



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	XV
ÍNDICE DE TABLAS Y ALGORITMOS	XV
Lista de Tablas	1
1. INTRODUCCIÓN	1
1.1. Motivación	1
1.2. Objetivos	3
1.3. Estructura de la memoria	4
2. ANTECEDENTES Y ESTADO DE LA CUESTIÓN	5
2.1. Robótica	5
2.2. El problema de la localización	6
2.3. Algoritmos de localización	9
2.4. Matemática aplicada a la informática gráfica	18
2.5. Escenario del problema	21
2.6. Herramientas Software utilizadas	21
3. METODOLOGÍA Y DESARROLLO	27
3.1. Metodología seguida para el desarrollo del trabajo	27
3.2. Estudio previo	29
3.3. Desarrollo del trabajo	37
4. EXPERIMENTOS Y RESULTADOS	49
4.1. Tiempos de cada una de las fases del filtro de partículas	49
4.2. Resultados obtenidos del experimento	50
4.3. Conclusiones del experimento	58
5. CONCLUSIONES Y PROPUESTAS	59
5.1. Conclusiones	59
5.2. Competencias cubiertas	59

5.3. Trabajo futuro	60
BIBLIOGRAFIA	62
CONTENIDO DEL CD	63

ÍNDICE DE FIGURAS

1.1.	Pioneer 3-DX	2
1.2.	Modelo en el entorno Gazebo de un robot Pioneer 3-DX	2
1.3.	Coche autónomo	2
2.1.	Localización al comienzo	7
2.2.	Localización tras realizar medidas	7
2.3.	Ejemplo de distribución Gaussiana	9
2.4.	Proceso del algoritmo de filtro de partículas	15
2.5.	Diagrama Resampling Wheel	16
2.6.	Obtención de nuevas partículas en Resampling Wheel	17
2.7.	Traslación en el espacio	18
2.8.	Rotación en el espacio	19
2.9.	Escalado en el espacio	20
2.10.	Robot Pioneer 3DX en el entorno de simulación	23
2.11.	Sistema de navegación de ROS	23
2.12.	Sistema de navegación de ROS (fuente: [1])	24
3.1.	Círculo realizado para el estudio previo de los algoritmos de localización	30
3.2.	Visualización en Rviz de la generación del mapa mediante SLAM	31
3.3.	Mapa generado con la técnica de SLAM	31
3.4.	Configuración en Rviz para mostrar elementos mientras el robot navega	33
3.5.	Error en metros en el método de odometría, eje x	34
3.6.	Error en metros en el método de odometría, eje y	34
3.7.	Error en metros en el método de Filtro de Partículas, eje x	34
3.8.	Error en metros en el método de Filtro de Partículas, eje y	35
3.9.	Error en metros en el método de Filtro de Kalman, eje x	35
3.10.	Error en metros en el método de Filtro de Kalman, eje y	35
3.11.	Comparación de errores en el eje x entre los diferentes métodos de localización	36
3.12.	Comparación de errores en el eje y entre los diferentes métodos de localización	37

3.13. Entorno simulado en Gazebo	38
3.14. Nube de puntos obtenida al realizar un mapeo con rtabmap	39
3.15. Nube de puntos obtenida recortando la nube de puntos obtenida mediante mapeo.	40
3.16. Nube de puntos obtenida por el sensor	41
3.17. Imagen obtenida desde arriba proyectando la nube de puntos del sensor sobre el plano z	41
3.18. Zona de interés seleccionada de la imagen	42
3.19. Función exponencial decreciente, $3^{1/x}$	43
3.20. Función exponencial decreciente, $f(x) = 3^{1/(x/100)}$	44
3.21. Porcentaje de datos en la distribución normal	45
3.22. Partículas tras ser inicializadas	45
3.23. Mejor partícula del conjunto y posición real	47
3.24. Esquema del filtro de partículas	48
4.1. Tiempos medios de cada una de las fases del filtro de partículas	50
4.2. Evolución del conjunto de partículas	51
4.3. Error total en metros y radianes de todo el conjunto de partículas con respecto a la posición real en cada iteración	51
4.4. Error en metros y radianes de la mejor partícula con respecto a la posición real en cada iteración	52
4.5. Error en metros del conjunto de partículas con respecto a la posición real en cada iteración	52
4.6. Error en metros y radianes de la localización basada en la odometría con respecto a la posición real en cada iteración	53
4.7. Error en metros y radianes de la localización del algoritmo con respecto a la posición real en cada iteración	54
4.8. Visualización de la información de los elementos del filtro de partículas en la última iteración	54
4.9. Entorno gazebo con obstáculos	55
4.10. Corrección de ambigüedad	55
4.11. Errores en metros y radianes del conjunto de partículas en el experimento en el que se añaden obstáculos	56
4.12. Errores en metros y radianes de localización en el experimento en el que se añaden obstáculos	56
4.13. Errores en metros y radianes de la odometría con respecto a la posición real del robot	57
4.14. Errores totales en metros y radianes del conjunto de partículas en la localización con errores de odometría y con obstáculos	57

4.15. Errores en metros y radianes de localización con errores de odometría y con obstáculos	57
---	----

ÍNDICE DE TABLAS Y ALGORITMOS

2.1.	Conjunto de celdas al principio del ejemplo	8
2.2.	Conjunto de celdas después de realizar la primera medida del ejemplo	8
2.3.	Filtro de Kalman	11
2.4.	Filtro de Partículas	14
2.5.	Algoritmo Resampling Wheel	17
3.1.	Fase de inicialización	44
3.2.	Fase de predicción	46
3.3.	Filtro de Partículas	47
4.1.	Tiempos medios en segundos de cada una de las fases del algoritmo	49

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[2].

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 [3], 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[4], donde se importará un modelo del robot Pioneer 3-DX que es el robot que nos facilita la universidad para realizar el proyecto (Véase la figura 1.1 y 1.2).



Figura 1.1: Pioneer 3-DX

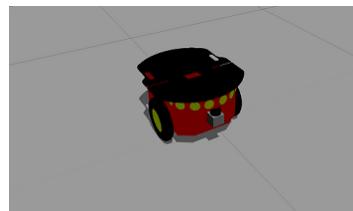


Figura 1.2: Modelo en el entorno Gazebo de un robot Pioneer 3-DX

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 como los que podemos encontrar en "Probabilistic Robotics" [5] y en el desarrollo de un algoritmo de localización que cometa el mínimo error posible para poder utilizarlo en un vehículo autónomo. Además se estudiarán conceptos de navegación y de visión artificial y reconocimiento de patrones útiles para el desarrollo del algoritmo de localización.



Figura 1.3: Coche autónomo

Para la localización se estudiará y desarrollará un método de localización basado en Filtros de Partículas [6], debido a que de igual forma que el Filtro de Kalman [7] 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 la sencillez para desarrollarlo y que ofrece una solución multimodal.

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 mover el 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 previamente el estudio de:

- Robótica móvil, 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 del Filtro de Kalman [7] 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.
- Mapeo de entornos en 3D utilizando diferentes sensores como lasers, cámaras estéreo y cámaras de profundidad.
- Matemática aplicada para la informática gráfica, para realizar transformaciones a los diferentes objetos en un entorno 3D.
- Métodos de comparación y registrado de imágenes.
- Librerías para implementar el componente de ROS, como los tutoriales básicos, TF, OpenCV y PCL.
- Librería para implementar paralelismo como OpenCV

- Hardware Pioneer 3AT.
- Comunicación entre otros módulos del sistema.

1.3. Estructura de la memoria

La presente memoria del Trabajo de Fin de grado se desarrolla presentando los contenidos teóricos y posteriormente su puesta en práctica, resultando en la siguiente estructura:

- **Capítulo 1: Introducción:** En este capítulo se da un primer acercamiento a la robótica, se describe la motivación y los objetivos del Trabajo de Fin de Grado.
- **Capítulo 2: Antecedentes y Estado de la cuestión:** Capítulo donde se da una explicación del panorama actual en este ámbito, se presentan diferentes métodos de localización junto con su respectivas bases teóricas. Además, se incluye un apartado sobre el estado actual de los métodos de comparación de imágenes, la matemática aplicada a la informática gráfica y por último se explica el escenario del problema de la localización y las herramientas utilizadas.
- **Capítulo 3: Metodología y Desarrollo** En este capítulo se describe la metodología seguida en el proyecto y el desarrollo del mismo.
- **Capítulo 4: Experimentos y resultados:** En este capítulo se muestran y explican los diferentes experimentos realizados con ROS en el entorno de simulación Gazebo junto con una conclusión y reflexión para analizar posibles mejoras.
- **Capítulo 5: Conclusiones y propuestas:** Capítulo donde se desarrolla una reflexión final sobre el presente Trabajo de Fin de Grado, se habla sobre la efectividad de la propuesta realizada y de trabajos futuros con respecto a este trabajo.
- **Bibliografía** Aparecen los documentos, artículos y páginas web de donde se ha obtenido la información.
- **Contenido del CD** Proporciona información de los archivos incluidos en el CD.

Capítulo 2

ANTECEDENTES Y ESTADO DE LA CUESTIÓN

2.1. Robótica

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 definitivo 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 [8]:

- **Robot industrial o manipulador:** Son artilugios 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 de realismo, 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 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 a partir de la información proporcionada por sus sensores.

2.2. El problema de la localización

La localización es la capacidad de una máquina para localizarse a sí 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 comete, 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 se debe 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 asociada a la estimación de la posición actual del robot.

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).

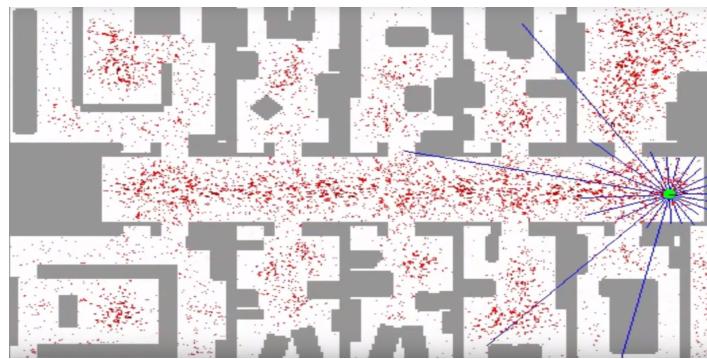


Figura 2.1: Localización al comienzo

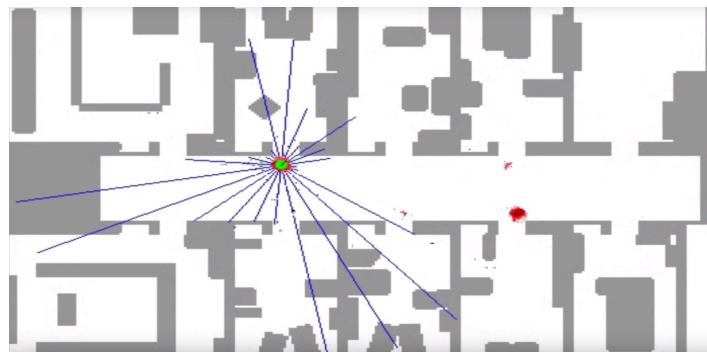


Figura 2.2: Localización tras realizar medidas

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 está 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 paramétrico debido al uso de una distribución normal 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 distribución normal tal y como podemos ver en la figura 2.3.

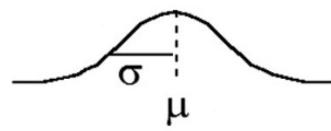


Figura 2.3: Ejemplo de distribución Gaussiana

La función de la distribución normal 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 n¹) 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 función 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, $n * n$. 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 $n * m$, 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 anterior 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:

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

¹número de estados posibles

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

El algoritmo del Filtro de Kalman da como salida la $\overline{\text{creencia}}(x_t)$ que es una distribución normal con media μ_t y varianza Σ_t y representa la localización.

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

- En las lineas 1 y 2 se implementa la fase de predicción que determina $\overline{\text{creencia}}(x_t)$.
- De la linea 3 a la 5 se implementa la actualización de las medidas, permitiendo conocer z_t para determinar $\text{creencia}(x_t)$. En la linea 3 la ganancia de Kalman, K_t , es calculada. K_t especifica el grado de verdad en z_t sobre $\overline{\text{creencia}}(x_t)$. En la linea 4, la medida z_t es comparada con la predicción de la medida de los sensores μ_t .

Algoritmo 2.3: 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 para estimar el estado de un sistema que cambia a lo largo del tiempo.

El Filtro de Partículas se ha convertido en una gran técnica 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 este 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 [6].

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.

La utilidad del filtro de partículas se obtiene cuando se descubre el problema del filtro de Kalman, este problema reside en que no se proporciona una estimación razonable para problemas altamente no lineales y que no siguen una distribución normal. 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, siendo estas partículas, los 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 se utiliza 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:

- Funciona con sistemas multi-modales, independientemente de la linealidad y del ruido debido a la gran cantidad de partículas usadas en el filtro.
- 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 debe 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 también ser creado mediante algún tipo de información a priori (posición aproximada).
- **Actualización de pesos:** 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 utilizando métodos de "re-sampling" probabilísticos, de forma que aquellas partículas que mejor se ajusten a las medidas (partículas de mayor peso) darán lugar a nuevas partículas con mayor probabilidad consiguiendo el conjunto de partículas que es probablemente el más correcto. Como ejemplo de método de "re-sampling" encontramos "Resampling Wheel" [9] que es el método de resampling más fácil de implementar o "KLD-Sampling" [10] que se utiliza en el filtro de partículas que proporciona ROS por defecto.
- **Predicción:** El Filtro de Partículas modifica el estado de las partículas basándose en la estimación del estado en el siguiente instante de tiempo. Posteriormente realiza una leve modificación al estado de cada una de ellas introduciendo algún tipo de ruido aditivo que aporte variabilidad al conjunto de partículas. 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.

Algoritmo 2.4: Filtro de Partículas

1	Algoritmo de Filtro de Partículas(X_{t-1}, u_t, z_t):
2	$\bar{X}_t = X_t = Random$
3	for m = 1 to M do:
4	sample $x_t^{[m]}$ $p(x_t u_t, x_{t-1}^{[m]})$
5	$w_t^{[m]} = p(z_t x_t^{[m]})$
6	$\bar{X}_t = \bar{X}_t + (x_t^{[m]}, w_t^{[m]})$
7	end for
8	for m=1 to M do:
9	draw i with probability $\alpha * w_t^{[i]}$
10	add $x_t^{[i]}$ to X_t
11	end for
12	return X_t

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

- En la línea 2, Se inicializa el conjunto de partículas del siguiente instante de tiempo y el conjunto de partículas predicho del siguiente instante a un conjunto generado aleatoriamente.
- En la línea 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 (velocidad, posición, etc).
- En la línea 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 línea 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 línea 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.

