajo (workspace) "*robot_guia*". El workspace consta de tres carpetas principales dos generadas por defecto al inicializar el workspace ("build" y "devel") y otra "src", creada por el usuario (abreviatura de "source"), en la cual se almacenan los paquetes y ficheros que se utilicen.

A continuación, en la carpeta "src", se crean los tres paquetes necesarios para el desarrollo de la aplicación propuesta: "mapeo", "navegación" y "guiado". Contendrán los distintos archivos (con extensiones tipo *.launch, .yaml, .world,* etc.) empleados a lo largo del proyecto.

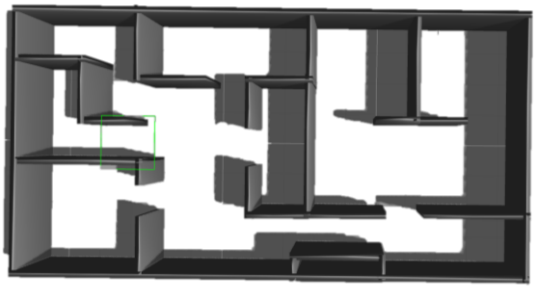


Figura 4.3. Modelo del entorno en Gazebo.

El último paso antes de abordar la simulación es crear en Gazebo el modelo del entorno por el que se desplazará el robot. En el diseño de dicho entorno se ha pretendido que se asemeje al interior de un edificio con distintas habitaciones (figura 4.3.). Se consigue

creando un modelo mediante la herramienta de Gazebo "building editor", este archivo (mundo_definitivo.world) se guarda en la carpeta "*worlds*" de cada paquete creado. Es un archivo escrito en lenguaje .xml que define las características físicas del entorno creado (luz, gravedad, etc.).

4.3. Simulación del robot.

Se trabajará con un robot tipo Turtlebot, ya que está diseñado para fines de investigación y de desarrollo mediante código abierto. Este robot dispone de una cámara Kinect de Microsoft que va unida a una base móvil tipo Kobuki mediante una estructura como se puede apreciar en la figura 4.4.

Figura 4.4. Robot Turtlebot real y el modelo simulado en Gazebo.

Es necesario instalar los paquetes que usa ROS para simular y controlar el Turtlebot, se pueden encontrar en la página oficial de ROS [29].

El paquete que se ha utilizado para la simulación del robot es "turtlebot_gazebo", dentro del cual hay un archivo de lanzamiento del Turtlebot (turtlebot_world.launch) que será usado en distintas fases de la aplicación. Este archivo ejecuta la simulación del robot en un entorno gráfico definido por defecto, pero como ya se ha mencionado, para desarrollar el proyecto se ha creado uno nuevo. Por tanto, es necesario modificar el archivo para que

simule el turtlebot en el entorno deseado, cambiando el "path" del entorno que viene por defecto (empty_world.world), por el del archivo "mundo_definitivo.world".

Para poder controlar el Turtlebot y desplazarlo por dicho entorno se usará el paquete "turtlebot_teleop", concretamente el archivo "keyboard_teleop.launch". Este archivo permite controlar mediante el teclado del ordenador los movimientos del robot.

El Turtlebot publica información sobre sus coordenadas de posición usando el paquete "tf". Este paquete permite realizar el seguimiento de múltiples marcos de coordenadas a través del tiempo, es decir, realiza las transformaciones entre los distintos marcos de coordenadas del sistema: entorno, posición del robot, mapa, etc.

Por último, para visualizar en RVIZ la información proporcionada por los sensores del robot se ha utilizado el paquete "turtlebot_rviz_launchers".

4.4. Mapeo en simulación.

A la hora de simular el mapeo es necesario ejecutar simultáneamente Gazebo (para visualizar el modelo y el entorno) y RVIZ (para visualizar la construcción del mapa), además del paquete necesario para mapear: "slam_gmapping".

El paquete "slam_gmapping" crea un mapa de ocupación de celdillas 2D del entorno mediante la técnica de SLAM, usando la información del láser (la información 3D de la Kinect se convierte en 2D) y de la odometría del robot. Relaciona la información del láser con la información de odometría del robot mediante el paquete "tf". Por defecto vienen preestablecidos unos valores del número de partículas usadas en el mapeo (80 partículas) y del intervalo entre actualizaciones del mapa (25 segundos) pero se ha comprobado que no dan los mejores resultados. Por tanto, se han probado distintos valores de partículas (30, 80, 100) y de período de actualización (5, 15, 25 segundos) hasta dar con la combinnación que mejores resultados daba: 100 partículas y 5 segundos entre cada actualización.

Para conseguir la ejecución simultánea de varios nodos se crea un archivo de lanzamiento "mapeo_turtlebot.launch" (anexo 1), en el cual se incluyen:

- **Gazebo** con el modelo del Turtlebot en el entorno creado.
- El nodo "**turtlebot_teleop_keyboard**" para desplazar el robot por el entorno.
- El nodo "**slam_gampping**" para crear el mapa.
- **RVIZ** con un archivo launch (view_mapping.launch) para la visualización de la creación del mapa.

4.4.1. Resultados obtenidos.

Al ejecutar el archivo "mapeo_turtlebot.launch" se abren las ventanas de Gazebo y RVIZ, como se muestra en las figuras 4.5. y 4.6.

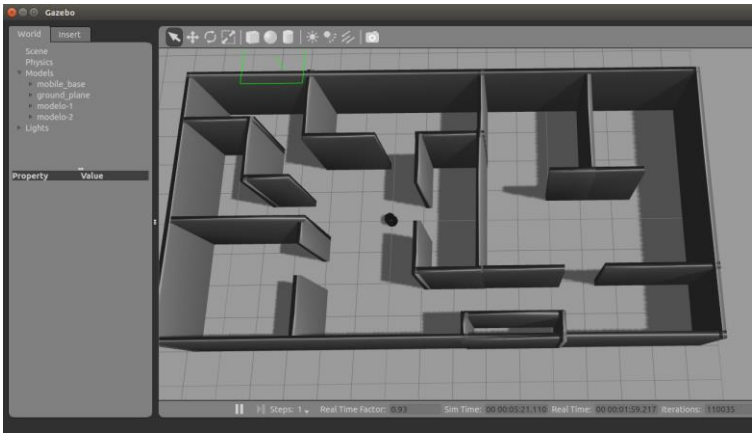


Figura 4.5. Simulación en Gazebo.

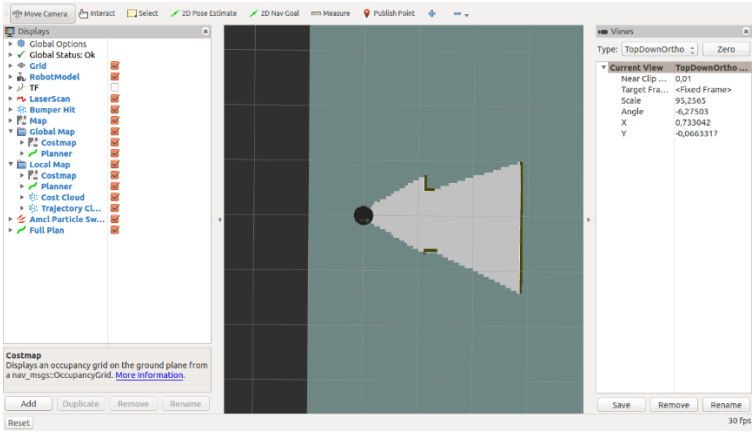


Figura 4.6. Visualización de la creación del mapa en RVIZ.

Utilizando las teclas que se muestran en la terminal (figura 4.7.) se desplaza el robot por el entorno para ir creando el mapa. Se puede ver el resultado tras dar una vuelta de 360º en la figura 4.8.

