



UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

**TECNOLOGÍA ESPECÍFICA DE
COMPUTACIÓN**

TRABAJO FIN DE GRADO

Diseño e implementación de un módulo de localización
y navegación para un vehículo autoguiado

Rafael Muñoz González

Febrero de 2019





UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

**TECNOLOGÍA ESPECÍFICA DE
COMPUTACIÓN**

TRABAJO FIN DE GRADO

**Diseño e implementación de un módulo de localización
y navegación para un vehículo autoguiado**

Autor: Rafael Muñoz González

Directores: Ismael García Varea

Cristina Romero González

Febrero de 2019

A mi familia y amigos

Declaración de Autoría

Yo, Rafael Muñoz González con DNI 06287234T , declaro que soy el único autor del trabajo de fin de grado titulado "Diseño e implementación de un módulo de localización y navegación para un vehículo autoguiado" y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 12 de Febrero de 2019

Fdo.: Rafael Muñoz González

Resumen

El presente Trabajo de Fin de Grado recoge el desarrollo teórico de un conjunto de métodos de localización para su uso práctico en diferentes entornos con robots móviles. Este conjunto de métodos de localización incluye el filtro de Kalman y el filtro de partículas. Además el trabajo se centrará en el desarrollo de un módulo de localización para vehículos auto-guiados, este módulo de localización estará basado en un filtro de partículas desarrollado específicamente por el autor mediante diferentes herramientas software como ROS y Gazebo, debido a esto también se hace una introducción a estas herramientas que permiten la simulación y el control de robots.

Agradecimientos

Este Trabajo de Fin de Grado realizado en la Universidad de Castilla La Mancha es un esfuerzo en el que participaron diversas personas, ya sea directamente o indirectamente, opinando, corrigiendo, animando y acompañando en momentos de frustración y de felicidad. Por ello me gustaría agradecer el apoyo de mis padres, de mi hermano y de mis amigos, gracias a quienes ellos soy quien soy y son a ellos hacia los que me gustaría expresar mi más sincero agradecimiento por apoyarme durante toda esta etapa.

Índice general

ÍNDICE DE FIGURAS	XI
Lista de Figuras	XIII
ÍNDICE DE TABLAS	XIII
Lista de Tablas	1
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	3
1.3. Metodología del desarrollo	3
1.4. Estructura de la memoria	5
2. Estado del Arte	7
2.1. La Robótica	7
2.2. El problema de la localización	8
2.3. Algoritmos de localización	11
2.4. Escenario del problema	20
2.5. Herramientas Software utilizadas	20
3. Desarrollo del trabajo y experimentos	25
3.1. Desarrollo del primer experimento	25
3.2. Desarrollo del segundo experimento	35
4. Conclusiones y Propuestas	49
4.1. Conclusiones	49
4.2. Competencias cubiertas	49
4.3. Trabajo futuro	50
4.4. Bibliografia	50
BIBLIOGRAFIA	51
CONTENIDO DEL CD	53

ÍNDICE DE FIGURAS

1.1.	Pioneer 3-AT	2
1.2.	Modelo en el entorno Gazebo de un robot Pioneer 3-AT	2
1.3.	Coche autónomo	2
1.4.	Diagrama de Gantt para la organización del trabajo	5
2.1.	Localización al comienzo	9
2.2.	Localización tras realizar medidas	9
2.3.	Ejemplo de distribución Gaussiana	11
2.4.	Proceso del algoritmo de filtro de partículas	17
2.5.	Diagrama Resampling Wheel	18
2.6.	Obtención de nuevas partículas en Resampling Wheel	20
2.7.	22
2.8.	Sistema de navegación de ROS	22
2.9.	Sistema de navegación de ROS	23
3.1.	Círculo de pruebas	26
3.2.	Visualización en rviz de la generación del mapa mediante SLAM	27
3.3.	Mapa generado con técnica de SLAM	28
3.4.	Configuración de rviz para mostrar el mapa con el detector de obstáculos	30
3.5.	Error en el método de odometría, eje X	31
3.6.	Error en el método de odometría, eje Y	31
3.7.	Error en el método de Filtro de Partículas, eje Y	32
3.8.	Error en el método de Filtro de Partículas, eje Y	32
3.9.	Error en el método de Filtro de Kalman, eje X	33
3.10.	Error en el método de Filtro de Kalman, eje Y	33
3.11.	Comparación de errores en el eje X entre los diferentes métodos de localización	34
3.12.	Comparación de errores en el eje Y entre los diferentes métodos de localización	35
3.13.	Entorno simulado en Gazebo	36
3.14.	Nube de puntos obtenida al realizar un mapeo con rtabmap	37

3.15. Nube de puntos obtenida recortando la nube de puntos obtenida mediante mapeo.	38
3.16. Área de 5 metros alrededor de la cámara	38
3.17. Nube de puntos obtenida por el sensor	40
3.18. Imagen obtenida desde arriba proyectando la nube de puntos del sensor sobre el plano z	40
3.19. Zona de interés seleccionada de la imagen	41
3.20. Función exponencial decreciente, $3^{1/x}$	42
3.21. Función exponencial decreciente, $f(x) = 3^{1/(x/100)}$	43
3.22. Porcentaje de datos en la distribución normal	44
3.23. Partículas tras ser inicializadas	44
3.24. Partículas tras la fase de predicción al haber aplicado una rotación al robot	45
3.25. Partículas elegidas para la próxima generación de nuevas partículas . .	46
3.26. Mejor partícula del conjunto y posición real	47
3.27. Esquema del filtro de partículas	47

ÍNDICE DE TABLAS

2.1.	Conjunto de celdas al principio del ejemplo	10
2.2.	Conjunto de celdas después de realizar la primera medida del ejemplo	10
2.3.	Algoritmo del Filtro de Kalman	13
2.4.	Algoritmo de Filtro de Partículas	16
2.5.	Selección de un Índice aleatorio	18
2.6.	Algoritmo Resampling Wheel	19

Capítulo 1

Introducción

La robótica autónoma es el tema en el cual se inspira este proyecto debido al auge que está teniendo estos últimos años. Este auge es debido a las grandes innovaciones, la gran cantidad de dinero invertido y las posibilidades que ofrece al mundo. Por ello introduciremos la importancia y la aplicación de la robótica en el panorama actual.

La robótica es el arte de percibir y manipular el mundo real a través de dispositivos controlados por ordenador, robots. Los robots son capaces de obtener información del entorno a través de sensores y ejecutar acciones con actuadores[1].

1.1. Motivación

La robótica tiene muchas aplicaciones, entre ellas se encuentran, el uso de robots industriales para realizar tareas repetitivas, el uso de robots con fines educativos, la utilización de robots en la vida cotidiana, robots utilizados en el campo de la medicina, etc.

El presente Trabajo de Fin de Grado tratará sobre la robótica y se centrará en el problema de la localización, para ello se utilizará una cámara, un mapa del entorno y se conseguirá que el robot sea capaz de conocer su posición con un mínimo margen de error.

Para resolver el problema de la localización se utilizarán paquetes y nodos que son generados por código en C++ utilizando el framework ROS[4], el cual provee de librerías y herramientas para ayudar a crear aplicaciones para robots, permitiéndonos la abstracción del hardware, la comunicación por mensajes y herramientas de visualización.

Además, para una buena visualización a la hora de enfrentarse al problema se utilizará un entorno de simulación llamado Gazebo[2], donde se importará un modelo del robot Pioneer 3-AT que es el robot que nos facilita la universidad para realizar el proyecto (Véase la figura 1.1 y 1.2).



Figura 1.1: Pioner 3-AT

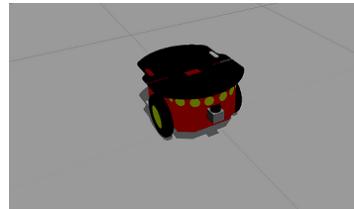


Figura 1.2: Modelo en el entorno Gazebo de un robot Pioneer 3-AT

1.1.1. Finalidad del Trabajo de Fin de Grado

El presente Trabajo de Fin de Grado se centrará en el estudio de algoritmos de localización que podemos encontrar en "Probabilistic Robotics" [7] y el desarrollo de un algoritmo de localización con el mínimo error posible para utilizar en un vehículo autónomo. Además se estudiarán conceptos de navegación y de visión artificial y reconocimiento de patrones.



Figura 1.3: Coche autónomo

Para la localización se estudiará y desarrollará el método de localización Filtro de Partículas [6], debido a que de igual forma que el Filtro de Kalman [3] los dos métodos permiten conocer el estado de un sistema dinámico cuando tenemos un modelo bayesiano. La ventaja de usar el Filtro de Partículas reside en como trabaja este método según aumenta la dimensionalidad, pues en el Filtro de Kalman según aumenta la dimensionalidad los problemas se vuelven intratables.

En cuanto a la visión artificial, utilizando una cámara podremos conocer las líneas de la carretera que utilizaremos para conocer la posición de nuestro robot mediante técnicas de visión artificial como la detección de histogramas, además, la navegación nos permitirá la navegación del robot por el entorno.

1.2. Objetivos

El objetivo de este TFG es el estudio y desarrollo de un algoritmo de localización para su posterior integración en el paquete de navegación que se proporciona en ROS.

Para la implementación del algoritmo se utilizará la técnica de filtros de partículas y como información de entrada la proporcionada por un sistema de visión, en concreto una cámara Asus Xtion Pro Live.

El resultado será un componente de localización para poder ser utilizado en la plataforma ROS.

Para el desarrollo del algoritmo de localización se ha realizado una investigación obteniendo conocimientos sobre:

- Estado del arte, para conocer el panorama actual del campo de la robótica, como ha ido evolucionando con el tiempo y sus últimas aplicaciones.
- Localización clásica, la cual incluye la localización por el método de filtro de kalman [3] y el filtro de partículas[6] .
- Localización respecto a la información visual, que se centra en usar la cámara y obtener las líneas de la carretera para saber la localización de nuestro vehículo.
- Navegación local, que se centra en el estudio de las velocidades lineales y angulares.
- Librerías para implementar el componente de ROS, como los tutoriales básicos, TF y PCL.
- Hardware Pioneer 3AT.
- Comunicación entre otros módulos del sistema.

1.3. Metodología del desarrollo

Para la realización de este trabajo se ha utilizado una metodología de desarrollo ágil, concretamente el framework Scrum adaptado[5] donde se realizan entregas parciales

y regulares del producto final priorizadas por el beneficio que aportan al proyecto. Se ha elegido el uso de Scrum porque está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos.

Los roles de los integrantes del equipo son:

- **Dueños del producto:** Ismael García Varea y Cristina Romero González. Coordinadores entre el equipo para establecer una buena comunicación y evitar problemas al establecer los requisitos y compromisos con el proyecto.
- **Equipo de trabajo y Scrum Master:** Rafael Muñoz González. Estudiante de 4º de Ingeniería Informática, defensor del trabajo de fin de grado.

La organización del trabajo tal y como queda esquematizado en el diagrama de Gantt (vease la figura 1.4) es la siguiente:

- **Inicio del proyecto:** En esta iteración se planteará el proyecto con su planificación, definiendo los requisitos del sistema y todos los aspectos técnicos del proyecto.
- **Estudio del estado del arte:** En esta iteración nos centraremos en el estudio de la base teórica sobre la que se sustenta el escrito, y la cual se usa en el desarrollo posterior del escrito, en la aplicación práctica del trabajo.
- **Desarrollo de un mapa para realizar pruebas y validación:** En esta iteración se conseguirá realizar o encontrar un mapa para permitir la realización de pruebas y validaciones en el desarrollo del sistema.
- **Desarrollo del módulo de localización:** En esta iteración nos centraremos en desarrollar el módulo de localización basado en el método del filtro de partículas.
- **Desarrollo del módulo de navegación:** En esta iteración se buscará la forma de interpretar y tratar los datos obtenidos por el módulo de percepción del sistema para evitar obstáculos y permitir la navegación autónoma del vehículo.
- **Pruebas y validación:** En esta iteración nos encargaremos de buscar fallos en el sistema para posteriormente solucionarlos y lograr el objetivo propuesto en el trabajo de fin de grado.
- **Presentación y defensa del trabajo de fin de grado.**

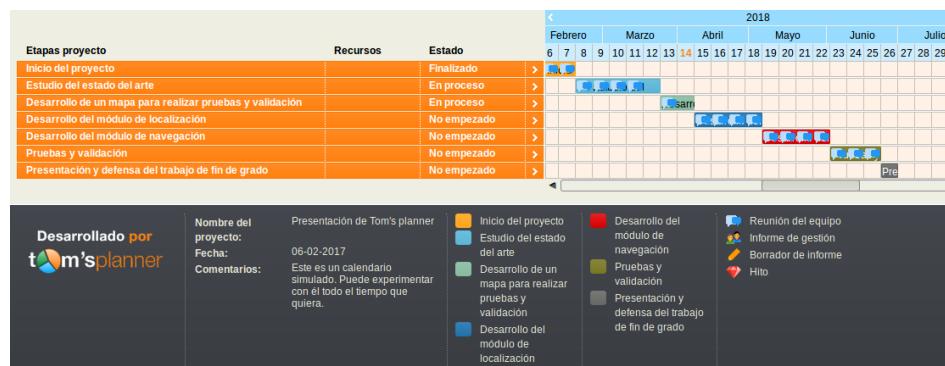


Figura 1.4: Diagrama de Gantt para la organización del trabajo

El funcionamiento de este framework ha consistido en la creación de un "Product Backlog." en el que se han añadido requisitos del sistema. Posteriormente se realiza una planificación del Sprint generando un "Sprint Backlog" para decidir que requisitos se van a ir implementando en el sprint, y se dividen cada uno de los objetos del "Sprint Backlog." en pequeñas tareas para centrar más el trabajo a realizar. Posteriormente se ha realizado un Sprint de un mes para completar el "Sprint Backlog". Realizándose un planteamiento de Scrum diario, revisiones del sprint para inspeccionar el Incremento y adaptar el "Product Backlog" si fuese necesario y retrospectiva de sprint al final de cada sprint para que el Equipo Scrum se inspeccione a si mismo y se creen mejoras para abordar durante el siguiente Sprint.

1.4. Estructura de la memoria

La presente memoria del Trabajo de Fin de grado se desarrolla presentando los contenidos teóricos y posteriormente su uso en la práctica. Se ha definido esta estructura para poner las bases para luego poder entender la práctica.

- **Capítulo 1: Introducción** Se da una primera explicación del problema, los objetivos y el procedimiento realizado en el Trabajo de Fin de Grado.
- **Capítulo 2: Estado del Arte** Se da una explicación del panorama actual en este ámbito, se explican los diferentes filtros estadísticos con su respectivas bases teóricas, se explica el escenario del problema y las herramientas utilizadas.
- **Capítulo 3: Experimentos y resultados** Se muestran y explican los diferentes experimentos realizados. Además se adjunta los resultados de los experimentos realizados con ROS en el entorno de simulación Gazebo junto con una conclusión y reflexión para analizar posibles mejoras.