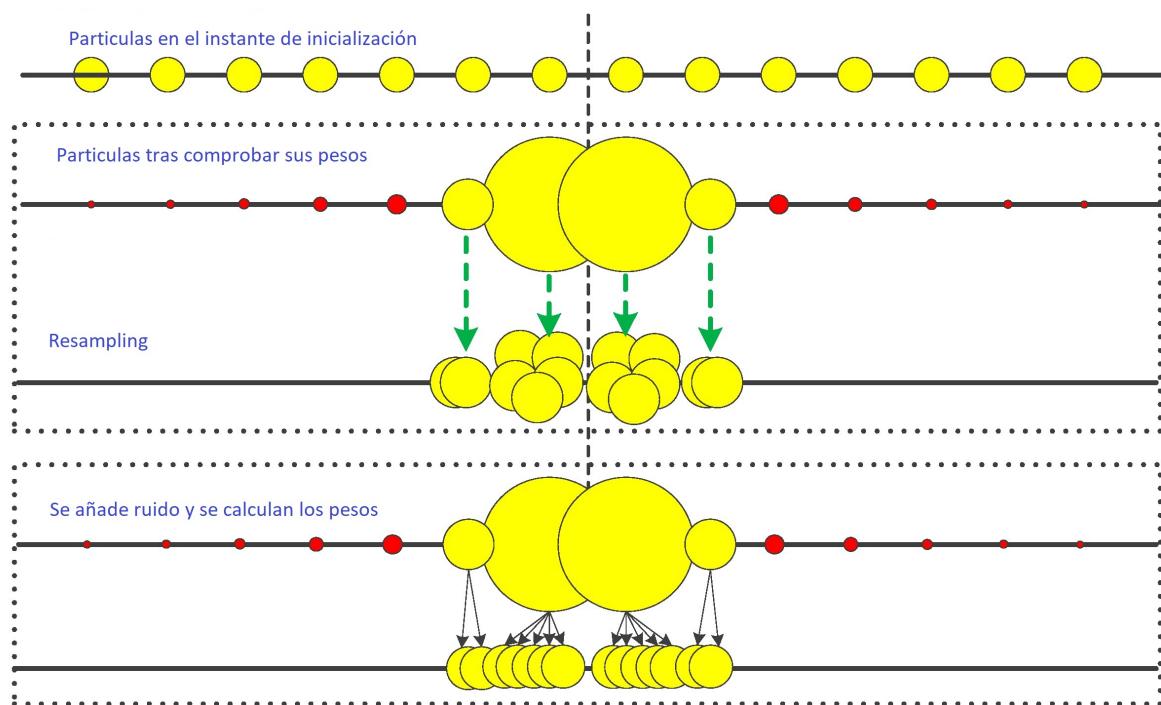


Figura 2.4: Proceso del algoritmo de filtro de partículas

Resampling Wheel

El método "Resampling Wheel" es el método de reemplazo más fácil de entender [9], pudiendo resumir su funcionamiento en la imagen 2.5 que se explicará posteriormente.

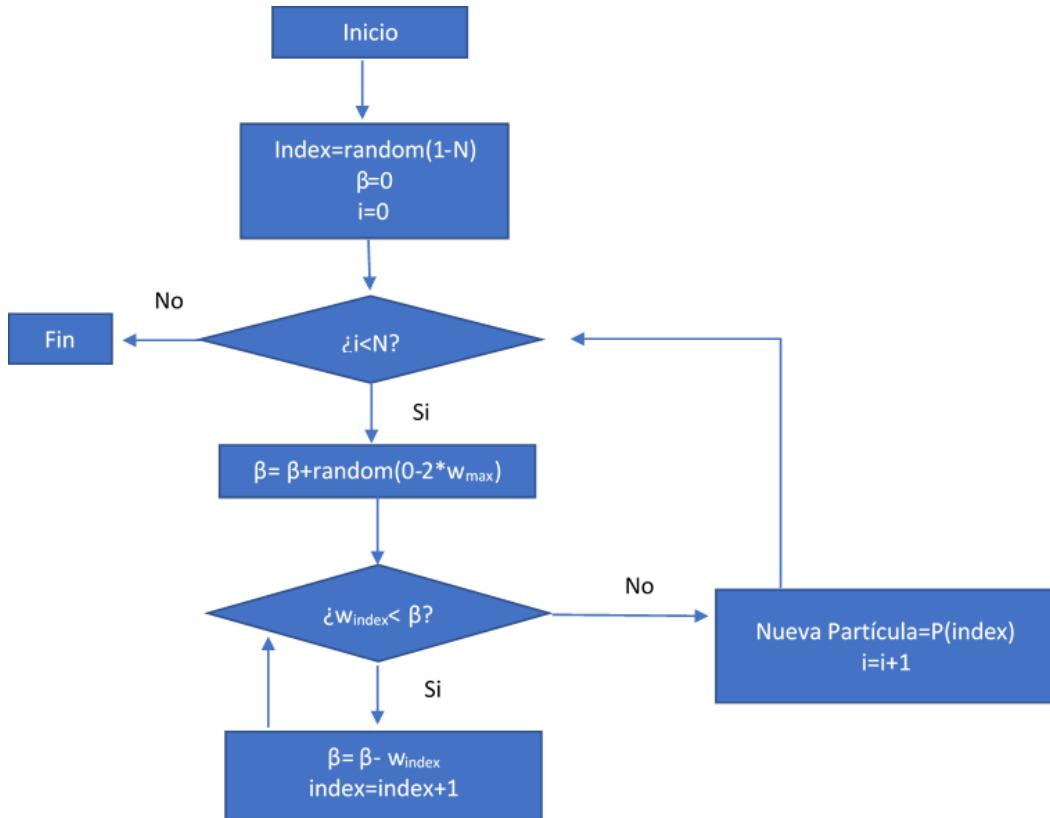


Figura 2.5: Diagrama Resampling Wheel

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.

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 β 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 β 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 2.5.

Algoritmo 2.5: 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).

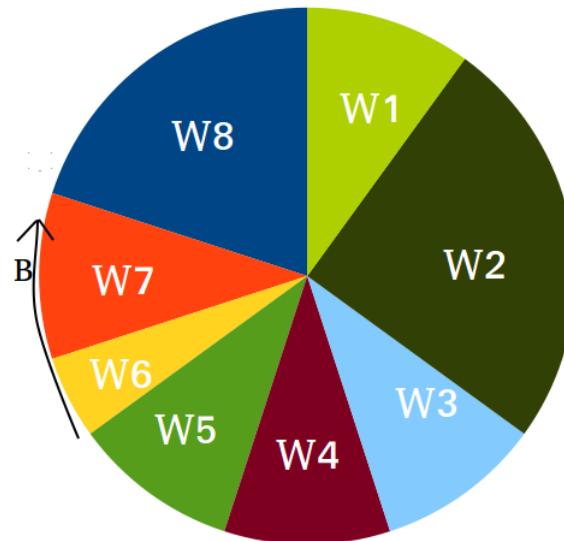


Figura 2.6: Obtención de nuevas partículas en Resampling Wheel

2.4. Matemática aplicada a la informática gráfica

2.4.1. Introducción

Puesto que en este trabajo se manejarán puntos en el espacio que serán modificados mediante traslaciones y rotaciones se explicará un poco de teoría en este apartado.

2.4.2. Transformaciones geométricas 3D

En este apartado se estudiarán los tres tipos de transformaciones geométricas en un espacio de 3 dimensiones.

Traslación

Comenzaremos explicando la traslación, la traslación consiste en la acción y efecto de mover un elemento en el espacio (figura 2.7).

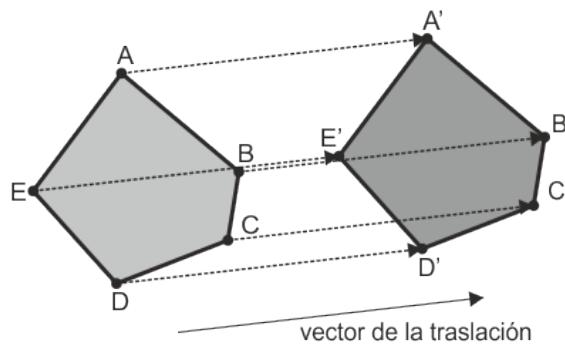


Figura 2.7: Traslación en el espacio

Para realizar esta traslación en un espacio en tres dimensiones es tan sencillo como aplicar la función 2.4.

$$\begin{aligned}x' &= x + tx \\y' &= y + ty \\z' &= z + tz\end{aligned}\tag{2.4}$$

Esta función se puede expresar matricialmente como la matriz de traslación MT²:

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

²Matriz de traslación

Rotación

En cuanto a la transformación geométrica de rotación consiste en la acción y efecto de girar un elemento en el espacio con respecto a uno de los ejes (figura 2.8).

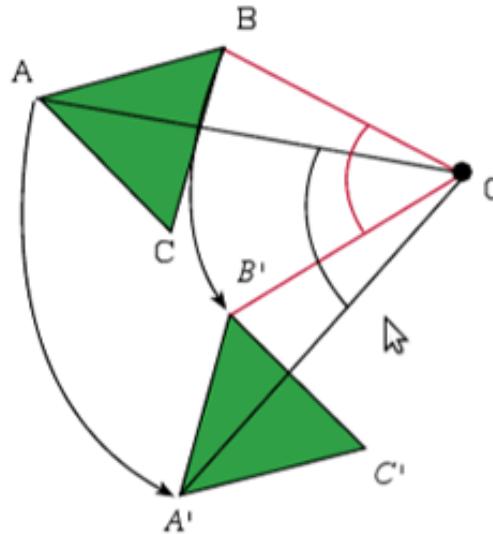


Figura 2.8: Rotación en el espacio

Para realizar esta rotación sobre el eje z en un espacio en tres dimensiones se debe aplicar la función 2.5, esta función cambiará dependiendo del eje sobre el que se quiera rotar.

$$\begin{aligned}x' &= x * \cos(\alpha) - y * \sin(\alpha) \\y' &= x * \sin(\alpha) + y * \cos(\alpha) \\z' &= z\end{aligned}\tag{2.5}$$

Estas funciones de rotación, dependiendo del eje del que se quiera rotar son expresadas matricialmente como las matrices MRX³, MRY⁴, MRZ⁵:

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

³Matriz de rotación para rotar sobre el eje X

⁴Matriz de rotación para rotar sobre el eje Y

⁵Matriz de rotación para rotar sobre el eje Z

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

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

Además hay que tener en cuenta que la rotación se aplica siempre con respecto al origen de coordenadas, por lo que en algunos casos hay que trasladar primero el elemento que queremos rotar al origen de coordenadas si no queremos que este sea trasladado de posición.

Escalado

En cuanto al escalado, este movimiento geométrico consiste en la modificación de las dimensiones, permitiendo agrandar el objeto en las diferentes dimensiones del espacio (figura 2.9).

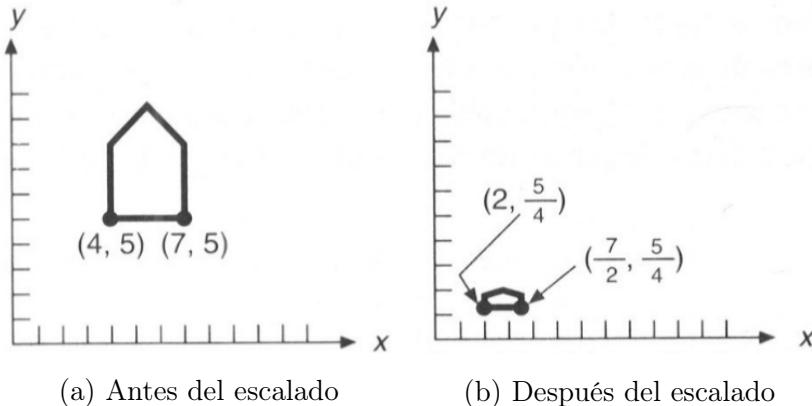


Figura 2.9: Escalado en el espacio

Para realizar esta transformación geométrica de escalado en tres dimensiones, utilizando sx^6, sy^7, sz^8 , se debe aplicar la función 2.6.

$$\begin{aligned} x' &= x * sx \\ y' &= y * sy \\ z' &= z * sz \end{aligned} \tag{2.6}$$

⁶Escala en el eje x en tanto por uno

⁷Escala en el eje x en tanto por uno

⁸Escala en el eje x en tanto por uno

Este escalado en el espacio puede ser expresada matricialmente como la matriz de escalado MS⁹:

$$MS = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.4.3. Composición de transformaciones

Por último, es destacable conocer que al utilizar matrices para aplicar transformaciones geométricas se pueden concatenar transformaciones.

Estas concatenaciones dependerán de como se aplique la transformación a los diferentes puntos, en nuestro caso vamos a utilizar la post-multiplicación, de forma que para concatenar transformaciones en una sola matriz se debe hacer de derecha a izquierda, es decir, para aplicar una transformación de traslación al punto seguida de una transformación de rotación se debe realizar la concatenación MR*MT*Punto. Esto lo utilizaremos en el apartado de obtener la nube de puntos de una partícula.

2.5. 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 desarrollar un método de filtro de partículas para localizar el robot.

2.6. Herramientas Software utilizadas

2.6.1. ROS

Introducción a ROS

Para la realización del trabajo se usa el "middleware" 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.

⁹Matriz de escalado

Arquitectura de ROS

ROS tiene una arquitectura específica[11] que debemos entender para poder realizar el trabajo con este sistema. En ROS encontramos diferentes elementos:

- **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.
- **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.
- **Nodos:** Los nodos son los procesos que realizan la computación en esta arquitectura. Estos nodos pueden utilizar o proporcionar servicios, topics u otras acciones para comunicarse con otros nodos, permitiendo, por ejemplo, la capacidad de obtener imágenes de sensores. Estos nodos son escritos utilizando librerías de ROS como pueden ser roscpp que utiliza C++ como lenguaje de programación, rospy que utiliza Python como lenguaje de programación u otros.

2.6.2. 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 2.10), 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 2.10: Robot Pioneer 3DX en el entorno de simulación

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 2.11).



Figura 2.11: Sistema de navegación de ROS

2.6.3. Funcionamiento del sistema de navegación de ROS

Para la utilización de un movimiento autónomo que permita evitar obstáculos con el robot es necesario conocer como funciona el sistema de navegación de ROS (figura 2.12).