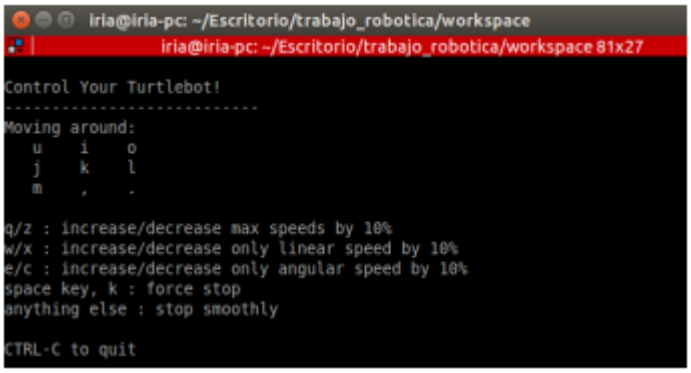


Figura 4.7. Captura de la terminal con los comandos de desplazamiento.

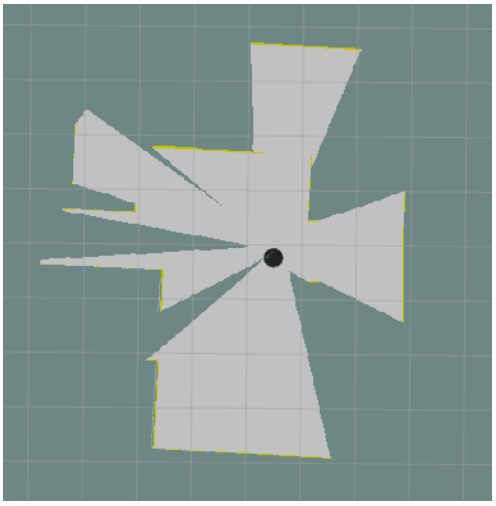


Figura 4.8. Mapa tras dar una vuelta de 360º.

Una vez finalizado el mapa, se guarda utilizando el paquete "map_saver". Esto genera dos archivos: uno de configuración (.yaml) y otro con la imagen del mapa (.pgm). Ambos archivos deben estar en la misma carpeta para poder ser usados.

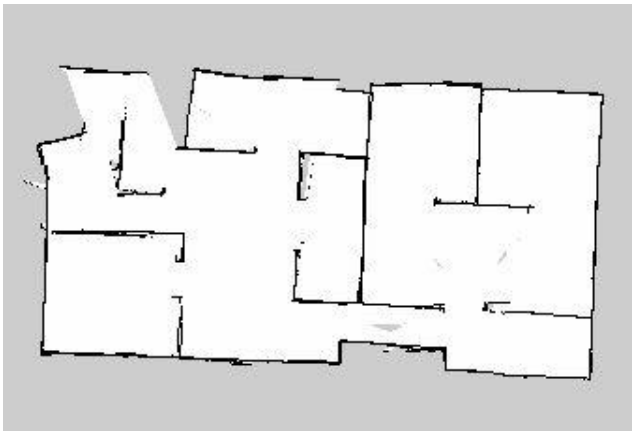


Figura 4.9. Imagen del mapa generado (.pgm).

4.5. Navegación en simulación.

Para solucionar el problema de la navegación se utilizará el stack de navegación llamado "navigation". Este stack contiene un conjunto de paquetes que proporcionan soluciones para todos los aspectos de la navegación. Usa la información de la odometría, de los sensores y de la posición final deseada del robot para proporcionar datos de velocidad que se envían como comandos a los controladores de éste.

- El paquete "costmap_2d" se encarga de implementar un mapa de costes 2D a partir del mapa previamente creado inflando los obstáculos. Se basa en los grafos de visibilidad (apartado 2.2.3.)

- El paquete "amcl" se encarga de la localización del robot durante la navegación. Se basa en la localización de Monte Carlo (apartado 2.2.2.).

- El paquete "navfn" proporciona un método de planificación global. El planificador asume un robot circular y opera en un costmap para encontrar un plan de coste mínimo. La función de navegación se calcula con el algoritmo de Dijkstra, pero tiene soporte para el algoritmo A * (apartado 2.2.3.).

- El paquete "dwa_local_planner" proporciona la implementación del algoritmo DWA (apartado 2.2.3.) para la planificación local.

- El paquete "move_base" es el que controla todos los anteriores generando así unos comandos de velocidad que envía a la base móvil (figura 4.10.), permitiendo así al robot llegar a la ubicación indicada, dentro de una tolerancia especificada por el usuario. Además, desempeña comportamientos de recuperación cuando el robot percibe que está atascado.

Figura 4.10. Esquema del paquete "move_base".

Es necesario configurar el stack de navegación según el robot y sus especificaciones para lograr una mayor precisión (dinámica y cinemática). En este caso, el stack del Turtlebot proporciona esto ya implementado: "turtlebot_navigation". En este paquete se encuentran los archivos necesarios para realizar una correcta navegación con el Turtlebot, entre ellos cabe destacar uno: "amcl.launch". Este archivo hace uso de dos paquetes esenciales en la navegación: "move_base" y "amcl", además incluye dos nodos: "safety_controller", que consigue una navegación más segura evitando que el robot se choque, que caiga en desniveles… y "velocity_smoother" que asegura una navegación más fluida.

En el paquete "navegación" dentro de la carpeta "map" se guardan los dos archivos del mapa que se ha creado antes. También se implementa el archivo "navegacion_turtlebot.launch" (anexo 1), en el cual se incluyen:

- **Gazebo** con el modelo del Turtlebot en el entorno creado.

- El archivo "amcl.launch" con los nodos "**amcl**" y **"move_base".**
- El nodo "**map_server**" para cargar el archivo del mapa.
- **RVIZ** con un archivo launch (view_navigation.launch) para la visualización de la planificación de trayectorias (local y global) sobre el mapa.

- **Navegación mediante coordenadas.**

Dado que la aplicación está pensada para que el robot opere dentro de un edificio guiando a la persona hasta una determinada ubicación, véase una consulta de un hospital o un despacho de una facultad, se han numerado las distintas habitaciones del entorno creado.

En la figura 4.13. se puede apreciar la distribución del mapa en 8 habitaciones, en rojo se representan los límites de las habitaciones que en el mapeo no han quedado bien reflejados.

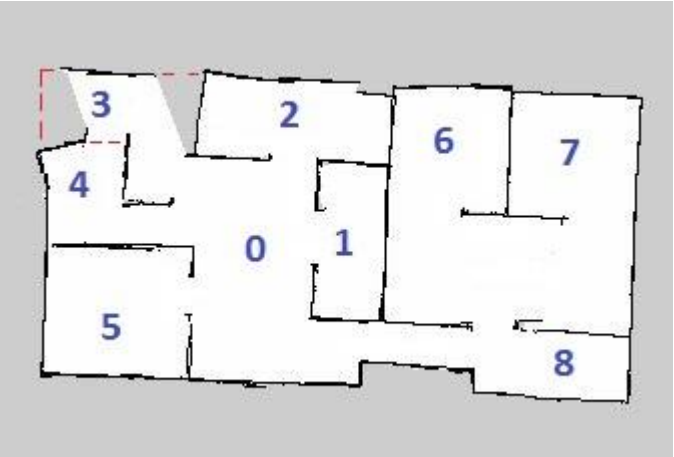


Figura 4.13. Mapa con la distribución de habitaciones.

Se han creado 8 archivos (.py) con las coordenadas de las distintas habitaciones y otro con las coordenadas del origen, que se pueden encontrar en la carpeta "goals" dentro de "robot_guia" (anexo 1). El archivo "goal_X.py" (siendo X el número de la habitación) contiene un código en Python que le indica al Turtlebot la pose final de la navegación.

4.5.1. Resultados obtenidos.

Al lanzar el archivo "navegacion_turtlebot.launch" se abren las ventanas de Gazebo y RVIZ, tal cual se muestra en la figura 4.11. En Gazebo aparece el mundo con el Turtlebot. Mientras, en RVIZ se observa el Turtlebot junto con el mapa creado, en verde la nube de puntos de localización (amcl) y la información detectada por los sensores en el instante inicial.