- **Capítulo 4: Conclusiones y propuestas** Se desarrolla una reflexión final sobre el presente Trabajo de Fin de Grado, específicamente se habla sobre la efectividad de la propuesta realizada y trabajos futuros.
- **Bibliografía** Aparece los documentos, artículos y páginas web de donde se ha obtenido la información.

Capítulo 2

Estado del Arte

2.1. La Robótica

La robótica es el arte de percibir y manipular el mundo real a través de dispositivos controlados por ordenador, robots. Los robots son capaces de obtener información del entorno a través de sensores y ejecutar acciones con actuadores[1].

La robótica es un área de estudio relativamente nueva, pues, a pesar de sus numerosas aplicaciones y aunque hayan pasado más de 50 años desde que surgió la robótica, esta todavía no ha dado el salto de extenderse por completo en la vida cotidiana, ya que tiene varios inconvenientes o problemas que impiden dicho salto. El primer impedimento, el robot debe conocer el entorno para actuar sobre él; el segundo, el robot debe saber su posición para poder moverse o realizar acciones con precisión; el tercero, la navegación, ya que se necesita de un planificador para realizar movimientos desde un punto inicial a uno final minimizando el tiempo empleado y evitando obstáculos.

En el campo de la robótica se distinguen distintos tipos de robots[?]: REFERENCIA

- **Robot industrial o manipulador:** Son artíluguos mecánicos y electrónicos destinados a realizar de forma automática determinados procesos de fabricación o manipulación. Se utilizan principalmente en la fabricación industrial.
- **Androïdes o humanoides:** Intentan reproducir total o parcialmente la forma y el comportamiento del ser humano.
- **Zoomórficos:** Reproducen con mayor o menor grado derealismo, los sistemas de locomoción de diversos seres vivos.
- **Robot móviles:** Los robots móviles están provistos de algún tipo de mecanismo que les permite desplazarse de lugar autónomamente, como pueden ser patas,

ruedas u orugas y reciben la información del entorno con sus propios sistemas sensores.

El presente TFG se centrará en los robots de tipo móviles, los cuales pueden moverse por cualquier medio utilizando su propia energía.

Las aplicaciones de estos robots suelen ser:

- **El desplazamiento de dichos robots por zonas donde existe la incapacidad o gran dificultad de desplazamiento por parte de las personas en dicha zona.** Por ejemplo robots espaciales que pueden sobrevivir sin aire ni agua o robots de investigación en los océanos ya que la presión no permite a los seres humanos descender más de 100 metros de profundidad.
- **La realización de tareas rutinarias en entornos, donde la eficiencia y la movilidad de estas máquinas reemplazan directamente la presencia humana.** Por ejemplo la agricultura o el transporte de materiales en fábricas.
- **Asistencia personal, rehabilitación y entretenimiento.** Por ejemplo una silla de ruedas con sensores e inteligencia computacional.

Además de estas aplicaciones, en estos tiempos están comenzando a aparecer los coches autónomos, que entrarían dentro de la clase de aplicaciones sobre la realización de tareas rutinarias en entornos, donde la eficiencia y la movilidad de estas máquinas reemplazan directamente la presencia humana.

Para el desarrollo de este tipo de aplicaciones necesitamos conocer, como bien se dijo anteriormente, la posición del robot obtenida mediante un algoritmo de localización mediante sensores.

2.2. El problema de la localización

La localización es la capacidad de una maquina para localizarse a si misma en el espacio. Cuando nosotros pensamos en localizar un robot en el espacio podría venirnos a la cabeza usar un sensor GPS, pero el problema con este método de localización es la cantidad de error que se obtiene utilizando un sensor GPS, el cual está entre 3 y 15 metros.

Para algunas tareas nos sería suficiente usar ese tipo de sensor, pero para un coche autónomo es algo impensable, necesitamos un error entre 2 y 10 centímetros para que sea un sistema seguro. Para ello deberemos mejorar el método de localización del GPS

utilizando algoritmos probabilísticos de localización que nos permiten tener un error mínimo conocido el mapa del espacio y un sensor que nos permita realizar medidas para distinguir una posición de otra, de esta manera el sistema es capaz de conocer su localización a través de medidas que van reduciendo la incertidumbre.

Estas medidas son tomadas cuando el robot se mueve, es decir, al comienzo el robot no sabrá cual es su localización y tendremos una distribución de probabilidad sobre la posición del robot con todos los estados igual de posibles.(Figura 2.1), pero después de realizar medidas, algunos estados aumentarán su probabilidad y otros la decrementaran.(Figura 2.2).

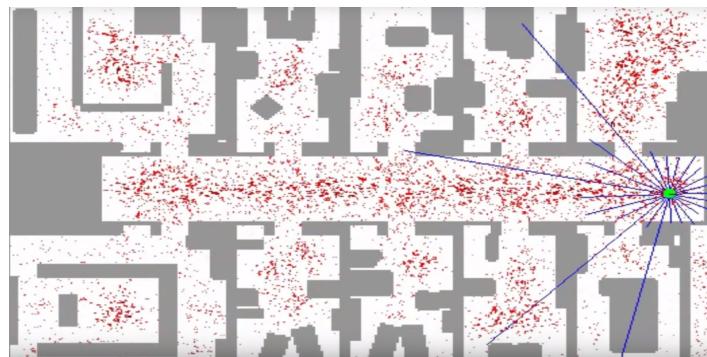


Figura 2.1: Localización al comienzo

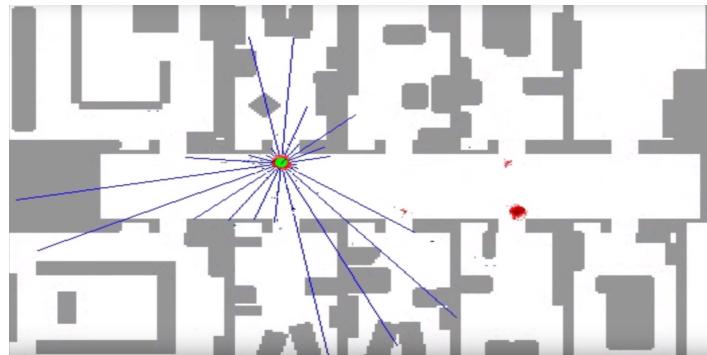


Figura 2.2: Localización tras realizar medidas

Un ejemplo de como funciona el método de localización basado en histogramas en un mundo con una sola dimensión sería:

Teniendo un robot que no conoce su localización y el mundo es definido por un conjunto de 10 celdas, 3 de las cuales tienen una puerta que nos permiten diferenciar esas 3 celdas de las demás. Como vemos en la tabla 2.1, al comienzo la distribución de probabilidad es uniforme, la probabilidad de que el robot esté en cualquiera de los estados es de 0.1.

Tabla 2.1: Conjunto de celdas al principio del ejemplo

1	2	3	4	5	6	7	8	9	10
Puerta	Vacio	Vacio	Vacio	Puerta	Vacio	Puerta	Vacio	Vacio	Vacio
P=0.1	P=0.1	P=0.1	P=0.1	P=0.1	P=0.1	P=0.1	P=0.1	P=0.1	P=0.1

Posteriormente, se realiza la primera medida y el robot conoce que la casilla en la que esta tiene una puerta. Debido a eso, la distribución de probabilidad cambia, incrementándose en los estados con puerta y decrementándose en los estados vacíos.

Se podría multiplicar la probabilidad de los estados correctos por 0.6 y de los incorrectos por 0.2, de esta forma evitamos hacer caso omiso a los sensores, ya que estos pueden fallar.

Después Normalizamos los valores de la nueva tabla para obtener una distribución de probabilidad como podemos ver en la tabla 2.2.

Tabla 2.2: Conjunto de celdas después de realizar la primera medida del ejemplo

1	2	3	4	5	6	7	8	9	10
Puerta	Vacio	Vacio	Vacio	Puerta	Vacio	Puerta	Vacio	Vacio	Vacio
P=0.1875	P=0.0625	P=0.0625	P=0.0625	P=0.1875	P=0.0625	P=0.1875	P=0.0625	P=0.0625	P=0.0625

Como se puede ver, cuando el robot se mueve, se produce una convolución y esto es debido al conocimiento de la distancia recorrida y la orientación del robot, de esta forma la distribución de probabilidad se aplanará un poco en puntos altos. Por lo que según se vayan realizando medidas, la localización irá mejorando.

Para la localización de los robots se suelen utilizar los filtro bayesianos, los cuales pueden ser no paramétricos o paramétricos.

Los paramétricos son aquellos que representan la creencia del estado como un vector de media y su co-varianza, formando así una distribución de probabilidad con forma de campana de Gauss en torno a una solución. En este grupo encontramos los Filtros de Kalman.

Por otro lado, los no paramétricos son aquellos filtros que representan la solución con un número finito de valores, y en el caso de que pudiese haber infinitos valores adoptarían a la perfección la curva de probabilidad de la solución, permitiendo resolver tanto problemas lineales como no lineales. En este grupo encontramos el Filtro de Histogramas y el Filtro de Partículas.

2.3. Algoritmos de localización

2.3.1. Filtro de Kalman

Introducción al Filtro de Kalman

El Filtro de Kalman es posiblemente la implementación más conocida de los filtros basados en filtros bayesianos debido a la simplicidad y eficiencia en el momento de filtrar y predecir estados. Este método fue inventado por Swerling (1958) y Kalman (1960).

Características del Filtro de Kalman

El Filtro de Kalman es un conjunto de ecuaciones matemáticas que implementan un estimador óptimo del tipo predictor-corrector.

El Filtro de Kalman tiene varias características que diferencian este método de localización al anteriormente descrito en el ejemplo (el método basado en histogramas) que convertía el espacio continuo en regiones discretas.

Entre estas diferencias se encuentra que es un filtro perimétrico debido al uso de una distribución Gaussiana para mostrar los posibles estados del espacio de forma que el espacio se representa con una función continua. Esta forma de representar el espacio utiliza dos variables, una media del eje x, μ_t , y una matriz de co-varianza Σ_t que representa el ancho de la Gaussiana tal y como podemos ver en la figura 2.3.

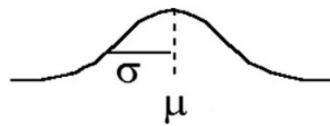


Figura 2.3: Ejemplo de distribución Gaussiana

La función de la distribución Gaussiana para representar la creencia de estado es la mostrada en la ecuación 2.1:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right] \quad (2.1)$$

En nuestro caso, preferimos obtener una Gaussiana con la menor co-varianza posible, pues esto significaría que el estado que concuerda con la media μ tendrá mayor probabilidad de ser el estado correcto donde se localiza el robot.

Pasos en el algoritmo de Filtro de Kalman

Este algoritmo de localización tiene dos pasos claramente definidos:

- **Predicción:** Es el primer paso del algoritmo. En esta etapa el filtro se encarga de anticipar eventos, es decir, el filtro consigue conocer el estado x_t a través del x_{t-1} (ambos estados están representados con un vector de tamaño n, SIENDO N EL NUMERO DE ESTADOS POSIBLES) y con las entradas de control μ_t (representadas en un vector de tamaño m). Además tenemos que añadir un error ϵ_t a nuestro sistema para dar la indeterminación producida por la transición de estados (vector del mismo tamaño que el vector de estados, n, media nula y una matriz de co-varianza R_t). De esta forma la ecuación de transición de estados que representa $p(x_t|x_{t-1}, u_t)$ sería la mostrada en la ecuación 2.2:

$$x_t = A_t x_{t-1} + B_t \mu_t + \epsilon_t \quad (2.2)$$

Donde A_t es una matriz cuadrada con el mismo tamaño que el vector de estados, $n \times n$. Dicha matriz une el estado en el instante $t - 1$ con el instante t sin entradas de control.

Por otro lado, B_t es una matriz de tamaño $n \times m$, la cual une el estado con las entradas de control.

- **Actualización:** En este paso se corrigen los errores para obtener la mejor estimación de estado del sistema. Esta corrección se realiza contrastando la creencia que se obtiene sobre el estado en el anterior paso con la información que nos dan los sensores.

De esta manera, el filtro predice las medidas de los sensores que se obtendrían en la posición obtenida en el anterior paso z'_t (vector de tamaño k) a través del estado x_t que el filtro cree como correcto y después lo compara con el vector real de medidas z_t , consiguiendo una creencia del estado actual con alta fidelidad.

La función que representaría $p(z_t|x_t)$ es la ecuación 2.3:

$$z_t = C_t x_t + \delta_t \quad (2.3)$$

Donde C_t es la matriz con tamaño $k \times n$ donde los modelos de sensores son representados.

Además, se debe añadir un error δ_t para simular el ruido que pueden tener los sensores. Este ruido es un vector aleatorio gaussiano del mismo tamaño que el vector de medidas, k, media nula y matriz de co-varianza Q_t .

Algoritmo del Filtro de Kalman

En la tabla 2.3 se puede ver como se implementa el Filtro de Kalman:

- En las lineas 2 y 3 se implementa la fase de predicción que determina $\bar{bel}(x_t)$.
- De la linea 4 a la 6 se implementa la actualización de las medidas, permitiendo conocer z_t para determinar $bel(x_t)$.
- En la linea 4 la ganancia de Kalman, K_t , es calculada. K_t especifica el grado de verdad en z_t sobre $\bar{bel}(x_t)$.
- En la linea 5, la medida z_t es comparada con la predicción de la medida de los sensores μ_t .

Tabla 2.3: Algoritmo del Filtro de Kalman

Algoritmo del Filtro de Kalman ($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
Etapa de predicción
1.- Proyección del estado hacia delante
$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
2.- Proyección de la covarianza hacia delante
$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
Etapa de actualización
3.- Calculo de la ganancia de Kalman
$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
4.- Actualización del estado con las medidas
$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
5.- Actualización de la covarianza
$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
return μ_t, Σ_t

2.3.2. Filtro de Partículas

Introducción al Filtro de Partículas

El filtro de partículas es un método empleado para estimar el estado de un sistema que cambia a lo largo del tiempo.

El Filtro de Partículas se ha convertido en una técnica establecida para resolver problemas que tengan que ver con modelos de espacios de estados.

La técnica que se utiliza en el Filtro de Partículas, la técnica de Monte Carlo, lleva existiendo desde 1950, pero debido a la potencia computacional de aquella época esté método fue pasado por alto.

Posteriormente, en 1993, N. Gordon, D. Salmond y A. Smith desarrollaron el filtro bootstrap para implementar filtros bayesianos a través del método de Monte Carlo [?].

El Filtro de Partículas ha sido aplicado a una gran cantidad de campos, entre los que se destacan la economía, el proceso de señales, el tracking, la neurociencia, las redes, la bioquímica, etc.

El problema usando el Filtro de Kalman reside en que no proporciona una estimación razonable para problemas altamente no lineales y no gaussianos. Por ello, cuando uno se encuentra con un problema de ese tipo se usa el filtro de partículas, pues funciona en tiempo real y permite aproximar la distribución a medida que se van realizando medidas.