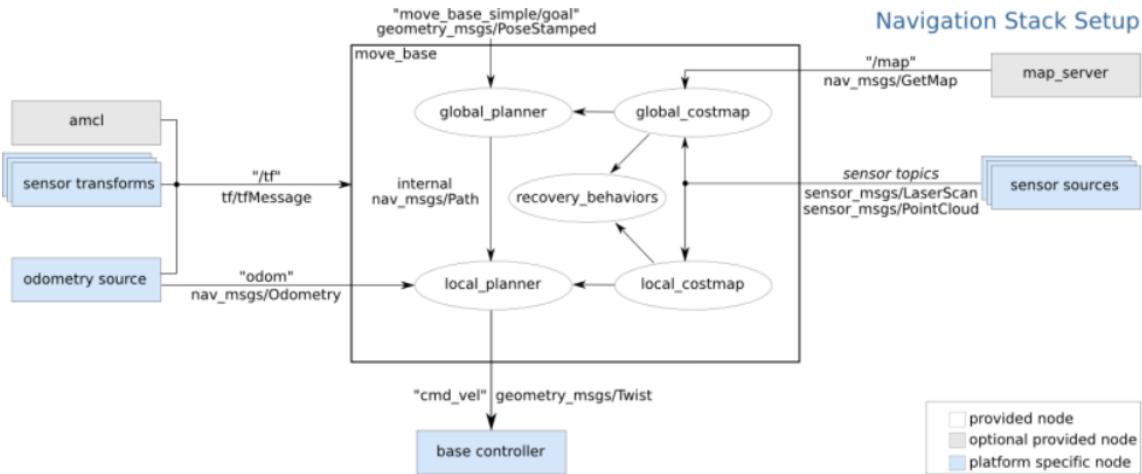


Figura 2.12: Sistema de navegación de ROS (fuente: [1])

Si se echa un vistazo a la figura 2.12, 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 y no debemos modificar son todos los nodos dentro del conjunto ‘‘move_base’’. Estos nodos tienen 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 posición real del robot en el entorno que será publicada en el topic ‘‘/odom’’.

El algoritmo de localización definido en la figura 2.12 es ‘‘amcl’’, que es el método de Monte Carlo que ofrece ROS para la localización y que utiliza la odometría y el árbol de transformaciones para conocer la posición del robot. Esta posición se usará en el nodo planificador permitiendo obtener las rutas a realizar y 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 obtenida mediante los sensores para que sea capaz de evitar obstáculos, esto lo realizaremos mediante el topic ‘‘/sensor_msgs/PointCloud’’ puesto que nuestro sensor es una cámara que tendrá una nube de puntos.

La salida del sistema de navegación será la publicación de un topic del tipo *geometry_msgs::Twist* [12], concretamente publicará en el topic ‘‘cmd_vel’’ que dirigirá nuestro robot al punto que se le ordene mediante un planificador que irá evitando obs-

táculos siempre y cuando hayamos configurado el robot para que se suscriba al topic “cmd_vel” y lo utilice para modificar su velocidad.

En este trabajo se modificará ”amcl” cambiándolo por un algoritmo de Filtro de Partículas que maneje la información visual. Además, se cambiarán las fuentes de los sensores y el mapa del entorno para adaptarlo a nuestro proyecto.

2.6.4. 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, permitiendo 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, como aplicar velocidades a los modelos y permitiendo el movimiento al robot.

Capítulo 3

METODOLOGÍA Y DESARROLLO

3.1. Metodología seguida para el desarrollo del trabajo

Para la realización de este trabajo se ha utilizado una metodología de desarrollo ágil, concretamente el framework Scrum adaptado[13] 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.

En Scrum los roles de los integrantes del equipo han sido:

- **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.

El funcionamiento de este framework se realiza por medio de Sprints. Un Sprint es un bloque de tiempo de hasta un mes de duración.

Antes de comenzar el Sprint se establecen los requisitos del proyecto y se almacenan en un "Product Backlog".

Este Sprint contiene 4 eventos diferenciados:

- **Planificación del Sprint:** Evento en el que participan todos los miembros del equipo Scrum. En este evento se establece un objetivo para el Sprint y se planifica el trabajo a realizar durante el Sprint almacenándose en un "Sprint Backlog".

- **Scrum Diario:** Evento en el que participa el Equipo de Desarrollo. En este evento se planea el trabajo a realizar durante las próximas 24 horas. Posteriormente se divide el trabajo a realizar en subtareas, optimizando el desempeño del equipo y permitiendo evaluar el progreso hacia el objetivo planificado en la anterior fase.
- **Revisión del Sprint:** Evento en el que participan todos los miembros del equipo Scrum. En este evento se realizan los siguientes elementos
 - El Dueño del Producto explica qué elementos del "Product Backlog" se han terminado y cuales no.
 - El Equipo de Desarrollo comenta los problemas que aparecieron y cómo se resolvieron.
 - El Equipo de Desarrollo hace una demostración del trabajo que se ha terminado y responde preguntas sobre el incremento realizado al proyecto.
 - El Dueño del Producto proyecta objetivos probables y fechas de entrega basándose en el progreso obtenido hasta la fecha.
 - El grupo completo colabora acerca de qué hacer a continuación, corrigiendo en caso de ser necesario el "Product Backlog".
- **Retrospectiva del Sprint:** En este evento el Equipo Scrum inspeccionan cómo fue el Sprint en cuanto a procesos y herramientas, buscando mejorar el proceso de desarrollo del proyecto.

La organización del trabajo ha sido la siguiente:

- **Inicio del proyecto:** Esta fase planteará el proyecto con su planificación, definiendo los requisitos del sistema y todos los aspectos técnicos del proyecto.
- **Estudio del estado del arte:** Esta etapa se centrará en el estudio de la base teórica sobre la que se sustenta el trabajo, 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:** Fase donde se conseguirá realizar o encontrar un mapa para permitir la realización de pruebas y validaciones en el desarrollo del sistema.
- **Mapeo del entorno:** Etapa que se centrará en obtener la nube de puntos del entorno mediante la información visual dada por la cámara del robot.
- **Desarrollo de proyección de nube de puntos en imágenes:** Etapa que se centrará en captar las nubes de puntos obtenidas por los sensores y el mapeo,

recortarlas, transformarlas para obtener la nube de puntos que se obtendría en caso de que una partícula este en una posición dada y proyectarla en una imagen.

- **Desarrollo de mecanismos para comparación de imágenes:** Etapa que se centrará en desarrollar diferentes técnicas para comparar imágenes y dar pesos a las diferentes partículas.
- **Desarrollo del módulo de localización:** Etapa que se centrará en desarrollar el módulo de localización basado en el método del filtro de partículas utilizando información visual.
- **Desarrollo del módulo de navegación:** Etapa transversal que se utilizará durante todo el proyecto, permitiendo dar la capacidad al robot de moverse por el entorno.
- **Pruebas y validación:** Fase encargada de buscar fallos en el sistema para posteriormente solucionarlos y lograr el objetivo propuesto en el trabajo de fin de grado. Además se realizarán pruebas de diferentes experimentos llevados a cabo para comprobar el funcionamiento del módulo de localización implementado en ROS.
- **Presentación y defensa del trabajo de fin de grado.**

3.2. Estudio previo

3.2.1. Introducción

Para una mejor comprensión de los métodos de localización se ha realizado un estudio previo de los diferentes métodos de localización realizando un experimento con un sensor de distancia, un modelo de nuestro robot Pioneer-3AT y una habitación cerrada, concretamente el laboratorio software 1 de la Escuela Superior de Ingeniería Informática en la Universidad de Castilla La Mancha.

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.

Se ha movido el robot en el interior del laboratorio software 1 de la Escuela Superior de Ingeniería Informática en la Universidad de Castilla La Mancha tal y como se

muestra en la figura 3.1.

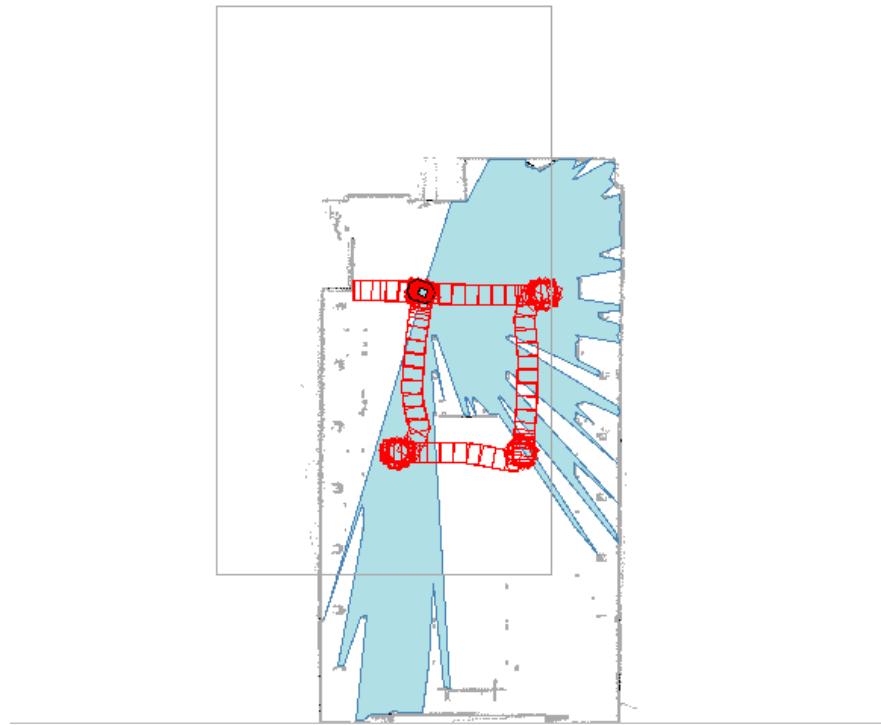


Figura 3.1: Circuito realizado para el estudio previo de los algoritmos de localización

3.2.2. Preparación del experimento

Los objetivos de este experimento han sido:

- Configurar Gmapping [14] para realizar SLAM[15].
- Crear y configurar el "stack" de navegación.
- Usar los métodos de localización y navegación.
- Evitar obstáculos en la navegación del robot mediante un planificador.

Configurando Gmapping

En este proyecto se necesita configurar Gmapping para generar el mapa que se utilizará después para la navegación y localización del robot.

Se ha utilizado un sensor láser (Lidar) del Pioneer 3-DX y la odometría para generar el mapa del entorno.

Para ello, se usó un comando que permite publicar mensajes de tipo *geometry_msgs::Twist* en el topic ''/cmd_vel'' simplemente mediante el teclado, controlando el robot mediante modificaciones en su velocidad lineal y angular:

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

Además, para ver el proceso de la generación del mapa se puede utilizar la herramienta Rviz [16], que permite ver una gran cantidad de elementos de forma gráfica como topics, el árbol de transformaciones, etc (figura 3.2).

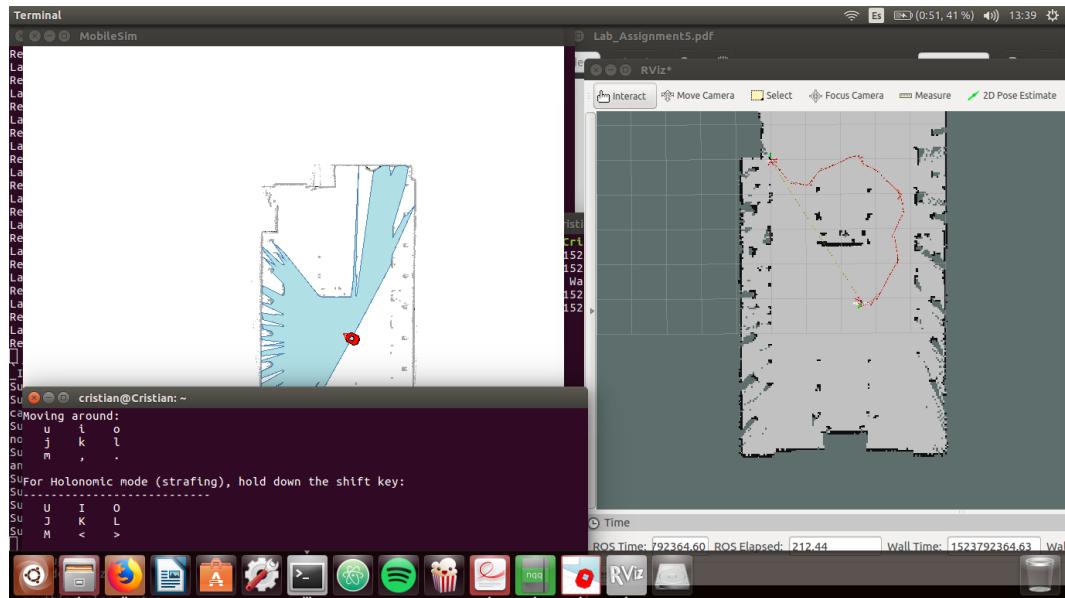


Figura 3.2: Visualización en Rviz de la generación del mapa mediante SLAM

Una vez completado el mapeo se almacena en el ordenador mediante el comando:

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

Generado el mapa se obtendría lo mostrado en la figura 3.3.

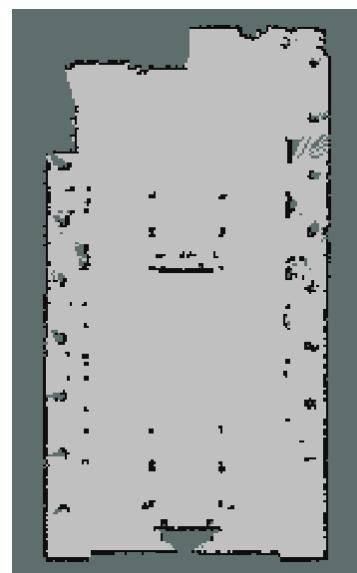


Figura 3.3: Mapa generado con la técnica de SLAM

Configurando el stack de navegación

Para la configuración del stack de navegación y dar navegación autónoma al robot evitando obstáculos, debemos establecer:

- **Un árbol de transformaciones:** Estableciendo un frame para el mapa, uno para la odometría del robot y otro para los sensores, permitiendo tener al sistema relaciones entre las coordenadas de los diferentes frames, como las traslaciones y rotaciones.
- **La información que utilizará para evitar obstáculos:** Estableciendo que tipo de sensor se utilizará para evitar obstáculos, en nuestro caso un sensor láser cuya estructura viene definida como "`sensor_msgs::LaserScan`" [17].
- **La posición del robot:** Es necesario conocer la posición del robot para que este pueda navegar. Esta posición se puede dar mediante un algoritmo de localización o utilizando la odometría del robot que permite conocer el movimiento que realiza el robot basándose en el movimiento de las ruedas. Esta posición tiene una estructura definida como "`nav_msgs::Odometry.msg`" [18].
- **El mapa del entorno:** Establecer un mapa del entorno permite planificar rutas que utilizaremos para la navegación autónoma del robot. Este mapa se establecerá mediante el nodo '`Map Server`' [19], que cargará el mapa que guardamos anteriormente utilizando Gmapping.

Navegación autónoma usando los diferentes métodos de localización

Para la utilización de los diferentes métodos de localización en la navegación autónoma se han utilizado nodos que ya proporciona ROS como "`amcl`" basado en un Filtro de Partículas y "`robot_pose_ekf`" basado en un Filtro de Kalman.

La navegación autónoma se ha realizado eligiendo una posición objetivo a la que llegar, posteriormente el robot obtiene la ruta óptima mediante el planificador y la sigue hasta llegar a su objetivo.