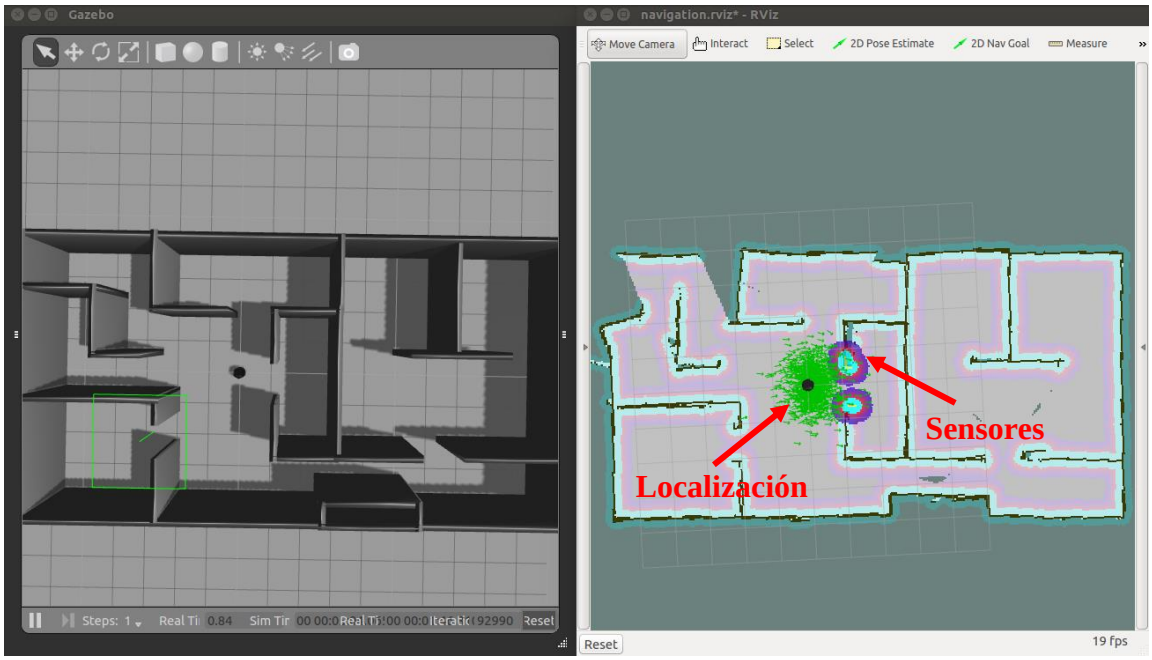


Figura 4.11. Simulación de navegación en Gazebo y visualización en RVIZ.

A continuación, es necesario indicarle al robot en RVIZ la pose final que debe alcanzar al terminar la navegación, lo cual se puede realizar de dos formas:

- Usando el botón "2D Nav Goal" y haciendo clic directamente en el mapa de RVIZ.

- Ejecutando un archivo "goal_X.py" en la terminal. Esta es la forma que se usa en el desarrollo de este proyecto.

Tras realizar esto, el robot comienza la navegación del robot hacia el objetivo. En la figura 4.12. se muestra un instante cualquiera del proceso de navegación, donde, en RVIZ, se aprecia el mapa de costes local (2d_costmap) alrededor del robot. También se puede ver

la trayectoria global (navfn) y la local (dwa_local_planner), líneas verde y roja, respectivamente.

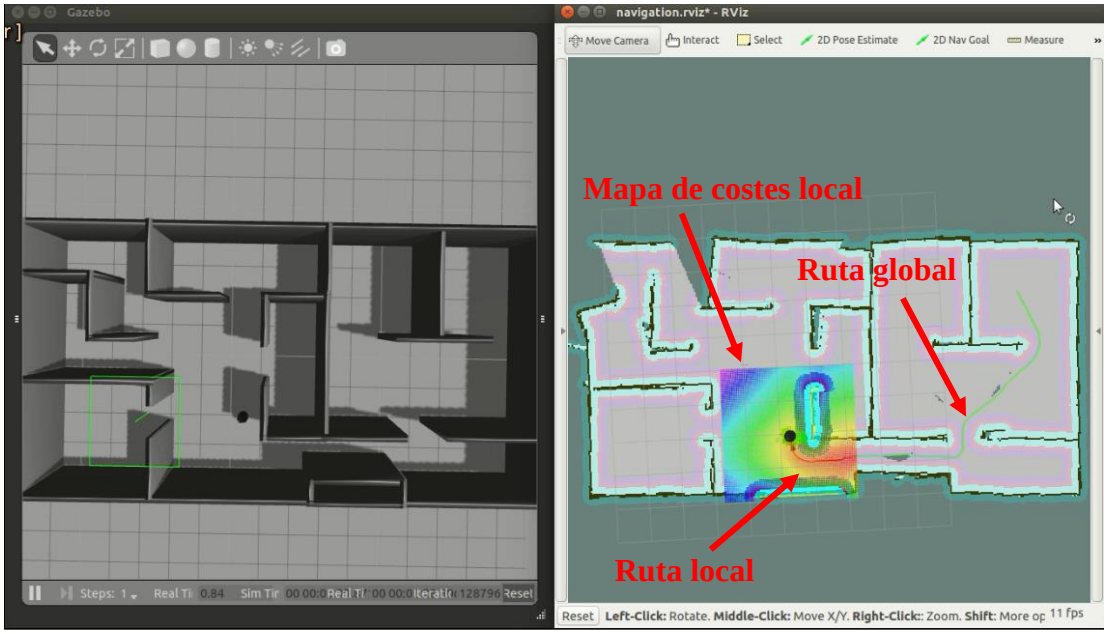


Figura 4.12. Instante del proceso de navegación.

Finalmente, se ha puesto a prueba el algoritmo de navegación para verificar que el robot es capaz de evitar obstáculos en tiempo real. Para ello, se ha repetido el proceso de navegación sobre el mismo entorno, pero con algunos obstáculos adicionales (cubos). De esta manera, el robot debería ser capaz de replanificar su ruta en caso de encontrarse con estos obstáculos, aunque no estén previstos en el mapa.

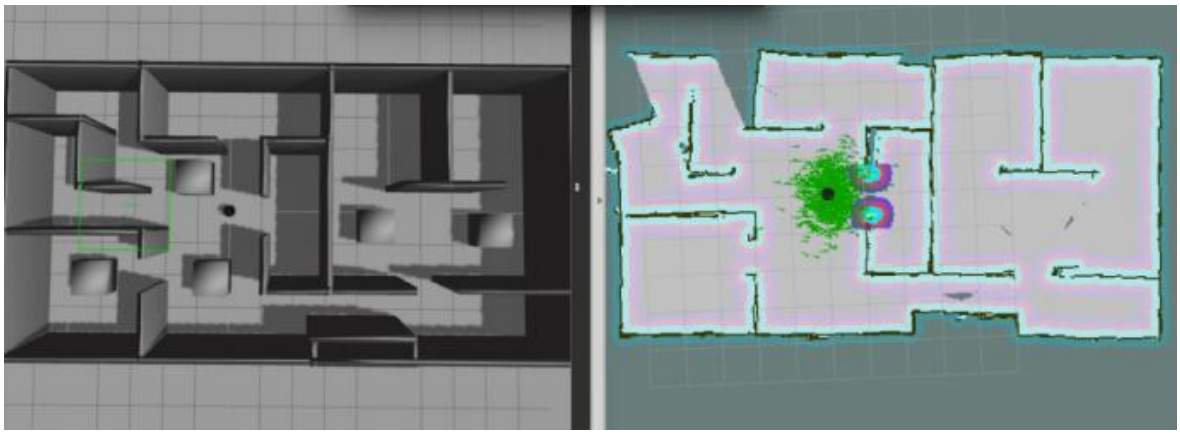


Figura 4.13. Instante inicial de la navegación con mapa incompleto.

A continuación se muestra la pantalla de RVIZ en sucesivos instantes del proceso de navegación, con el fin de prestar atención a la ruta planificada.