El Filtro de partículas es un filtro no paramétrico, estos son aquellos filtros que representan la solución con un número finito de valores, y en el caso de que pudiese haber infinitos valores adoptarían a la perfección la curva de probabilidad de la solución, permitiendo resolver tanto problemas lineales como no lineales. El filtro de partículas consiste en utilizar un conjunto de muestras (partículas) y valores (pesos) asociados a cada una de esas muestras.

Las partículas son posibles estados del sistema que pueden ser representados como puntos en el espacio de estados del sistema [?].

Características del Filtro de Partículas

El Filtro de Partículas es una aplicación del Filtro de Bayes para estimar una distribución de probabilidad que refleje la probabilidad de que el robot se encuentre localizado en un estado. Esta distribución es representada por un conjunto de ejemplos, y como se dijo anteriormente, si el conjunto de ejemplos fuese infinito obtendríamos una distribución de probabilidad que adoptaría perfectamente la curva de probabilidad de la solución.

El Filtro de Partículas tiene las siguientes características:

- El Filtro de Partículas funciona con sistemas multi-modales y es capaz de encargarse de cualquier tipo de modelo, independientemente de la linealidad y del

ruido debido a la gran cantidad de partículas usadas en el filtro.

- El Filtro de Partículas es costoso computacionalmente hablando, por lo que si queremos que un sistema sea representado exactamente necesitaremos un número infinito de partículas.
- Usar un número razonable de partículas permite encontrar una solución con un error mínimo.
- Se puede encontrar una relación entre el coste y la precisión del algoritmo.

Etapas del Filtro de Partículas

- **Inicialización:** El Filtro de Partículas elige aleatoriamente un conjunto de puntos los cuales serán los posibles estados del sistema. El conjunto de partículas puede ser creado aleatoriamente o se puede utilizar algún tipo de información a priori para crearlo (tamaño del objeto, posición aproximada).
- **Actualización:** El Filtro de Partículas asigna pesos a cada partícula en función de la similitud del estado de cada partícula con respecto a un estado de referencia.
- **Estimación:** El Filtro de Partículas crea un nuevo conjunto de partículas que se corresponden a la estimación a priori del estado en el siguiente instante de tiempo. Este conjunto de partículas es creado usando métodos de "re-sampling" probabilísticos, de forma que aquellas partículas que mejor se ajusten a las medidas darán lugar a nuevas partículas con mayor probabilidad. Como ejemplo de método de "re-sampling" encontramos "Resampling Wheel" [?] o "KLD-Sampling" (Usado en amcl) [?] que se encargan de conseguir el conjunto de partículas que es probablemente el más correcto.
- **Predicción:** El Filtro de Partículas realiza una leve modificación al estado de cada una de ellas introduciendo algún tipo de ruido aditivo que aporte variabilidad al sistema, con el fin de estimar el estado del objeto en el instante siguiente. Al terminar esta etapa de predicción se obtiene un nuevo conjunto de partículas al que se le vuelve a aplicar la etapa de actualización, repitiéndose este bucle hasta que termine la secuencia de datos, caso en el cual se volvería a la etapa de inicialización.

Algoritmo del Filtro de Partículas

Una vez definidas las etapas del algoritmo se pasará a explicar el algoritmo en su totalidad.

Tabla 2.4: Algoritmo de Filtro de Partículas

```

Algoritmo de Filtro de Partículas( $X_{t-1}, u_t, z_t$ ):
 $\overline{X}_t = X_t = \emptyset$ 
for m = 1 to M do:
    sample  $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$ 
     $w_t^{[m]} = p(z_t | x_t^{[m]})$ 
     $\overline{X}_t = \overline{X}_t + (x_t^{[m]}, w_t^{[m]})$ 
end for
for m=1 to M do:
    draw i with probability  $\alpha * w_t^{[i]}$ 
    add  $x_t^{[i]}$  to  $X_t$ 
end for
return  $X_t$ 

```

Para la comprensión del algoritmo se realizará un análisis de la implementación basándonos en la tabla 2.4:

- En la linea 2, Se inicializa el conjunto de partículas del siguiente instante y el conjunto de partículas predicho del siguiente instante al conjunto vacío.
- En la linea 4, las partículas evolucionan en una fase de predicción, es decir, para cada una de las partículas del conjunto de partículas del instante anterior X_{t-1} el filtro predice el estado de dicha partícula en el siguiente instante utilizando las entradas del sistema (aceleración,etc).
- En la linea 5 se asigna el peso a cada partícula a través de una función inversamente proporcional al error cometido entre las medidas realizadas por sensores y las medidas predichas. Además se debe tener en cuenta la normalización de los pesos para conseguir una distribución de probabilidad, para ello utilizaremos una constante de normalización a la cual llamaremos α .
- En la linea 6, el estado y el peso es añadido al conjunto de partículas predicho en el próximo instante.
- Surge un problema, después de un cierto número de pasos del algoritmo sólo un pequeño número de partículas tienen pesos no despreciables, para solucionar esto se debe entrar en la fase de re-sampling. Así que en la linea 9, para cada partícula que se necesita en el conjunto de partículas se requerirá una partícula, estas partículas son seleccionadas con un método de re-sampling, consiguiendo así más partículas cercanas a la partícula de mayor peso para obtener una solución

más precisa. Además, en la fase de predicción del algoritmo se añadirá ruido por lo que no tendremos las mismas partículas en nuestro conjunto.

En la figura 2.4 se puede ver el proceso de trabajo del algoritmo de filtro de partículas de una forma gráfica.

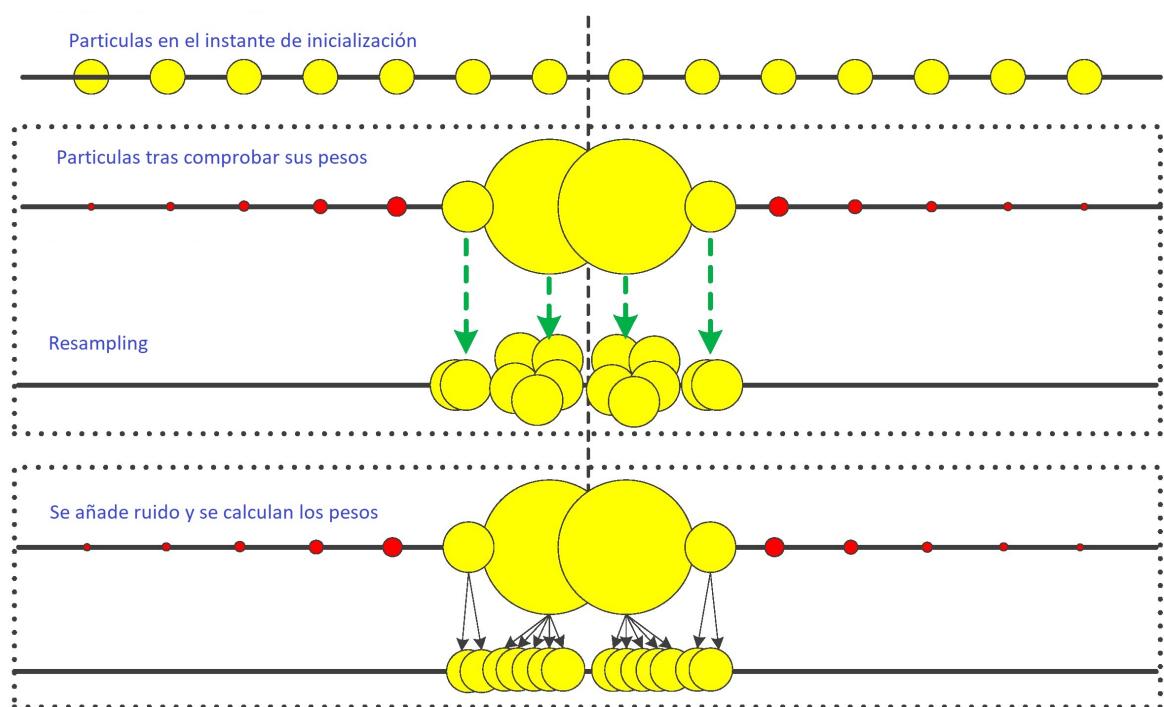


Figura 2.4: Proceso del algoritmo de filtro de partículas

Resampling Wheel

El método "Resampling Wheel" es el método de reemplazo más fácil de entender [?] REFERENCIA, pudiendo resumir su funcionamiento en la imagen 2.5 que se explicará posteriormente.

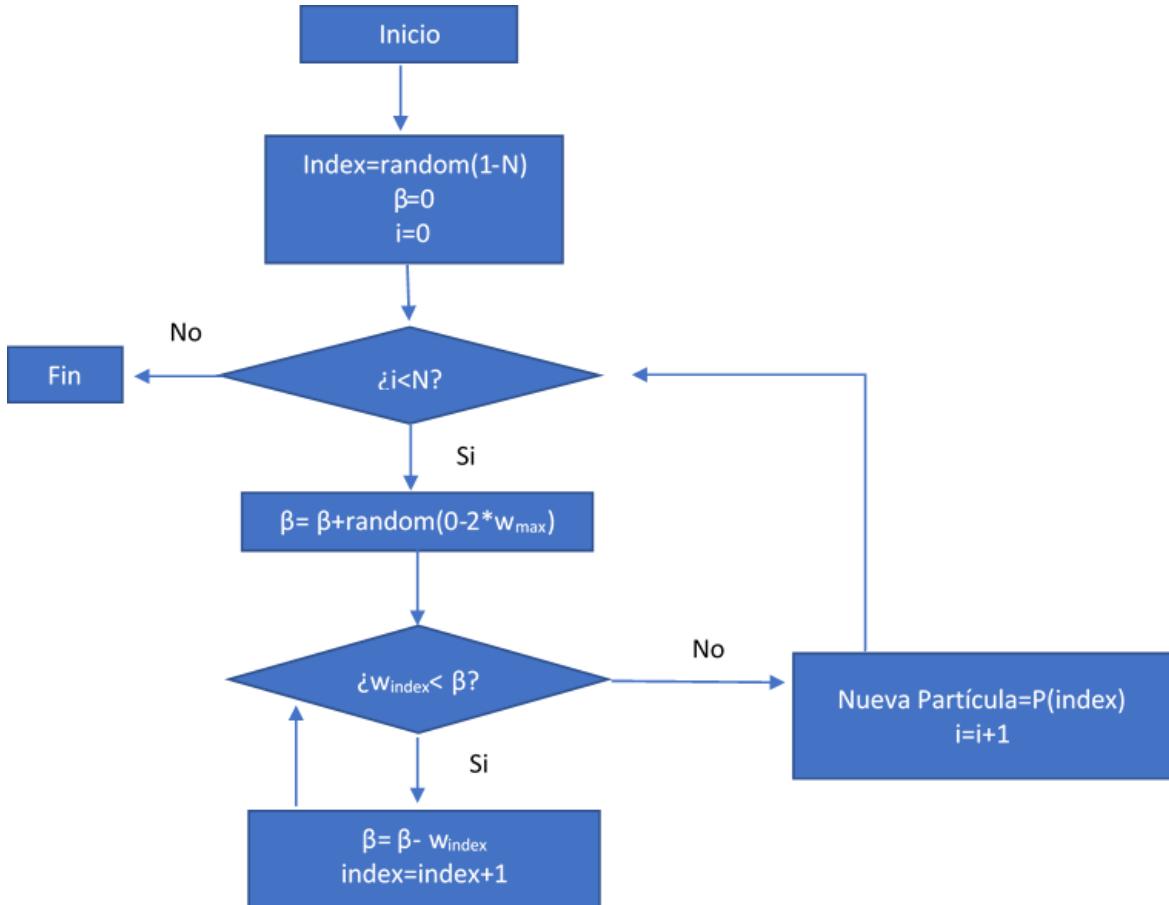


Figura 2.5: Diagrama Resampling Wheel

Este método coloca todas las partículas con sus respectivos pesos en una ruleta. Posteriormente se genera un número aleatorio para usarlo como índice en la ruleta (tabla 2.5).

Tabla 2.5: Selección de un Índice aleatorio

$$\text{Index}=\text{Random}[1,N];$$

Se coloca como inicio de la ruleta aquella partícula que tuvo el índice elegido aleatoriamente, posteriormente se debe girar la ruleta N veces, siendo N el número de partículas del conjunto de partículas, esto es debido a que debemos sustituir este conjunto por otro con el mismo número de partículas.

Este método de remplazo utiliza una función β que irá incrementando su valor con un número aleatorio entre el 0 y el doble del peso máximo de todas las partículas. Ese valor decidirá cual es la partícula que se mete en el nuevo conjunto teniendo en cuenta la partícula por la que se empieza en la ruleta y los pesos, de forma que si beta es menor o igual que el peso de la partícula por la que se empieza en la ruleta se elige esa

partícula, por el contrario, si la función beta es mayor que el peso de la partícula se tomara el peso de diferencia y se comparará con la siguiente partícula comprobándose de nuevo si el peso de diferencia es mayor que el peso de la partícula siguiente.

De esta manera la función β selecciona nuevas partículas para el nuevo conjunto de partículas en el instante de tiempo X_t tal y como podemos ver en la tabla 2.6.

Tabla 2.6: Algoritmo Resampling Wheel

```

 $\beta = 0;$ 
Index=Random[1,N];
//Para cada partícula
For (i=0;i<N;i++){
     $\beta = \beta + Random(0, 2 * w_{max});$ 
    While w[Index]<Beta{
         $\beta = \beta - w_t^{[Index]}$ 
        Index+=1;
    }
     $X_t^{[i]} = \bar{X}_t^{[Index]}$ 
}
Return  $X_t$ 
```

Para ver mejor el funcionamiento se especifica un ejemplo:

Suponiendo que se tienen 8 partículas en el sistema, el índice generado por el número aleatorio es 6. El peso máximo de todas las partículas es el peso de la partícula 5. Cuando se calcula la función β se obtiene que el nuevo índice es el 7, por lo que se debe añadir la partícula número 7 en el nuevo conjunto de partículas (figura 2.6).

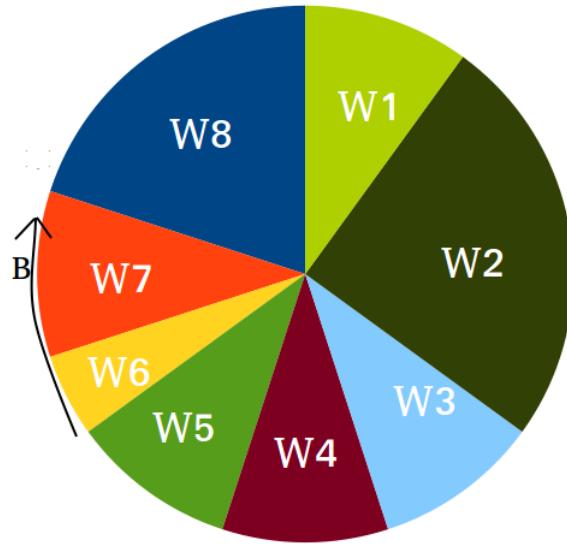


Figura 2.6: Obtención de nuevas partículas en Resampling Wheel

2.4. Escenario del problema

El problema que se ha tratado de resolver en este trabajo es el de localizar un robot, mediante una cámara en un entorno. Para ello se tendrá que obtener el mapa del entorno, elaborar un método para comparar imágenes y utilizar el método de filtro de partículas para localizar el robot.