Mientras se realiza la navegación se puede utilizar Rviz para ver el mapa con los obstáculos (sombreados en el mapa), la ruta que define el planificador (línea de color verde), el conjunto de partículas (flechas rojas) y los frames que suponen la posición real del robot y el mapa tal y como se puede ver en la figura 3.4.

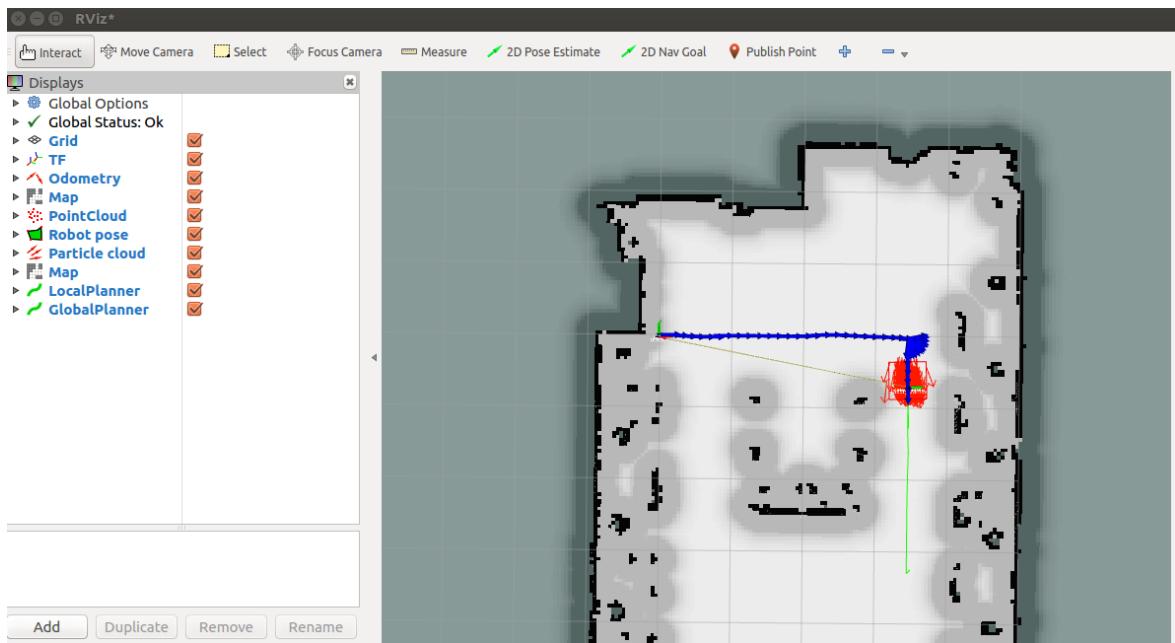


Figura 3.4: Configuración en Rviz para mostrar elementos mientras el robot navega

3.2.3. Resultados de los diferentes métodos de localización

Tras la navegación autónoma se ha realizado un análisis del error de la posición de los distintos métodos de localización.

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.

Los resultados se han mostrado utilizando un gráfico de barras donde el eje X es la iteración y el eje Y el error cometido.

En el gráfico obtenido por medio del método de localización basado en la odometría encontramos que los mayores errores han sido **0.052 m en el eje X** (movimiento vertical) tal y como podemos ver en la figura 3.5 y **0.058 m 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.

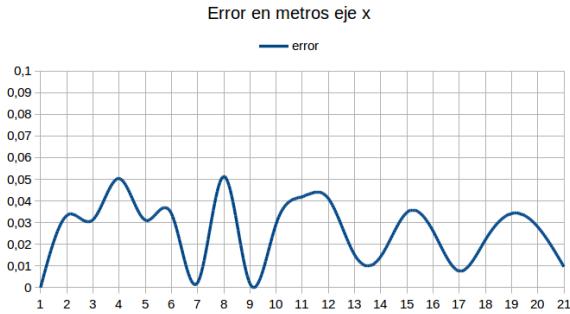


Figura 3.5: Error en metros en el método de odometría, eje x

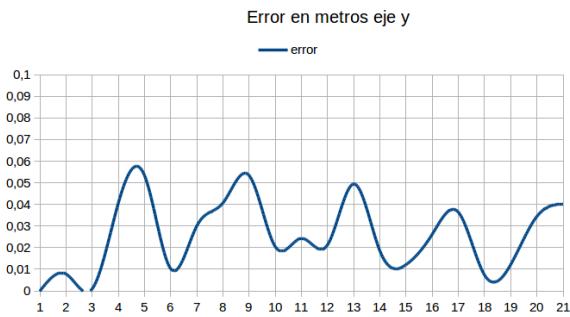


Figura 3.6: Error en metros en el método de odometría, eje y

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

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 m en el eje X**, tal y como podemos ver en la figura 3.7 y **0.07 m 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.

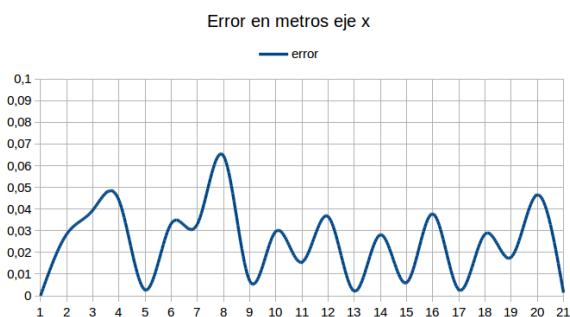


Figura 3.7: Error en metros en el método de Filtro de Partículas, eje x

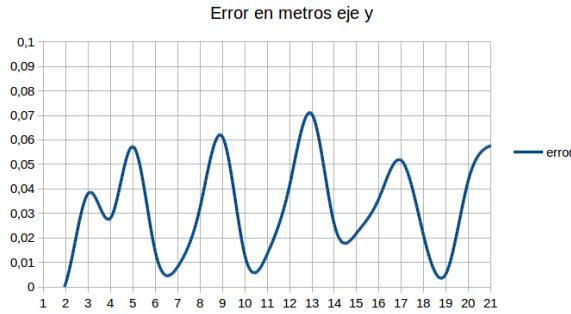


Figura 3.8: Error en metros en el método de Filtro de Partículas, eje y

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

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 m en el eje X**, tal y como podemos ver en la figura 3.9 y **0,073 m en el eje Y**, tal y como podemos ver en la figura 3.10, con respecto a los objetivos a los que tenía que llegar el robot.

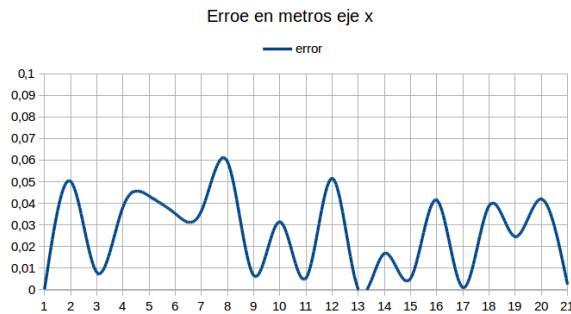


Figura 3.9: Error en metros en el método de Filtro de Kalman, eje x

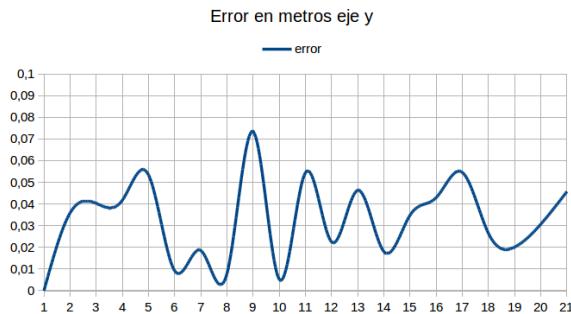


Figura 3.10: Error en metros en el método de Filtro de Kalman, eje y

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

3.2.4. Conclusiones del estudio previo de los algoritmos de localización

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 como conclusión que el método de localización basado en la odometría es el mejor en caso de que el movimiento de las ruedas no falle y de que se sepa la posición inicial exacta al comienzo de la prueba.

En el caso de que se probase en un entorno real, donde la odometría fallase o la posición al inicio no se conozca, los errores en el método de localización basado en la odometría serían mucho mayores que los otros algoritmos de localización. Esto será mostrado en el apartado de experimentos del filtro de partículas basado en información visual.

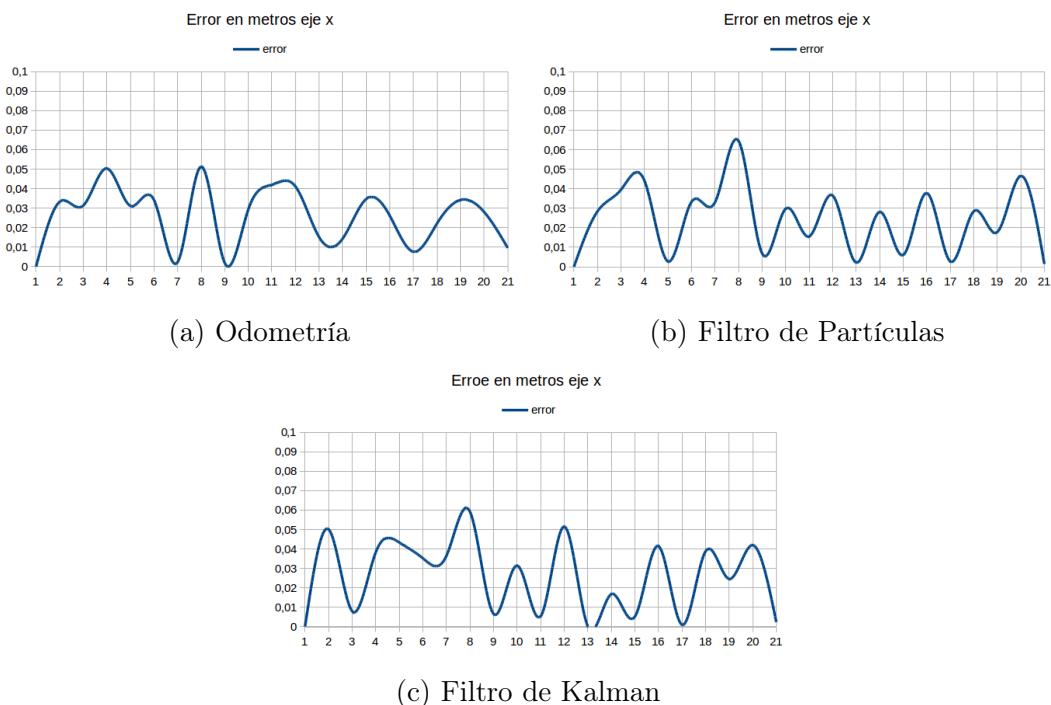


Figura 3.11: Comparación de errores en el eje x entre los diferentes métodos de localización

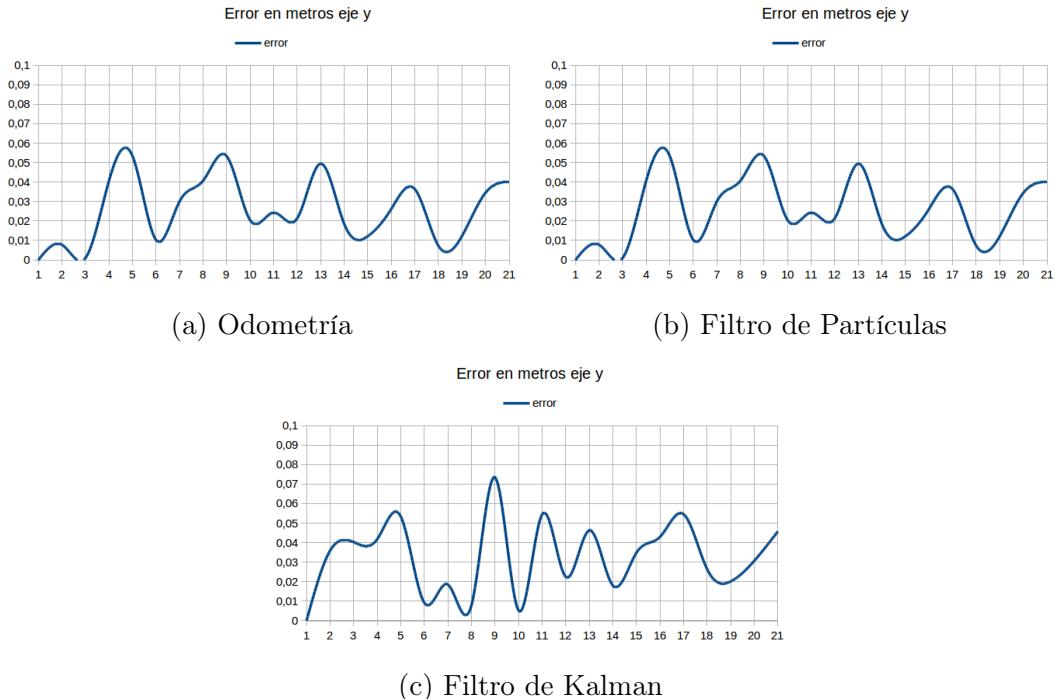


Figura 3.12: Comparación de errores en el eje y entre los diferentes métodos de localización

3.3. Desarrollo del trabajo

3.3.1. Introducción

En esta sección se explicará el desarrollo del trabajo para realizar un filtro de partículas basado en información visual.

Se incluirá:

- Preparar el entorno de pruebas en Gazebo y ROS para utilizarlo en el desarrollo y validación del filtro de partículas.
 - Mapear el entorno, para obtener un mapa que utilizar en la localización del robot.
 - Obtener imágenes de los sensores y de las partículas.
 - Calcular el peso de las partículas mediante las imágenes obtenidas.
 - Aplicar los anteriores pasos a la teoría del filtro de partículas para desarrollar el módulo de localización basado en un filtro de partículas con información visual.

3.3.2. Preparación del entorno

Para el desarrollo y validación del filtro de partículas se ha preparado un entorno (figura 3.13) utilizando 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.