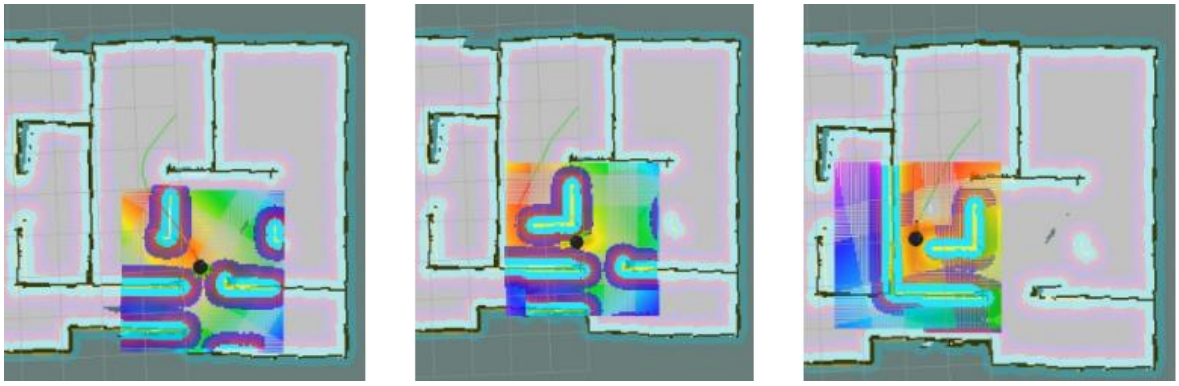


Figura 4.14. Evolución de la planificación local con el tiempo.

4.6. Guiado de la persona en simulación.

Una vez lograda la navegación autónoma del robot sobre el mapa del entorno creado, se procede a simular el guiado de la persona.

Para poder simular a la persona se empleará una herramienta de RVIZ llamada "Interactive Markers", ésta permite mostrar uno o varios objetos en 3D e interactuar con ellos mediante el ratón. Para realizar la simulación del guiado solo es necesario conocer la posición de la persona en cada instante y gracias a esta herramienta se puede obtener información de la posición del marcador.

El paquete "interactive_marker_tutorials" contiene varios archivos .cpp con distintos tipos de marcadores interactivos, para el propósito de la simulación de la persona se ha optado por el marcador llamado "quadrocopter" contenido en el "archivo basic_controls.cpp", el cual se puede desplazar en todas direcciones. En el paquete "guiado", dentro de la carpeta "src", se crea el archivo "marcador_persona.cpp" con el código de dicho marcador interactivo, creando así el nodo que simula a la persona. Este nodo publica la información de la posición de ésta bajo el tópico /basic_controls/feedback. En la figura 4.15. se puede ver el marcador en RVIZ junto al Turtlebot.

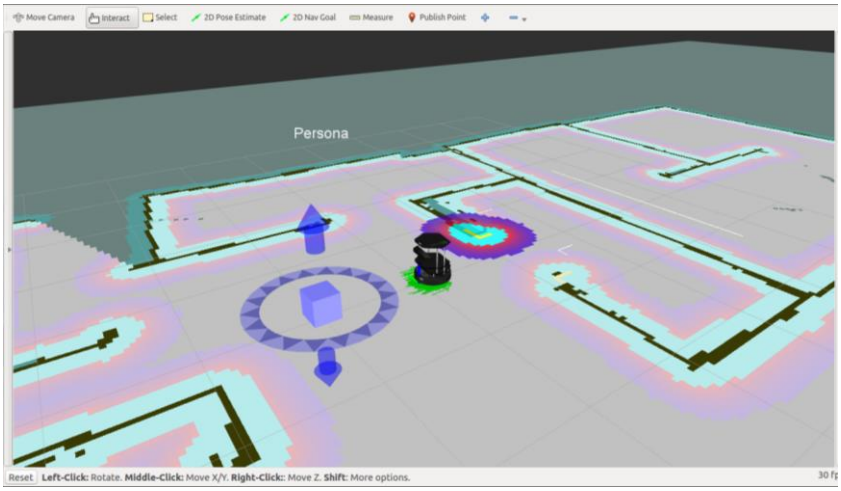


Figura 4.15. Simulación de la posición de la persona en RVIZ.

Una vez que se dispone de un nodo publicando la información de la posición de la persona, es necesario obtener la posición del robot, esto se consigue accediendo a la información que publica el tópico /odom.

Por tanto, ahora es necesario crear un nodo que se subscriba a ambas posiciones y compute la distancia entre ambas para compararla, y generar así un comando de velocidad que debe de publicar en el nodo del controlador de velocidades del robot.

Figura 4.16. Esquema de comunicación entre el nodo y los tópicos para la adaptación de movimientos.

En la figura 4.16. se muestra los tópicos a los que se subscribe el nodo "adaptar_movimiento": /odom (posición del robot) y /basic_controls/feedback (posición de la persona) y el tópico en el que publica la velocidad a la que debe ir el robot: /navigation_velocity_smoother/raw_cmd_vel. El algoritmo empleado para ello se ha explicado en el apartado 3.3. (tabla 3.2.).

4.6.1. Resultados obtenidos.

Una vez se le indica al robot la ubicación a la que debe dirigirse acompañando a la persona, se inicia la navegación con el nodo de adaptación de movimientos. Se pueden dar tres situaciones:

- Si la distancia entre ellos es mayor de 2 metros y menor que 3 metros el robot disminuye su velocidad de navegación hasta 0,3m/s y se muestra por terminal el mensaje "Reducción de velocidad" (figura 4.17.).

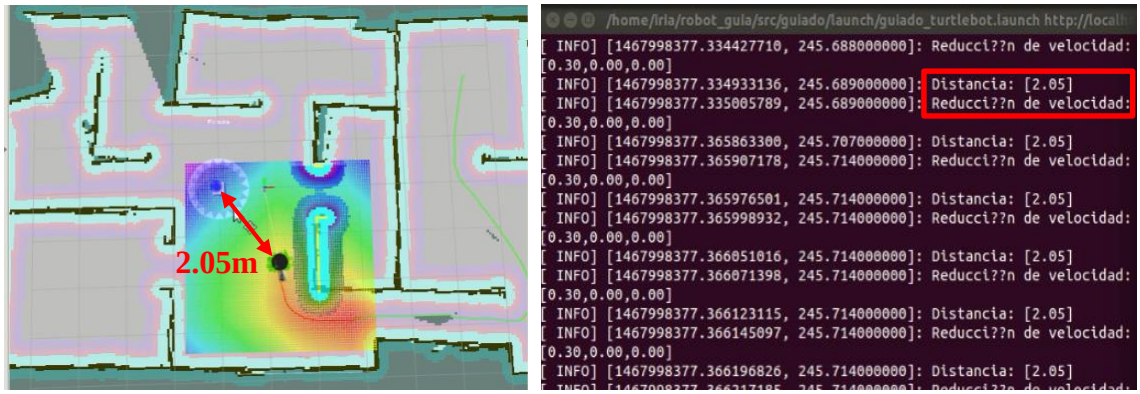


Figura 4.17. Instante durante el guiado en el que la distancia entre el robot y la persona es mayor que 2 metros y el robot reduce su velocidad.

- Si la distancia es mayor que 3 metros el robot se detiene completamente y se muestra por terminal el mensaje "Parar" (figura 4.18.).

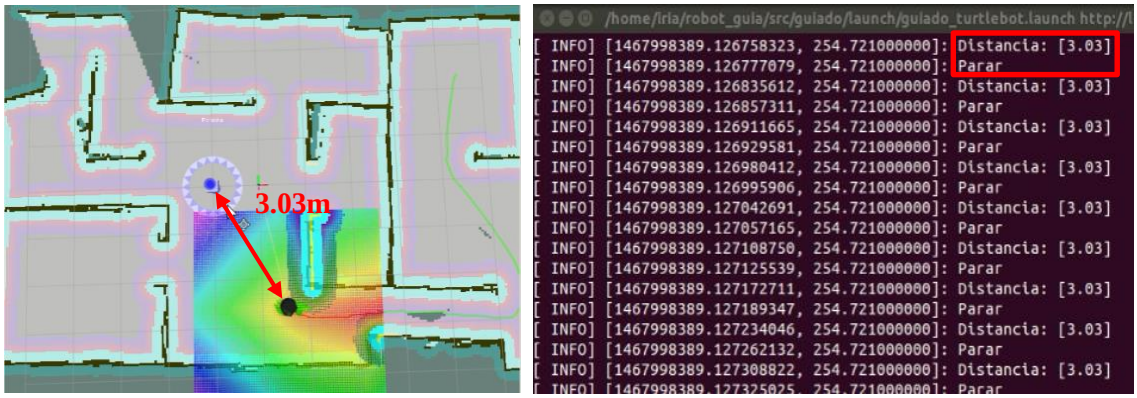


Figura 4.18. Instante durante el guiado en el que la distancia entre el robot y la persona es mayor que 3 metros y el robot se detiene.