2.5. Herramientas Software utilizadas

2.5.1. ROS

Introducción a ROS

Para la realización del trabajo se usa el sistema operativo ROS que permite y facilita el control de actuadores y sensores permitiendo un tratamiento de los datos obtenidos por los sensores y la elaboración de respuestas con los actuadores. De esta forma se pueden realizar tareas como la creación de mapas, comunicación entre los diferentes nodos del sistema, etc.

Arquitectura de ROS

ROS tiene una arquitectura específica que debemos entender para poder realizar el trabajo con este sistema operativo. En ROS encontramos diferentes elementos: REFERENCIA

- **ROS Master:** El ROS Master se encarga de proveer nombres y registrar servicios al resto de nodos del sistema de ROS, también se encarga de gestionar las publicaciones y suscripciones a los topics.
- **Nodos:** Los nodos son los archivos ejecutables que usa ROS para comunicarse con otros nodos mediante mensajes que son enviados por unos canales llamados tópicos, de esta forma, toda la computación se realiza en estos elementos mediante scripts que pueden ser creados en el lenguajes de programación Python y C++.
- **Tópicos:** Los tópicos son los canales por los que se publican mensajes, de forma que un nodo puede publicar en estos tópicos o suscribirse a ellos para recibir mensajes de otros nodos.
- **Mensajes:** Es la forma de comunicación entre los diferentes nodos. Se debe definir el tipo del mensaje necesariamente.
- **Servicios:** Es otro método de comunicación que tienen la característica de tener dos fases, la petición y la respuesta. Este método de comunicación no lo se utilizará.

2.5.2. Gazebo

Además de ROS se ha utilizado un software para la simulación del robot, su entorno y los diferentes sensores mediante Gazebo.

En Gazebo se ha modelado el robot a localizar (figura 2.7), añadiéndole una cámara para poder localizarlo mediante sensores visuales y permitiendo su control con el teclado para poder cambiar su posición en el entorno.



Figura 2.7

Además, se ha diseñado un escenario sencillo con carreteras y casas para poder simular un entorno donde trabajar con el problema de la localización (figura 2.8).



Figura 2.8: Sistema de navegación de ROS

2.5.3. Funcionamiento del sistema de navegación de ROS

Para la realización del trabajo es necesario conocer como funciona el sistema de navegación de ROS 2.9).

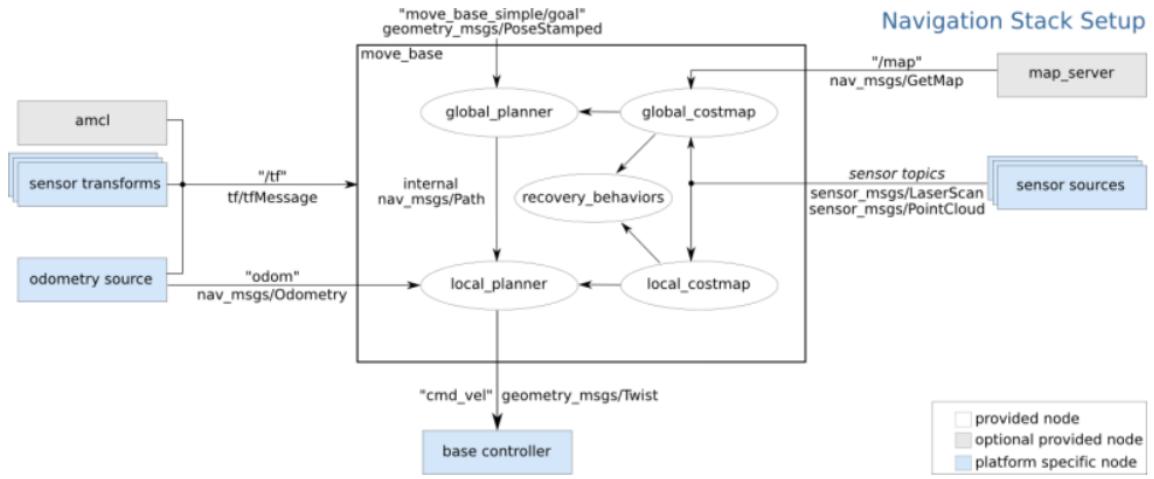


Figura 2.9: Sistema de navegación de ROS

Si se echa un vistazo a la figura 2.9, se puede ver como el sistema de navegación utiliza un conjunto de elementos que no se tendrá que modificar puesto que ROS los proporciona por defecto y no es necesario que se cambie nada para el trabajo de la localización. Este conjunto de elementos que proporciona ROS el cual no se debe modificar es "move_base" tiene que ver con los planificadores locales y globales.

Por otro lado se puede encontrar que para conocer el estado en el que se encuentra el robot se utiliza la odometría que será publicada en el topic "/odom", la cual consiste en la estimación de la posición del robot mediante la distancia recorrida por las ruedas, esta estimación tiene un problema relacionado con la suposición de que toda superficie es siempre perfecta, por lo que a la larga el error de la odometría irá creciendo y por ello debemos utilizar un algoritmo de localización.

Este algoritmo de localización está definido en la figura 2.9 como amcl, el cual es el método de Monte Carlo y utiliza la odometría y transformaciones para conocer la posición del robot la cual se usará para indicársela al planificador y poder navegar por el entorno.

Se puede observar también que el planificador necesita conocer su entorno, por lo que necesitamos indicarle el mapa del entorno mediante el topic "/map".

Además, se necesita indicarle al stack de navegación la información que obtenida mediante los sensores para que sea capaz de evitar obstáculos, esto lo realizaremos mediante un topic "/sensor_msgs/PointCloud" puesto que nuestro sensor es una cámara que tendrá una nube de puntos.

La salida del sistema de navegación será un "cmd_vel" que dirigirá nuestro robot al

punto que se le ordene mediante un planificador que irá evitando obstáculos.

En este trabajo modificaremos "amcl" cambiándolo por un algoritmo de Filtro de Partículas que maneje la información visual, además de cambiar también las fuentes de los sensores y el mapa del entorno.

2.5.4. Uniendo ROS y Gazebo

ROS y Gazebo pueden complementarse, por lo que se puede utilizar Gazebo como entorno de simulación, encargándose de la representación gráfica del entorno y de la representación física en un espacio. De esta manera, se pueden simular modelos con sensores que permitirían mandar datos a ROS como si del mundo real se tratase. De la misma manera ROS sería capaz de actuar sobre ese entorno simulado mediante los actuadores que le proporciona Gazebo.

Capítulo 3

Desarrollo del trabajo y experimentos

3.1. Desarrollo del primer experimento

3.1.1. Introducción

Para una mejor comprensión de los métodos de localización se ha realizado un experimento previo sobre el estudio de los diferentes métodos de localización mediante un sensor de distancia, un modelo de nuestro robot Pioneer-3AT y una habitación cerrada, concretamente el laboratorio software 1. Además se ha acotado el experimento eliminando una de las dimensiones, la altura, pues esta no resultaba de utilidad y facilitaba el estudio.

En el experimento se han utilizado tres diferentes métodos de localización para su posterior comparación, permitiendo así un acercamiento a los diferentes métodos y sus ventajas y desventajas.

Para la comparación de los diferentes métodos de localización se ha realizado un circuito del robot en la habitación (figura 3.1), el cual consistía en realizar mover el robot formando un cuadrado, esto se ha repetido 5 veces y se ha escrito en un csv las posiciones obtenidas cada vez que el robot llegaba a un vértice del cuadrado.

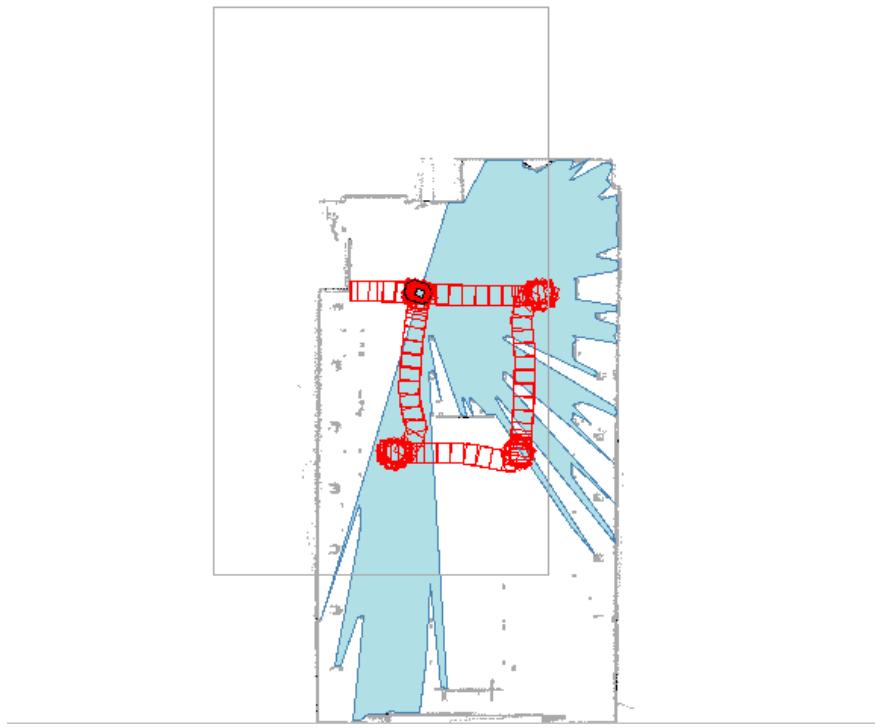


Figura 3.1: Circuito de pruebas

Tal y como vemos en la imagen (figura 3.1) el circuito realizado por el robot no es un cuadrado, esto es debido a que el planificador detecta los obstáculos, evitándolos y permitiendo que el robot llegue al destino correspondiente.

3.1.2. Preparación del primer experimento

Los objetivos de este primer experimento han sido:

- Configurar Gmapping[REFERENCIA] para realizar un SLAM[REFERENCIA].
- Crear y configurar el "stack"de navegación[REFERENCIA].
- Usar los métodos de localización y navegación.
- Evitar obstáculos en la navegación del robot mediante un planificador.

Configurando Gmapping

En este proyecto necesitamos configurar Gmmapping para generar el mapa que vamos a utilizar después para la navegación y localización del robot.

Tal y como dijimos anteriormente vamos a utilizar el laboratorio Software 1 y mediante el sensor láser (Lidar) del Pioneer 3-AT y la odometría generaremos el mapa del

entorno.

Para ello, se utilizará el comando que permite mandar mensajes de tipo geometry_msgs/Twist[REFERENCIA] , permitiendo controlar el robot aumentando la velocidad lineal y angular y publicándola en el topic /cmd_vel simplemente mediante el teclado:

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Además, para ver por como va la generación del mapa utilizaremos la herramienta rviz[REFERENCIA] , que nos permite ver una gran cantidad de elementos de forma gráfica como topics, el árbol de transformaciones, etc. Podemos ver como se va generando el mapa en la figura 3.2.

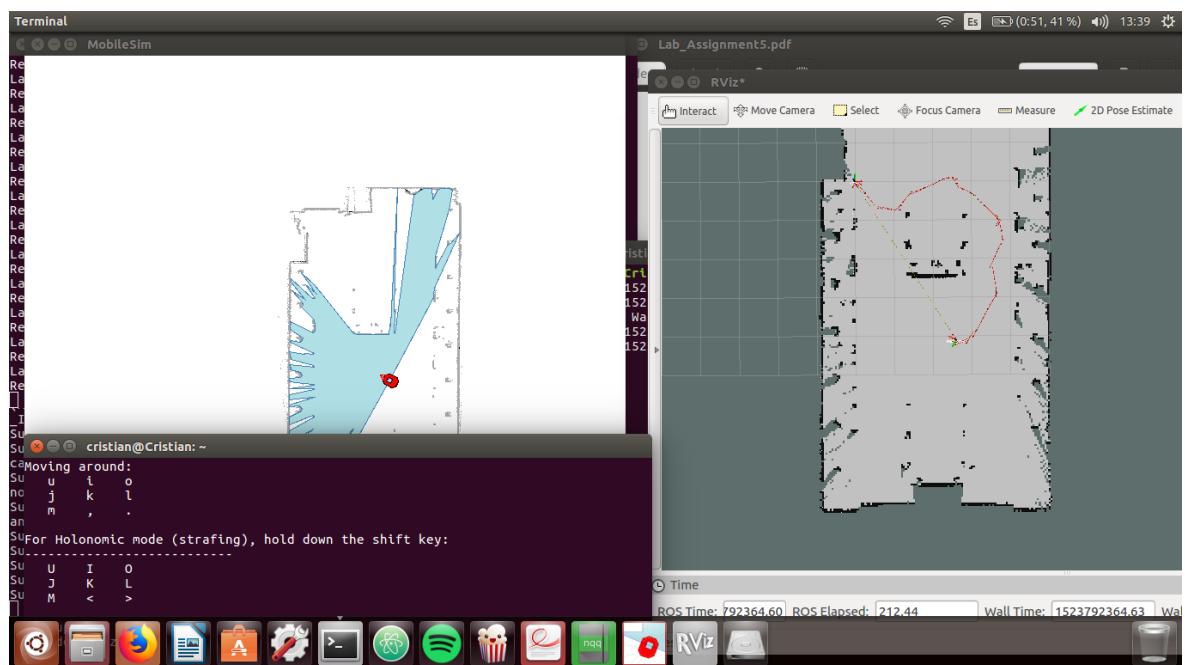


Figura 3.2: Visualización en rviz de la generación del mapa mediante SLAM

Cuando hayamos completado el mapa que estamos visualizando el rviz, podremos almacenarlo en el ordenador generandone dos archivos uno de formato .pgm (contiene la información sobre el mapa generado, creando una imagen con los espacios libres de color blanco y los espacios ocupados de color negro) y otro .yaml(representa la información del map) mediante el comando:

```
$ rosrun map_server map_saver [-f mapname]
```

Una vez generado el mapa mediante la técnica de SLAM obtenemos el mapa tal y

como vemos en la figura 3.3.