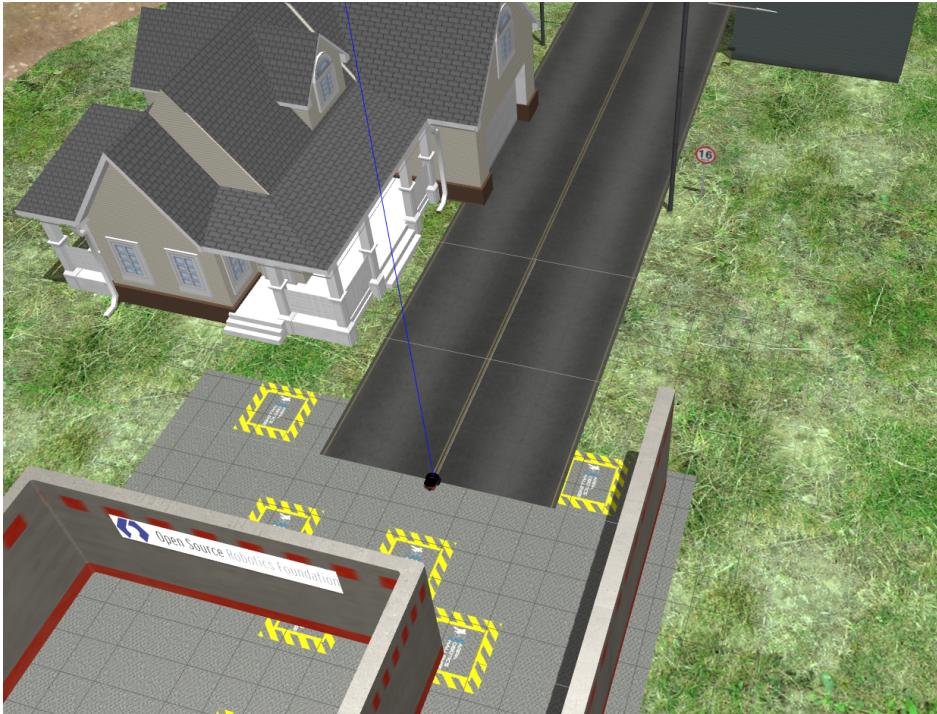


Figura 3.13: Entorno simulado en Gazebo

3.3.3. Mapeo del entorno

Para el desarrollo del módulo de localización es necesario realizar un mapeo 3d del entorno, para ello se ha utilizado el nodo `"rtabmap_ros"` [20] que permite realizar esta acción suscribiéndose mediante información visual.

Rtabmap utiliza un algoritmo basado en "SURF" [21] para obtener los puntos de interés de la información visual. Estos puntos de interés son utilizados agrupar toda la información en una sola nube de puntos con toda la información del entorno.

Para mapear el entorno 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 obtenga la velocidad que le pasamos mediante el comando anterior se ha añadido al modelo del robot el plugin `"differential_drive_controller"` [22].

Podemos ver como se va mapeando el entorno con la herramienta Rviz que permite

suscribirse a topics y visualizarlos. Rtabmap publicará la nube de puntos del entorno en un topic por lo que con Rviz se puede llevar un seguimiento tal y como vemos en la imagen 3.14.

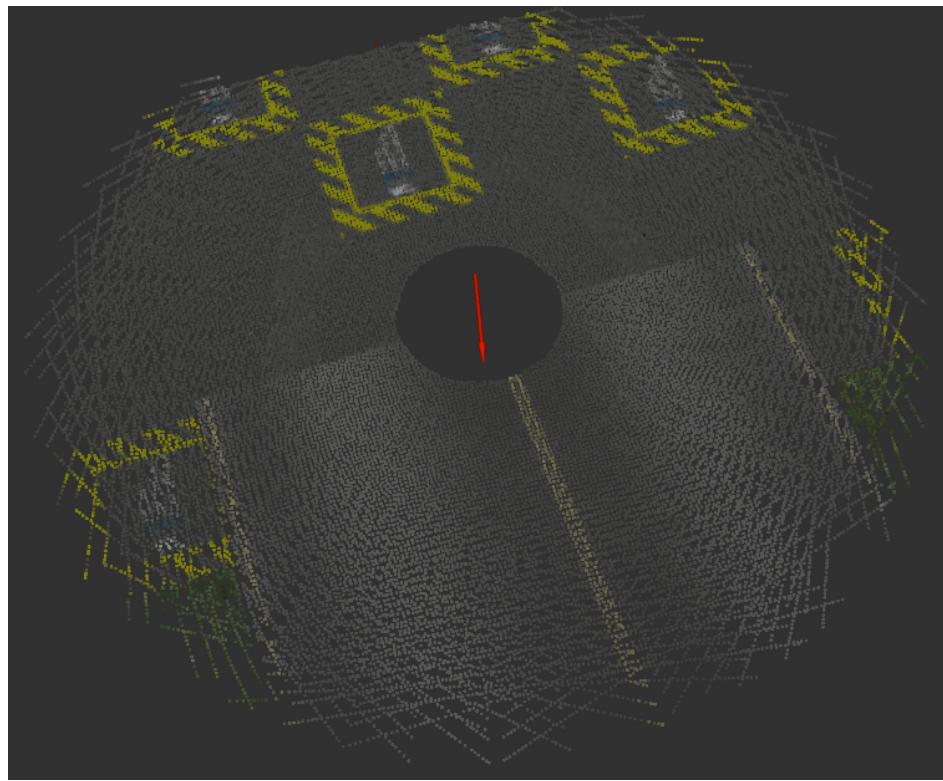


Figura 3.14: Nube de puntos obtenida al realizar un mapeo con rtabmap

3.3.4. Obtención de la nube de puntos de las partículas

Una vez que mapeado todo el entorno se necesita conocer la nube de puntos que se obtendría suponiendo que el robot se encuentra localizado en la posición (x,y,θ) siendo x la posición en el eje de la profundidad, y la posición en el eje horizontal y θ la dirección del robot.

La forma para obtener dicha nube de puntos consiste en trasladar la nube de puntos $(-x,-y,0)$ aplicando la transformación $M1$ a todos los puntos del entorno.

$$M1 = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Posteriormente rotar la nube de puntos $(0,0,\theta)$ aplicando la transformación $M2$ a todos los puntos del entorno.

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

Una vez aplicadas estas transformaciones solo es necesaria la nube de puntos que obtendría el sensor basándose en la profundidad y la anchura, así que se recorta en estas dimensiones, obteniendo como resultado la nube de puntos mostrada en la figura 3.15.

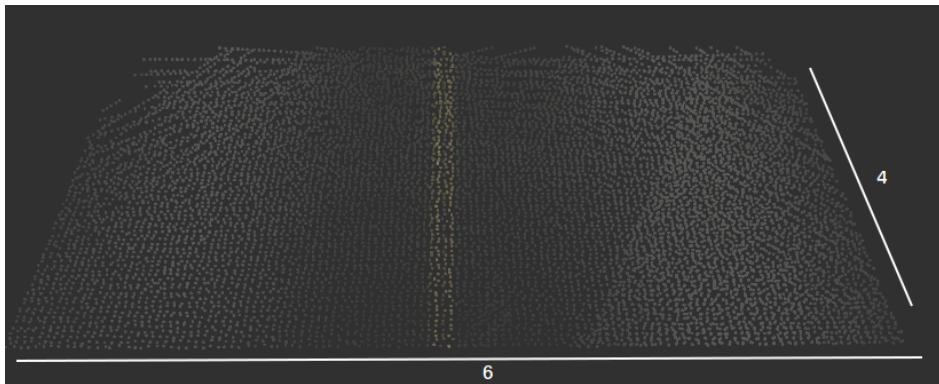


Figura 3.15: Nube de puntos obtenida recortando la nube de puntos obtenida mediante mapeo.

Para reducir la complejidad computacional se ha recortado la nube de puntos antes de comenzar a realizar las diferentes transformaciones, esta reducción se ha realizado mediante un recorte de 6 metros alrededor de la imagen, consiguiendo así que las transformaciones no sean aplicadas a los puntos que no son potencialmente importantes para visualizar la imagen del sensor.

Para reducir aún más la complejidad computacional, se ha realizado conjuntamente y simultáneamente la translación de la nube de puntos (M1) y la rotación de la nube de puntos (M2) mediante la multiplicación de matrices, obteniendo una matriz MT que realiza las anteriores operaciones en conjunto y simultáneamente.

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

MT se aplicaría a todos los puntos de la nube de puntos recortada anteriormente:

$$PN^1 = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & \cos(\alpha) * PosX - \sin(\alpha) * PosY \\ \sin(\alpha) & \cos(\alpha) & 0 & \cos(\alpha) * PosY + \sin(\alpha) * PosX \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

3.3.5. 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 3.16) y se ha proyectado sobre el plano z, obteniendo así una imagen en 2d (figura 3.17).

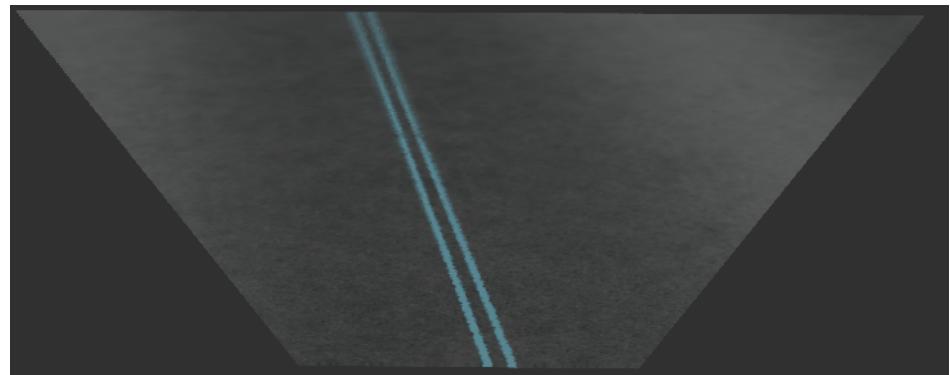


Figura 3.16: Nube de puntos obtenida por el sensor

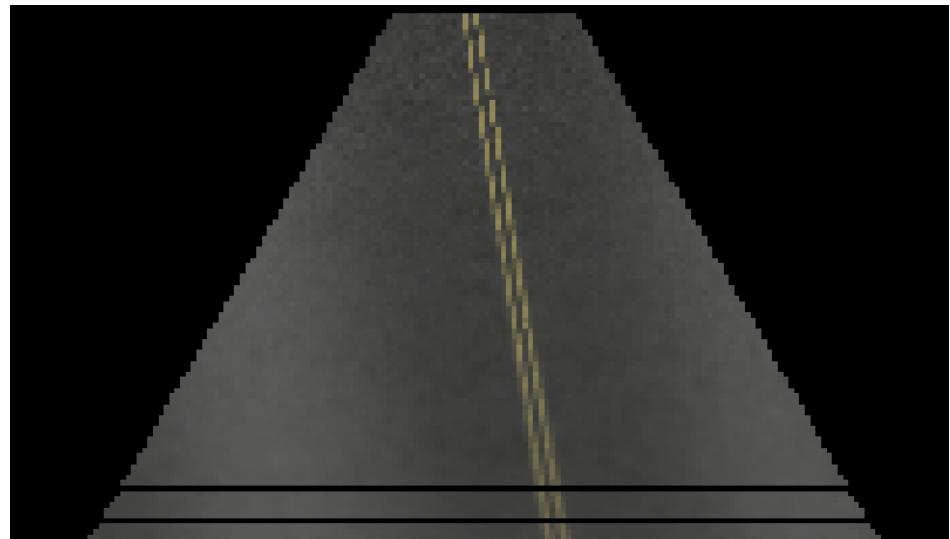


Figura 3.17: Imagen obtenida desde arriba proyectando la nube de puntos del sensor sobre el plano z

¹PN = Punto nuevo

3.3.6. Detección de líneas

En este experimento también se ha tratado de detectar las líneas para facilitar los cálculos.

Para detectar las líneas se han utilizado las imágenes producto de la proyección de las nubes de puntos, se han pasado a blanco y negro, se ha utilizado el algoritmo "Canny Edge Detection" [23], que permite encontrar bordes en la imagen y se han excluido los elementos de la imagen que no nos resultan útiles, para ello se ha elegido una zona de interés y se elimina todo elemento que no esté dentro de esta zona [3.18](#).

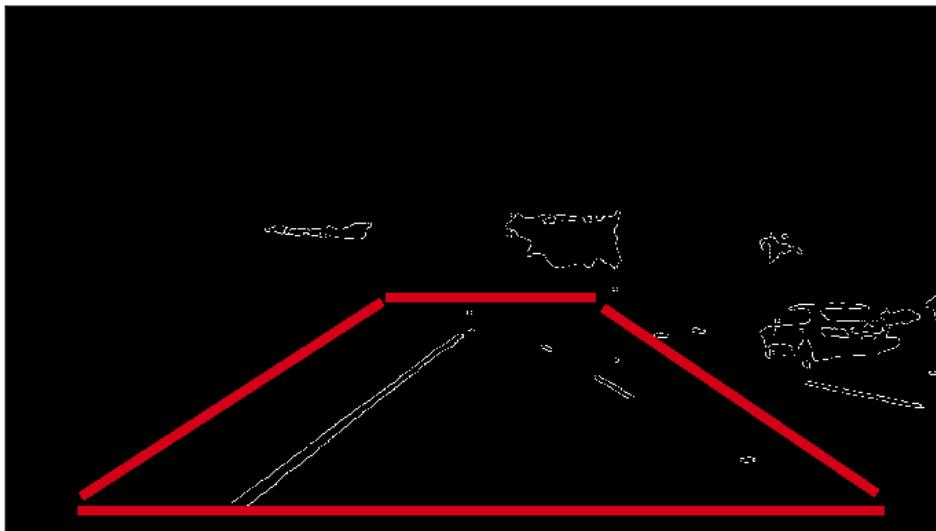


Figura 3.18: Zona de interés seleccionada de la imagen

Posteriormente se han seleccionado las regiones de los elementos que tienen bordes de la imagen en blanco y negro, obteniéndose la linea de la carretera en su totalidad, no solo los bordes y tendríamos las líneas detectadas para basarnos en la visualización a la hora de localizar el robot en el entorno.

Debido a las pruebas que se han realizado posteriormente se ha decidido no utilizar las líneas detectadas para la localización y utilizar la imagen integra.

3.3.7. 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 comparan las imágenes obtenidas anteriormente del sensor con las de las partículas.

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 partícula en pequeñas sub-imágenes, concretamente 16 sub-imágenes.

Para cada una de las sub-imágenes se calculan los valores medios de los canales RGB y posteriormente se comparan con los valores medios de las sub-imágenes obtenidas de la imagen del sensor mediante la función de coincidencia 3.1 donde FC es la función que devuelve el índice de coincidencia, S es la subimagen, RGB Son los componentes RGBs, VIS es el valor de la imagen del sensor y VIP el valor de la imagen de la partícula.

$$FC = \sum_{i=0}^S \sum_{j=0}^{RGB} |VIS_{ij} - VIP_{ij}| \quad (3.1)$$

Una vez conocido el índice de coincidencia se da 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 3.19.