- Si la persona se encuentra a una distancia menor de 2 metros del robot este navega a la velocidad óptima (figura 4.19.).

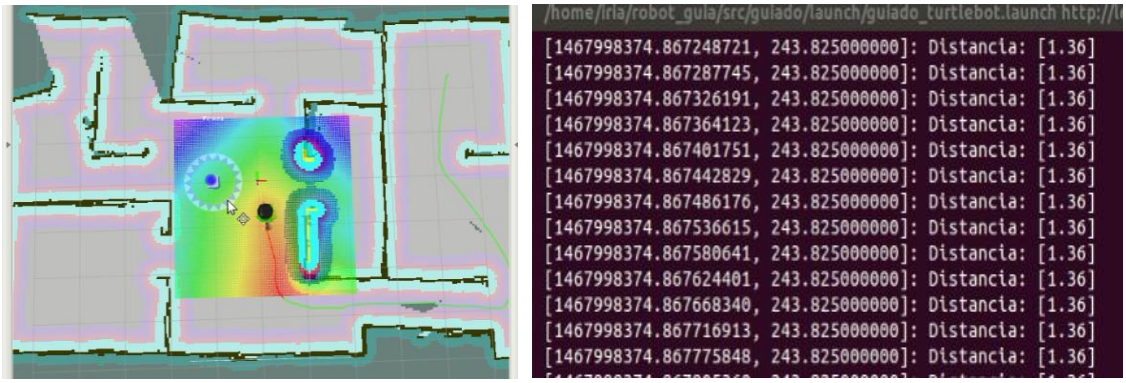


Figura 4.19. Instante durante el guiado en el que la distancia entre el robot y la persona está dentro del margen admisible.

Todos los diagramas de nodos de los distintos procesos: mapeo, navegación y guiado se encuentran en el anexo 1, no se han incluido en la memoria por su gran tamaño.

4.7. Localización de la persona.

Para realizar los experimentos siguientes se han usado imágenes del instituto Berkeley de California [30]. Se ha optado por una serie de movimientos realizados por una persona que está en frente a la Kinect, en concreto se han usado ocho tipos de movimiento diferentes, con dos repeticiones de cada uno. Estas imágenes se han convertido en archivos de vídeo, los cuales se incluyen en el anexo 2 del trabajo.

Tras la implementación en MATLAB del algoritmo explicado en el apartado 3.2. (incluido en el anexo 2 del trabajo) se procede a exponer los resultados obtenidos.

4.7.1. Resultados obtenidos.

- **Detectar la cara.**

En la fase de detección facial se pueden dar 3 casos:

1. Cara detectada correctamente: se detecta correctamente la localización de la cara, señalizada con un recuadro amarillo, como se puede apreciar en la figura 4.20.

Figura 4.20. Cara detectada correctamente (vídeo 1.1 del anexo 2).

2. Cara no detectada: el algoritmo no detecta ninguna cara, como se muestra en la figura 4.21., por tanto, es un falso negativo.

Figura 4.21. Cara no detectada (vídeo 4.1 del anexo 2).

3. Cara detectada y falso positivo: Se detecta correctamente la cara, pero también se produce un falso positivo y detecta otra zona de la imagen como otro rostro, como se aprecia en la figura 4.22.

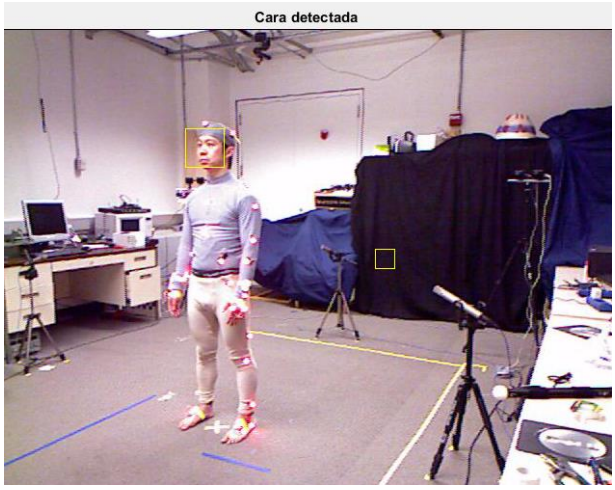


Figura 4.22. Falso negativo (vídeo 3.1 del anexo 2).

- **Identificar rasgos faciales.**

En esta fase se puede apreciar (como en la figura 4.23.) que multitud de puntos de características han sido detectados (verde), aunque se pueden dar casos en los que se detecten puntos que no pertenecen a la cara. Para poder continuar con el algoritmo debe localizar al menos dos de ellos.

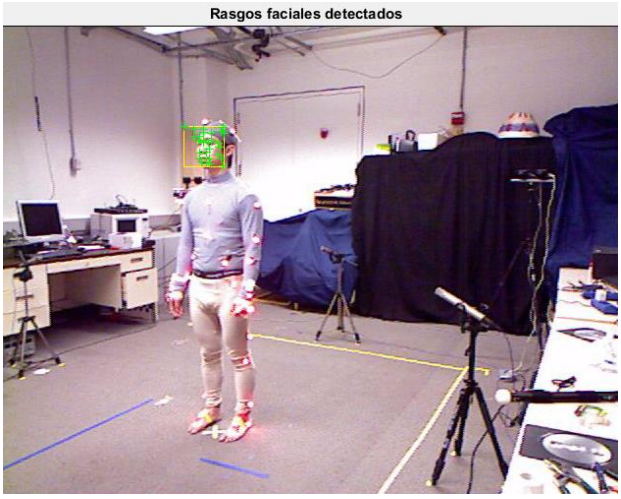


Figura 4.23. Rasgos faciales detectados (vídeo 1.1 del anexo 2).

- **Seguimiento de la cara.**

Una vez localizados estos puntos se inicia el seguimiento facial. Como se aprecia en la figura 4.24. el número de puntos de seguimiento se reduce respecto al inicial (contando

siempre con al menos 2 puntos), esto se debe a que están sujetos a desaparecer y aparecer otros nuevos en el siguiente frame.

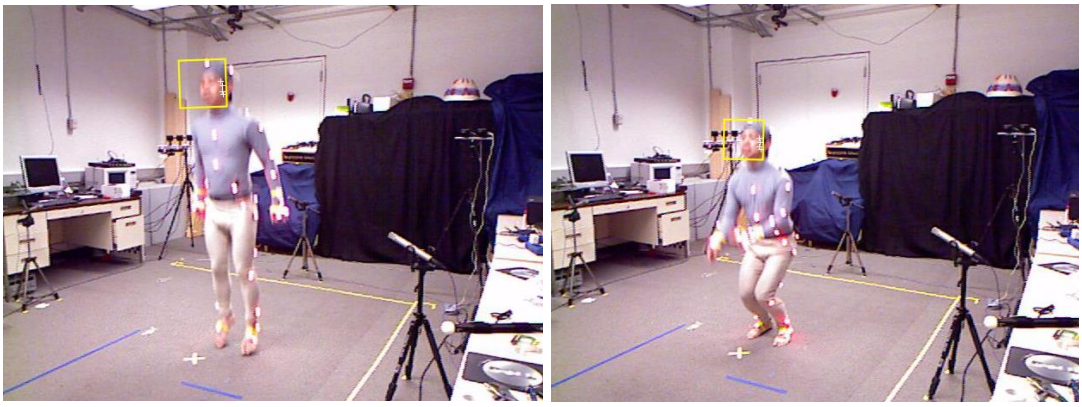


Figura 4.24. Seguimiento facial en distintos frames (vídeo 1.1 del anexo 2).