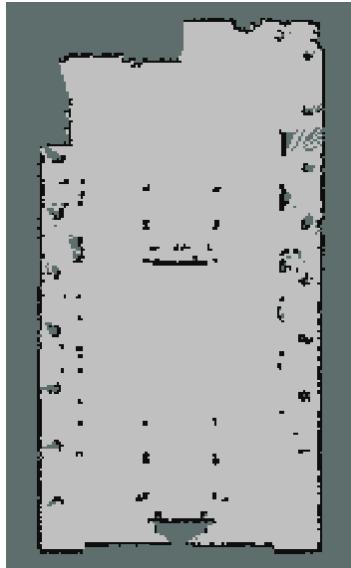


Figura 3.3: Mapa generado con técnica de SLAM

Configurando el stack de navegación

Para la configuración del stack de navegación 2.9, debemos configurar adecuadamente los siguientes elementos del stack de navegación:

- **El árbol de transformaciones:** Se debe establecer un frame para el mapa, uno para la odometría del robot y otro para los sensores, de esta manera permite tener al sistema relaciones entre las coordenadas de los diferentes frames, como las traslaciones y rotaciones.
- **La información que utilizará para evitar obstáculos:** Se debe establecer que tipo de sensor se utilizará para evitar obstáculos, en nuestro caso el sensor_msgs/LaserScan[REFERENCIA].
- **EL topic que nos muestra la posición del robot:** Se debe tener un topic de tipo nav_msgs/Odometry.msg [REFERENCIA] que muestre la localización del robot, esta puede calcularse mediante la odometría, u otros filtros bayesianos que permitan la localización.
- **Base Controller:** Este elemento no debemos modificarlo ya que ROS nos proporciona un planificador por defecto.
- **El mapa del entorno:** Este mapa lo debemos establecer mediante Map Server [REFERENCIA], que carga un mapa de un archivo y lo publica en diferentes topics y servicios.

Usando los diferentes métodos de localización

Para la utilización de los diferentes métodos de localización se han utilizado nodos que ya proporciona ROS como amcl que es un filtro de partículas, robot_pose_ekf que es un Filtro de Kalman. Estos nodos proporcionan como salida un topic del tipo nav_msgs/Odometry.msg de forma que podemos utilizar este topic como entrada del move_base mediante archivos launch que permiten modificar los topics a los que se suscribe un nodo mediante remap como el que podemos ver en el código 3.1.

```

0 <node pkg="robot_pose_ekf" type="robot_pose_ekf" name="robot_pose_ekf">
1   <remap from="/odom" to="/RosAria/pose" />
2   <param name="output_frame" value="odom"/>
3   <param name="base_footprint_frame" value="base_link"/>
4   <param name="freq" value="30.0"/>
5   <param name="sensor_timeout" value="1.0"/>
6   <param name="odom_used" value="true"/>
7   <param name="imu_used" value="false"/>
8   <param name="vo_used" value="false"/>
9
10 </node>
11 <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
12   <param name="controller_frequency" value="5.0" />
13   <rosparam file="$( find my_robot_name_2dnav )/costmap_common_params.yaml" command="load" ns="global_costmap" />
14   <rosparam file="$( find my_robot_name_2dnav )/costmap_common_params.yaml" command="load" ns="local_costmap" />
15   <rosparam file="$( find my_robot_name_2dnav )/local_costmap_params.yaml" command="load" />
16   <rosparam file="$( find my_robot_name_2dnav )/global_costmap_params.yaml" command="load" />
17   <rosparam file="$( find my_robot_name_2dnav )/base_local_planner_params.yaml" command="load" />
18 </node>
```

Código 3.1: move_base.launch file

Tras ejecutar el launch podemos utilizar rviz para ver el mapa con los obstáculos,

y con la distribución de probabilidad de la localización del robot añadiendo los topics correspondientes tal y como se puede ver en la figura 3.4.

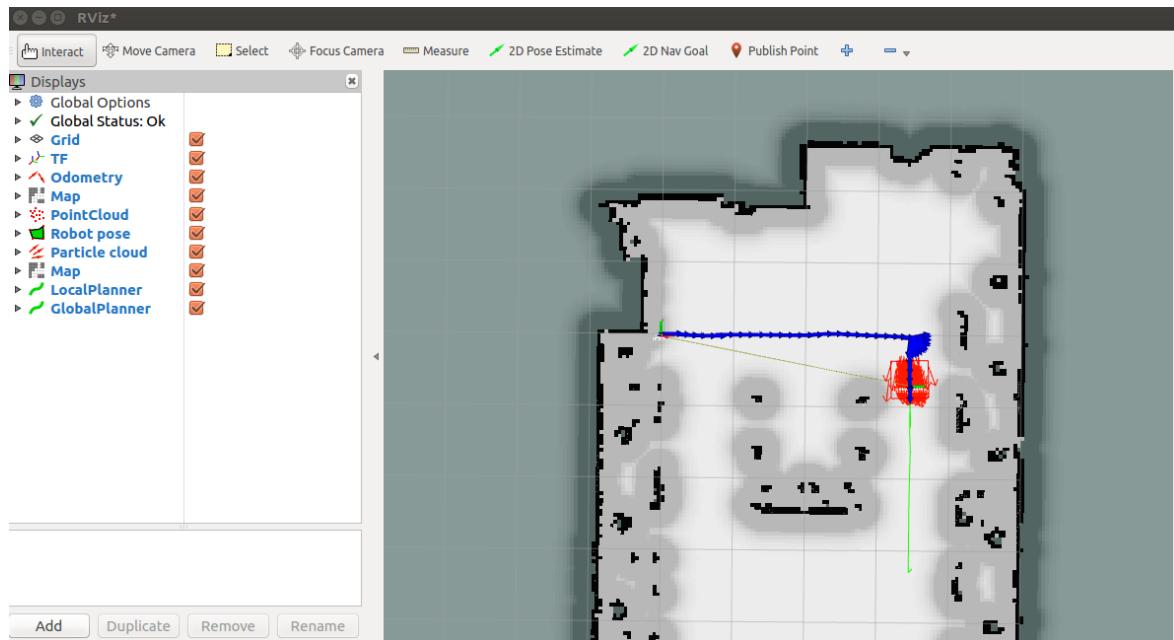


Figura 3.4: Configuración de rviz para mostrar el mapa con el detector de obstáculos

Una vez configurado todo el stack de navegación, se realizará un análisis del error de la posición de los distintos métodos de localización.

3.1.3. Resultados usando la odometría como método de localización

Este método se basa en la estimación de la posición de vehículos con ruedas durante la navegación. Para realizar esta estimación se usa información sobre la rotación de las ruedas para estimar cambios en la posición a lo largo del tiempo.

Para la comparación de gráficas se ha utilizado un gráfico de barras donde el eje X es el número de vueltas que el robot ha dado y el eje Y el error cometido en cada gráfico

Además, ya que esta vez hemos utilizado un mapa basado en un plano cenital [REFERENCIA] y teniendo dos dimensiones posibles de movimiento se ha analizado el error obtenido en cada uno de los posibles movimientos, el movimiento en el eje x y el movimiento en el eje y.

En el gráfico obtenido mediante el csv obtenido por método de localización basado en la odometría encontramos que los mayores errores han sido **0.052 en el eje X**

(movimiento vertical) tal y como podemos ver en la figura 3.5 y **0.058 en el eje Y** (movimiento horizontal) tal y como podemos ver en la figura 3.6 con respecto a los objetivos a los que tenía que llegar el robot.

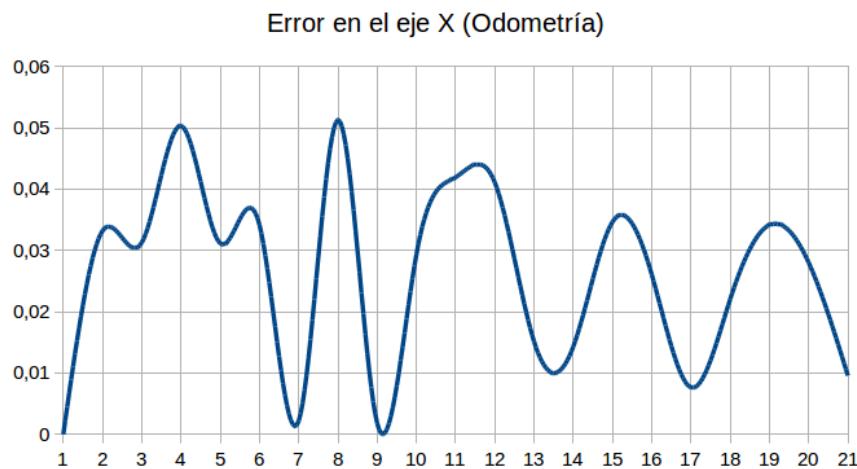


Figura 3.5: Error en el método de odometría, eje X

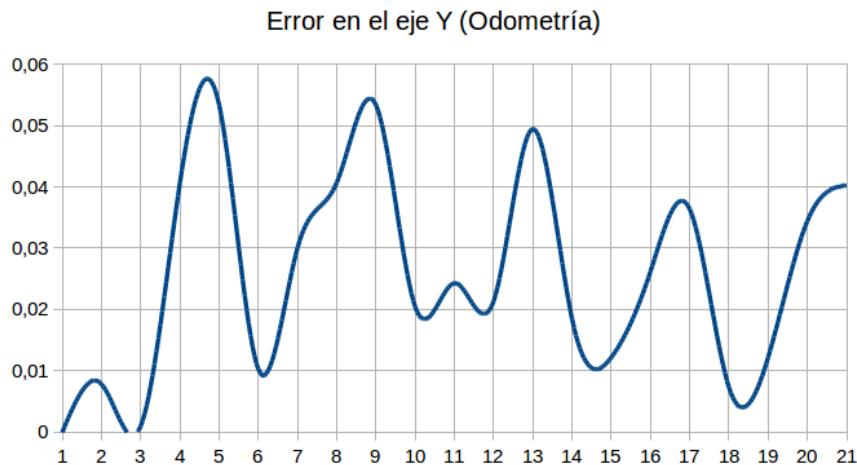


Figura 3.6: Error en el método de odometría, eje Y

De esta forma obtenemos un error medio en el eje X de **0.0257** con varianza **0,000237** y un error medio en el eje Y de **0.0258** con varianza **0,000281**.

3.1.4. Resultados usando el Filtro de Partículas como método de localización

Estudiando el caso del Filtro de Partículas como método de localización encontramos que los mayores errores han sido **0.065 en el eje X**, tal y como podemos ver en

la figura 3.7 y **0.07** en el eje **Y**, tal y como podemos ver en la figura 3.8, con respecto a los objetivos a los que tenía que llegar el robot.

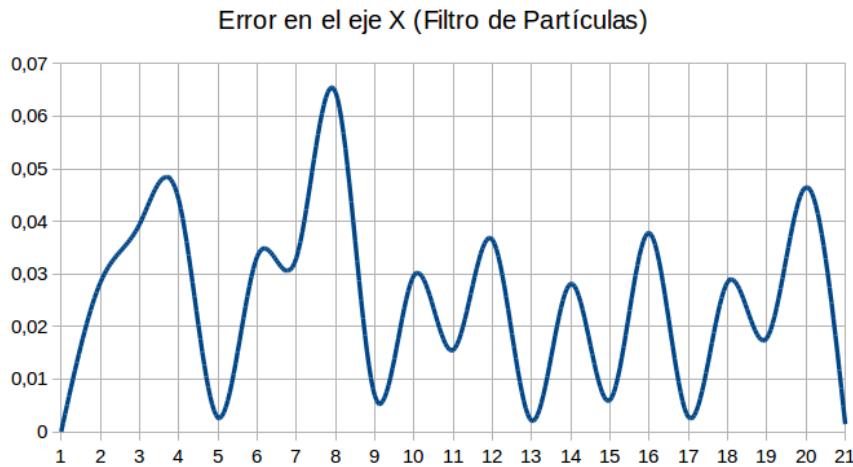


Figura 3.7: Error en el método de Filtro de Partículas, eje Y

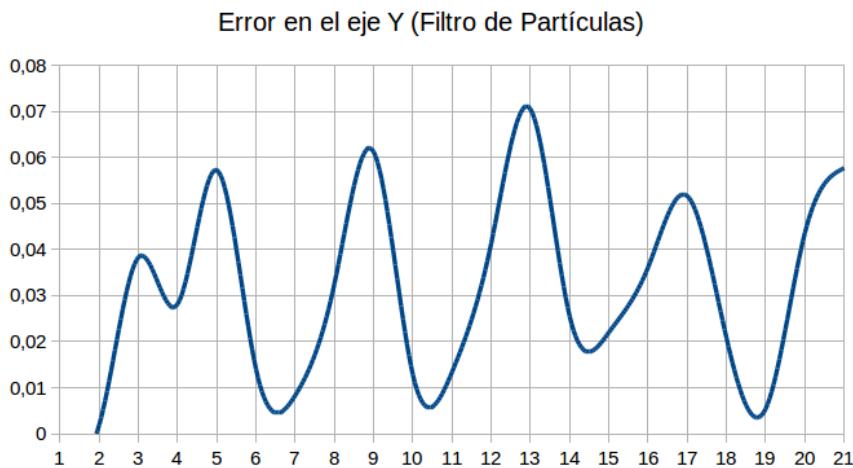


Figura 3.8: Error en el método de Filtro de Partículas, eje Y

Obteniendo así, un error medio en el eje X de **0,0240** con varianza **0,00033** y un error medio en el eje Y de **0,0306** con varianza **0,000439**.

3.1.5. Resultados usando el Filtro de Kalman como método de localización

Estudiando el caso del Filtro de Partículas como método de localización encontramos que los mayores errores han sido **0.061** en el eje **X**, tal y como podemos ver en la

figura 3.9 y **0.073** en el eje Y, tal y como podemos ver en la figura ??, con respecto a los objetivos a los que tenía que llegar el robot.

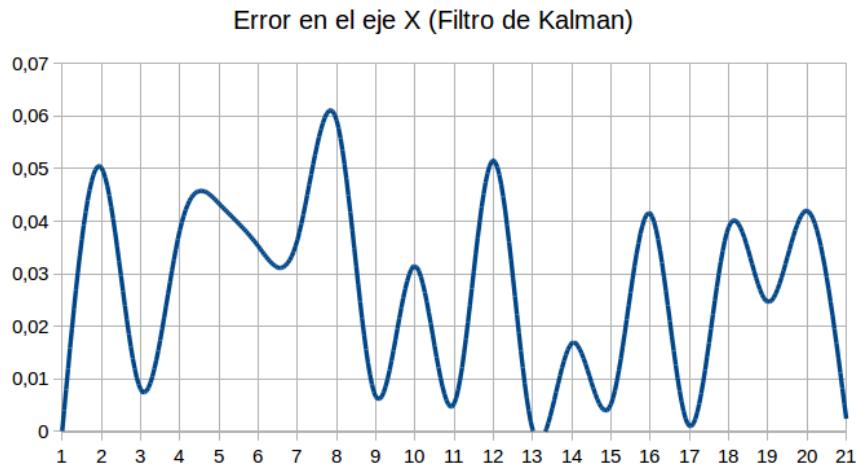


Figura 3.9: Error en el método de Filtro de Kalman, eje X

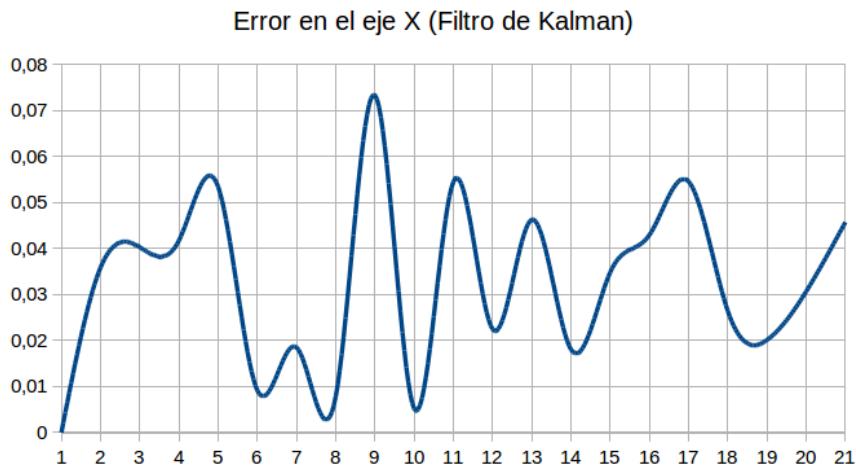


Figura 3.10: Error en el método de Filtro de Kalman, eje Y

Obteniendo así, un error medio en el eje X de **0.0256** con varianza **0,00039** y un error medio en el eje Y de **0.0325** con varianza **0,000368**.