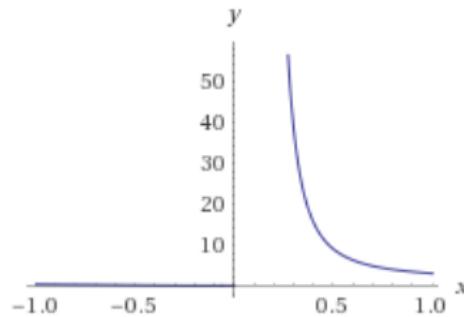


Figura 3.19: Función exponencial decreciente, $3^{1/x}$

Para obtener una función parecida a la figura 3.19 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.

$$f(IC) = IC/100 \quad (3.2)$$

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 3.20

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

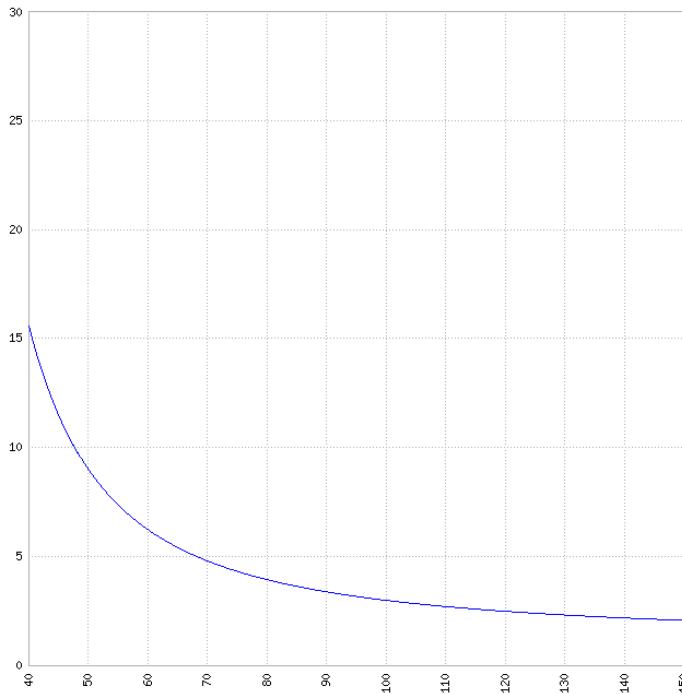


Figura 3.20: Función exponencial decreciente, $f(x) = 3^{1/(x/100)}$

3.3.8. Desarrollo del filtro de partículas

Tras establecer el entorno y desarrollar un método para calcular los pesos de las partículas, se comienza el desarrollo del módulo de ROS para la localización basado en el filtro de partículas con información visual.

El filtro de partículas diseñado seguirá la teoría básica de un filtro de partículas.

Fase de inicialización

El filtro de partículas tendrá una fase de inicialización donde el conjunto de partículas se genera aleatoriamente o mediante una posición aproximada utilizando una distribución normal con μ como media y σ^2 como varianza (algoritmo 3.1).

Algoritmo 3.1: Fase de inicialización

```

for each number of particles do:
     $x_t^{[m]} \sim N(\mu, \sigma^2)$ 
    set  $x_t^{[m]}$  to  $X_t$ 
end for

```

En el desarrollo del filtro de partículas se ha inicializado el conjunto de partículas asignando partículas aleatoriamente siguiendo una distribución normal $N(\mu, \sigma)$, donde μ

es la posición dada por el sensor GPS y σ es la varianza que se necesite para corregir el error del GPS, consiguiendo distribuir las partículas por esa posición.

Para el cálculo de σ ha sido necesario conocer la proporción de los datos en una distribución normal (figura 3.21). Pudiendo asignar el valor de σ según el error del GPS.

3.21.

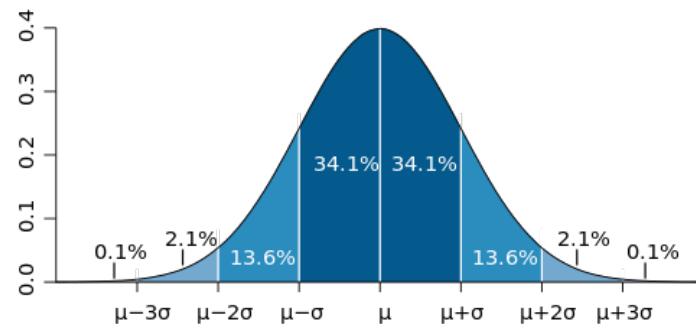


Figura 3.21: Porcentaje de datos en la distribución normal

Una vez inicializado el conjunto de partículas quedaría como se muestra en la figura 3.22 donde se muestra la posición del robot como una flecha roja grande y el conjunto de partículas disperso por el entorno mediante flechas pequeñas rojas.

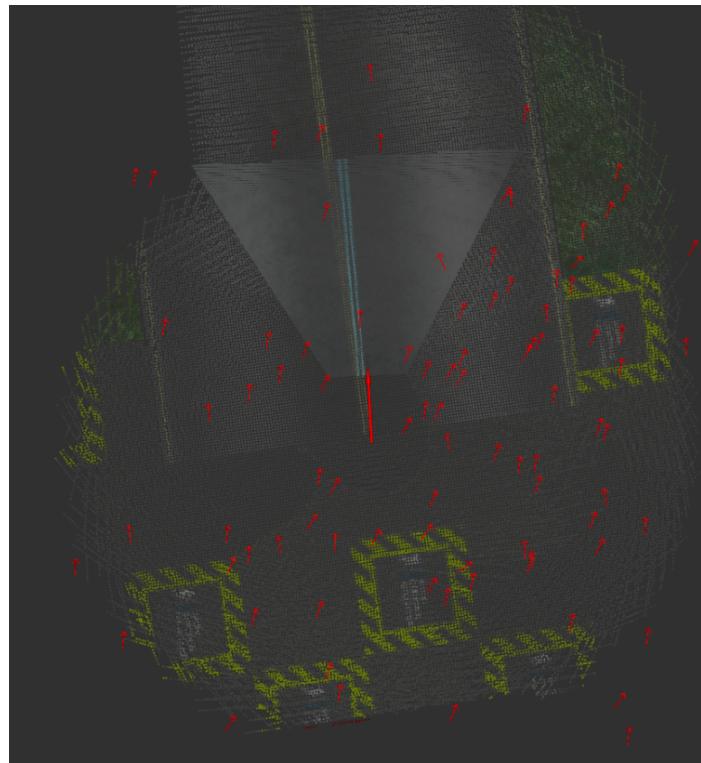


Figura 3.22: Partículas tras ser inicializadas

Fase de predicción

El filtro de partículas contiene una fase de predicción que será la encargada de, dependiendo de las entradas del robot u_t , predecir su localización (algoritmo 3.2) modificando las partículas mediante la aplicación de las funciones 3.4, 3.5 y 3.6.

Algoritmo 3.2: Fase de predicción

$$x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$$

$$P^2_{X_t} = P_{X_{t-1}} + (|X_t - X_{t-1}| + |Y_t - Y_{t-1}|) * \cos(\alpha^3) \quad (3.4)$$

$$P_{Y_t} = P_{Y_{t-1}} + (|X_t - X_{t-1}| + |Y_t - Y_{t-1}|) * \sin(\alpha) \quad (3.5)$$

$$P_{Theta_t} = P_{Theta_{t-1}} + Theta_t - Theta_{t-1} \quad (3.6)$$

Tras la predicción de la posición del robot se añade un ruido gaussiano de media la posición de la partícula y varianza la que necesaria para que las partículas no converjan a un punto fijo.

Fase de actualización

A continuación el filtro de partículas realia la fase de actualización de pesos de las partículas (ecuación 3.7) mediante la comparación de imágenes obtenidas por el sensor y mediante el mapeo utilizando la función de coincidencia 3.8.

$$w_t^{[m]} = p(z_t | x_t^{[m]}) \quad (3.7)$$

$$FC = \sum_{i=0}^S \sum_{j=0}^{RGB} |VIS_{ij} - VIP_{ij}| \quad (3.8)$$

Posteriormente, la posición de la partícula calculada en la fase de predicción con su respectivo peso es añadida al conjunto de partículas (ecuación 3.9).

$$\bar{X}_t = \bar{X}_t + (x_t^{[m]}, w_t^{[m]}) \quad (3.9)$$

²Partícula

³Dirección de la partícula

Fase de "re-sampling"

El algoritmo, tras pasar la fase de actualización de pesos mediante la comparación de imágenes, llega a la fase de "re-sampling" donde se aplicará el algoritmo 3.3.

Algoritmo 3.3: Filtro de Partículas

```

for each number of particles do:
    draw i with probability  $\alpha * w_t^{[i]}$ 
    add  $x_t^{[i]}$  to  $X_t$ 
end for
return  $X_t$ 
```

En esta fase se aplica el método de resample "Resample Wheel" donde aquellas partículas que mejor se ajusten a las medidas (partículas de mayor peso normalizado) darán lugar a nuevas partículas con mayor probabilidad consiguiendo el conjunto de partículas que es probablemente el más correcto.

En esta fase también se obtiene la localización según el algoritmo, pues nos quedaremos con la partícula de mayor peso que indicará la posición del robot tal y como se ve en la figura 3.23 como una flecha de color verde junto con la posición real del robot mostrada como una flecha de color rojo.



Figura 3.23: Mejor partícula del conjunto y posición real

Tras esta fase se pasa a la fase de predicción para seguir localizando el robot realizando el esquema mostrado en la figura 3.24.

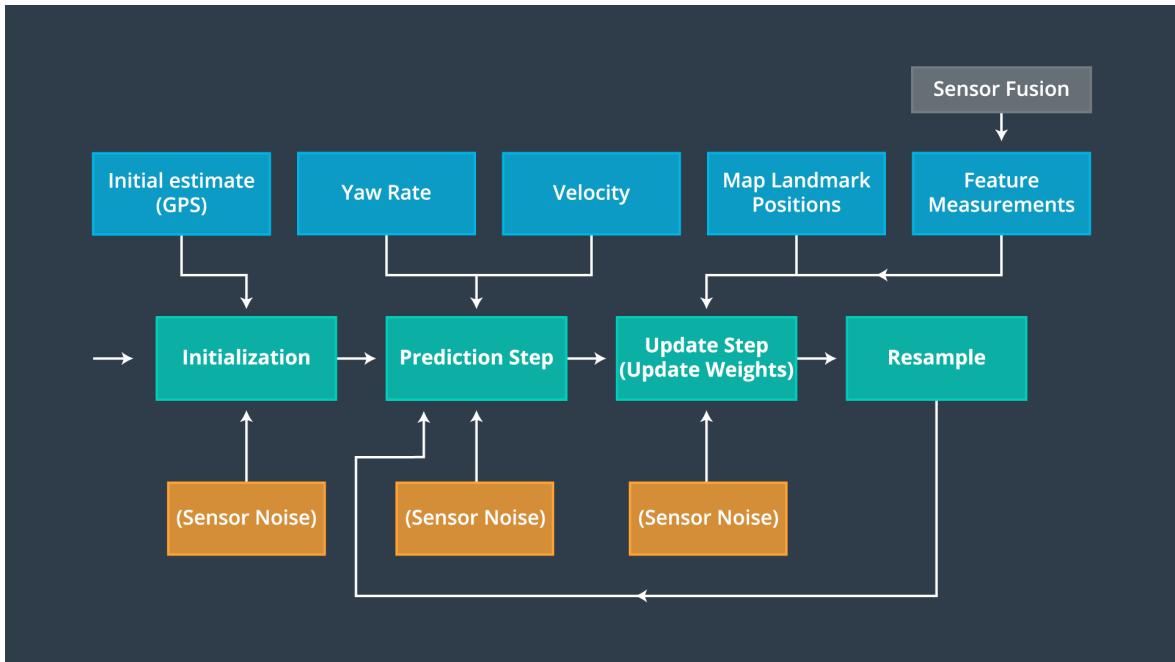


Figura 3.24: Esquema del filtro de partículas

3.3.9. Paralelización del código

En el proyecto se ha intentado añadir concurrencia al código, concretamente se ha añadido concurrencia con OpenMP [24] al apartado encargado de proyectar las nubes de puntos de las partículas en imágenes vistas desde arriba.

Esto no ha supuesto una mejora debido a la utilización de varios hilos por parte de otros nodos en el ordenador, pues ya estabamos utilizando todos los núcleos del procesador para realizar la localización.

Por ello, para mejorar la velocidad de computo tendríamos dos opciones más, utilizar CUDA [25] para aprovechar la tarjeta gráfica al realizar cálculos o utilizar Grid Computing.

Capítulo 4

EXPERIMENTOS Y RESULTADOS

En este capítulo se mostrarán los tiempos de cada una de las fases del filtro de partículas, los experimentos realizados y los resultados de los mismos.

4.1. Tiempos de cada una de las fases del filtro de partículas

Tras desarrollar el filtro de partículas se ha querido comprobar la complejidad computacional del algoritmo, procediendo a calcular los tiempos de cada fase para detectar posibles mejoras en las fases más pesadas.

Los tiempos obtenidos realizando una media de 20 iteraciones en el algoritmo han sido los mostrados en la tabla 4.1.

Tabla 4.1: Tiempos medios en segundos de cada una de las fases del algoritmo

Tiempos	Media	Varianza
Fase de inicialización	0,000161	0,000005
Fase de predicción	0,0002330526	1,40E-09
Fase de actualización de pesos	23,63	1,42
Calcular la imagen de la partícula	0,231965	0,0003605
Calcular la imagen del sensor	0,0479708	2,086677E-05
Comparar imágenes	0,000447	4,718639E-08
Fase de resample	9,99E-05	9,03E-10
Iteración en el filtro de partículas	23,63	1,42296

En la figura 4.1 se muestra una gráfica con la media de los tiempos de 20 iteraciones en el bucle del filtro de partículas.

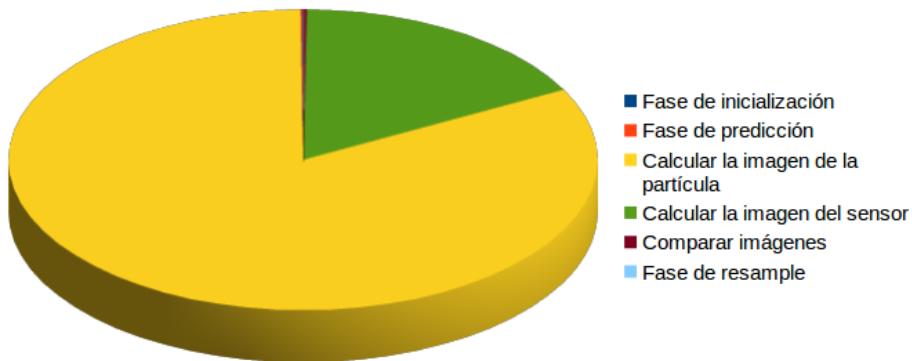


Figura 4.1: Tiempos medios de cada una de las fases del filtro de partículas

Se puede ver que las fases más pesadas computacionalmente son aquellas encargadas de obtener la imagen de la partícula y del sensor debido a la gran cantidad de operaciones que se realizan, y que crecerán proporcionalmente según la resolución que requiera la imagen y según la cantidad de puntos de la nube de puntos de todo el entorno.

Por ello y cómo se dijo anteriormente se intentó utilizar la concurrencia sin éxito con OpenMP y quedando como posible opción lanzar los hilos mediante CUDA consiguiendo paralelizar utilizando la GPU.