Como se puede apreciar en la figura 4.25. en ciertas ocasiones y fundamentalmente debido a oclusiones de la cara estos puntos desaparecen completamente y se pierde el seguimiento de la cara.

Figura 4.25. Último frame con detección de cara (vídeo 3.2 del anexo 2).

En la tabla 4.1 se pueden observar las estadísticas de los resultados obtenidos a lo largo del trabajo. Como se ha mencionado anteriormente se ha trabajado con 8 tipos diferentes de movimientos y dos repeticiones de cada uno. En todos ellos el sujeto se encontraba inicialmente con el rosto de frente a la cámara.

Nº Secuencia	Nº Repetición	Falsos Positivos	Falsos Negativos	% Seguimiento
1	**1**	0	0	100
	2	0	0	100
2	**1**	0	0	100
	2	0	0	100
3	**1**	1	-	-
	2	0	0	13.63
4	**1**	0	1	-
	2	0	1	-
5	**1**	0	0	100
	2	0	0	100
6	**1**	0	0	100
	2	0	0	100
7	**1**	0	0	100
	2	0	0	100
8	**1**	0	0	100
	2	0	0	100
TOTAL (%)	**-**	**6,25%**	**13,33%**	**93,36%**

Tabla 4.1. Validación de resultados.

5. Discusión.

Ante los resultados obtenidos durante la simulación, se puede afirmar que el proyecto cumple con los objetivos propuestos. Por un lado, el robot es capaz de construir un mapa de su entorno, consigue navegar en dicho entorno de forma autónoma y segura y además consigue adaptar sus movimientos a los de la persona que está guiado.

Sin embargo, el comportamiento del sistema no siempre se ajusta al deseado. Durante el proceso de mapeo el robot experimenta algunos problemas de localización y el mapa creado puede llegar a diferir mucho de la realidad. Esto ocurre, por ejemplo, en los pasillos, que son zonas uniformes donde coger referencias es difícil y el filtro de partículas falla. Ante esta situación, ha sido necesario hacer pasar al robot varias veces por las zonas de pasillo hasta mapearlas correctamente. Además, la localización también ha traído problemas en la zona superior izquierda del mapa (figura 5.1.), donde el robot no ha sido capaz de mapear fielmente las dos estancias (al tratarse de dos habitaciones contiguas tan parecidas, el robot no siempre sabe en cuál de ellas se encuentra).

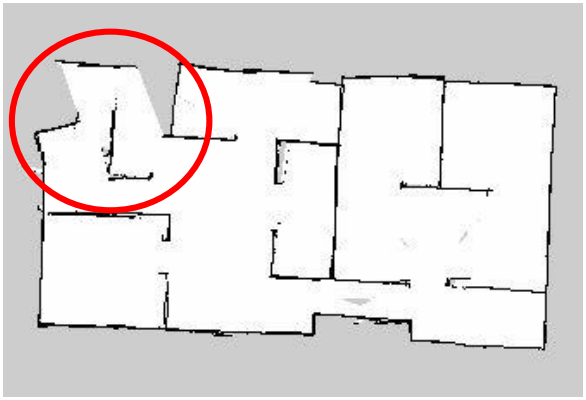


Figura 5.1. Zona del mapa dónde el robot ha tenido más problemas de localización.

En cuanto al mapeo, también cabe destacar la importancia de parámetros como la velocidad de refresco del mapa o el número de partículas empleadas en el filtrado, los cuales se han ido ajustando hasta obtener una precisión y velocidad de mapeo aceptables. El resultado más óptimo para la actualización del mapa entre 5, 15 y 25 segundos ha sido el obtenido con 5 segundos. En cuanto al número de partículas, entre los valores probados: 30, 80 y 100 partículas, el mejor resultado se produjo usando 100 partículas (figura 5.2.).

Ambos parámetros se encuentran en el archivo “gmapping.launch.xml”, incluido en el anexo 1.

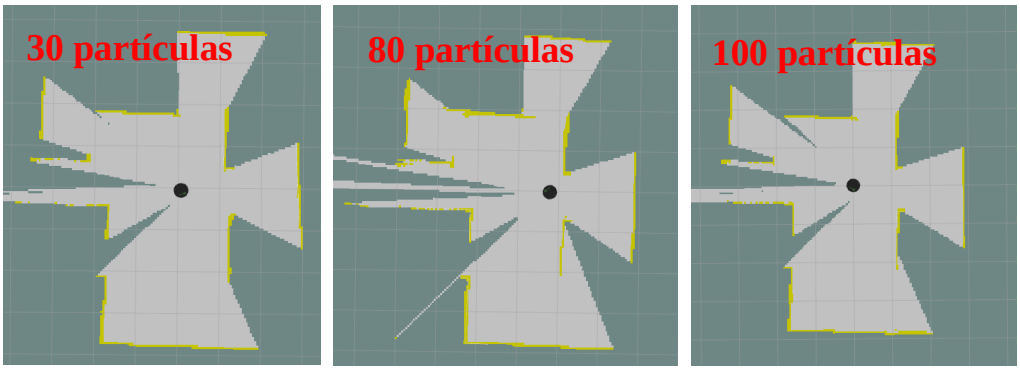


Figura 5.2. Resultados del mapeo tras dar una vuelta de 360º.

Otro aspecto a tener en cuenta es que sólo se ha probado el funcionamiento del sistema en el mundo creado. Por tanto, sólo se garantiza su correcto comportamiento en dicho entorno. Es probable que, en otras situaciones, con pasillos más largos o grandes habitaciones sin obstáculos, el problema de la localización empeore.

Como se ha visto en el apartado anterior, la tarea de navegación proporciona unos buenos resultados incluso cuando se han incluido obstáculos no previstos. Hay que tener en cuenta que los obstáculos simulados eran inmóviles, mientras que en la aplicación real éstos pueden ser móviles (personas). Por tanto, no se puede asegurar un comportamiento totalmente efectivo y libre de colisiones en esa situación.

Durante la tarea de guiado, cuando el nodo “adaptar_movimiento” envía al robot los comandos de reducción de velocidad, éstos interfieren con los que le está enviando el nodo “move_base”, por lo que el robot no navega de forma fluida cuando la persona está alejada entre 2 y 3 metros de él.

En la parte de visión por computador, se comprueba que los resultados obtenidos cumplen el objetivo propuesto y han sido bastante buenos. En la mayoría de casos se ha detectado la cara correctamente y el seguimiento ha sido del 100%. Aunque hay que mencionar que ha habido casos en los que se han producido algunos fallos:

- Un falso positivo en el movimiento 3, primera repetición (vídeo 3.1 del anexo 2): habiendo un solo rostro en la imagen se han detectado dos.

- Dos falsos negativos en el movimiento 4, en ambas repeticiones: habiendo un rostro, el algoritmo no lo ha detectado; se cree que se debe que el sujeto estaba demasiado ladeado inicialmente.

Una posible solución ambos fallos sería realizar un entrenamiento del algoritmo de Viola-Jones para detectar caras de perfil.

Otro fallo que se ha observado es la detención del seguimiento debido a la pérdida de puntos característicos del rostro tras una oclusión del mismo. En el movimiento 3, repetición 2, sí detecta la cara del sujeto, pero tras realizar un movimiento en el cual queda ocultado el rostro, se pierde el seguimiento del rostro. A pesar de volver a tener una imagen clara del rostro tras este movimiento, ya no es detectado, aspecto sujeto a mejoras en el futuro.

Hay que tener en cuenta que en este proyecto se ha trabajado con imágenes tomadas con una cámara fija, pero en el prototipo final la cámara será móvil (sobre el robot), por tanto, se debería analizar cómo es la respuesta obtenida con la cámara en movimiento.

6. Conclusiones y trabajo futuro.

6.1. Conclusiones.

Tras la finalización del proyecto, se pueden extraer una serie de conclusiones que se exponen a continuación.

La aplicación robótica de guiado de personas propuesta es capaz de desarrollar una navegación autónoma en un entorno mientras se adapta a los desplazamientos de la persona a la que acompaña. Además, el módulo de visión por computador es capaz de detectar y localizar a la persona guiada.