3.1.6. Conclusiones del primer experimento

Tras analizar los errores cometidos en el eje X por los diferentes métodos de localización según las gráficas mostradas en la figura 3.11 y los errores cometidos en el eje Y por los diferentes métodos de localización según las gráficas mostradas en la figura 3.12, se ha obtenido que el método de localización basado en la odometría es el peor de

todos, pues es el que mayor error tiene, y además, en este caso el error es muy pequeño debido a que se ha realizado una simulación, por lo que el movimiento de las ruedas no tienen error, en el caso de que se probase en un entorno real los errores en el método de la odometría serían mucho mayores que los otros algoritmos de localización, por ello es muy importante aplicar otros tipos de localización si necesitamos que nuestro robot sepa con precisión su localización.

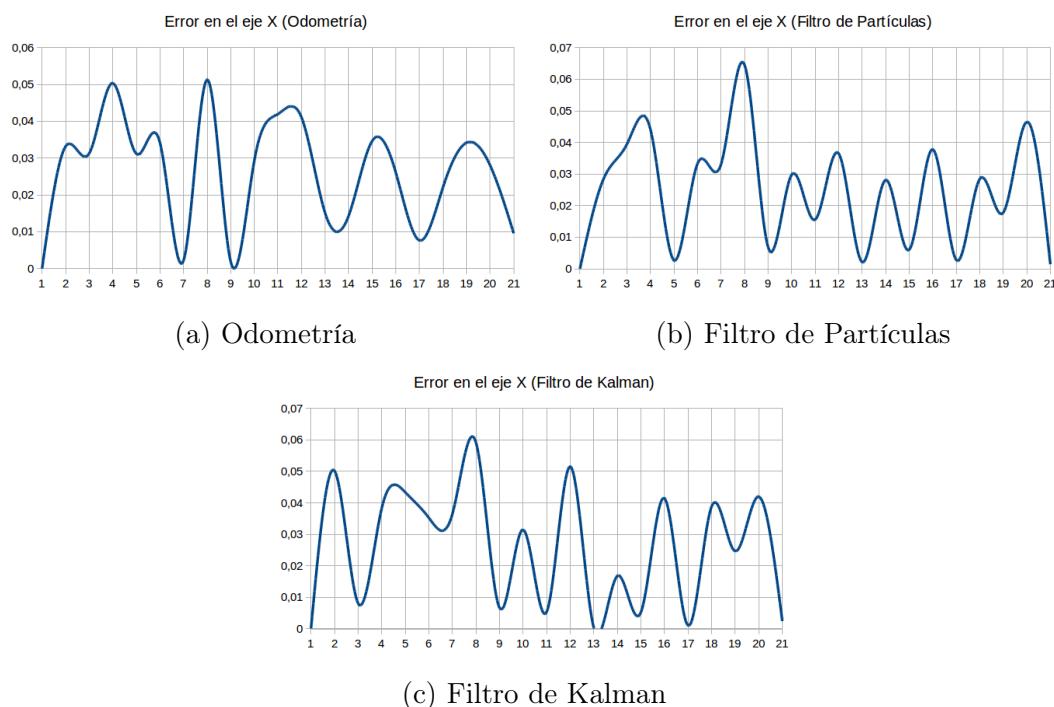


Figura 3.11: Comparación de errores en el eje X entre los diferentes métodos de localización

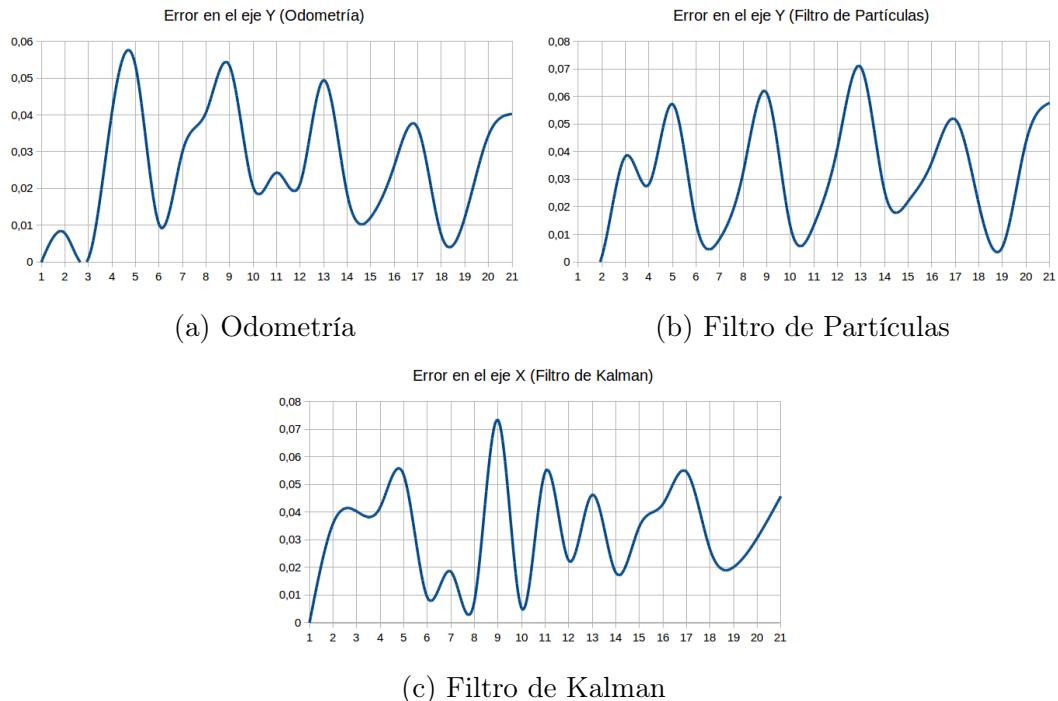


Figura 3.12: Comparación de errores en el eje Y entre los diferentes métodos de localización

3.2. Desarrollo del segundo experimento

3.2.1. Introducción

3.2.2. Preparación del entorno

Para la realización del experimento se ha utilizado la herramienta gazebo, donde se ha cargado un modelo de una carretera, un modelo del Pioneer 3-AT al cual se le ha añadido una cámara de tipo profundidad (Kinect) y un controlador para poder mover el robot a través del entorno. Para la utilización de estos modelos se ha creado un modelo del mundo con formato sdf que abriremos en gazebo mediante el comando:

```
$ roslaunch mybot_gazebo mybot_world.launch
```

Una vez ejecutado el comando podemos ver el entorno del mundo en la imagen 3.13.

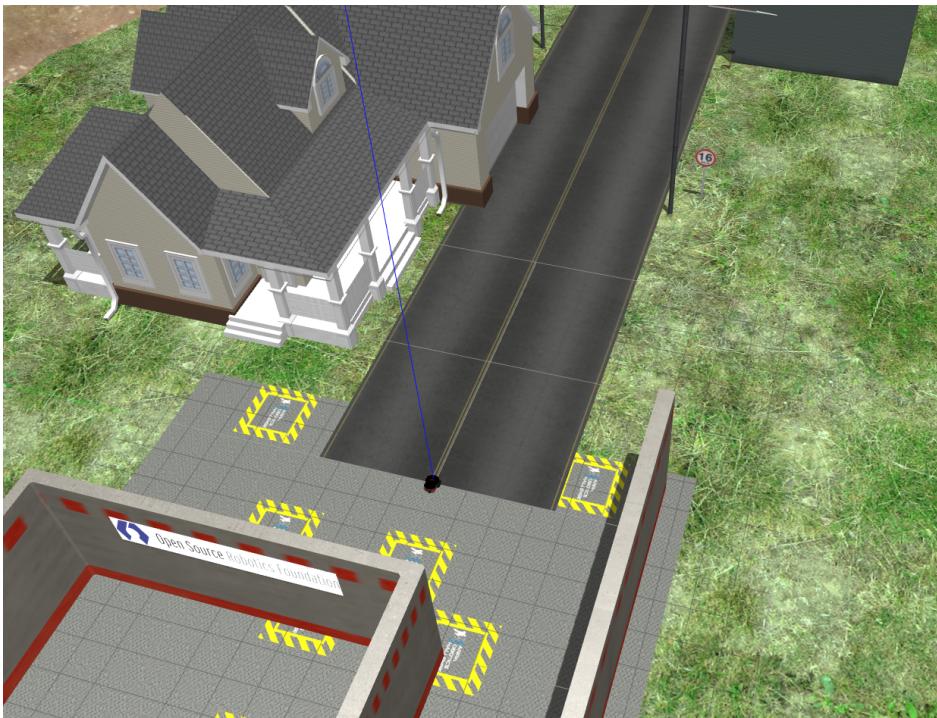


Figura 3.13: Entorno simulado en Gazebo

3.2.3. Mapeo del entorno

Para la realización de este experimento es necesario realizar un mapeo 3d del entorno, para ello se ha utilizado el nodo rtabmap_ros [REFERENCIA] que permite realizar esta acción suscribiéndose a la nube de puntos que obtiene la información del entorno de forma visual mediante nuestra cámara de profundidad (Kinect).

Posteriormente Rtabmap utilizará un algoritmo basado en SURF[REFERENCIA] para obtener los puntos de interés de la nube de puntos y registrar las nubes de puntos obtenidas en cada instante de tiempo en una sola nube de puntos, es decir, agrupando las nubes de puntos para formar una única nube de puntos con toda la información del entorno. Para la configuración de Rtabmap se ha creado un archivo rtabmap.launch. donde se establecen los parámetros como la asignación de los topics donde el nodo se debe suscribir, el algoritmo para obtener los puntos de interés, la memoria utilizada, la profundidad máxima y mínima, etc.

Una vez configurado Rtabmap se ha movido el robot mediante el nodo teleop_twist_keyboard que publica la velocidad que se desee obtener en el topic /cmd_vel. Además, para que el robot se suscriba a este topic y añada la velocidad correspondiente se ha añadido en el archivo del mundo, concretamente en la parte del modelo del robot el plugin "differential_drive_controller"[REFERENCIA].

El proceso de mapear el entorno se puede visualizar mediante la herramienta Rviz

que permite suscribirse a topics y visualizarlos. Rtabmap publicará la nube de puntos del entorno en el topic `/rtabmap/cloud_ground` por lo que con Rviz se puede llevar un seguimiento tal y como vemos en la imagen [3.14](#).

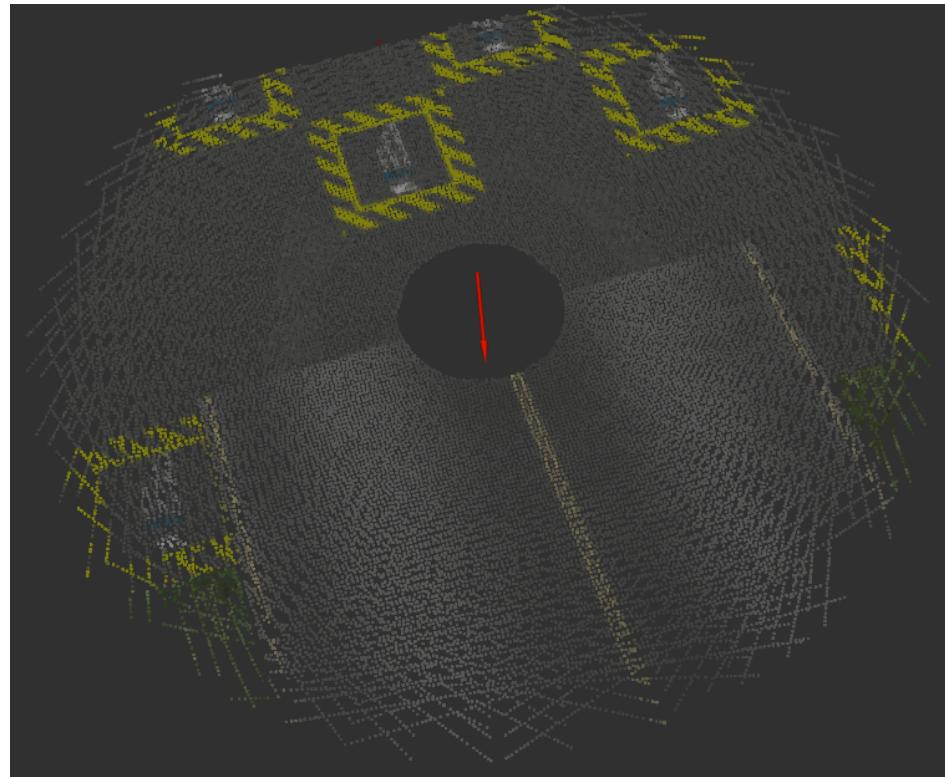


Figura 3.14: Nube de puntos obtenida al realizar un mapeo con rtabmap

3.2.4. Obtención de la nube de puntos de las partículas

Una vez que ya hemos mapeado todo el entorno se necesita conocer la nube de puntos que se obtendría suponiendo que el robot esta en la posición (x,y) con orientación theta. Para ello, se debe recortar la nube de puntos obtenida por el mapeo, trasladando la nube de puntos para que su origen tenga las coordenadas (x,y) , posteriormente rotando theta y volviendo a trasladar para dejar en el centro de coordenadas el punto inferior izquierdo. Posteriormente se recortarían las dimensiones correspondientes a la nube de puntos del sensor, en nuestro caso, nuestro sensor obtiene una profundidad de 5 (el primer metro no aprovechable debido a la altura de la cámara), y una anchura de 6.

El resultado obtenido al recortar la nube de puntos del mapeo sería la mostrada en la figura [3.15](#).

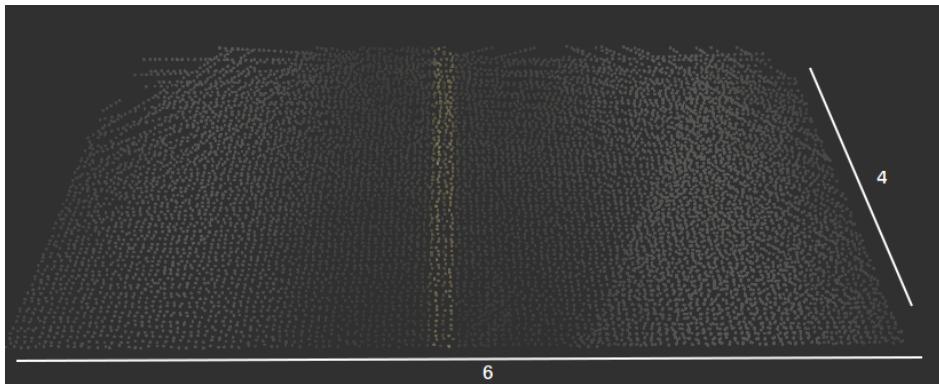


Figura 3.15: Nube de puntos obtenida recortando la nube de puntos obtenida mediante mapeo.