4.2. Resultados obtenidos del experimento

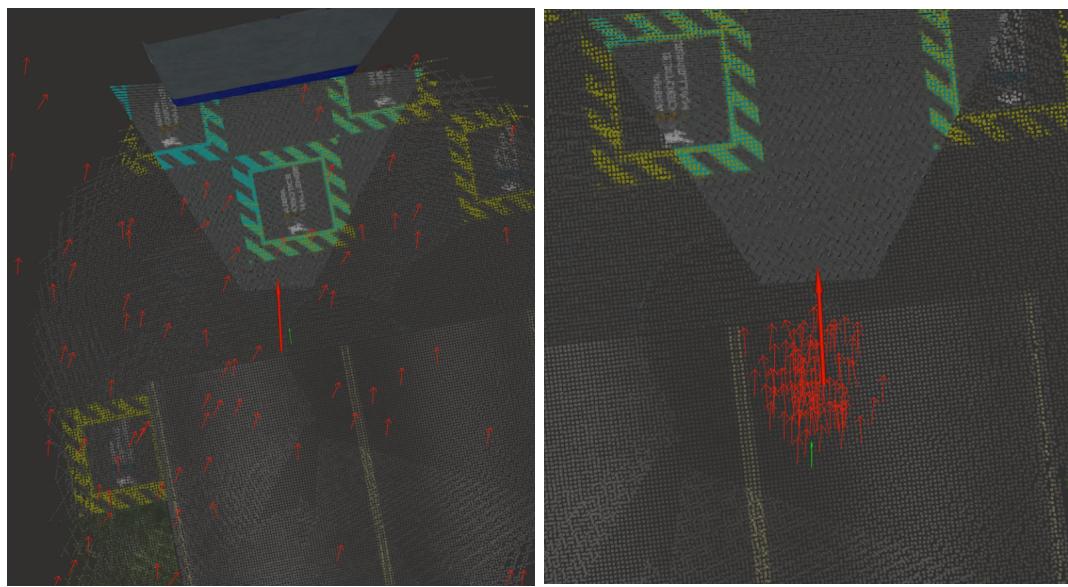
En esta sección se probará el funcionamiento del filtro de partículas en diferentes condiciones para probar la robustez del método de localización.

Estas pruebas de robustez han consistido en añadir error en la odometría y añadir obstáculos en el entorno que no han sido mapeados anteriormente.

4.2.1. Resultados de la localización sin error en la odometría

Tras desarrollar el filtro de partículas se ha decidido probar en diferentes entornos, el primero es utilizando una odometría perfecta, es decir, el movimiento que realiza el robot es perfecto, no existen defectos como deslizamientos en las ruedas.

En este experimento se ha movido el robot realizando un cuadrado en el entorno. Se puede ver la evolución del conjunto de partículas tras varias iteraciones en la figura 4.2.



(a) Conjunto de partículas tras iniciarse en el experimento
(b) Conjunto de partículas tras finalizar el experimento

Figura 4.2: Evolución del conjunto de partículas

Si comparamos la posición y orientación de todas las partículas con respecto a la posición correcta en cada iteración se ve la evolución que va teniendo el conjunto de partículas, teniendo un error cada vez menor tal y como podemos ver en la figura 4.3.

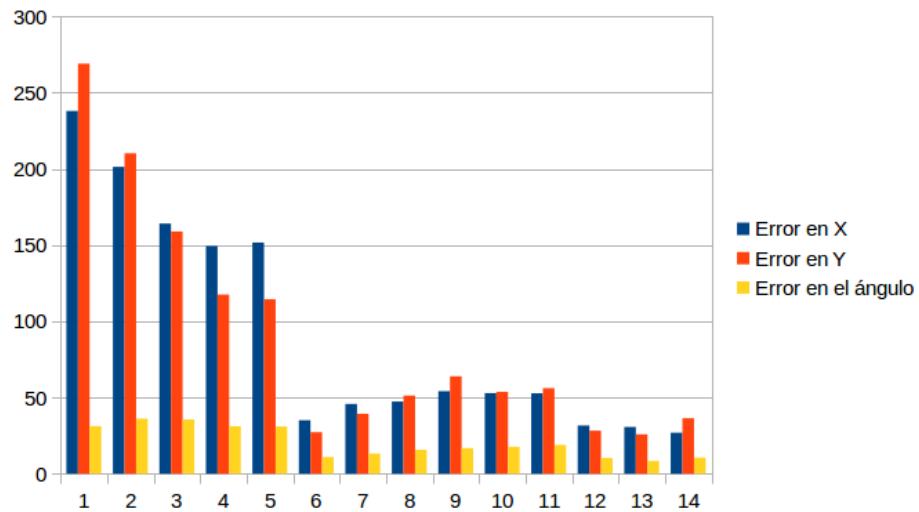


Figura 4.3: Error total en metros y radianes de todo el conjunto de partículas con respecto a la posición real en cada iteración

De igual manera, si se compara la posición y orientación de las partículas elegidas como localización real del algoritmo con la posición real del robot obtendríamos algo similar a lo anterior tal y como se muestra en la imagen 4.4.

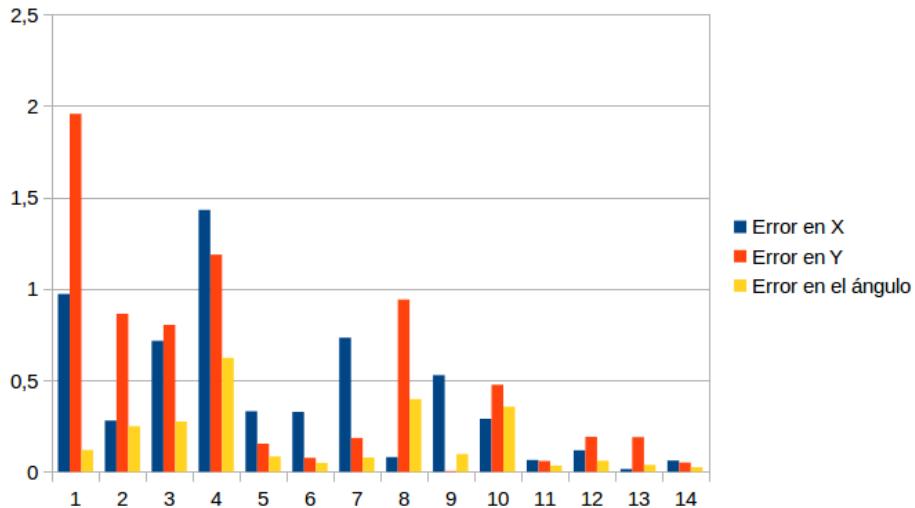


Figura 4.4: Error en metros y radianes de la mejor partícula con respecto a la posición real en cada iteración

4.2.2. Resultados de la localización añadiendo error en la odometría

Para probar la efectividad de este filtro de partículas se ha decidido simular que la odometría obtenida tiene error, es decir, se simulará que el robot detecta que realiza según la odometría movimientos lineales y angulares mayores que los que realmente realiza, concretamente se ha añadido un error de 0.01 m/s a la hora de realizar movimientos lineales y 6°/s a la hora de realizar movimientos angulares.

Tras realizar un cuadrado en el entorno con el robot se ha obtenido un error en las partículas mostrado en la figura 4.5.

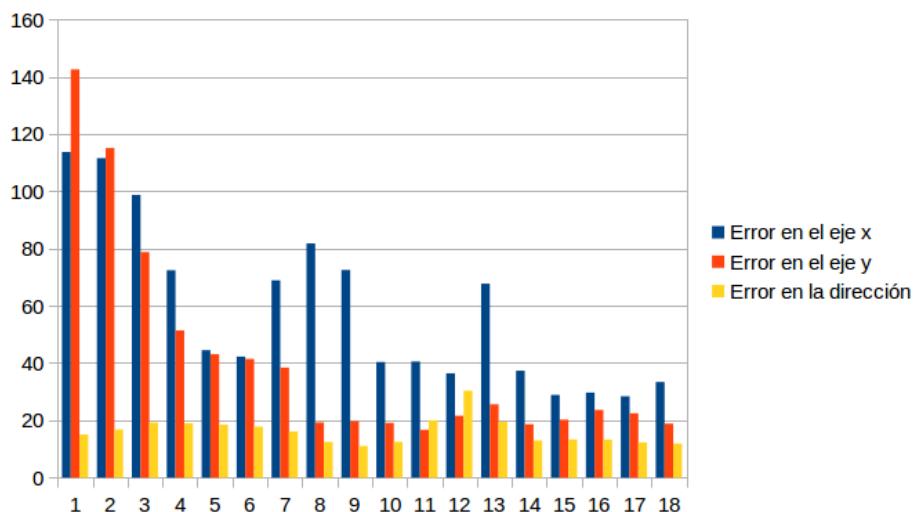


Figura 4.5: Error en metros del conjunto de partículas con respecto a la posición real en cada iteración

El error de la localización basada en la odometría con respecto a la posición real del robot se muestra en la figura 4.6.

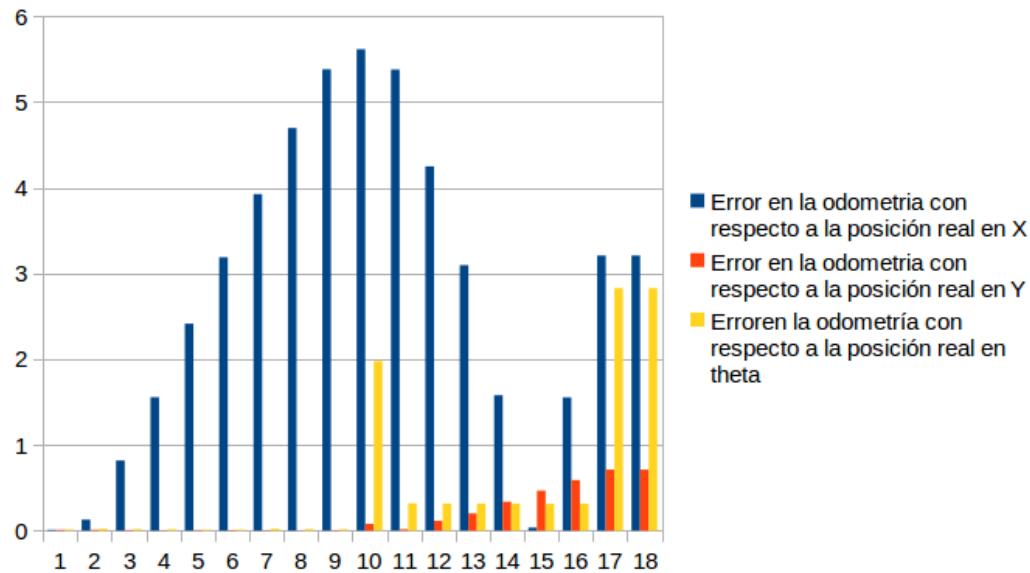


Figura 4.6: Error en metros y radianes de la localización basada en la odometría con respecto a la posición real en cada iteración

Se puede ver como el error va aumentando según el tiempo va transcurriendo, por lo que a mayor tiempo ejecutando el filtro de partículas y añadiendo movimiento al robot, mayor será el error de la odometría, llegando a conseguir que el robot se pierda completamente.

La localización obtenida por el algoritmo de localización basado en el filtro de partículas con información visual obtendría unos errores con respecto a la posición real mostrados en la figura 4.7, como se puede ver, el error se reduce en comparación con respecto a la odometría, aún así, debido a el tiempo de cálculo de una iteración, las partículas modifican en gran cantidad su posición y dirección, de forma que en algunos momentos se pierde, pero tras un tiempo el robot vuelve a encontrar su posición con este algoritmo. Esto mejoraría reduciendo el tiempo de cálculo de obtener las imágenes de las partículas o paralelizando los cálculos sobre esta obtención de las imágenes de las partículas.

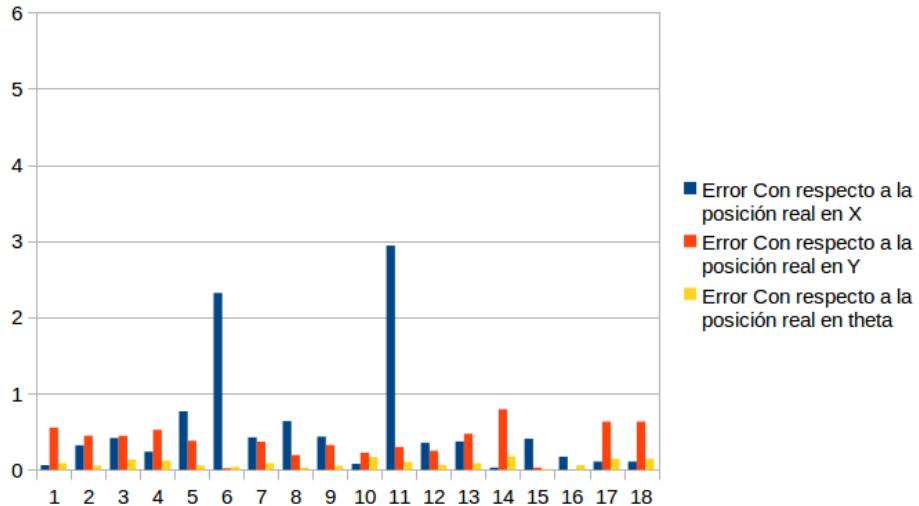


Figura 4.7: Error en metros y radianes de la localización del algoritmo con respecto a la posición real en cada iteración

Al final, si utilizamos el visualizador de topics Rviz, y publicamos la posición real del robot con una flecha azul, la posición obtenida según la odometría del robot como una flecha roja de gran tamaño, las diferentes partículas del conjunto como flechas rojas y la posición que el algoritmo devuelve en el último instante del recorrido obtenemos la visualización gráfica mostrada en la figura 4.8.