Para el guiado, primero es necesaria la obtención del mapa del entorno en el que va a navegar el robot, se consigue mediante la creación de un mapa de ocupación de celdillas. A continuación, se desarrolla la tarea de navegación usando distintas técnicas para cada tarea implicada: localización de Monte Carlo, planificación global de rutas mediante grafos de visibilidad y planificación local con el algoritmo DWA. En último lugar se utiliza un algoritmo de control de velocidades del robot para adecuarse al desplazamiento de la persona.

Para la localización de la persona, primero se ha detectado la cara de la persona y se han localizado los rasgos característicos del rosto mediante técnicas de detección de esquinas, por último, el seguimiento facial se ha realizado usando el método de segmentación por discontinuidad de flujo.

En cuanto a la aplicación final obtenida, de acuerdo con los resultados expuestos anteriormente, se puede afirmar que:

- El proceso de mapeo logra crear un mapa bastante ajustado al entorno simulado.
- Durante la navegación el robot es capaz de esquivar eficazmente obstáculos no previstos replanificando su trayectoria.

- En la simulación del guiado, el robot se adecuaba a los movimientos de la persona simulada, dependiendo éstos de la distancia entre ambos.

- En el módulo de visión ha conseguido detectar y seguir el rostro de una persona, bajo ciertas condiciones.

6.2. Trabajo futuro.

Tras finalizar este proyecto y estudiar la solución obtenida, surgen varias ideas para mejorar la presente aplicación.

En primer lugar, poder probar esta aplicación sobre una plataforma robótica real y comprobar los aspectos mencionados en el apartado anterior (Discusión) es lo más inmediato. Para ello habría que integrar el algoritmo desarrollado en la parte de visión con el sistema de ROS, MATLAB dispone de una toolbox llamada “Robotics System Toolbox” para realizar esta conexión. También se podría lograr implementando en OpenCV el algoritmo propuesto, ya que este programa viene ya integrado en ROS.

Se podría mejorar el algoritmo de detección de personas para tener la posibilidad de etiquetar a la persona guiada identificándola y diferenciándola claramente del resto de personas que pueda haber en el entorno.

Aunque esta aplicación inicialmente se ha planteado para detectar y seguir a una sola persona, se podría implementar una mejora en el algoritmo de localización de personas para poder trabajar con grupos de personas. Abriendo así el abanico de posibilidades de aplicación para el sistema.

Una ampliación interesante consistiría en utilizar también la imagen en color del entorno proporcionada por la Kinect. Se podría utilizar para realizar una grabación durante la navegación y así poder estudiar la ruta seguida por el robot, así como los posibles errores cometidos durante la navegación. Además, al llegar al punto de destino indicado se podría

tomar una fotografía que se envíe directamente a un sistema de control y así verificar que el robot ha cumplido su tarea correctamente.

Por último, la necesidad de una interfaz de comunicación entre el robot y el usuario es clara, por ello el desarrollo de la misma sería un punto interesante para desarrollar en el futuro.

7. Planificación y presupuesto.

7.1. Planificación.

Para llevar a cabo el trabajo se han desempeñado, durante los pasados 9 meses, las siguientes tareas:

Concepto	Horas de trabajo
Instalación de ROS	1h
Instalación de los paquetes de ROS usados	1h
Planteamiento de objetivos y del problema a resolver	4h
Estudio del guiado de personas	10h
Aprendizaje de ROS	80h
Creación del entorno en Gazebo	2h
Estudio del mapeo	10h
Creación del paquete “mapeo”	1h
Implementación del mapeo	8h
Estudio de la navegación	15h
Creación del paquete “navegación”	1h
Implementación de la navegación	10h
Estudio de la detección y seguimiento facial	15h
Diseño del algoritmo de detección y seguimiento facial	35h
Pruebas y ensayos con el algoritmo de detección y seguimiento facial	7h
Generación de resultados	5h
Creación del paquete “guiado”	2h
Estudio de la adaptación de movimientos	20h
Diseño del algoritmo de adaptación de movimientos	15h
Implementación del guiado	25h
Redacción de la documentación	65h
TOTAL	**332h**

En total, se establece una medición de mano de obra de TRESCIENTAS TREINTA Y DOS HORAS de trabajo.

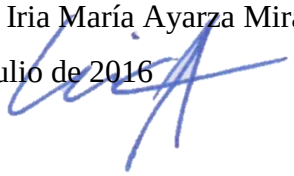
7.2. Presupuesto.

En base a las horas de trabajo requeridas y a los materiales empleados, se procede a calcular el presupuesto del trabajo desarrollado.

Concepto	Coste unitario	Medición	Coste
Mano de obra	25€/h	332h	8.300€
Ordenador (Intel i7-6500U, 8GB RAM, 1TB y gráfica R5 de 2GB)	30€/mes	9 meses	270€
Ubuntu Linux 14.04	0€/h	0€/h	0€
ROS Indigo	0€/h	0€/h	0€
Matlab + Image Processing Toolbox (licencia estudiante)	70€	1 ud.	70€
Database de los movimientos (Kinect)	0€	1 ud.	0€
SUB-TOTAL			**8.640€**
Gastos generales (%5)			432€
Beneficio (7%)			604,8€
IVA (21%)			1.814,4€
TOTAL + IVA			**11.491,2€**

Para las mediciones y precios dados, se calcula un presupuesto total de ONCE MIL CUATROCIENTOS NOVENTA Y UN EUROS CON VEINTE CÉNTIMOS.

Fdo. Iria María Ayarza Mira
10 Julio de 2016



8. Listado de anexos.

8.1. Contenidos del anexo 1.

Dentro de este anexo se encuentra el código utilizado para el desarrollo del proyecto en ROS, junto con la documentación asociada a las distintas partes del proyecto. Se especifican a continuación las carpetas y su contenido.

1. **Robot_Guía**: código y archivos de ROS empleados en el proyecto. Dentro de esta carpeta hay otras cuatro:

   - Goals: almacena los programas que le indican al robot las ubicaciones de: el origen de coordenada del mapa (goal_0.py) y las distintas habitaciones del mapa (goal_1.py, goal_2.py, goal_3.py, goal_4.py, goal_5.py, goal_6.py, goal_7.py, goal_8.py).

   - Guiado: en *launch* está el archivo de lanzamiento de la aplicación final (guiado_turtlebot.launch); en *src* hay dos archivos: el código que simula a la persona (marcador_persona.cpp) y el código que adapta los movimientos del robot a los de la persona (adaptar_movimiento.cpp).

   - Mapeo: en *launch* está el archivo de lanzamiento de la aplicación de mapeo (mapeo_turtlebot.launch); en *worlds* se encuentra el archivo del entorno creado en Gazebo (mundo_definitivo.world); en *rviz* está la configuración de este programa para realizar el mapeo (mapeo.rviz).

   - Navegación: en *launch* está el archivo de lanzamiento de la aplicación de navegación (navegacion_turtlebot.launch); en *worlds* se encuentra el archivo del entorno creado en Gazebo (mundo_definitivo.world); en map están los archivos del mapa creado (mapa_definitivo.yaml y mapa_definitivo.png).

2. **Imágenes y vídeos**: contiene imágenes y vídeos documentando los tres procesos desarrollados en ROS.

    - Mapeo: contiene una imagen del momento inicial del mapeo en Rviz (momento_inicial_rviz_mapeo.png), el rosgraph del mapeo (rosgraph_mapeo.png), una imagen del mapa tras dar una vuelta de 360º (mapa_360.png), el mapa creado (mapa_definitivo.jpg) y un vídeo del proceso del mapeo (mapeo.mp4).

    - Navegación: contiene dos imágenes, una del momento inicial de la navegación en Gazebo y Rviz (momento_inicial_navegacion.png) y otra del rosgraph de la navegación (rosgraph_navegacion.png). También incluye cuatro vídeos mostrando distintas formas de navegación del robot: indicando al robot directamente el punto en el mapa con el ratón y sin obstáculos no previstos (1_navegacion_2dgoal.mp4), indicando al robot directamente el punto en el mapa y con un obstáculo no previsto (2_navegacion_2dgoal_obstaculo.mp4), indicando al robot ir a un punto del mapa del que no dispone información completa (3_navegacion_2dgoal_sininfo.mp4) y, por último, indicando al robot mediante código la ubicación final en el mapa y con un obstáculo no previsto (4_navegacion_goal_obstaculo.mp4).