Para reducir la complejidad computacional se ha reducido la nube de puntos antes de comenzar a realizar las diferentes transformaciones, esta reducción se ha realizado mediante un recorte de 5 metros alrededor de la imagen puesto que al no tener en cuenta el ángulo del robot la profundidad de la cámara podría estar en cualquier dirección tal y como vemos en la figura 3.16.

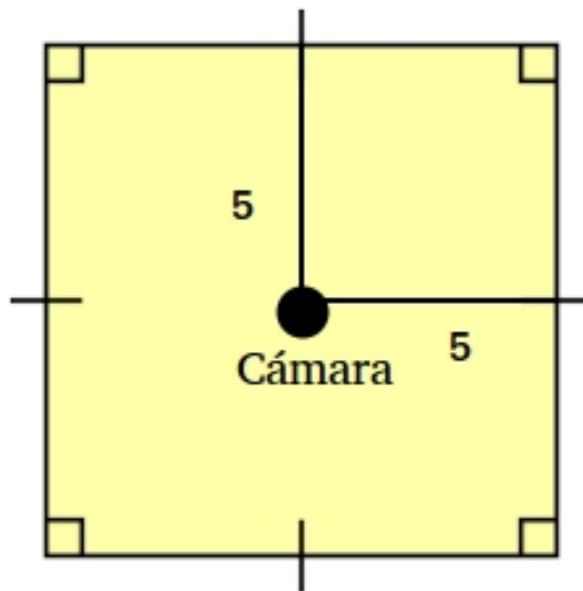


Figura 3.16: Área de 5 metros alrededor de la cámara

Tras realizar este cambio, las transformaciones no serán aplicadas a los puntos que no son potencialmente importantes para visualizar la imagen del sensor.

Además, para reducir la complejidad computacional también se ha realizado conjuntamente y simultáneamente la traslación de la nube de puntos para dejar el robot en el centro de coordenadas(M1), la rotación de la nube de puntos para dejar el robot con rotación 0° (M2) y la traslación para dejar el punto inferior izquierda de la nube de puntos que queremos en el centro de coordenadas (M3) mediante la multiplicación

de matrices, obteniendo una matriz que realiza las anteriores operaciones en conjunto y simultáneamente.

$$M1 = \begin{bmatrix} 1 & 0 & 0 & PosX \\ 0 & 1 & 0 & PosY \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M2 = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M3 = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M3 * M2 * M1 = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & \cos(\alpha) * PosX - \sin(\alpha) * PosY - 1 \\ \sin(\alpha) & \cos(\alpha) & 0 & \cos(\alpha) * PosY + \sin(\alpha) * PosX - 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Posteriormente se aplica esa matriz a todos los puntos de la nube de puntos recorrida anteriormente.

$$PuntoNuevo =$$

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & \cos(\alpha) * PosX - \sin(\alpha) * PosY - 1 \\ \sin(\alpha) & \cos(\alpha) & 0 & \cos(\alpha) * PosY + \sin(\alpha) * PosX - 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} PAnteriorX \\ PAnteriorY \\ PAnteriorZ \\ 1 \end{bmatrix}$$

3.2.5. Obtención de las imágenes vistas desde arriba

Para este experimento es necesario obtener las imágenes que se obtendrían por el sensor y por las diferentes partículas, para ello se ha utilizado la nube de puntos del sensor/partícula (Figura 3.17) y se ha trasladado para que el punto más cercano y más a la izquierda sea el origen de coordenadas y poder así proyectar la nube de puntos sobre el plano z, obteniendo así una imagen en 2d tal y como podemos ver en la figura 3.18.



Figura 3.17: Nube de puntos obtenida por el sensor

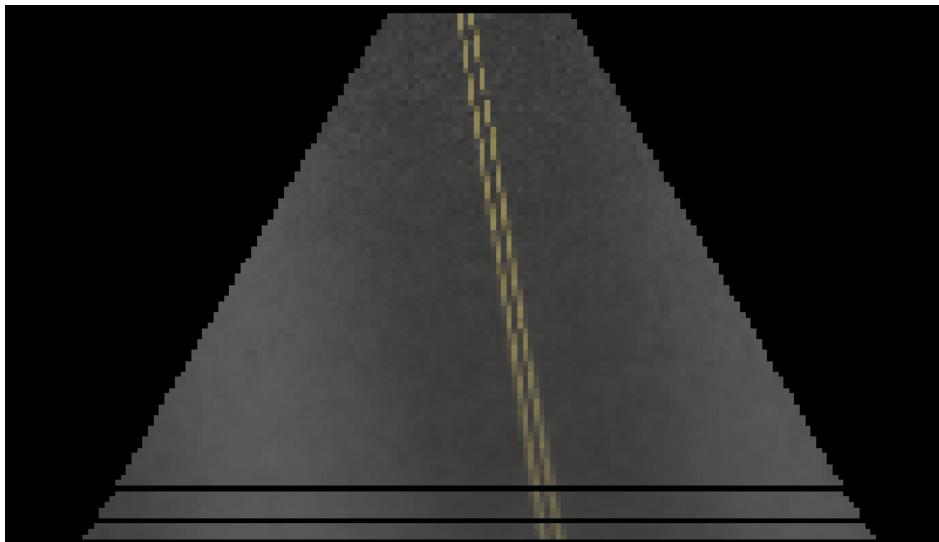


Figura 3.18: Imagen obtenida desde arriba proyectando la nube de puntos del sensor sobre el plano z

3.2.6. Detección de líneas

En este experimento también se ha tratado de detectar las líneas para facilitar los cálculos, para ello se han utilizado las imágenes en escala de gris obtenidas mediante la proyección de las nubes de puntos a una imagen 2d y se ha creado un nuevo nodo que permite detectar las líneas de la carretera, eliminando lo demás de su entorno,

para ello se ha pasado a blanco y negro la imagen y posteriormente se ha realizado el Canny Edge Detection[REFERENCIA], que permite encontrar bordes en la imagen. A continuación se han excluido los elementos de la imagen que no nos resultan útiles, para ello se elige una zona de interés y se elimina todo elemento que no esté dentro de esta zona [3.19](#).

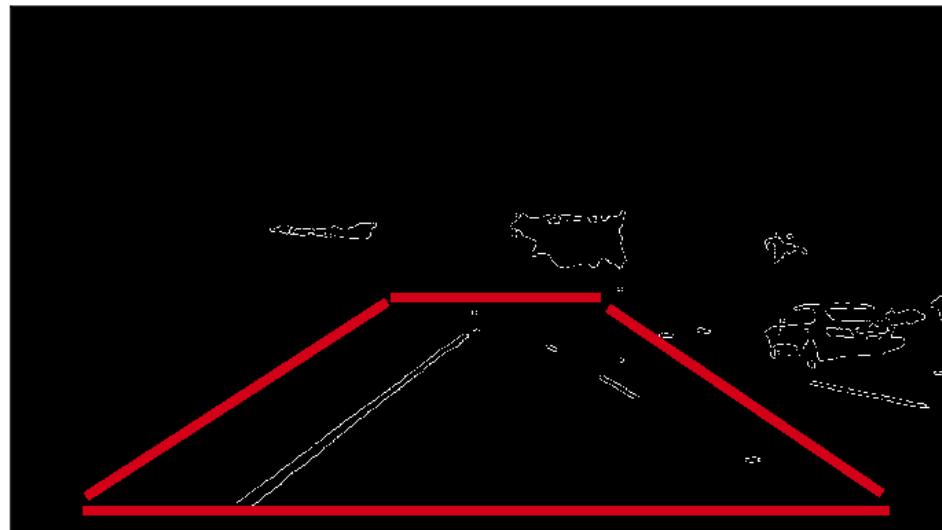


Figura 3.19: Zona de interés seleccionada de la imagen

Una vez encontrados los bordes de la imagen, se ha cogido la imagen inicial en blanco y negro y se han seleccionado los elementos que tienen bordes, permitiendo obtener la linea de la carretera en su totalidad, no solo los bordes.

De esta forma ya tendríamos las lineas detectadas para basarnos en la visualización a la hora de localizar el robot en el entorno pero debido a las pruebas que se han realizado posteriormente se ha decidido no utilizar las lineas detectadas para la localización y utilizar la imagen integra.

3.2.7. Calculo de pesos de las partículas

Para la realización del filtro de partículas se necesita una forma de comparar las medidas obtenidas con las esperadas, para ello se ha obtenido una imagen de la nube de puntos obtenida por la cámara y se comparará con aquella imagen obtenida sobre la nube de puntos esperada.

Calculo de pesos mediante comparación por histogramas

La comparación se ha decidido hacer de forma rápida, por lo que se ha utilizado un método basado en los histogramas de las imágenes, de forma que se divide la imagen que obtendría la particula en pequeñas sub-imágenes, concretamente 4 sub-imágenes.

Para cada una de las sub-imágenes se calcularán los valores medios de los canales RGB y posteriormente se compararán con los valores medios de las subimágenes obtenidas de la imagen del sensor, permitiendo así la capacidad de establecer un porcentaje de coincidencia entre las imágenes mediante una función de coincidencia [3.2.7](#).

$$\text{Indice de coincidencia} =$$

$$\sum_{i=0}^{\text{Sub-imagenes}} \sum_{j=0}^{\text{ComponentesRGB}} |\text{ValorImagenSensor}_{ij} - \text{ValorImagenParticula}_{ij}| \quad (3.1)$$

Una vez conocido el índice de coincidencia se pasará a dar el peso a las partículas en base a este índice. Para calcular el peso se ha decidido buscar una función exponencial y decreciente, buscando una función parecida a la de la figura [3.20](#).

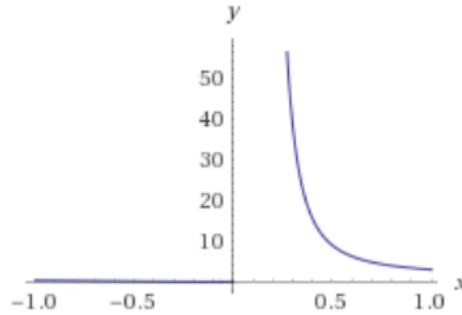


Figura 3.20: Función exponencial decreciente, $3^{1/x}$

Para obtener una función parecida a la figura [3.20](#) se ha analizado el resultado de los índices de coincidencia. Estos índices de coincidencia tienen un rango de 40 a 90 por lo que se ha buscado una función que utilice como entrada ese índice de coincidencia y se obtenga una salida que esté entre 0.4 y 1 aproximadamente, para pasársela como entrada a la función exponencial decreciente, obteniendo la función [3.2](#).

$$f(\text{IndiceCoincidencia}) = \text{IndiceCoincidencia}/100 \quad (3.2)$$

Combinando estas dos funciones, obtendríamos la función que devolvería el peso de la partícula [3.3](#), que tiene como entrada el índice de coincidencia, base un factor mayor que 0 según la pendiente que se quiera en la función. Su representación visual se vería en la imagen [3.21](#)

$$f(\text{IndiceCoincidencia}) = \text{factor}^{1/(\text{IndiceCoincidencia}/100)} \quad (3.3)$$

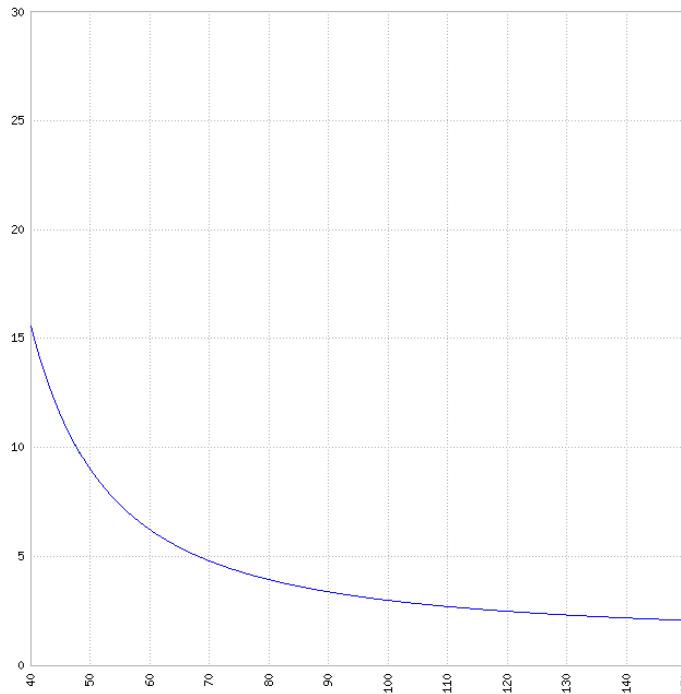


Figura 3.21: Función exponencial decreciente, $f(x) = 3^{1/(x/100)}$

3.2.8. Desarrollo del filtro de partículas

Una vez establecido el entorno, la detección de líneas y el análisis de las imágenes con un método para calcular los pesos de las partículas, se explicará el desarrollo del filtro de partículas mediante una iteración.

Suponemos que en el primer instante el robot se sitúa en la posición (0,0,0) con una rotación de 0° y se aplica una velocidad angular y lineal de 0 radianes/s y 0 m/s.

El filtro de partículas inicializa el conjunto de partículas aleatoriamente o puede comenzar iniciando el conjunto de partículas mediante una posición aproximada, la cual se puede obtener de un sensor GPS que da la posición del robot con un ruido gaussiano de media la posición del robot y varianza 3, 3, 0.25 para cada una de las variables x,y,theta. De esta forma se simulará el error de hasta 9 m en el eje horizontal, vertical y 42° en el rotación del robot tal y como podemos ver en la figura 3.22.

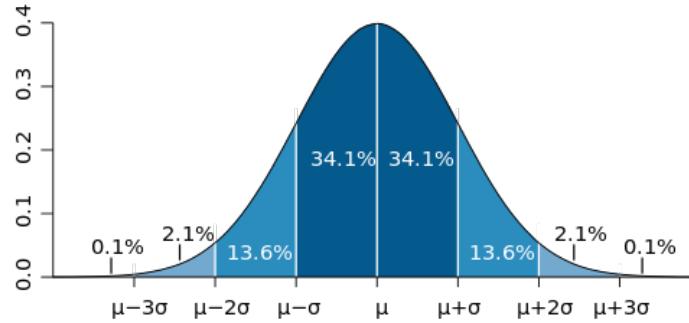


Figura 3.22: Porcentaje de datos en la distribución normal

Posteriormente utilizando la medida del GPS se asigna aleatoriamente un conjunto de partículas mediante una distribución normal con media la posición dada por el sensor GPS y varianza 3, 3, 0.25 para cada una de las variables x,y,theta. De esta forma, la separación entre las partículas será de 3m en las coordenadas horizontales y verticales y de 42° en la rotación tal y como podemos ver en la imagen 3.23.