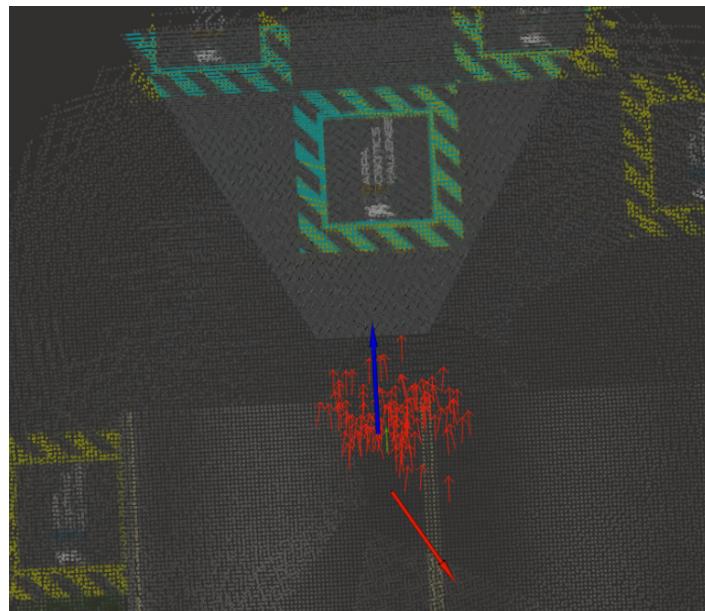


Figura 4.8: Visualización de la información de los elementos del filtro de partículas en la última iteración

4.2.3. Resultados de la localización sin error en la odometría y añadiendo obstáculos

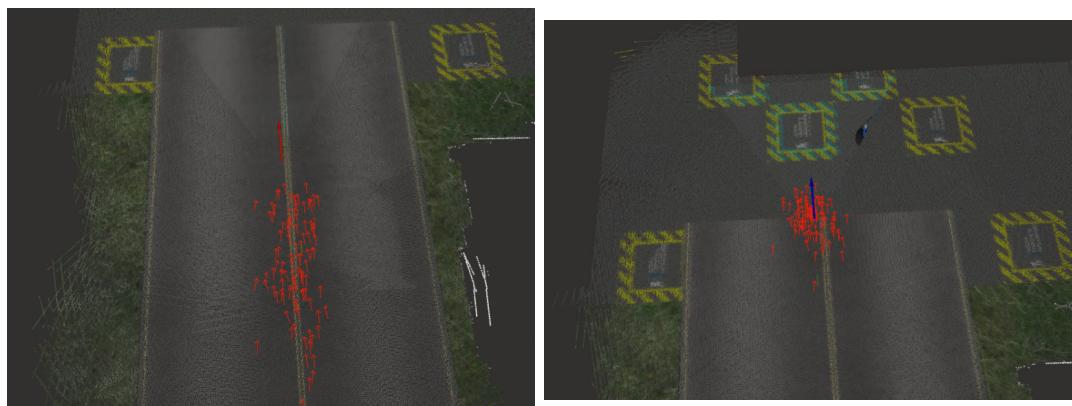
Ahora pasaremos a ver el funcionamiento del algoritmo de localización añadiendo obstáculos para dificultar la localización del robot. Este aspecto no supone un mayor problema debido al método de comparación de imágenes basado en histogramas, pues no tiene en cuenta aquellas zonas de la imagen donde no hay elementos proyectados.

Para añadir obstáculos lo que se ha realizado es añadir algunos elementos como conos en el entorno que no han sido mapeados anteriormente tal y como se puede ver en la figura 4.9.



Figura 4.9: Entorno gazebo con obstáculos

La prueba realizada en este caso ha sido la de avanzar por la carretera para demostrar también que partículas pueden obtener imágenes muy similares y por tanto tener pesos parecidos y crear ambigüedad en la localización. Esto se resolvería posteriormente, cuando alguna partícula junto al sensor encuentre un elemento distintivo y las demás partículas pierdan su peso, ejemplo que se puede ver en la figura 4.10.



(a) Conjunto de partículas con ambigüedad en la localización (b) Conjunto de partículas resolviendo ambigüedad en la localización

Figura 4.10: Corrección de ambigüedad

Analizando el error del conjunto de partículas en este experimento (figura 4.11) y el error de la localización (figura 4.12), se puede ver la parte donde aparece ambigüedad, produciéndose un error elevado en el eje x y como esta se va corrigiendo.

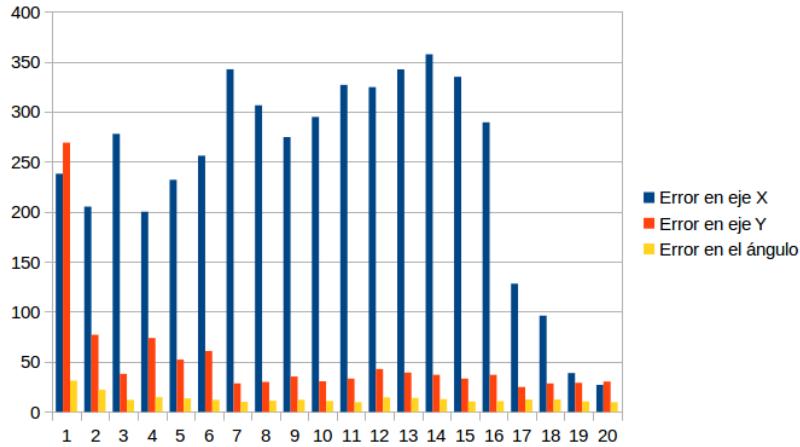


Figura 4.11: Errores en metros y radianes del conjunto de partículas en el experimento en el que se añaden obstáculos

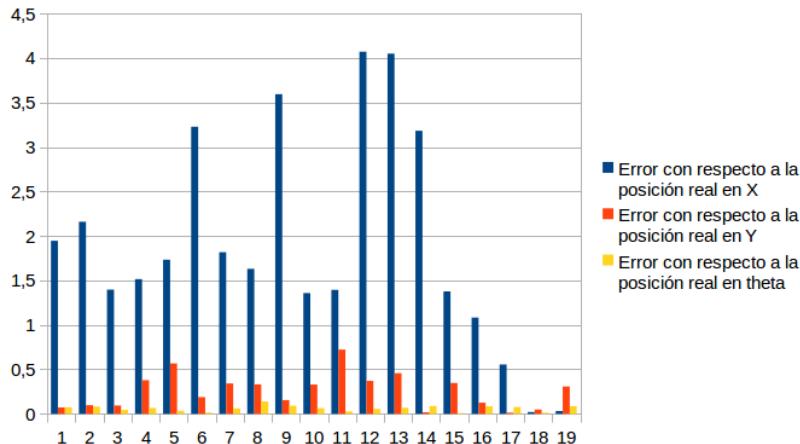


Figura 4.12: Errores en metros y radianes de localización en el experimento en el que se añaden obstáculos

4.2.4. Resultados añadiendo error en la odometría y añadiendo obstáculos

En esta sección se utilizará el mismo entorno que se utilizó en la sección anterior, aquel entorno en el que se añadían obstáculos pero con el añadido de que se utilizará una odometría con error para ver como responde el algoritmo de localización.

Analizando el error de la odometría (figura 4.13) se ve como este va aumentando según el tiempo va transcurriendo, consiguiendo que el robot se pierda completamente.

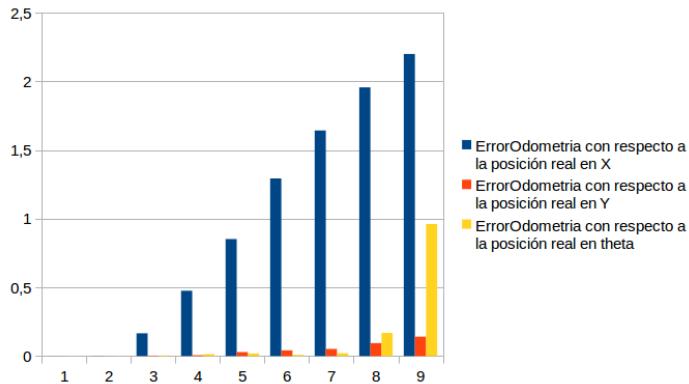


Figura 4.13: Errores en metros y radianes de la odometría con respecto a la posición real del robot

Por el contrario, el error de localización junto con el del conjunto de partículas se estabiliza como se ve en las figuras 4.15 y 4.14.

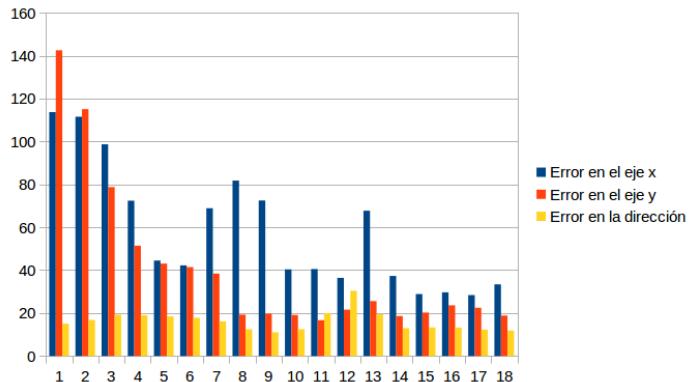


Figura 4.14: Errores totales en metros y radianes del conjunto de partículas en la localización con errores de odometría y con obstáculos

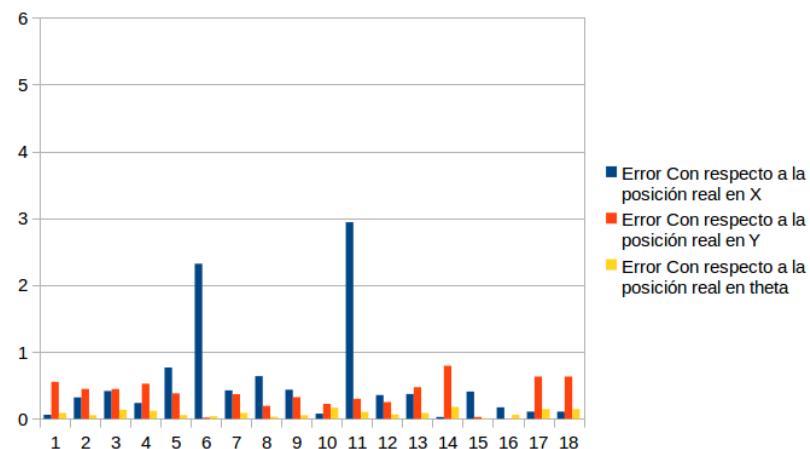


Figura 4.15: Errores en metros y radianes de localización con errores de odometría y con obstáculos

4.3. Conclusiones del experimento

Con las diferentes pruebas realizadas en el apartado anterior sobre el filtro de partículas que utilizaba información visual para localizar al robot, se ha visto la importancia del uso de un algoritmo de localización, pues al usarse se ha visto como el error acumulado en cada iteración se va corrigiendo y no va aumentando continuamente tal y como haría un método de localización basado únicamente en la odometría en el caso de que este tuviese error.

Además, en estas pruebas se ha podido comprender la gran utilidad de ROS en el ámbito de la robótica.

En cuanto a la representación de los datos gráficamente es algo que no implementa ROS y que debemos hacerlo manualmente en contraposición a Matlab.

En 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 distintas soluciones a los problemas.

En el filtro de partículas se debe tener en cuenta que a mayor cantidad de partículas más duro se hace el trabajo computacionalmente, más exacta y precisa es la solución al problema de la localización del robot, por ello, 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.

Por último, tras programar el filtro de partículas ha sorprendido 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 5

CONCLUSIONES Y PROPUESTAS

5.1. Conclusiones

El trabajo realizado se ha centrado en el 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 y otras son poco precisas.

Se ha aprendido una gran cantidad de conceptos relativos a la robótica y se ha estudiado el uso de herramientas como Gazebo y ROS.

Se ha tratado con un algoritmo complejo computacionalmente hablando debido a la utilización de imágenes para la localización, lo que ha hecho plantear diversas formas de mejorar la eficiencia del código y plantear el paralelismo en el proyecto.

5.2. Competencias cubiertas

En este trabajo se han utilizado técnicas de las asignaturas de la rama de computación como son la visión artificial, la robótica y la informática gráfica. Además se han tocado transversalmente otras asignaturas como la estadística.

La robótica ha sido el centro del trabajo, centrando el trabajo en herramientas como ROS, Gazebo y estudiando una parte del temario de una forma más profunda, concretamente la de los filtros paramétricos y los no paramétricos, comprobándose empíricamente la gran utilidad para la localización.

Se ha comprobado que además de un buen algoritmo de localización se necesita tener buenos métodos para el cálculo de pesos y buenos sensores para que estos funcionen con excepcionalidad.

Se han tratado con sensores de diversos tipos como cámaras estéreo, cámaras de profundidad y sensores lidar.

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

Por otro lado, las técnicas de informática gráfica también han sido utilizadas para realizar transformaciones geométricas en las nubes de puntos en un entorno en 3 dimensiones.

Otras disciplinas tratadas en el trabajo han sido el diseño de algoritmos y la aplicación de técnicas de sistemas inteligentes, evaluando y reduciendo la complejidad computacional de un problema, consiguiendo así, resolver, desarrollar e implementar una solución a dicho problema.

5.3. Trabajo futuro

En el trabajo futuro se estudiarán más algoritmos para la comparación de imágenes y obtener así mejores resultados.

También se buscará reducir la complejidad computacional reduciendo los elementos mapeados de la imagen de forma que solamente se utilicen los elementos que dan mayor información del entorno, como las líneas de la carretera.

Por último se buscará aumentar la velocidad del algoritmo para mejorarlo sustancialmente mediante paralelismo utilizando herramientas como CUDA, pudiendo, de esta manera, reducir el tiempo en un número inversamente proporcional al número de hilos que se consigan ejecutar en paralelo.

Bibliografía

- [1] ROS, “Navigation Stack,” 2017. [xi](#), [24](#)
- [2] ElectronicTeacher, “What is Robotics?,” 2000. [1](#)
- [3] ROS, “Tutoriales de ROS, Robot Operating System,” 2010. [1](#)
- [4] Gazebo, “Gazebo, Robot simulation made easy,” 2009. [2](#)
- [5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. 2005. [2](#)
- [6] S. Thrun, “Particle Filters in Robotics,” tech. rep., 2002. [2](#), [3](#), [12](#)
- [7] T. Lacey, “Tutorial: The Kalman Filter,” tech. rep., 2015. [2](#), [3](#)
- [8] J. R. d. G. Pascual, “Robótica: Estado del arte,” tech. rep., 2007. [5](#)
- [9] S. Thurn, *Particle Filter, Resampling Wheel*, vol. 2012. 2005. [13](#), [16](#)
- [10] D. Fox, “KLD-sampling: Adaptive particle filters,” *Advances in neural information processing systems*, vol. 14, no. 1, pp. 713–720, 2001. [13](#)
- [11] Erle Robotics S.L., *Conceptos de ROS*. PhD thesis, 2013. [22](#)
- [12] ROS, “Twist Message,” 2018. [24](#)
- [13] K. Schwaber and J. Sutherland, “La Guía Definitiva de Scrum: Las Reglas del Juego,” tech. rep., 2016. [27](#)
- [14] ROS, “Gmapping,” 2019. [30](#)
- [15] W. Burgard, C. Stachniss, K. Arras, and M. Bennewitz, “Introduction to Mobile Robotics SLAM,” Tech. Rep. June, 2010. [30](#)
- [16] ROS, “Rviz,” 2018. [31](#)
- [17] ROS, “LaserScan Message,” 2018. [32](#)
- [18] ROS, “Odometry Message,” 2018. [32](#)

- [19] ROS, “Map Server,” 2018. [32](#)
- [20] ROS, “Rtab-map,” 2019. [38](#)
- [21] A. Mordvintsev and K. Abid, “Introduction to SURF,” 2013. [38](#)
- [22] Gazebo, “Gazebo Plugins,” 2014. [38](#)
- [23] OpenCV, “Canny Edge Detection,” *Página oficial OpenCV modules*, no. 2002, p. 1, 2015. [42](#)
- [24] OpenMP, “Application Programming Interface,” tech. rep., 2016. [48](#)
- [25] Nvidia, “CUDA,” 2007. [48](#)

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.