    - Guiado: contiene una imagen inicial del guiado en Rviz (momento_inicial_rviz_guiado.png), el rosgraph del guiado (rosgraph_guiado.png) y un vídeo documentando el proceso de guiado (guiado.mp4).

8.2. Contenidos del anexo 2.

Dentro de este anexo se encuentran dos carpetas, cada una de ellas contiene lo especificado a continuación:

1. **Código Matlab**: en esta carpeta se encuentran los dos archivos de código utilizados durante el trabajo, que son:

    - "crear_video.m" es un archivo con el código para convertir las secuencias de imágenes en vídeo.

    - "detección_seguimiento_facial.m" es un archivo con el código empleado en el proyecto para detectar y seguir la cara de la persona.

2. **Vídeos**: en esta carpeta se pueden encontrar los vídeos de los distintos movimientos de la persona usados en el desarrollo del proyecto. Diferenciando dos subcarpetas:

    - Vídeos iniciales: son los vídeos de la persona antes de aplicar el método (video_1.1, video_1.2, video_2.1, video_2.2, video_3.1, video_3.2, video_4.1, video_4.2, video_5.1, video_5.2, video_6.1, video_6.2, video_7.1, video_7.2, video_8.1, video_8.2,).

    - Vídeos procesados: son los vídeos una vez tratados, en los cuales se aprecia el resultado del método empleado (video_proc_1.1, video_ proc_1.2, video_ proc_2.1, video_ proc_2.2, video_ proc_3.2, video_ proc_5.1, video_ proc_5.2, video_ proc_6.1, video_ proc_6.2, video_ proc_7.1, video_ proc_7.2, video_ proc_8.1, video_ proc_8.2,).

9. Bibliografía.

[1] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The Museum Tour-Guide Robot RHINO," in Autonome Mobile Systeme 1998: 14. Fachgespräch Karlsruhe, 30. November–1. Dezember 1998, H. Wörn, R. Dillmann, and D. Henrich, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 245–254.

[2] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "MINERVA: A Tour-Guide Robot that Learns," in KI-99: Advances in Artificial Intelligence: 23rd Annual German Conference on Artificial Intelligence Bonn, Germany, September 13–15, 1999 Proceedings, W. Burgard, A. B. Cremers, and T. Cristaller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 14–26.

[3] D. Karreman, G. Ludden, and V. Evers, "Visiting Cultural Heritage with a Tour Guide Robot: A User Evaluation Study in-the-Wild," in Social Robotics: 7th International Conference, ICSR 2015, Paris, France, October 26-30, 2015, Proceedings, A. Tapus, E. André, J.-C. Martin, F. Ferland, and M. Ammi, Eds. Cham: Springer International Publishing, 2015, pp. 317–326.

[4] L. Susperregi, I. Fernandez, A. Fernandez, S. Fernandez, I. Maurtua, and I. L. de Vallejo, "Interacting with a Robot: A Guide Robot Understanding Natural Language Instructions," in Ubiquitous Computing and Ambient Intelligence: 6th International Conference, UCAmI 2012, Vitoria-Gasteiz, Spain, December 3-5, 2012. Proceedings, J. Bravo, D. López-de-Ipiña, and F. Moya, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 185–192.

[5] A. Martínez, H. Hassan, C. Domínguez, P. López. "Prototipo de robot de servicio para guiado de personas por visión". Universidad Politécnica de Valencia, Valencia, 2014.

[6] Víctor Fernando Muñoz Martínez. "Planificación de Trayectorias para robots móviles", 1995. Capítulo 2: Navegación, pp. 21-26.

[7] Ingemar J. Cox, Blanche. "Position Estimation for an Autonomous Robot Vehicle". Autonomous robot vehicles, 1990. Springer, pp. 221-228.

[8] G. Grisetti, C. Stachniss, and W.Burgard. "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters", IEEE Transactions on Robotics, 2007, Volume 23, pp. 34-46.

[9] J. Álvarez Álvarez, "Problemas característicos de la robótica móvil", 1st ed. EPI Gijón, 2015.

[10] Y. Chao, "A developed dijkstra algorithm and simulation of urban path search," in Computer Science and Education (ICCSE), 2010 5th International Conference on, Aug 2010, pp. 1164–1167.

[11] A. Stentz and I. C. Mellon, "Optimal and efficient path planning for unknown and dynamic environments," International Journal of Robotics and Automation, vol. 10, pp. 89-100, 1993.

[12] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: The case for A*," Int. J. Geogr. Inf. Sci., vol. 23, no. 4, pp. 531–543, Apr. 2009.

[13] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," Artificial Intelligence, vol. 155, no. 1-2, pp. 93 – 146, 2004.

[14] Marija Dakulović, Ivan Petrović, "Two-way D∗ algorithm for path planning and replanning", Robotics and Autonomous Systems, Volume 59, Issue 5, 2011, pp. 329-342.

[15] A. Bundy and L. Wallen, "Breadth-First Search," in Catalogue of Artificial Intelligence Tools, A. Bundy and L. Wallen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1984, pp. 13–13.

[16] A. Bundy and L. Wallen, “Depth-First Search,” in Catalogue of Artificial Intelligence Tools, A. Bundy and L. Wallen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1984, pp. 29–29.

[17] "Evasión de Obstáculos", Revista.unam.mx, 2016. [Online]. Available: http://www.revista.unam.mx/vol.6/num1/art02/art02-2.htm [Accessed: 28- Jun- 2016].

[18] S. M. Lavalle, J. J. Kuffner, and Jr., “Rapidly-exploring random trees: Progress and prospects,” Algorithmic and Computational Robotics: New Directions, 2000, pp. 293-308.

[19] Fox, D.; Burgard, W. and Thrun, S. "The dynamic window approach to collision avoidance". Robotics & Automation Magazine 1997, IEEE, pp: 23–33.

[20] Viola, Paul A. and Jones, Michael J. “Rapid Object Detection using a Boosted Cascade of Simple Features”, IEEE CVPR 2001.

[21] Demirkir, C. Sankur, B. “Signal Processing and Communications Applications”. IEEE 14th. 2006, pp. 1-4.

[22] H. Allende-Cid, R. Salas, H. Allende, and R. Ñanculef, “Robust Alternating AdaBoost,” in Progress in Pattern Recognition, Image Analysis and Applications: 12th. Proceedings, L. Rueda, D. Mery, and J. Kittler, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 427–436.

[23] H. Fassold, J. Rosner, P. Schallauer and W. Bailer. “Realtime KLT Feature Point Tracking for High Definition Video” 2009.

[24] B. D. Lucas and T. Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision" International Joint Conference on Artificial Intelligence 1981, pp. 674-679.

[25] C. Tomasi and T. Kanade. "Detection and Tracking of Point Features". Technical Report CMU-CS-91-132 Carnegie-Mellon University, 1991.

[26] Jianbo Shi and Carlo Tomasi. "Good Features to Track. IEEE Conference on Computer Vision and Pattern Recognition", 1994, pp. 593–600.

[27] "ROS | Erle Robotics Gitbook", Erlerobotics.gitbooks.io, 2015. [Online]. Available: https://erlerobotics.gitbooks.io/erlerobot/content/es/ros/ROS.html [Accessed: 01- Jul- 2016].

[28] "ROS/Tutorials - ROS Wiki", Wiki.ros.org. [Online]. Available: http://wiki.ros.org/ROS/Tutorials [Accessed: 1- Jul- 2016].

[29] "Turtlebot Installation ROS Indigo", Wiki.ros.org. [Online]. Available: http://wiki.ros.org/turtlebot/Tutorials/indigo/Turtlebot%20Installation [Accessed: 01- Jul- 2016].

[30] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal y R. Bajesy "Berkekly MHAD: A Comprehensive Multimodel Human Action Database". In Proceedings of the IEEE Workshop on Applications on Computer Vision (WACV), 2013.