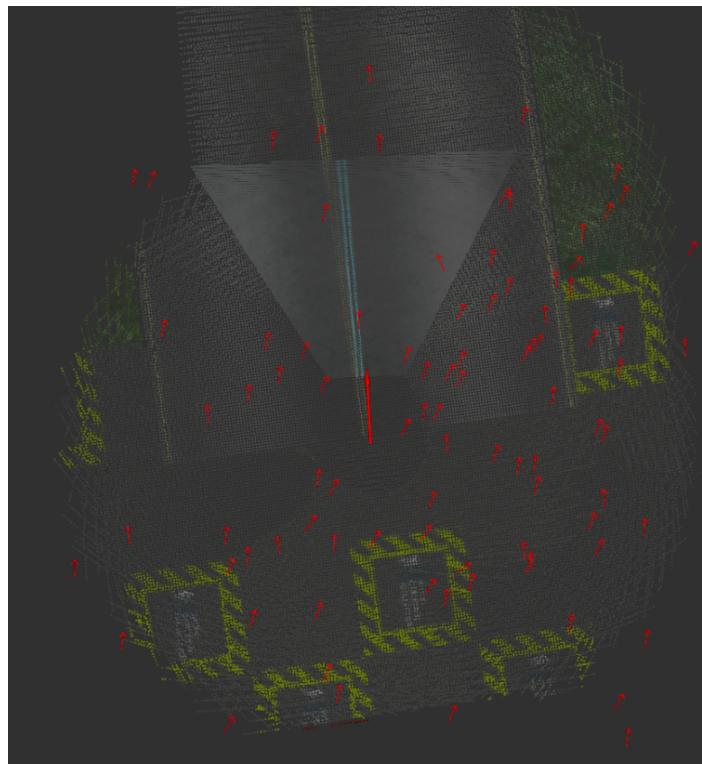


Figura 3.23: Partículas tras ser inicializadas

Después se predice la posición de las partículas utilizando la odometría del robot, esto se realiza aplicando la distancia o rotación que ha recorrido el robot según el movimiento de las ruedas a cada una de las partículas tal y como se muestra en las ecuaciones 3.4, 3.5 y 3.6. Posteriormente se le añade un ruido gaussiano de media la posición de la partícula y varianza 0.3,0.3,0.1, dando así a dos partículas que deberían

estar en la misma posición una diferencia entre ellas en las diferentes dimensiones de un máximo de 17° y de hasta 1 metro de distancia en x e y. En la figura 3.24 se ve como el robot ha aplicado un giro de casi 160° y todas las partículas han rotado su posición esos 160° .

$$\text{Particula}X_t = \text{Particula}X_{t-1} + X_t - X_{t-1} \quad (3.4)$$

$$\text{Particula}Y_t = \text{Particula}Y_{t-1} + Y_t - Y_{t-1} \quad (3.5)$$

$$\text{ParticulaTheta}_t = \text{ParticulaTheta}_{t-1} + \Theta_t - \Theta_{t-1} \quad (3.6)$$

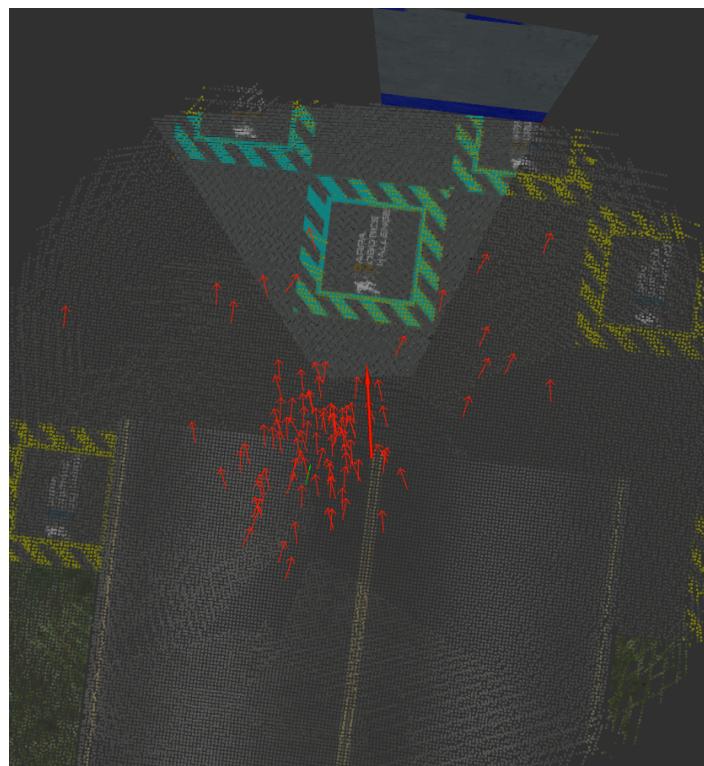


Figura 3.24: Partículas tras la fase de predicción al haber aplicado una rotación al robot

A continuación se actualizan los pesos de las partículas basándose en las imágenes que captarían las partículas y la imagen captada por el sensor tal y como se explicó anteriormente. Posteriormente se realizaría el Resample Wheel donde el filtro de partículas se quedaría con las partículas de mayor peso, obteniéndose un nuevo conjunto de partículas con las mejores partículas. Cabe destacar que en este nuevo conjunto de partículas puede haber partículas repetidas debido a su peso tal y como vemos en la figura 3.26 pero que posteriormente en la fase de predicción estas darían lugar a partículas diferentes debido al ruido que se le añade.

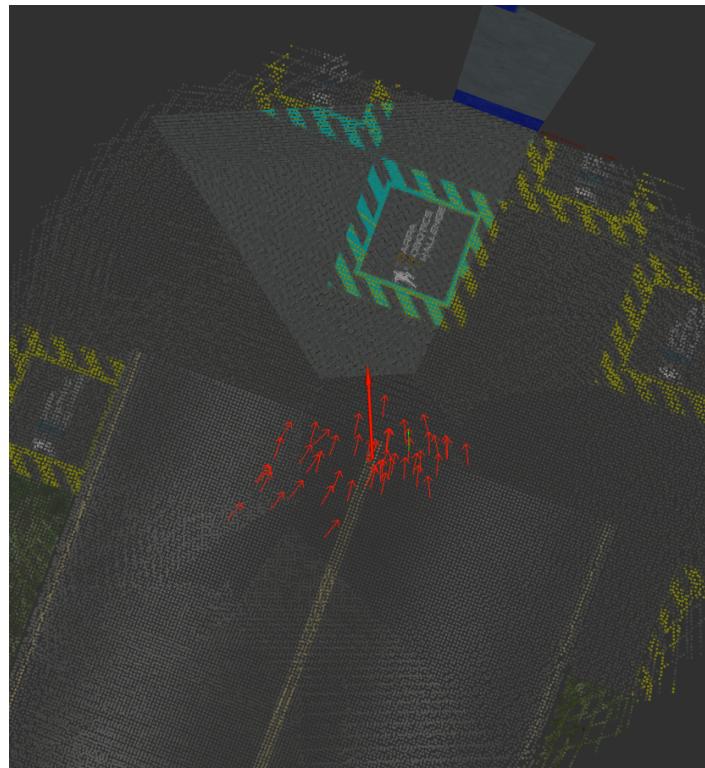


Figura 3.25: Partículas elegidas para la próxima generación de nuevas partículas

Una vez obtenido el nuevo conjunto nos quedaremos con la partícula con mayor peso que servirá para decidir la posición del robot según el algoritmo, la cual se puede ver en la figura 3.26 como una flecha de color verde junto con la posición real del robot mostrada como una flecha de color rojo.



Figura 3.26: Mejor partícula del conjunto y posición real

Después de esto se volvería a la fase de predicción para seguir localizando el robot independientemente del movimiento que el robot realice entrando ya en el bucle que se muestra en la figura 3.27.

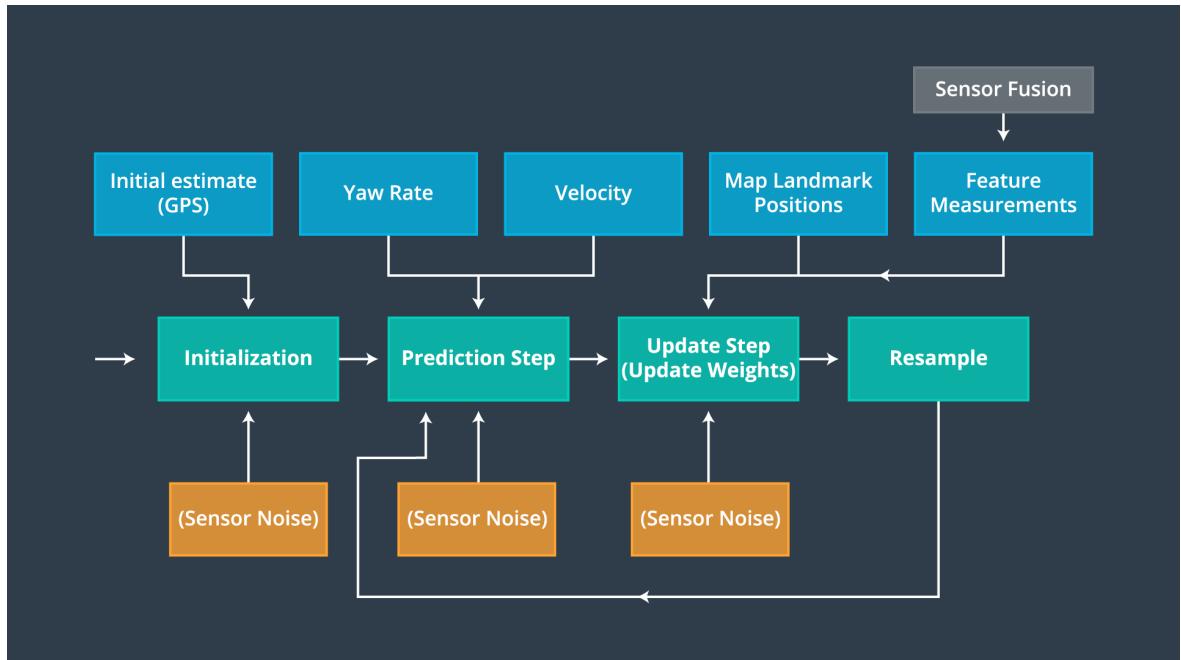


Figura 3.27: Esquema del filtro de partículas

3.2.9. Conclusiones del segundo experimento

Las conclusiones de este segundo experimento se sacan de la figura ??), donde se puede ver el error acumulado en cada iteración y pudiendo deducirse que el error se va corrigiendo y no va aumentando continuamente tal y como haría un método de localización basado en la odometría. Por lo que se ha demostrado la importancia del uso de un algoritmo de localización.

FIGURA DE ESTADÍSTICAS

Además, en este experimento hemos podido comprender la gran utilidad de ROS a la hora de manejar elementos de un robot pero la representación de los datos en estadísticas es algo que no implementa y que debemos hacerlo manualmente. En cuanto a la unión de ROS con Gazebo se ha comprobado el gran potencial que tiene, pues permite realizar cualquier tipo de proyecto relacionado con la robótica sin necesidad de hardware, únicamente utilizando un entorno de simulación que aunque al principio es bastante complejo, tiene muchas características y tiene mucha flexibilidad para poder crear muchas soluciones a los problemas.

Y por supuesto hemos aprendido la gran importancia de los filtros bayesianos para la localización de un robot y como en el filtro de partículas se debe tener en cuenta que a mayor cantidad de partículas más duro se hace el trabajo computacionalmente y más exacta y precisa será la solución al problema de la localización del robot. De esta manera se debe elegir una solución en la que el trabajo de computación no sea muy elevado y la precisión y exactitud del filtro sean adecuadas. Otra característica que hemos encontrado en los filtros de partículas tras programar el nuestro propio ha sido la sencillez y la gran capacidad de modificación que este tiene, permitiendo crear un filtro de partículas especializado en cada tipo de problema específico.

Capítulo 4

Conclusiones y Propuestas

4.1. Conclusiones

El trabajo realizado se ha centrado estudio, planteamiento y desarrollo de varias soluciones, para probar diferentes métodos de localización en un coche autónomo.

Algunas de estas soluciones implican mayor complejidad computacional, otras son poco precisas, pero todas permiten localizar al robot mediante información visual. Además en este trabajo de Fin de Grado se ha aprendido una gran cantidad de conceptos relativos a la robótica y se ha estudiado el uso de herramientas como Gazebo y ROS.

4.2. Competencias cubiertas

En este trabajo se han utilizado técnicas de las asignaturas de la rama de computación como son la visión artificial y la robótica.

La robótica ha sido el centro del trabajo, estudiando una parte del temario de una forma más profunda, esta parte ha sido el estado del arte de los filtros paramétricos y los no paramétricos y se ha comprobado empíricamente que son de gran utilidad para la localización pero es necesario tener unos buenos métodos para el cálculo de pesos y buenos sensores para que estos funcionen con excepcionalidad.

Posteriormente se han utilizado herramientas clave en la robótica como son ROS y Gazebo donde se han demostrado las capacidades de estos filtros.

Además se han tratado con sensores de diversos tipos y se han utilizado diversos métodos de localización y navegación.

En cuanto a la visión artificial se han utilizado técnicas para la detección de elementos de una imagen y la comparación de imágenes mediante técnicas relacionadas con los histogramas de una imagen.

Otras disciplinas tratadas en el trabajo han sido el diseño de algoritmos y la aplica-

ción de técnicas de sistemas inteligentes, evaluando la complejidad computacional de un problema, resolviendo, desarrollando e implementando una solución a dicho problema.

4.3. Trabajo futuro

4.4. Bibliografía

Bibliografía

- [1] ELECTRONICTEACHER. What is Robotics, 2000. [1](#), [7](#)
- [2] GAZEBO. Gazebo, Robot simulation made easy. [2](#)
- [3] LACEY, T. Tutorial: The Kalman Filter. ch. 11. [2](#), [3](#)
- [4] ROS. Tutoriales de ROS, Robot Operating System. [1](#)
- [5] SCHWABER, K., AND SUTHERLAND, J. La Guía Definitiva de Scrum: Las Reglas del Juego. [3](#)
- [6] THRUN, S. Particle Filters in Robotics. [2](#), [3](#)
- [7] THRUN, S., BURGARD, W., AND FOX, D. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. 2005. [2](#)

Contenido del CD

En el contenido del CD que acompaña a la memoria podemos encontrar los siguientes recursos:

- Memoria del trabajo en formato PDF.
- Código fuente del trabajo dentro del directorio Código fuente.
- Libros y artículos a los que se ha hecho referencia durante la memoria y que se han utilizado como bibliografía. Los cuales podemos encontrar en el directorio Bibliografía.
- Páginas Web que han servido de bibliografía. Las podemos encontrar dentro del directorio Bibliografia/Enlaces Web.
- Fichero con la explicación de los diferentes nodos del código fuente, lo podemos encontrar en el directorio de Código Fuente/Readme.